INAUGURAL-DISSERTATION

zur Erlangung der Doktorwürde der

Naturwissenschaftlich-Mathematischen Gesamtfakultät

der Ruprecht-Karls-Universität Heidelberg

vorgelegt von

Diplom-Mathematiker Thorsten Bonato

aus Münster (Westfalen)

Tag der mündlichen Prüfung: 14. Juni 2011

Contraction-based Separation and Lifting for Solving the Max-Cut Problem

Gutachter: Prof. Dr. Gerhard Reinelt
Prof. Dr. Dr. h. c. Hans Georg Bock

Zusammenfassung

Das Max-Cut-Problem ist ein auf ungerichteten gewichteten Graphen definiertes **NP**-schweres kombinatorisches Optimierungsproblem. Es besteht darin, eine Teilmenge der Knoten des Graphen zu bestimmen, so dass das summierte Gewicht der Kanten zwischen der Teilmenge und ihrem Komplement maximiert wird. In der vorliegenden Arbeit stellen wir einen neuen Separierungsansatz vor, der im Rahmen eines Branch-and-Cut-Algorithmus verwendet werden kann, um Max-Cut-Probleme optimal zu lösen. Die Methode basiert auf der Kontraktion des Graphen und erlaubt die effiziente Separierung sogenannter Ungerader-Kreis-Ungleichungen. Darüber hinaus beschreiben wir Techniken, um im bereits kontrahierten Graphen gegebenenfalls fehlende Kanten hinzuzufügen. Dies ermöglicht es, Max-Cut-Probleme auf dünn besetzten Graphen unter Verwendung von Methoden für vollständige Graphen zu lösen, die ansonsten nicht anwendbar gewesen wären. Wir untersuchen die theoretischen Aspekte dieses kombinierten Ansatzes und erläutern seine Umsetzung innerhalb eines Branch-and-Cut-Systems. Abschließend bewerten wir die Leistungsfähigkeit unserer Separierungsroutine anhand einer Vielzahl unterschiedlicher Testinstanzen.

Abstract

The max-cut problem is an **NP**-hard combinatorial optimization problem defined on undirected weighted graphs. It consists in finding a subset of the graph's nodes such that the aggregate weight of the edges between the subset and its complement is maximized. In this doctoral thesis we present a new separation approach to be used within a branch-and-cut algorithm for solving max-cut problems to optimality. This method is based on graph contraction and allows the fast separation of so-called odd-cycle inequalities. In addition, we describe techniques to add possibly missing edges to an already contracted graph. This allows solving max-cut problems on sparse graphs by using methods that were originally intended for complete graphs and could not have been applied otherwise. We investigate the theoretical aspects of this combined approach and also explain its realization within a branch-and-cut framework. Finally, we evaluate the performance of our separation procedure on a variety of test instances.

Acknowledgment

Above all I want to thank my advisor Prof. Gerhard Reinelt for raising my interest in combinatorial optimization as well as for his support and guidance over the years. His scientific thoroughness and profound knowledge were truly inspiring.

I had the honor of working with Dr. Giovanni Rinaldi. He was always glad to share his expertise in general as well as his insights into the max-cut problem in particular. I want to thank him for his enthusiasm and for the productive yet enjoyable time I spent at his institute in Rome.

The members of my work group provided a pleasant working environment. Over the years, I shared the office with Dr. Marcus Oswald and Stefan Wiesberg. Thanks for the relaxed atmosphere and for your opinions on both important and unimportant matters. I am grateful to Marcus for his ability to convey complex concepts and for always being cheerful, enthusiastic, and supportive. Furthermore, I owe Stefan respect and gratitude for all the effort he put into proofreading my entire thesis; of course, I take full responsibility for any remaining errors. Dr. Hanna Seitz with her talent to encourage and motivate people was always a pleasure to work with. I am thankful for our conversations and for her help in putting large problems into perspective. Our secretaries, Catherine Proux-Wieland and Karin Tenschert, as well as our system administrators, Georgios Nikolis, Adrian Dempwolff, and Lucas Appelmann, took care of the administrative needs and kept the machines running. I deeply appreciate all their help.

Furthermore, I would like to thank Dr. Christoph Buchheim, Dr. Frauke Liers, and Dr. Marcus Oswald for sharing their insights into target cuts and for providing their target cut software framework.

It is also a pleasure to thank the members of the Operations Research work group at the University of Klagenfurt for the great time I spent with them. I owe special gratitude to the head of the group, Prof. Franz Rendl, as well as to Dr. Angelika Wiegele and Anna Perdacher for their warm welcome, their expertise, and their support.

Last but certainly not least, my heartfelt thanks go to my parents Gisela and Adolf Bonato to whom I am indebted the most.

Contents

Introduction

The max-cut problem consists in partitioning the nodes of an undirected weighted graph into two sets such that the aggregate weight of the edges between these sets is maximized. This well-known combinatorial optimization problem is the reformulation, in graph theoretical terms, of the unconstrained 0/1 quadratic problem which aims at optimizing a quadratic objective function over the set of all 0/1 vectors of fixed dimension. In general, the max-cut problem is **NP**-hard, although selected special cases can be solved in polynomial time. It has a number of interesting applications such as the optimal design of very-large-scale-integration (VLSI) circuits or the study of minimum energy configurations of spin glasses—alloys of magnetic impurities diluted in a nonmagnetic metal—which is among the most investigated topics in the statistical physics literature.

In this doctoral thesis we are interested in finding provably optimal solutions of the max-cut problem as opposed to approximate solutions. To do so, we use the established and well-working branch-and-cut method, a generic solution technique whose performance for a given type of optimization problem is mainly determined by two key elements: Firstly, a close yet manageable approximation of the polyhedron associated with the problem. Secondly, efficient methods to solve the corresponding separation problem which is to decide for an arbitrary point in the ambient space whether or not it lies inside the polyhedron just mentioned.

For complete graphs, the max-cut problem and the associated cut polytope have been extensively studied over the last decades. Their counterparts on arbitrary graphs, in particular sparse ones, on the other hand, have received much less attention. Moreover, the transferability of methods from the complete to the sparse case is limited. This is mainly because the respective methods often require certain structures that are unlikely to be found in a sparse graph. A generic possibility to work around this problem is to make the graph artificially complete by adding zero-weighted edges. However, this technique is only effective in conjunction with an efficient way to exploit the original sparse structure. Otherwise, it will ultimately lead to the same computational complexity as the problem on the complete graph.

In this study, we investigate a new contraction-based separation approach for the max-cut problem that is primarily intended for problems on sparse graphs. The key idea is to contract edges based on their value in a given linear programming (LP) solution. In its simplest form, this technique presents an efficient way to separate so-called odd-cycle inequalities. In addition, we describe sophisticated methods to add missing edges to an already contracted graph as well as to compute suitable values to extend the corresponding LP solution accordingly. This allows us to apply solution techniques that were originally intended for problems on complete graphs and could not have been used on a sparse graph otherwise.

The remainder of this thesis is structured as follows: In the first chapter, we introduce

fundamental concepts, methods, and results from the fields of graph theory, complexity theory, linear and integer programming, as well as combinatorial optimization. This lays the groundwork for understanding the rest of the thesis.

In Chapter 1 we precisely define the max-cut problem, including its reformulation in terms of quadratic optimization. We proceed with the description of two interesting applications coming from circuit layout design and statistical physics, respectively. Following a brief introduction to approximate solution techniques, we give a survey of the associated cut polytope and its facial structure. These combined results will help in devising a branch-and-cut algorithm later on. Finally, we outline the literature and previous work on the max-cut problem that is relevant in the scope of this thesis.

Our key contribution, the new shrink separation approach, is presented in Chapter 2. Here, we elaborate on the single steps of the method and their respective underlying theory. Afterwards, we describe an actual realization of the shrink separation and point out its deviations from the theoretical conceptual design. Finally, we investigate some of the algorithm's numerical aspects.

Chapter 3 deals with the computational experiments that we carried out to test the performance of the shrink separation. After introducing the considered test instances, we specify the experiments' setup, including the hard- and software used, chosen parameters, and tested separation scenarios. We proceed by summarizing the results for the different classes of test instances before concluding with a case study that takes an in-depth look at a particularly interesting set of instances generated from real-world data.

The last chapter comprises a recapitulation of our contributions and findings in this study, followed by our conclusions and some suggestions regarding future research directions.

Finally, Appendices A and B contain the collective tables with detailed information on the characteristics of the test instances and the results of the computational experiments, respectively.

Chapter 0

Preliminaries and Notations

In this chapter we introduce fundamental concepts and general terminology used throughout this thesis. Definitions with a more restricted scope are provided in subsequent chapters. Some elementary definitions are given to fix the terminology and to make the presentation more self-contained.

0.1 Graphs

In this thesis we study problems that are defined on graphs. Various forms of graphs are also encountered in the solution approaches to these problems. We now introduce selected topics of graph theory. The statement below is mainly adopted from the introductory chapters of [Die05, GY04].

0.1.1 Nodes, Edges, and Density

An **undirected graph** is a pair $G = (V, E)$ consisting of a nonempty set V of **nodes** and a set E of **edges** which are unordered pairs of nodes. Unless otherwise stated, the graphs in this thesis are generally assumed to be undirected. Therefore, we will omit the term "undirected" from now on. A **weighted** graph additionally associates a **label**, or **weight**, with every edge in the graph. Weights are usually real numbers.

The node set of a graph G is referred to as $V(G)$, its edge set as $E(G)$. These notations are independent of any actual names the sets may have in a given context. We denote an edge $e = \{u, v\}$ by uv. Also, we always equate a node v and the respective 1-element set $\{v\}$.

The cardinality of the node set V, written as $|V|$, is called the **order** of G. A graph of order n can have at most $\binom{n}{2}$ edges, in which case we call the graph **complete** and denote it by K_n. The ratio $|E|/\binom{n}{2}$ of actual and potential edges is called the **edge density**, or simply the **density**, of G. Graphs with a density near 0 and 1 are referred to as **sparse** and **dense**, respectively. However, the decision whether a given graph is considered to be sparse or dense is not absolute; it usually depends on both the context and the type of graph at hand.

A graph is called **finite** if both V and E are finite. In the scope of this thesis we will exclusively deal with finite graphs.

0.1.2 Incidence and Adjacency

A node v is called **incident** with an edge e, and vice versa, if $v \in e$. The two nodes incident with an edge are its **ends**, and an edge **joins** its ends. Two nodes are **adjacent**, or **neighbors**, if they are joined by an edge. Two edges $e \neq f$ are **adjacent** if they share a common end.

An edge joining a node with itself is called a **loop**. A collection of at least two edges sharing the same ends is referred to as **multiple edge** or **parallel edge**. A graph without loops or multiple edges is called **simple**.

0.1.3 Cuts and Degree

Let $U, W \subseteq V$ be two node sets. We call the set of edges with precisely one end in U a **cut** of G and denote it by $\delta(U)$. In this context, the set U and its **complement** $U^{\complement} := V \setminus U$ are referred to as the **shores** of the cut. We write $\delta(v)$ instead of $\delta(\{v\})$ for a node $v \in V$ and call $\delta(v)$ the **star** of v. We will also use the abbreviation $(U : W) := \delta(U) \cap \delta(W)$ for the set of edges with one end in U and the other end in W.

The **degree** $\deg(v)$ of a node v is the number of edges incident with v, loops counting twice. Thus, for graphs without loops the degree of v is identical to $|\delta(v)|$.

0.1.4 Paths, Cycles, and Connectivity

A **path** is a nonempty graph $P = (V, E)$ with the node set $V = \{v_i \mid i = 0, \ldots, k\}$ of pairwise distinct nodes and the edge set $E = \{v_i v_{i+1} \mid i = 0, \ldots, k-1\}$. The nodes v_0 and v_k are **linked** by P and are called its **ends**. The remaining nodes $v_1, \ldots, v_{k-1}$ are called the **inner** nodes of P. A path is often referred to by the sequence of its nodes, i. e., $P = v_0 v_1 \ldots v_k$, and is called a path **from** v_0 **to** v_k or simply a (v_0, v_k)**-path**. The number of edges of a path is its **length**. In weighted graphs, however, the length of a path commonly refers to the aggregate weight of its edges rather than their number.

If $P = v_0 \ldots v_{k-1}$ is a path of length at least 2 then the graph $C := (V, E \cup v_{k-1} v_0)$ is called a **cycle**. As with paths, a cycle is often referred to by the (cyclic) sequence of its nodes, e. g., the above cycle C could be written as $v_0 \ldots v_{k-1} v_0$. The **length** of a cycle is the number of its edges (or nodes) and a cycle of length k is also called a k**-cycle**. A **chord** of a cycle C is an edge that joins two nodes of C which are not adjacent in the cycle.

Two nodes u, v in a graph G are **connected** if they are linked by a path in G. The graph G itself is **connected** if this is true for any two of its nodes. The **distance** between two nodes u and v in a graph G is the length of a shortest (u, v)-path in G; if no such path exists, we set the distance to infinity.

0.1.5 Subgraphs, Connected Components, and Contractibility

Let $G = (V, E)$ and $G' = (V', E')$ be two graphs. If $V' \subseteq V$ and $E' \subseteq E$ then G' is a **subgraph** of G (and G a **supergraph** of G'), written as $G' \subseteq G$.

If $G' \subseteq G$ and G' contains all the edges $uv \in E$ with $u, v \in V'$ then G' is an **induced subgraph** of G. We say that V' **induces** or **spans** G' in G and write $G' = G[V']$. Thus, for any node set $U \subseteq V$, the (node-)induced subgraph $G[U]$ is the graph on U whose edges are exactly the edges of G with both ends in U. Finally, $G' \subseteq G$ is a **spanning** subgraph of G if V' spans all of G, i. e., if $V' = V$.

A maximal (with respect to edge inclusion) connected subgraph of G is called a **connected component** of G. A complete subgraph of G is called a **clique** of G.

We call the operation of identifying a pair of adjacent nodes—while preserving all other adjacencies between nodes—an **elementary contraction**, or simply a **contraction**. We assume that multiple edges arising from a contraction are replaced by single edges. A graph G is called **contractible** to another graph G' if G' can be obtained from G by a sequence of contractions.

0.1.6 Trees and Forests

An **acyclic** graph, i. e., a graph not containing any cycles, is called a **forest**. A connected forest is called a **tree**. Thus, a forest is a graph whose connected components are trees. The nodes of degree 1 in a tree are its **leaves**, the remaining nodes are its **inner nodes**.

Sometimes it is convenient to consider one node of a tree as special. Such a node is called the **root** of this tree. Note that the root of a tree is never called a leaf, even if it has degree 1. A tree with a fixed root is called a **rooted tree**.

In a rooted tree T, the nodes at distance k from the root have **height** k and form the k-th **level** of T. The root has height 0. The **height** of T itself is the maximum height of its nodes. The nodes on level $k+1$ which are adjacent to a node v on level k are called the **children** of v and v is called the **parent** of its children. We refer to children of the same parent as **siblings**. If every node in T has at most k children, $k \geq 2$, we call it a k**-ary** tree. In case of a 2-ary tree, we use the term **binary tree** instead.

0.1.7 Embeddings and Genera

Let S_k denote the **orientable surface** formed by adding k **handles** to the sphere. The sphere itself is denoted by S_0. It is topologically equivalent to the flat plane since we can create a hole in the sphere's surface and then stretch out the surface onto the plane. The **torus** is S_1, the **double-torus** S_2, and so on. The **genus** of S_k is the number of handles, k.

An **embedding** of a graph into a surface is a drawing of the graph on the surface in such a way that its edges may intersect only at their ends. In other words, the graph can be drawn on the surface without any edges crossing. The **genus** of a graph is the minimum integer g such that the graph can be embedded into S_g.

0.1.8 Selected Classes of Graphs

A graph $G = (V, E)$ is **bipartite** if its node set V can be partitioned into two nonempty subsets V_1 and V_2 such that each edge has one end in V_1 and the other end in V_2. G is **complete bipartite** if each node in V_1 is joined to each node in V_2. We denote the complete bipartite graph with $|V_1| = m$ and $|V_2| = n$ by $K_{m,n}$.

A graph is **planar** if it can be drawn in a plane without any edges crossing, i. e., if it has genus 0. We call a nonplanar graph G **almost planar** if it contains a node v such that G becomes planar by removing v and all its incident edges.

A graph is k**-regular** if all its nodes have the same degree k. A 3-regular graph is called **cubic**.

A **grid graph** is a graph that can be mapped to a grid, i. e., each node corresponds to a grid point and each edge corresponds to a tie between the respective grid points of its ends.

0.2 Affine Geometry and Polyhedra

A basic knowledge of polyhedral theory is essential to understand this thesis. We now give an overview of the relevant concepts of affine geometry and polyhedral theory. It is mainly based on the introductory chapters of [Brø83, Zie06]. We assume familiarity with the standard linear theory of the **real vector space** $\mathbb{R}^d$, in particular basic notions such as **subspaces**, **linear independence**, **dimension**, **scalar product**, and so forth. As a convention, vectors (e. g.: "$\boldsymbol{x}$") are typeset bold while scalars (e. g.: "α"), including single vector entries (e. g.: "x_i"), are typeset in regular font weight.

0.2.1 Affine Subspaces

A **linear subspace** of $\mathbb{R}^d$ is a nonempty set $L \subseteq \mathbb{R}^d$ such that $\lambda_1 \boldsymbol{x}_1 + \lambda_2 \boldsymbol{x}_2$ is in L for all $\boldsymbol{x}_1, \boldsymbol{x}_2 \in L$ and all $\lambda_1, \lambda_2 \in \mathbb{R}$. Thus, a linear subspace always contains the **origin 0** which is the appropriately sized vector of all zeros.

For a pair of sets $X, Y \subseteq \mathbb{R}^d$, we define the **Minkowski sum** $X + Y$ as the set of all elements $\boldsymbol{x} + \boldsymbol{y}$ with $\boldsymbol{x} \in X$ and $\boldsymbol{y} \in Y$. An **affine subspace** of $\mathbb{R}^d$ is a translate of a linear subspace, i. e., a subset $A = \boldsymbol{x} + L$ where $\boldsymbol{x}$ is a vector in $\mathbb{R}^d$ and L is a linear subspace of $\mathbb{R}^d$. Note that for a given affine subspace A, the linear subspace L is unique whereas $\boldsymbol{x}$ can be chosen arbitrarily in A.

The **dimension** of an affine subspace is the dimension of its corresponding linear vector space. Affine subspaces of dimension $0, 1, 2$, and $d-1$ in $\mathbb{R}^d$ are called **points**, **lines**, **planes**, and **hyperplanes**, respectively.

Let $\boldsymbol{\lambda}, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n \in \mathbb{R}^d$ and let $\mathbf{1}$ denote the appropriately sized vector of all ones. We call the **linear combination**

$$\sum_{i=1}^{n} \lambda_i \boldsymbol{x}_i \tag{0.1}$$

an **affine combination** if the scalar product $\mathbf{1}^T \boldsymbol{\lambda} = \sum_{i=1}^{n} \lambda_i$ equals 1. Furthermore, we call (0.1) a **conic combination** if $\boldsymbol{\lambda} \geq \mathbf{0}$, and a **convex combination** if it is both conic and affine. We say that the vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ are **affinely independent** if (0.1) with $\mathbf{1}^T \boldsymbol{\lambda} = 0$ can only have the value $\mathbf{0}$ when $\boldsymbol{\lambda} = \mathbf{0}$. In other words, none of the vectors is an affine combination of the remaining ones.

For any subset $X \subseteq \mathbb{R}^d$, the **affine hull** $\operatorname{aff}(X)$ is the set of all affine combinations of points of X. The **conic hull** $\operatorname{cone}(X)$ and the **convex hull** $\operatorname{conv}(X)$ are defined analogously.

0.2.2 Hyperplanes, Halfspaces, Polyhedra, and Polytopes

For $\boldsymbol{a} \in \mathbb{R}^d$ and $\alpha \in \mathbb{R}$, we denote the hyperplane defined by the equation $\boldsymbol{a}^T \boldsymbol{x} = \alpha$ by

$$H(\boldsymbol{a}, \alpha) := \{\boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{a}^T \boldsymbol{x} = \alpha\}.$$

The corresponding **inequality** $\boldsymbol{a}^T \boldsymbol{x} \leq \alpha$ defines a **closed halfspace**

$$K(\boldsymbol{a}, \alpha) := \{\boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{a}^T \boldsymbol{x} \leq \alpha\}$$

which is **bounded** by the hyperplane $H(\boldsymbol{a}, \alpha)$. In other words, the bounding hyperplane is the collection of all points for which the inequality is **tight**, i. e., for which the inequality is satisfied with equality. For a given inequality $\boldsymbol{a}^T \boldsymbol{x} \leq \alpha$, we call $\boldsymbol{a}^T \boldsymbol{x}$ the **left hand**

side and α the **right hand side**. Where appropriate, we will use the abbreviation $(\boldsymbol{a}, \alpha)$ instead of $\boldsymbol{a}^T\boldsymbol{x} \leq \alpha$.

The intersection of a finite number of closed halfspaces is called a **polyhedron**. Every polyhedron is closed and convex. The convex hull of a nonempty finite set is called a **polytope**. A polytope is a bounded polyhedron. Consequently, there are two ways of describing a polytope: as the convex hull of a nonempty finite set, also called **$\mathcal{V}$-representation**, or as the bounded intersection of finitely many closed halfspaces, also called **$\mathcal{H}$-representation**. A nonempty subset $C \subseteq \mathbb{R}^d$ is a **cone** if with any finite set of vectors in C it also contains all their conic combinations. In particular, every cone contains the origin $\mathbf{0}$. A cone is called **polyhedral** if it can be obtained as the conic hull of a finite set of vectors.

The concepts of polyhedron, polytope, and polyhedral cone are related by means of the **decomposition theorem for polyhedra**.

Theorem 0.1 (Decomposition Theorem for Polyhedra)**.** *A subset $Q \subseteq \mathbb{R}^d$ is a polyhedron if and only if $Q = P + C$ for some polytope P and some polyhedral cone C.*

Let P be a polyhedron in $\mathbb{R}^d$ and let $K := K(\boldsymbol{a}, \alpha)$ be a closed halfspace in $\mathbb{R}^d$ with the corresponding bounding hyperplane H. We say that K and its defining inequality $\boldsymbol{a}^T\boldsymbol{x} \leq \alpha$, respectively, are **valid** for P if $P \subseteq K$. If, in addition, $P \cap H \neq \emptyset$, we call K and H a **supporting halfspace** and a **supporting hyperplane** of P, respectively. If a supporting hyperplane H of P does not contain the entire polyhedron P, we call it a **proper supporting hyperplane**.

A **face** F of a polyhedron P is the intersection of P and the bounding hyperplane of a valid halfspace $K(\boldsymbol{a}, \alpha)$ of P, i. e.,

$$F = P \cap H(\boldsymbol{a}, \alpha).$$

We also say that the inequality $\boldsymbol{a}^T\boldsymbol{x} \leq \alpha$ **defines** the face F. If $H(\boldsymbol{a}, \alpha)$ is a proper supporting hyperplane of P, we call F **proper**. There are only two **improper**, or **trivial**, faces of P, namely the empty set and the polyhedron P itself.

The **dimension** $\dim(F)$ of a face F of P is the dimension of its affine hull $\operatorname{aff}(F)$. Since $\operatorname{aff}(F)$ is an affine subspace of $\mathbb{R}^d$—and thus a translate of a unique linear subspace L—the dimension of the face F equals the dimension of L. As a convention, we set the dimension of the empty set to -1. The faces of dimension $0, 1, \dim(P) - 2$, and $\dim(P) - 1$ are called the **vertices**, **edges**, **ridges**, and **facets** of P, respectively. A vertex $\boldsymbol{v}$ of P is also referred to as **extreme point**. This is because $\boldsymbol{v}$ cannot be obtained as a proper convex combination of two distinct points in P. In other words, there are no two distinct points $\boldsymbol{a}, \boldsymbol{b} \in P$ such that

$$\boldsymbol{v} = \lambda\boldsymbol{a} + (1 - \lambda)\boldsymbol{b},$$

for a suited scalar λ in the open range $(0, 1)$.

0.3 Algorithms and Complexity

Complexity theory investigates the difficulty of problems and the efficiency of the algorithmic methods to solve them. Below, we give a brief and very informal introduction to algorithms and complexity, in particular to polynomial-time solvability and **NP**-completeness. The following explanations are mainly adopted from [Sch03]. For an in-depth description of the topic we refer to [GJ79].

0.3.1 Polynomial-time Solvability

An **algorithm** can be interpreted as a finite set of **instructions**, where each instruction performs a sequence of **elementary steps** on certain data. A **polynomial-time algorithm**, or simply **polynomial algorithm**, is an algorithm that terminates after a number of elementary steps that is bounded by a polynomial in the size of the input. A problem is called **polynomial-time solvable**, or **polynomial**, if it can be solved by a polynomial algorithm.

For a more detailed specification of an algorithm's complexity, we use the so-called $\mathcal{O}$- or **Landau notation**. Let f, g be two real-valued functions with common domain $X \subseteq \mathbb{R}$. We say that the function $f(x)$ is $\mathcal{O}\big(g(x)\big)$ if there exists a nonnegative constant c such that $f(x) \leq cg(x) + c$ for all $x \in X$. Using this notation, we call an algorithm polynomial if it terminates after $\mathcal{O}\big(p(x)\big)$ elementary steps, where p is a polynomial in the size of the input.

0.3.2 NP-completeness and Reducibility

A **decision problem** is a problem that can be answered by 'yes' or 'no'. Note that a decision problem is completely described by the inputs with positive answer.

We introduce two **complexity classes** which are collections of certain decision problems. The class **P** is the collection of all polynomial-time solvable decision problems. The class **NP** is the collection of all decision problems for which one can verify in polynomial-time that a positive answer to a given input is indeed correct. Obviously, **P** is a subset of **NP**. Yet it is still unknown whether this inclusion is proper.

A decision problem Π is called **polynomial-time reducible**, or **polynomially reducible**, to the decision problem Π' if there exists a polynomial algorithm A that transforms inputs x of Π into inputs $x' := A(x)$ of Π' such that

$$x \text{ has positive answer } \Leftrightarrow\ x' \text{ has positive answer, } \quad \text{for all inputs } x \text{ of } \Pi.$$

The above polynomial reduction characterizes the decision problem Π' to be at least as hard as Π.

A decision problem Π in **NP** is called **NP-complete** if each problem in **NP** is polynomially reducible to it. Less formally speaking, the **NP**-complete problems are the most difficult problems in **NP**.

0.3.3 Complexity of Optimization Problems

In the scope of this thesis, we are not concerned with decision problems but with **optimization problems** of the form

$$\max_{x \in X}\ f(x),$$

where X is a collection of elements derived from the input and f is a function with domain X. Since an optimization problem is not a decision problem, we cannot directly apply the notions of complexity introduced so far. Instead, we assign to the optimization problem an **associated decision problem**:

"Given a value λ, is there an $x \in X$ with $f(x) \geq \lambda$?"

This decision problem is **polynomially reducible** to its associated optimization problem, since the optimization also answers the yes-no question. Thus, the notion of polynomial reduction to an optimization problem extends the previously introduced concept of polynomial reduction to a decision problem in a natural fashion.

Finally, we say that an optimization problem P is **NP-hard** if there exists an **NP**-complete decision problem that is polynomially reducible to P. This is equivalent to demanding that all decision problems in **NP** have to be polynomially reducible to P.

0.4 Integer Programming and Combinatorial Optimization

Optimization, or **mathematical programming**, is concerned with finding the best elements from some set of available alternatives. To give an example, a possible optimization problem is to minimize or maximize a function over a set of feasible solutions. We now introduce selected subfields of mathematical programming that are relevant for this thesis, namely linear programming, integer programming, and combinatorial optimization. The following statement is based on the introductory chapters of [NW99, Thi95].

0.4.1 Linear Programming, Duality, and the Simplex Method

One of the fundamental models in mathematical optimization is **linear programming**. Let $\mathbb{R}^{m\times n}$ denote the set of $(m \times n)$ real matrices and let $\mathbb{R}^n_+$ denote the set of non-negative n-dimensional real vectors. The **standard form** of a **linear programming problem**, or simply **linear program** (LP), is as follows:

$$\begin{aligned} \max\ & \boldsymbol{c}^T\boldsymbol{x} \\ \text{s.t.}\ & A\boldsymbol{x} \le \boldsymbol{b} \\ & \boldsymbol{x} \in \mathbb{R}^n_+, \end{aligned} \tag{0.2}$$

where $\boldsymbol{x} := (x_1, \ldots, x_n)$ is the vector of **variables**, $A = (a_{ij})$ is a matrix in $\mathbb{R}^{m\times n}$, and $\boldsymbol{b}$ and $\boldsymbol{c}$ are vectors in $\mathbb{R}^m$ and $\mathbb{R}^n$, respectively.

The conditions $A\boldsymbol{x} \le \boldsymbol{b}$ and $\boldsymbol{x} \in \mathbb{R}^n_+$ are referred to as **constraints**. We call the set $S := \{\boldsymbol{x} \in \mathbb{R}^n_+ \mid A\boldsymbol{x} \le \boldsymbol{b}\}$ the **feasible set** of the LP. It is the intersection of finitely many closed halfspaces and is sometimes also referred to as **feasible polyhedron** P_{LP}. The elements of S are the **feasible solutions** of the LP. If there exists at least one feasible solution, i. e., $S \neq \emptyset$, then we call the LP **feasible**.

The linear function $\boldsymbol{c}^T\boldsymbol{x}\colon \mathbb{R}^n \to \mathbb{R}$ is called the **objective function**. A feasible solution $\boldsymbol{x}^* \in S$ for which the value of the objective function is maximal, i. e.,

$$\boldsymbol{c}^T\boldsymbol{x}^* \ge \boldsymbol{c}^T\boldsymbol{x}, \text{ for all } \boldsymbol{x} \in S,$$

is an **optimum solution**. In this case, $\boldsymbol{c}^T\boldsymbol{x}^*$ is the **optimum value** of the LP. A feasible LP may not have an optimum solution. We say that an LP is **unbounded** if for any scalar $\lambda \in \mathbb{R}$ there is a feasible solution $\boldsymbol{x} \in S$ such that $\boldsymbol{c}^T\boldsymbol{x} > \lambda$.

The standard form (0.2) can be seen as the prototypical linear program. Any LP can be converted into standard form. To give some examples: a minimization problem can be transformed into a maximization problem by multiplying the objective function by -1; or an equality constraint $\boldsymbol{a}_i^T\boldsymbol{x} = b_i$ can be expressed by the two inequality constraints $\boldsymbol{a}_i^T\boldsymbol{x} \le b_i$ and $-\boldsymbol{a}_i^T\boldsymbol{x} \le -b_i$. Therefore, we can assume a given LP to be in standard form without loss of generality.

With every LP in standard form, also called the **primal** problem, we associate its **dual** problem

$$\begin{aligned} \min \ & \boldsymbol{b}^T \boldsymbol{u} \\ \text{s.t.} \ & A^T \boldsymbol{u} \geq \boldsymbol{c} \\ & \boldsymbol{u} \in \mathbb{R}^n_+. \end{aligned}$$

A primal and its corresponding dual are closely related. Note that the dual of the dual is again the primal. Hence, the denotation of an LP as primal or dual is arbitrary and depends on the given context. Feasible solutions of the dual provide upper **bounds** on the optimum value of the primal, while feasible solutions of the primal yield lower bounds on the optimum value of the dual. A fundamental result of linear programming duality is the following **LP duality theorem**.

Theorem 0.2 (LP Duality Theorem)**.** *Given a primal problem (P) and its corresponding dual problem (D).*

(i) If both (P) and (D) are feasible then (P) and (D) have optimum solutions and the optimum values of (P) and (D) are equal.

(ii) If (P) (resp. (D)) is infeasible then (D) (resp. (P)) is either infeasible or unbounded.

(iii) If (P) (resp. (D)) is unbounded then (D) (resp. (P)) is infeasible.

There are several approaches for solving LPs systematically. The most prominent solution techniques are the **simplex method** and the class of **interior point methods**. There is also the **ellipsoid method**, but it has no practical relevance. It is of theoretical interest, though, since it can be used to prove the polynomial-time solvability of LPs.

However, in this brief introduction we focus exclusively on the simplex method. It was originally created by the American mathematician George B. Dantzig [Dan51]. The initial method has since undergone numerous improvements and has become one of the most important solution technique for LPs in practice. Its basic idea is best explained geometrically. Maximizing the linear function $\boldsymbol{c}^T \boldsymbol{x}$ over the feasible polyhedron P_{LP} is equivalent to translating the hyperplane $H = H(\boldsymbol{c}, 0)$ alongside the vector $\boldsymbol{c}$ until it becomes a supporting hyperplane H' of P_{LP}. Then, all the points in the face $H' \cap P_{\text{LP}}$ are optimum solutions. Note that hyperplanes such as H and H' are also referred to as **iso-value planes**. This is because all the points in an iso-value plane have the same objective function value, namely the right hand side value of the equation that defines the hyperplane. If the LP is feasible and bounded then at least one of the vertices of P_{LP} is an optimum solution. Hence, the search for optimum solutions can be restricted accordingly.

The simplex method exploits this fact by starting at a vertex of P_{LP} and then iteratively moving from the current vertex to one of the adjacent vertices with a better objective value. The method terminates if none of the adjacent vertices can improve the objective value of the current vertex. Due to the convexity of the feasible polyhedron, this locally optimal solution is also globally optimal.

However, the number of vertices of P_{LP} can be exponential in the number of variables and constraints. To give an example, the n-dimensional unit cube can be described using n variables and $2n$ inequalities, but it has 2^n vertices. In 1972, Klee and Minty [KM72] gave an example of an LP in which the feasible polyhedron P_{LP} is a distortion of an n-dimensional cube. They showed that the simplex method as formulated by Dantzig visits all 2^n vertices before arriving at the optimal vertex. This proves the exponential worst-case complexity of the algorithm. Since then, similar examples could be constructed for almost every known variant of the algorithm.

Nevertheless, the simplex method is remarkably efficient in practice. In 2004, Spielman and Teng [ST04] introduced the notion of **smoothed complexity** to provide a more realistic analysis of the performance of algorithms. Using continuous interpolation between the worst-case and average-case analyses of algorithms, they showed that "the simplex algorithm usually takes polynomial time."

In conclusion, the field of linear programming is very well studied and provides powerful methods to solve LPs efficiently. Unfortunately, only a minority of practical problems can be modeled as pure LPs. In most cases, some or all of the variables are restricted to integral values. By adding integrality conditions to a subset of the variables we obtain the class of **linear mixed integer programming problems**. However, in the scope of this thesis we only consider the special case where all the variables have to be integral.

0.4.2 Integer and Binary Programming

Let $\mathbb{Z}_+^n$ denote the set of nonnegative n-dimensional integral vectors. A **linear integer programming problem**, or simply **integer program** (IP), has the form

$$\begin{aligned} \max \ & \boldsymbol{c}^T\boldsymbol{x} \\ \text{s.t.} \ & A\boldsymbol{x} \leq \boldsymbol{b} \\ & \boldsymbol{x} \in \mathbb{Z}_+^n. \end{aligned} \tag{0.3}$$

Except for the **integrality constraints** $\boldsymbol{x} \in \mathbb{Z}_+^n$, its structure is similar to that of an LP. Yet, in contrast to an LP, the feasible set $S := \{\boldsymbol{x} \in \mathbb{Z}_+^n \mid A\boldsymbol{x} \leq \boldsymbol{b}\}$ is not a polyhedron. For an IP, we define the **associated polyhedron** P_{IP} as the convex hull of the feasible set. Clearly, the associated polyhedron P_{IP} satisfies

$$S \subseteq P_{\text{IP}} \subseteq \{\boldsymbol{x} \in \mathbb{R}_+^n \mid A\boldsymbol{x} \leq \boldsymbol{b}\},$$

with the inclusions being proper in general.

By further restricting the variables to be either 0 or 1, we obtain a **linear binary programming problem**, or **binary program**,

$$\begin{aligned} \max \ & \boldsymbol{c}^T\boldsymbol{x} \\ \text{s.t.} \ & A\boldsymbol{x} \leq \boldsymbol{b} \\ & \boldsymbol{x} \in \{0,1\}^n. \end{aligned}$$

Binary programs are an important special case of integer programs and arise, for instance, in the context of combinatorial optimization.

0.4.3 Combinatorial Optimization

Combinatorial optimization problems are concerned with the optimization of an objective function over collections of subsets of a finite set. A generic combinatorial optimization problem can be defined as follows. Let E be a finite set of n elements and let $\boldsymbol{c} \in \mathbb{R}^n$ be the vector of the **weights** assigned to the elements of E. For an arbitrary subset $F \subseteq E$, we define its **aggregate weight** $\boldsymbol{c}(F) := \sum_{e \in F} c_e$. Suppose we are given a collection $\mathcal{F}$ of subsets $F \subseteq E$. These subsets can be interpreted as the feasible solutions out of the **power set** 2^E of all possible subsets of E. The corresponding **combinatorial optimization problem** is

$$\max_{F \in \mathcal{F}} \boldsymbol{c}(F).$$

To solve this problem, we need a characterization of the set $\mathcal{F}$ of feasible solutions. This can be accomplished by using binary vectors to represent the elements of $\mathcal{F}$.

The **incidence vector** $\boldsymbol{\chi}^F$ of a subset $F \subseteq E$ is a vector in $\{0,1\}^n$ with

$$\chi_e^F := \begin{cases} 1 & \text{if } e \in F, \\ 0 & \text{if } e \in E \setminus F. \end{cases}$$

Moreover, we define the **associated polytope** $P_{\mathcal{F}}$ of the combinatorial optimization problem as the convex hull of the incidence vectors $\boldsymbol{\chi}^F$ of all $F \in \mathcal{F}$, i. e.,

$$P_{\mathcal{F}} = \operatorname{conv}\{\boldsymbol{\chi}^F \mid F \in \mathcal{F}\}.$$

Since the incidence vectors are binary vectors, they are exactly the vertices of the polytope $P_{\mathcal{F}}$. In principle, we can now reformulate the combinatorial optimization problem as

$$\begin{aligned} \max \ & \boldsymbol{c}^T \boldsymbol{x} \\ \text{s.t.} \ & \boldsymbol{x} \in P_{\mathcal{F}} \\ & \boldsymbol{x} \in \{0,1\}^n. \end{aligned}$$

Unfortunately, we do not know any efficient algorithm to solve such an optimization problem whose feasible set is only described by a $\mathcal{V}$-representation. However, since every polytope also has an $\mathcal{H}$-representation, there exists a finite set of inequalities $A\boldsymbol{x} \leq \boldsymbol{b}$, also called a **linear description**, such that $P_{\mathcal{F}} = \{\boldsymbol{x} \in \mathbb{R}^n_+ \mid A\boldsymbol{x} \leq \boldsymbol{b}\}$. So, theoretically we could transform the combinatorial optimization problem into a binary program. In fact, there are techniques, like, for example, the **Fourier-Motzkin elimination**, to transform the different representations of the polytope $P_{\mathcal{F}}$ into one another. Yet, this is only viable for very small problem instances. In general, a linear description of a combinatorial optimization problem is either not completely known or it comprises too many inequalities to be represented explicitly.

However, we do not necessarily need a complete linear description for finding an optimum solution; it is sufficient to know the local facet structure of the associated polytope $P_{\mathcal{F}}$ near a vertex that represents such a solution. This idea is exploited by the branch-and-cut method.

0.4.4 Branch-and-Cut

The **branch-and-cut method** is an algorithmic approach for solving integer programs. Thus, it is also applicable to combinatorial optimization problems that are modeled as binary programs as described above. Branch-and-cut is a conjunction of **cutting plane methods**, used to dynamically generate valid inequalities, with a **divide-and-conquer** technique called **branch-and-bound** that recursively splits the initial problem into **subproblems**. The first description of this combined method can be found in [Mil76] for the traveling salesman problem. However, a branch-and-cut algorithm of the form to be outlined shortly was first published by Grötschel, Jünger, and Reinelt [GJR84] for the linear ordering problem. The actual term "branch-and-cut" has been introduced by Padberg and Rinaldi [PR87, PR91] for an algorithm for solving the traveling salesman problem.

A key element of the branch-and-cut method is the use of relaxations to obtain bounds on the optimum value of the problem to be solved. Let S be the feasible set of an integer

program of the form (0.3). We call a maximization problem

$$\max_{\boldsymbol{x} \in R} \; f(\boldsymbol{x})$$

a **relaxation** of the IP if

$$S \subseteq R \quad \text{and} \quad \boldsymbol{c}^T \boldsymbol{x} \leq f(\boldsymbol{x}), \text{ for all } \boldsymbol{x} \in S.$$

So, an optimum solution of the relaxation provides an upper bound on the optimum value of the integer program.

We obtain the so-called **linear programming relaxation**, or **LP relaxation**, by replacing the integrality constraints of the IP with the weaker conditions $\boldsymbol{x} \in \mathbb{R}^n_+$. The LP relaxation is a pure linear program and can be solved efficiently using, for instance, the simplex method. A corresponding optimum solution will be referred to as **LP solution**. However, since we dropped the integrality constraints, the LP solution may be **fractional** and hence infeasible for the IP. Also, even an integral LP solution does not necessarily represent a feasible solution of the IP. Yet, in both cases we can tighten the relaxation by adding appropriate inequalities to improve the approximation of the polyhedron P_{IP} associated with the integer program. Such a tightening inequality $\boldsymbol{a}^T \boldsymbol{x} \leq \alpha$ has to be valid for the polyhedron P_{IP} while the LP solution has to lie outside the respective closed halfspace $K(\boldsymbol{a}, \alpha)$. We also say that the inequality has to be **satisfied** by all points of P_{IP} while being **violated** by the LP solution. Descriptively speaking, the bounding hyperplane $H(\boldsymbol{a}, \alpha)$ **separates**, or **cuts**, the LP solution from the polyhedron P_{IP}. Hence, we call $H(\boldsymbol{a}, \alpha)$ a **separating hyperplane**, or **cutting plane**.

The task to decide whether a separating hyperplane exists and, if so, compute one, is called **separation problem**, or simply **separation**. An important theorem in this context is that, under some technical conditions, the optimization problem is polynomial-time solvable if and only if the corresponding separation problem is polynomial [GLS93].

In theory, we could solve the IP by iterating the process of tightening and solving the LP relaxation until we obtain an optimum solution. In practice, however, these pure cutting plane methods tend to progress asymptotically towards the solution and need too many iterations. Nevertheless, cutting plane generation proved to be very effective in conjunction with branch-and-bound.

The workflow of a generic branch-and-cut algorithm is depicted in Figure 0.1[1] on page 14. The algorithm uses a rooted tree, the so-called **branch-and-cut tree**, to store and manage the subproblems, also called **nodes**. Each node maintains a **local upper bound** `lub` on the optimum value of its corresponding subproblem. We initialize the **root node** with the IP (0.3) to be solved. So, the `lub` of the root node is in fact a **global upper bound** `gub`. In addition, we maintain a **global lower bound** `glb` that equals the objective value of the best known feasible solution at any one time. We set the initial values of `gub` and `glb` to plus and minus infinity, respectively, and enter the bounding phase.

For a given subproblem, the **bounding** starts by solving the LP relaxation of the IP and checking whether it is feasible. If not, we can **prune** the subproblem, i. e., we can remove the node from the branch-and-cut tree. Otherwise, we update the `lub` with the optimum value of the LP relaxation. Next, we check whether the `lub` is less than or equal to the `glb`. If so, none of the feasible solutions of the subproblem can increase the

[1] This figure is adapted from [Thi95].

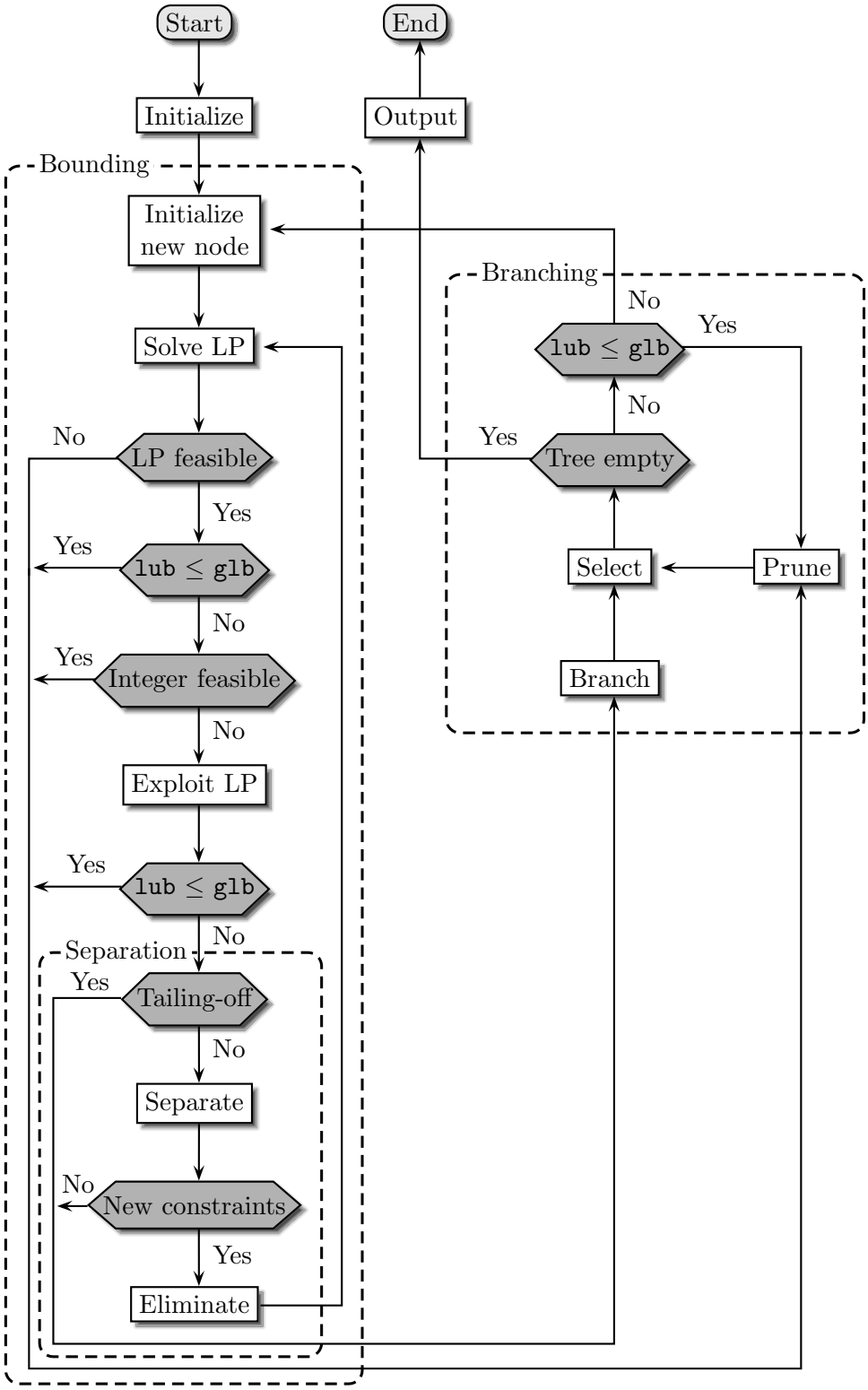


Figure 0.1: Workflow of a generic branch-and-cut algorithm.

`glb` and thus the node can be pruned. Otherwise, we proceed by testing whether the LP solution is integral and feasible, in which case the subproblem has been solved and can be pruned. If its optimum value increases the `glb`, we update the bound accordingly and prune all remaining nodes with a `lub` less than or equal to the new `glb`.

If the LP solution is fractional, or integral but infeasible, we enter the **separation**. The generated cutting planes are added to the LP relaxation and we move to the next iteration of the bounding phase. There is also the option to try and increase the `glb` by heuristically deriving an integral feasible solution from an infeasible LP solution. This can be done in the step "exploit LP".

To avoid a slow-down of computation by unnecessary growth of the number of constraints in the LP relaxation, we **eliminate** some of the inequalities from time to time. For example, we could remove the inequalities that are no longer violated by the current LP solution. The eliminated constraints are stored in a **constraint pool**. After a number of iterations, we can check whether some of the inequalities in the pool have become violated again after their removal from the LP relaxation. This checking is called **pool separation**.

If the separation fails to find cutting planes or if the `lub` does not decrease significantly over a specified number of iterations ("tailing-off") then the algorithm **branches** at the current node. In other words, it splits the subproblem into two new subproblems such that the union of their respective feasible sets equals the feasible set of the split problem. This can be accomplished, for instance, by selecting a fractional component $\bar{x}_j$ of the LP solution and adding the constraint $x_j \leq \lfloor \bar{x}_j \rfloor$ to the first new subproblem and $x_j \geq \lceil \bar{x}_j \rceil$ to the second one. Here, $\lfloor \bar{x}_j \rfloor$ is the largest integer not greater than $\bar{x}_j$ while $\lceil \bar{x}_j \rceil$ is the smallest integer not less than $\bar{x}_j$. This strategy is called **branching on a single variable**. After the branching, we select one of the remaining subproblems from the branch-and-cut tree and reenter the bounding phase.

The branch-and-cut algorithm terminates as soon as the branch-and-cut tree is empty. However, in particular for hard to solve problems it is common practice to abort the algorithm once a specified time or iteration limit is exceeded. In this respect, the advantage of the branch-and-cut algorithm is that, together with a feasible solution, it also provides an estimate of the quality of the solution. The quality is measured by the **relative gap** which is defined as $(\texttt{gub}-\texttt{glb})/\texttt{glb}$. In particular, if the numerator $\texttt{gub} - \texttt{glb}$, also known as the **absolute gap**, is 0—or less than 1 in case of an integral objective function—then the solution is optimal.

Another advantage of branch-and-cut is its use of separation techniques to dynamically generate valid inequalities. This allows us to start the optimization with a partial and/or approximate linear description of the IP. The algorithm will then adaptively refine this description in the vicinity of the current LP solution.

In conclusion, the branch-and-cut method is a powerful tool for solving integer programs. However, the overall success of the method heavily depends on the quality of the lower and upper bounds provided by primal heuristics and the separation, respectively. This is because the quality of the bounds determines the effectiveness of the pruning, which is the only means of preventing the branching from degenerating into the total enumeration of all feasible solutions.

0.4.5 Target Cuts

When building a branch-and-cut solver for a specific combinatorial optimization problem, one usually implements separation procedures for selected classes of inequalities that are valid or even facet defining for the associated polyhedron. In most cases, these inequalities follow the so-called **template paradigm**. This means that all the inequalities in a given class share a similar structure. However, there are ways to separate facet defining inequalities without putting any restrictions on their actual structure. In the following we present such a procedure for the separation of so-called **target cuts**. These inequalities were introduced by Buchheim, Liers, and Oswald [BLO08]. They are enhancements of the so-called **local cuts**, which were proposed by Applegate, Bixby, Chvátal, and Cook [ABCC01] in the context of the traveling salesman problem. Compared to their predecessors, target cuts have the key benefit of being facet defining. We now provide a brief survey of the theory of target cuts. For a detailed explanation of the topic, we refer to Buchheim, Liers, and Oswald [BLO08].

Assume we are given a polytope $P := \operatorname{conv}\{\boldsymbol{x}_1, \dots, \boldsymbol{x}_t\}$ in $\mathbb{R}^m$ and an infeasible point $\boldsymbol{x}^*$ that we want to separate from it. A crucial precondition for the efficiency of both the local cut and the target cut generation procedure is the sufficient reduction of the problem size. Therefore, we need an appropriate projection $\pi\colon \mathbb{R}^m \to \mathbb{R}^r$ with $r \leq m$. This could be, for instance, the orthogonal projection. We then consider the projected polytope $\overline{P} := \pi(P)$ which is the convex hull of the points $\pi(\boldsymbol{x}_1), \dots, \pi(\boldsymbol{x}_t)$ in $\mathbb{R}^r$. Also, we denote the projection of the infeasible point by $\overline{\boldsymbol{x}}^*$. In general, if r is substantially less than m then many of the points $\overline{\boldsymbol{x}}_i := \pi(\boldsymbol{x}_i)$ will correspond, such that $\overline{P} = \operatorname{conv}\{\overline{\boldsymbol{x}}_1, \dots, \overline{\boldsymbol{x}}_s\}$ with s substantially less than t. As a consequence, if r is sufficiently small then the polytope $\overline{P}$ can be dealt with efficiently. Note that the remainder of the procedure differs depending on whether or not the polytope $\overline{P}$ is full-dimensional. However, we will focus entirely on the full-dimensional case since the other one is irrelevant in the scope of this thesis.

Our task consists in answering the question whether or not $\overline{\boldsymbol{x}}^*$ is contained in the projected polytope $\overline{P}$. If not, we want to derive a respective cutting plane. To this end, we first choose a point $\overline{\boldsymbol{q}}$ in the interior of $\overline{P}$. This could be, for instance, the barycenter of its vertices. Then, the task can be accomplished by solving the following linear program:

$$\begin{aligned} \min \ & \sum_{i=1}^{s} \lambda_i \\ \text{s.t.} \ & \sum_{i=1}^{s} \lambda_i (\overline{\boldsymbol{x}}_i - \overline{\boldsymbol{q}}) = \overline{\boldsymbol{x}}^* - \overline{\boldsymbol{q}} \\ & \boldsymbol{\lambda} \geq \mathbf{0}. \end{aligned} \tag{0.4}$$

By construction, the feasible solutions of the LP (0.4) are exactly all the possible ways to obtain $\overline{\boldsymbol{x}}^* - \overline{\boldsymbol{q}}$ as a conic combination of the points $\overline{\boldsymbol{x}}_i - \overline{\boldsymbol{q}}$. Strictly speaking, we need a convex combination rather than a conic one, i. e., we require $\boldsymbol{\lambda}$ to satisfy $\sum_{i=1}^{s} \lambda_i = 1$. However, since $\overline{\boldsymbol{q}}$ is an interior point of $\overline{P}$, we can obtain the origin $\mathbf{0}$ as a convex combination of the points $\overline{\boldsymbol{x}}_i - \overline{\boldsymbol{q}}$. This means that for any conic combination with $\sum_{i=1}^{s} \lambda_i < 1$ we can derive a respective convex combination that yields the same point. Therefore, $\overline{\boldsymbol{x}}^*$ is contained in $\overline{P}$ if and only if the optimum value of (0.4) is at most 1.

In order to obtain a cutting plane for $\overline{\boldsymbol{x}}^*$, we introduce the dual of the LP (0.4):

$$\begin{aligned} \max\ & \boldsymbol{a}^T(\overline{\boldsymbol{x}}^* - \overline{\boldsymbol{q}}) \\ \text{s.t.}\ & \boldsymbol{a}^T(\overline{\boldsymbol{x}}_i - \overline{\boldsymbol{q}}) \leq 1, \text{ for } i = 1, \ldots, s \\ & \boldsymbol{a} \in \mathbb{R}^r. \end{aligned} \tag{0.5}$$

In this context, the fact that $\overline{\boldsymbol{q}}$ is an interior point of $\overline{P}$ has several important consequences. First of all, it ensures that all the candidate inequalities are indeed of the form $\boldsymbol{a}^T(\boldsymbol{x} - \overline{\boldsymbol{q}}) \leq 1$ (up to positive scaling) and thus correspond exactly to the feasible solutions of (0.4). Furthermore, it guarantees that (0.4) is bounded. Finally, it enables the handling of **homogeneous** inequalities which have right hand side zero. For more information we refer to Section 2.3 about polarity in [Zie06].

If the optimum value of the LP (0.4)—and thus of its dual (0.5)—is greater 1 then the corresponding optimum solution $\boldsymbol{a}^*$ of the dual yields the desired cutting plane. Theorem 0.3 summarizes the relevant characteristics.

Theorem 0.3 (cf. Theorem 1 in [BLO08])**.** *Let $\boldsymbol{a} \in \mathbb{R}^r$ and $\overline{P}$ be full-dimensional. Then, $\boldsymbol{a}^T(\boldsymbol{x} - \overline{\boldsymbol{q}}) \leq 1$ is valid for $\overline{P}$ if and only if $\boldsymbol{a}$ is feasible for the dual* (0.5)*. It defines a facet of $\overline{P}$ if and only if $\boldsymbol{a}$ is a vertex of the polytope of feasible solutions of the dual.*

There is also a geometrical interpretation of the target cut generation procedure. Consider the following **non-linear program**:

$$\begin{aligned} \min\ & \mu \\ \text{s.t.}\ & \sum_{i=1}^{s} \lambda_i \leq 1 \\ & \sum_{i=1}^{s} \lambda_i \mu(\overline{\boldsymbol{x}}_i - \overline{\boldsymbol{q}}) = \overline{\boldsymbol{x}}^* - \overline{\boldsymbol{q}} \\ & \boldsymbol{\lambda} \geq \boldsymbol{0}, \mu \geq 0. \end{aligned} \tag{0.6}$$

It is equivalent to the LP (0.4) and can be interpreted as follows: Let $\mu\overline{P}$ denote the polytope $\overline{P}$ scaled by the factor μ with respect to the scaling center $\overline{\boldsymbol{q}}$. The optimum value of (0.6) yields the minimal factor μ by which the polytope $\overline{P}$ has to be scaled in order to contain the infeasible point $\overline{\boldsymbol{x}}^*$. If $\mu > 1$ then the point $\overline{\boldsymbol{x}}^*$ lies outside $\overline{P}$. A graphical depiction is given in Figure 0.2[2].

Although the problem size has already been reduced by considering the projection $\overline{P}$ of the initial polytope P, the formulation of the dual (0.5) still features one column for each vertex of $\overline{P}$. For large numbers of vertices it can be beneficial to use a so-called **delayed column generation procedure**. This approach requires an oracle for maximizing an arbitrary linear function over $\overline{P}$. We start with a small set of vertices $\overline{\boldsymbol{x}}_1, \ldots, \overline{\boldsymbol{x}}_h$ and solve the corresponding reduced linear program to obtain a target cut $\boldsymbol{a}^T\boldsymbol{x} \leq \alpha$ for the polytope $\overline{P}_h = \operatorname{conv}\{\overline{\boldsymbol{x}}_1, \ldots, \overline{\boldsymbol{x}}_h\}$. Next, we call the oracle to decide whether or not the target cut is violated by any vertex of $\overline{P}$. This is done by using $\boldsymbol{a}^T\boldsymbol{x}$ as the linear function to be maximized. If the returned solution $\overline{\boldsymbol{y}}$ satisfies $\boldsymbol{a}^T\overline{\boldsymbol{y}} \leq \alpha$ then the target cut is valid for the polytope $\overline{P}$. Otherwise, we define the additional vertex $\overline{\boldsymbol{x}}_{h+1} := \overline{\boldsymbol{y}}$,

[2]This figure is adapted from [BLO08].

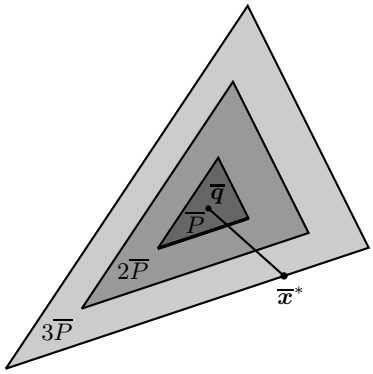


Figure 0.2: Scaling of the polytope $\overline{P}$.

add the corresponding column to the reduced linear program, and search for the next target cut.

In general, the number of columns generated in the course of the above procedure is much smaller than the number of vertices of $\overline{P}$. However, whether or not delayed column generation will improve the performance ultimately depends on the efficiency of the oracle.

Chapter 1

The Max-Cut Problem

The max-cut problem is one of the most studied combinatorial optimization problems. It is a very generic problem that can be used to model various applications. Moreover, it is equivalent to other interesting optimization problems as we will see in Section 1.1.

The max-cut problem can be defined as follows.

Definition 1.1 (Max-Cut Problem)**.** *Let $G = (V, E)$ be an undirected graph with edge weights $\boldsymbol{c} \in \mathbb{R}^{|E|}$. The **max-cut problem** consists in finding a set of nodes $U \subseteq V$ such that the aggregate weight $\boldsymbol{c}\big(\delta(U)\big)$ of the induced cut $\delta(U)$ is maximal.*

The max-cut problem is **NP**-hard in general. Yet, there are certain special cases for which it can be solved in polynomial time. We will elaborate on this topic in Section 1.6.

In this chapter we present heuristics to compute approximate solutions to the max-cut problem. We also cover fundamental polyhedral aspects that are necessary to derive a branch-and-cut algorithm. First, however, we start with two equivalent optimization problems.

1.1 Equivalent Optimization Problems

We now introduce two optimization problems that can be reduced to the max-cut problem, namely unconstrained quadratic $-1/+1$ respectively $0/1$ optimization. The equivalence between quadratic $0/1$ optimization and the max-cut problem has been pointed out by Hammer [Ham65] (see also [PR74, Han79]). The corresponding reduction consists of two steps: First, the quadratic $0/1$ optimization problem is transformed into a quadratic $-1/+1$ optimization problem, which is then modeled as a max-cut problem. For a better understanding, the following sections present these two steps in reverse order.

1.1.1 Unconstrained Quadratic $-1/+1$ Optimization

The **unconstrained quadratic $-1/+1$ optimization problem** is a discrete quadratic optimization problem of the form

$$\begin{aligned} \min \ & \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} q_{ij} x_i x_j + \sum_{i=1}^{n} p_i x_i \\ \text{s.t.} \ & x_i \in \{-1, +1\}, \ \text{for } i = 1, \dots, n. \end{aligned}$$

By introducing an auxiliary variable x_0 and defining $q_{0j} := p_j$, for $j = 1, \ldots, n$, we can eliminate the linear term and reformulate the problem as

$$\begin{aligned} \min \;\; & f(\boldsymbol{x}) := \sum_{i=0}^{n-1} \sum_{j=i+1}^{n} q_{ij} x_i x_j \\ \text{s.t.} \;\; & x_i \in \{-1, +1\}, \;\; \text{for } i = 0, \ldots, n. \end{aligned} \tag{1.1}$$

Obviously, we have $f(\boldsymbol{x}) = f(-\boldsymbol{x})$. Also, we can fix the auxiliary variable x_0 to 1 without loss of generality.

In order to model this quadratic problem as a max-cut problem, we construct the graph $G = (V, E)$ with node set $V := \{0, \ldots, n\}$, edge set $E := \{ij \mid 0 \leq i < j \leq n\}$ and edge weights $c_{ij} := q_{ij}$. Each assignment of values -1 or $+1$ to the variables x_i defines a partition of the node set V into $V^- := \{i \in V \mid x_i = -1\}$ and $V^+ := \{i \in V \mid x_i = +1\}$. Hence, we can write $f(\boldsymbol{x})$ as

$$\begin{aligned} \sum_{i=0}^{n-1} \sum_{j=i+1}^{n} q_{ij} x_i x_j &= \sum_{ij \in E(V^-)} c_{ij} + \sum_{ij \in E(V^+)} c_{ij} - \sum_{ij \in \delta(V^+)} c_{ij} \\ &= -2 \cdot \sum_{ij \in \delta(V^+)} c_{ij} + \boldsymbol{c}(E). \end{aligned}$$

The quadratic $-1/+1$ optimization problem (1.1) can thus be modeled as a max-cut problem on the graph G. It has the optimum value $-2c^* + \boldsymbol{c}(E)$, where c^* is the weight of a maximum cut of G.

1.1.2 Unconstrained Quadratic 0/1 Optimization

The second quadratic optimization problem that is equivalent to the max-cut problem is **unconstrained quadratic 0/1 optimization**. It has the form

$$\begin{aligned} \min \;\; & f(\boldsymbol{x}) := \boldsymbol{x}^T Q \boldsymbol{x} \\ \text{s.t.} \;\; & x_i \in \{0, 1\}, \;\; \text{for } i = 1, \ldots, n, \end{aligned}$$

where $Q = (q_{ij})$ is a symmetric $(n \times n)$ matrix. Without loss of generality, we can assume that $f(\boldsymbol{x})$ does not contain an additional linear term $\boldsymbol{p}^T \boldsymbol{x}$. This is because $x_i^2 = x_i$ and thus the linear term can be eliminated by replacing q_{ii} with $q'_{ii} := q_{ii} + p_i$.

The reduction to a max-cut problem consists of two steps. We first reduce the quadratic 0/1 optimization problem to a quadratic $-1/+1$ optimization problem, which can then be modeled as a max-cut problem as explained in Section 1.1.1.

For the first step, we exploit the symmetry of the matrix Q and get

$$f(\boldsymbol{x}) = 2 \cdot \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} q_{ij} x_i x_j + \sum_{i=1}^{n} q_{ii} x_i.$$

Next, we perform the variable transformation $s_i = 2x_i - 1$. Obviously, the variables s_i satisfy the following condition:

$$s_i = \begin{cases} +1 & \text{if } x_i = 1, \\ -1 & \text{if } x_i = 0. \end{cases}$$

Hence, we obtain

$$\begin{aligned} f(\boldsymbol{s}) &= \frac{1}{2}\Bigg(\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} q_{ij}(s_i+1)(s_j+1) + \sum_{i=1}^{n} q_{ii}(s_i+1)\Bigg) \\ &= \frac{1}{2}\Bigg(\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} q_{ij}(s_i s_j + s_i + s_j) + \sum_{i=1}^{n} q_{ii}s_i + C\Bigg), \end{aligned}$$

with a constant $C := \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} q_{ij} + \sum_{i=1}^{n} q_{ii}$.

To uncover the inherent structure of a generic quadratic $-1/+1$ optimization problem, we rewrite the term $\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} q_{ij}s_j$ in the form $\sum_{i=1}^{n}\sum_{j=1}^{i-1} q_{ji}s_i$, which equals $\sum_{i=1}^{n}\sum_{j=1}^{i-1} q_{ij}s_i$ due to the symmetry of Q. Thus, we get

$$f(\boldsymbol{s}) = \frac{1}{2}\Bigg(\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} q_{ij}s_i s_j + \sum_{i=1}^{n}\Big(\sum_{j=i+1}^{n} q_{ij} + \sum_{j=1}^{i-1} q_{ij} + q_{ii}\Big)s_i + C\Bigg).$$

Note that the middle term $\sum_{i=1}^{n}(\sum_{j=i+1}^{n} q_{ij} + \sum_{j=1}^{i-1} q_{ij} + q_{ii})s_i$ features two so-called **empty sums**, namely $\sum_{j=1}^{0} q_{1j}$ and $\sum_{j=n+1}^{n} q_{nj}$. The value of any empty sum is conventionally taken to be zero. Also, since the expression describing the terms of the summation—in this case q_{ij}—is never instantiated in an empty sum, its value is irrelevant.

Finally, we introduce $p_i := \sum_{j=1}^{n} q_{ij}$ for all $i = 1, \ldots, n$ and obtain

$$f(\boldsymbol{s}) = \frac{1}{2}\Bigg(\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} q_{ij}s_i s_j + \sum_{i=1}^{n} p_i s_i + C\Bigg).$$

Thus, the quadratic 0/1 optimization problem can be reduced to a quadratic $-1/+1$ optimization problem. It has the optimum value $\frac{1}{2}(q^* + C)$, where q^* is the optimum value of the quadratic $-1/+1$ optimization problem.

In the second step, the quadratic $-1/+1$ optimization problem is modeled as a max-cut problem as described in Section 1.1.1. Altogether, we obtain

$$\tfrac{1}{2}(-2c^* + \boldsymbol{c}(E) + C) = -c^* + \tfrac{1}{2}(\boldsymbol{c}(E) + C)$$

as the optimum value of the quadratic 0/1 optimization problem, where c^* is the weight of a maximum cut.

1.2 Applications

Due to its generic nature, the max-cut problem can be used to model a variety of applications. In the following we focus on two particular examples. From the field of statistical physics, we present the problem of finding ground states of Ising spin glasses. The second problem is the minimization of so-called vias—a problem that arises in very-large-scale-integration (VLSI) circuit design as well as in the design of printed circuit boards. The subsequent descriptions are mainly based on [BGJR88]. The figures are adapted from [Rei07].

1.2.1 Ising Spin Glasses

In statistical physics, a **spin glass** is an alloy of magnetic impurities diluted in a nonmagnetic metal. Each magnetic impurity i has a magnetic orientation, called **spin**, which is represented by a three-dimensional unit vector $\boldsymbol{s}_i \in \mathbb{R}^3$. Between each pair ij of impurities there is an interaction J_{ij} that depends on the nonmagnetic material as well as the distance r_{ij} between the impurities. The absolute value of these interactions decreases rapidly with the distance. Hence, it is customary to only consider interactions between close impurities, i. e., the so-called **nearest neighbor interactions**. The resulting models are referred to as **short range models**.

Given a spin configuration ω, the energy of the system is measured by the **Hamiltonian**

$$H(\omega) = -\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} J_{ij}\, \boldsymbol{s}_i^T \boldsymbol{s}_j - h \sum_{i=1}^{n} \boldsymbol{s}_i^T \boldsymbol{f},$$

where $\boldsymbol{f} \in \mathbb{R}^3$ represents the orientation of an external magnetic field of strength h. At a temperature of $0°K$, the spin glass attains a minimum energy configuration called **ground state**. Since the necessary conditions are hard to realize experimentally, researchers rely on models to simulate the behavior of a spin glass. Regarding ground states, the simulation consists in minimizing the Hamiltonian associated with the system. However, the complexity of the Hamiltonian has led to various simplifications. We will focus on one particular simplification called the Ising model.

In the **Ising model** we assume the spins and the external magnetic field to be **Ising spins**, i. e., scalars s_i and f with a value of either $+1$ or -1. Ising spins correspond to the notion of the magnetic north pole being 'up' or 'down'. This seems to be quite restrictive at first glance. However, there are, in fact, substances that show this up/down behavior of the spins and for which the Ising model is accurate.

As further simplification, we assume the spins to be regularly distributed on a two- or three-dimensional grid. Interactions are only considered between neighbors in the grid graph. There are two particular grid models that have been studied intensively. The first one is the **Gaussian model**, in which the interaction values are chosen from a Gaussian distribution. The second one is the **$\pm J$-model**, where interactions only attain the values $+J$ and $-J$, with J being a fixed positive number, according to some distribution. Note that in the models just introduced, the spins are regularly distributed on a grid while the interaction values are random. In a real spin glass, on the other hand, the magnetic impurities themselves are randomly distributed.

We model a spin glass system using a graph $G = (V, E)$ which we call the **interaction graph** associated with the system. The nodes and edges of G represent the impurities and the pairwise interactions between impurities, respectively. To each node i we assign an Ising spin $s_i \in \{-1, +1\}$ and to each edge ij we assign an interaction value J_{ij}. Without loss of generality, we assume the Ising spin f of the external magnetic field to be $+1$. Let h denote the strength of this field. The associated Hamiltonian of the system is a quadratic function in ± 1-variables of the following form:

$$H(\omega) = -\sum_{ij \in E} J_{ij} s_i s_j - h \sum_{i=1}^{n} s_i.$$

Therefore, the problem of finding a ground state can be formulated as an unconstrained

quadratic $-1/+1$ optimization problem, which in turn can be modeled as a max-cut problem as explained in Section 1.1.1.

1.2.2 Via Minimization

The construction of a semiconductor chip comprises several phases, among which are component placement, wire routing, and layer assignment. We assume that the single components have been placed on the chip and all connecting wires have been routed. This is called a **transient routing**. An example of a transient routing for six components and eleven connections is depicted in Figure 1.1. As indicated by the figure, a transient

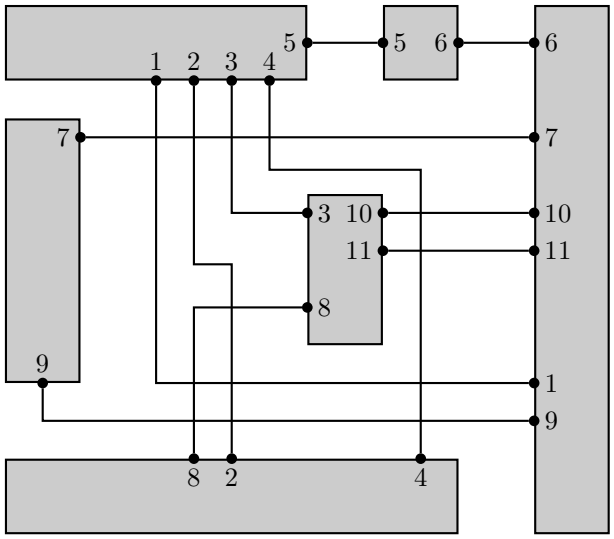


Figure 1.1: Example of a transient routing for six components and eleven connections.

routing may contain wire crossings. In order to avoid short circuits, the crossing wires are assigned to different layers. In this brief introduction we only consider the two-layer case.

Physically, a change of layers is achieved by placing a so-called **via**. In **very-large-scale-integration** (VLSI), a via is a contact that requires special treatment during the production process—it occupies additional space, is an obstacle in compaction, and decreases the yield in the fabrication process. In printed circuit board design, a via is a hole to be drilled—it causes additional costs and also contributes to failure of the board due to cracking. Therefore, it is desirable to find a feasible layer assignment with a minimum number of necessary vias. In the following we present a reduction of the via-minimization problem to the max-cut problem. This reduction was described independently by Pinter [Pin84] for VLSI and by Chen, Kajitani, and Chan [CKC83] for two-layer printed circuit boards.

The placement of vias is restricted by certain design rules. In general, we can partition each wire into **free** and **critical segments**, respectively, such that vias are allowed on free segments while being forbidden on the critical ones. The partition of the wires for our example is depicted in Figure 1.2. The critical segments, which are marked by solid lines, are numbered from 1 to 17. Given such a partition of the wires, we construct the corresponding **layout graph** $G = (V, E)$. Its nodes represent the critical segments. We

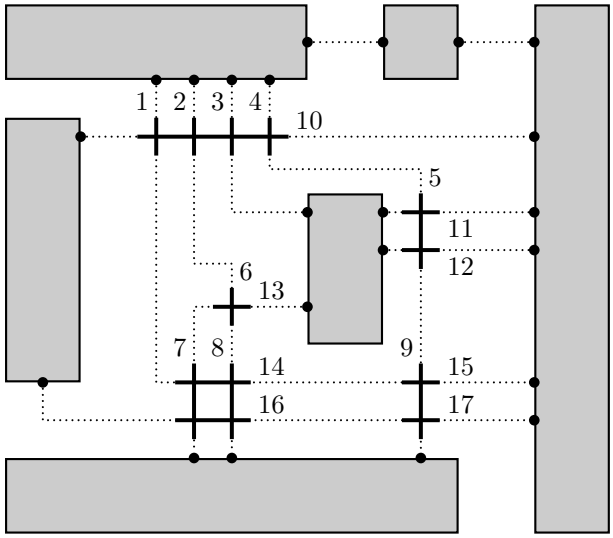


Figure 1.2: Partition of wires into free and critical segments.

distinguish two kinds of edges: The **conflict edges** join nodes whose critical segments have to be placed on different layers; the **continuation edges** join nodes whose critical segments are connected by a free segment. We define the edge set E of the layout graph as the disjoint union of the set A of conflict edges and the set B of continuation edges. Figure 1.3 shows the layout graph for our example. Conflict edges are represented by solid lines and continuation edges by dashed lines, respectively.

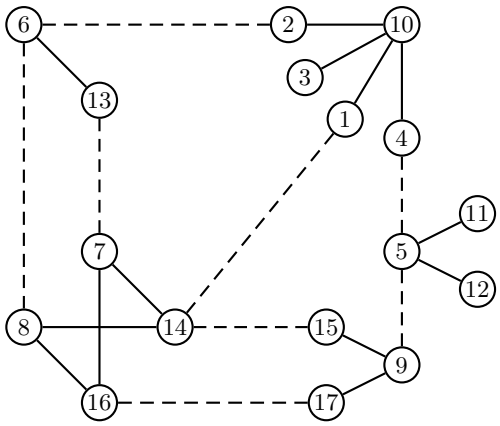


Figure 1.3: Layout graph with conflict edges (solid) and continuation edges (dashed).

We then consider the so-called **conflict graph** $H = (V, A)$. It has the same node set as the layout graph, but its edge set is restricted to the conflict edges. If H is not bipartite then there is no feasible layer assignment and the transient routing has to be changed. Otherwise, the conflict graph decomposes into bipartite connected components $(V_1, A_1), \ldots, (V_k, A_k)$. Clearly, the assignment of one node of a component (V_i, A_i) to a layer implies the assignment of all other nodes of V_i. As a result, we can contract each component (V_i, A_i) to an arbitrary representative node $v_i \in V_i$ without loss of generality. When applied to the layout graph G, this contraction removes all conflict

edges and possibly merges some of the continuation edges. Altogether, we obtain a **reduced layout graph** $R = (W, F)$ with the node set $W = \{v_1, \ldots, v_k\}$ and the edge set $F = \{v_i v_j \mid \exists\, u \in V_i, v \in V_j \text{ such that } uv \in E\}$. The reduced layout graph R is planar since, by definition, continuation edges do not cross. To each edge $v_i v_j$ in F we assign a pair of weights α_{ij} and β_{ij}, where

$$\begin{aligned} \alpha_{ij} &:= \text{number of vias necessary between } V_i \text{ and } V_j \text{ if } v_i \text{ and } v_j \text{ are} \\ &\quad\ \text{assigned to the same layer, and} \\ \beta_{ij} &:= \text{number of vias necessary between } V_i \text{ and } V_j \text{ if } v_i \text{ and } v_j \text{ are} \\ &\quad\ \text{assigned to different layers.} \end{aligned}$$

Figure 1.4(a) shows the reduced layout graph for our example. The edges are labeled with the weights $(\alpha_{ij}, \beta_{ij})$.

Finally, a layer assignment corresponds to a cut $(W_1 : W_2)$ of the reduced layout graph R. For a given cut, the number of necessary vias ν is defined as follows:

$$\nu(W_1, W_2) := \sum_{\substack{v_i v_j \in F \\ v_i, v_j \in W_1}} \alpha_{ij} + \sum_{\substack{v_i v_j \in F \\ v_i, v_j \in W_2}} \alpha_{ij} + \sum_{\substack{v_i v_j \in F \\ v_i \in W_1, v_j \in W_2}} \beta_{ij}.$$

Using the constant $C := \sum_{v_i v_j \in F} \alpha_{ij}$, we obtain

$$\nu(W_1, W_2) - C = \sum_{\substack{v_i v_j \in F \\ v_i \in W_1, v_j \in W_2}} (\beta_{ij} - \alpha_{ij}).$$

Therefore, the problem of minimizing the number of vias is equivalent to the max-cut problem on the reduced layout graph R with the edge weights $c_{ij} := \alpha_{ij} - \beta_{ij}$. For a given maximum cut $\delta(S)$ of R, the minimum number of necessary vias in the corresponding layer assignment is $C - \sum_{v_i v_j \in \delta(S)} c_{ij}$.

In our example, a maximum cut is, for instance, the cut $\big(\{1, 6\} : \{5, 7, 9\}\big)$ with weight zero, as illustrated in Figure 1.4(b). The corresponding optimum layer assignment, shown

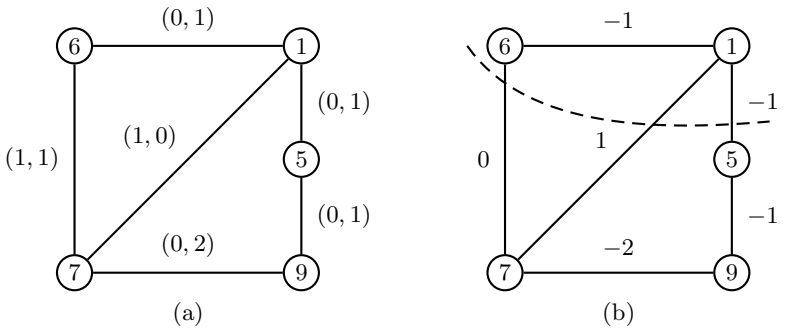


Figure 1.4: Reduced layout graph (a) and corresponding maximum cut (b).

in Figure 1.5, requires two vias and places the critical segments {1–4,6,11,12,14–17} on the first layer and {5,7–10,13} on the second one.

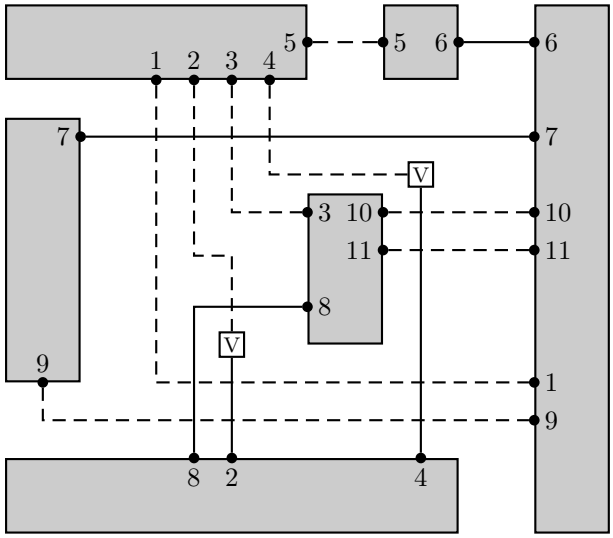


Figure 1.5: An optimum layer assignment using two vias.

1.3 Heuristics

A **heuristic** is a technique designed to obtain a feasible solution to a given problem with little computational effort. In general, a heuristic does not provide the means to estimate the quality of the solution. We distinguish two basic heuristic approaches: the construction of feasible solutions from scratch and the improvement of existing solutions, respectively. The following general survey is mainly based on [Rei94].

Construction heuristics are methods that determine a feasible solution according to some construction rule but do not try to improve upon this solution. In other words, a feasible solution is built successively and the parts already built remain in a certain sense unchanged throughout the algorithm. Possible construction heuristics for the max-cut problem include the following ones:

- **greedy**: nodes are successively assigned to the shores of the nascent cut. In each step, we assign a remaining node that yields a maximum increase in the weight of the cut with respect to the nodes already assigned.
- **spanning tree**: constructs a maximum weight spanning tree with respect to the absolute values of the edge weights. Based on this tree, a cut is constructed that contains all of the tree's edges with a positive original edge weight but none of those with a negative one.
- **random**: nodes are assigned randomly to the shores of the nascent cut.

The feasible solutions computed by construction heuristics are often of only moderate quality. Though these heuristics might be useful for special applications, they are not satisfactory in general. Their results, however, can be used as initial solutions for improvement heuristics.

Improvement heuristics are characterized by a certain type of basic move to alter a given feasible solution. This move is applied iteratively to the solution until a certain stop criterion is met. Common improvement heuristics for the max-cut problem are, for instance, the following ones:

- k-**opt**: checks whether the weight of the cut can be increased by assigning k nodes to their respective opposite shore of the cut.
- **local enumeration**: chooses small subsets of the nodes and uses total enumeration to determine the best possible assignment of the chosen nodes to the shores of the cut.
- **Kernighan-Lin**: builds complicated modifications that are composed of simple moves, for instance, 1-opt or 2-opt moves, where not all of these moves necessarily have to increase the weight of the cut.

In the scope of this thesis, we are concerned with solving max-cut problems to optimality using the branch-and-cut method. In this regard, good heuristics are a vital component as they affect the effectiveness of the pruning and thus the overall performance of the branch-and-cut method. In the subsequent sections we elaborate on two of the heuristics just mentioned, namely the spanning tree heuristic and the Kernighan-Lin heuristic. For more information on the remaining heuristics, we refer to [Rei07].

1.3.1 Spanning Tree Heuristic

The **spanning tree heuristic** is a construction heuristic. It is particularly suitable for sparse graphs. Initially, we calculate a maximum weight spanning tree T_{max} of the graph G with respect to the absolute edge weights $w_e = |c_e|$, for all $e \in E$. From T_{max} we can derive a cut of G that contains all of the tree's edges with positive c_e but none of those with negative c_e. The pseudocode of the heuristic is given in Algorithm 1.1. The overall running time is mainly due to the computation of the maximum spanning tree. Using an efficient implementation of Kruskal's algorithm (see, for instance, [CLRS09]), the heuristic runs in time $\mathcal{O}(|E| \log |E|)$. Easier implementations yield a running time of $\mathcal{O}(|V|^2)$.

Algorithm 1.1: Spanning Tree Construction Heuristic

Input: Undirected weighted graph $G = (V, E, \boldsymbol{c})$.
Output: Cut $(S : T)$ of G.

`// Initialization`
Define absolute edge weights $w_e = |c_e|$, for all $e \in E$;

`// Compute a maximum weight spanning tree`
Compute a maximum weight spanning tree T_{max} of $G' := (V, E, \boldsymbol{w})$;
Mark an arbitrary node $r \in V$ as root of T_{max} and set $S = \{r\}$ and $T = \emptyset$;

`// Construct the cut based on` T_{max}
Traverse T_{max} in a breadth-first-search manner starting at r. Let uv be the current edge with u being the predecessor of v within T_{max}, i. e., u has already been assigned to either S or T;

if $c_{uv} > 0$ **then**
  Assign v to the opposite shore of u;
else
  Assign v to the same shore as u;

With some minor adjustments we can use the spanning tree heuristic as a **rounding heuristic**. The purpose of a rounding heuristic is to find a feasible solution that is preferably close to a given fractional LP solution $\boldsymbol{z}$. To this end, we set the initial weights to $w_e := |z_e - \frac{1}{2}|$, for all $e \in E$. Thus, the larger the edge weight w_e, the closer the LP value of the respective variable is to either 0 or 1. From a maximum weight spanning tree with respect to the weight vector $\boldsymbol{w}$ we then construct a cut of G. This cut contains all of the tree's edges with an LP value near 1 and none of those with an LP value near 0. The pseudocode of this rounding heuristic is given in Algorithm 1.2.

Algorithm 1.2: Spanning Tree Rounding Heuristic

Input: Undirected graph $G = (V, E)$ and fractional LP solution $\boldsymbol{z}$.
Output: Cut $(S : T)$ of G.

`// Initialization`
Define edge weights $w_e := |z_e - \frac{1}{2}|$, for all $e \in E$;

`// Compute a maximum weight spanning tree`
Compute a maximum weight spanning tree T_{max} of $G' := (V, E, \boldsymbol{w})$;
Mark an arbitrary node $r \in V$ as root of T_{max} and set $S = \{r\}$ and $T = \emptyset$;

`// Construct the cut based on` T_{max}
Traverse T_{max} in a breadth-first-search manner starting at r. Let uv be the current edge with u being the predecessor of v within T_{max}, i. e., u has already been assigned to either S or T;

if $z_{uv} > \frac{1}{2}$ **then**
  Assign v to the opposite shore of u;
else
  Assign v to the same shore as u;

1.3.2 Kernighan-Lin Heuristic

Two common observations regarding improvement heuristics in general are the following ones:

- the more flexible and powerful the possible modifications are, the better results are usually obtained,
- simple moves quickly get stuck in local optima of only moderate quality that cannot be left anymore.

The approach of the **Kernighan-Lin heuristic** is based on the experience that sometimes a modification slightly decreasing the weight of a cut can open up new possibilities for achieving considerable improvements afterwards. Its basic principle is to build complicated modifications that are composed of simple moves where not all of these moves necessarily have to increase the weight of the cut. Naturally, to obtain reasonable running times, the effort to find the parts of the composed move has to be limited.

Originally, Kernighan and Lin [KL70] developed this heuristic to compute minimum cuts with shores of equal cardinality. However, the general approach can easily be adapted to the max-cut problem. The pseudocode of a Kernighan-Lin heuristic using 1-opt moves is presented in Algorithm 1.3.

Algorithm 1.3: Kernighan-Lin Improvement Heuristic

Input: Undirected weighted graph $G = (V, E, \boldsymbol{c})$ and initial cut $(S : T)$ of G.
Output: Improved cut $(S' : T')$ of G.

Set $S' = S$ and $T' = T$;

`// Repeat until first iteration without improvement occurs`
repeat
  `// Initialization`
  Set $p = 1$, $V_p = V$;
  for *all nodes* $v \in V$ **do**
    `// Determine change of cut weight if` v `switches shores`
    Let $N(v)$ be the set of nodes adjacent to v.
    if $v \in S'$ **then**
      Set $D_v = \sum_{u \in N(v) \cap S'} c_{uv} - \sum_{u \in N(v) \cap T'} c_{uv}$;
    else
      Set $D_v = \sum_{u \in N(v) \cap T'} c_{uv} - \sum_{u \in N(v) \cap S'} c_{uv}$;
  `// Assign` $|V| - 1$ `nodes to their respective opposite shore`
  while $p < |V|$ **do**
    `// Determine best remaining node to switch shores`
    Set $v_p^* = \arg\max_{v \in V_p} D_v$ and $V_{p+1} = V_p \setminus \{v_p^*\}$;
    `// Update` D_u `for remaining neighbors of` v_p^*
    for *all nodes* $u \in V_{p+1}$ *adjacent to* v_p^* **do**
      if u *and* v_p^* *are in different shores* **then**
        Set $D_u = D_u + 2c_{uv_p^*}$;
      else
        Set $D_u = D_u - 2c_{uv_p^*}$;
    Set $p = p + 1$;
  `// Determine best composed modification along the search path`
  Determine $k \in \{1, \ldots, |V| - 1\}$ such that $\Delta = \sum_{i=1}^{k} D_{v_i^*}$ is maximal;
  `// Update` $(S' : T')$
  if $\Delta > 0$ **then**
    Assign the nodes $v_1^*, \ldots, v_k^*$ to their respective opposite shore;
until $\Delta \leq 0$;

In our max-cut solver we use a combination of Algorithms 1.2 and 1.3 to compute lower bounds on the optimum value. First, we use the spanning tree rounding heuristic to obtain a feasible solution near the current fractional LP solution. Then, we apply the Kernighan-Lin heuristic to improve upon this preliminary solution. In our experience, this combined approach provides solutions of high quality that quickly converge towards an optimum solution. In particular, the resulting lower bounds are significantly better than the ones obtained from the rounding heuristic alone or from initializing the Kernighan-Lin heuristic with random solutions.

1.4 Cut Polytope and Polyhedral Results

The polyhedral study of the polytope associated with the max-cut problem serves an important purpose. Not only does it provide a better understanding of the structure of feasible solutions to the problem, but it is also essential to the development of a branch-and-cut algorithm. The results below are mainly based on [BM86, DL97, Rei07]. The figures are adapted from [Rei07].

For the max-cut problem on a given graph $G = (V, E)$, we define the associated **cut polytope** $\mathrm{CUT}(G)$ as the convex hull of the incidence vectors of all the cuts of G, i. e.,

$$\mathrm{CUT}(G) := \mathrm{conv}\{\boldsymbol{\chi}^{\delta(S)} \mid S \subseteq V\}.$$

Figure 1.6 shows the cut polytope for the complete graph of order three. Note that the origin $\mathbf{0}$ is a vertex of $\mathrm{CUT}(G)$.

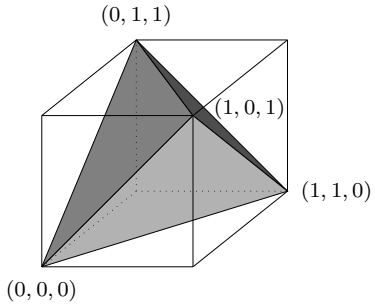


Figure 1.6: Cut polytope $\mathrm{CUT}(K_3)$.

The cut polytope is full dimensional [BGM85], i. e., $\dim\big(\mathrm{CUT}(G)\big) = |E|$. It has $2^{|V|-1}$ vertices. This is exactly the number of possibilities to partition the $|V|$ nodes into two shores, one of which may be empty. The number of facets, however, grows super-exponentially in the order of the graph.

Table 1.1 shows statistics on the facet structure of $\mathrm{CUT}(K_n)$ for $n = 3, \ldots, 9$ as published by Christof and Reinelt [CR97]. More detailed information is available in the SMAPO "library of linear descriptions of SMAll combinatorial POlytopes" [Chr95]. The facet classes mentioned in the rightmost column of Table 1.1 classify the facets with respect to two symmetries, namely the σ-symmetry—given by the permutations of the nodes—and the symmetry of the switching operation which we will introduce in Section 2.1.1. The linear descriptions of $\mathrm{CUT}(K_n)$ for $n \leq 7$ are proven to be complete [Gri90] whereas the completeness of those for $n = 8$ and $n = 9$ is only conjectured.

n	#Vertices	#Facets in total	#Facet classes
3	4	4	1
4	8	16	1
5	16	56	2
6	32	368	3
7	64	116,764	11
8	128	$\geq$ 217,093,472	$\geq$ 147
9	256	$\geq$ 12,246,651,158,320	$\geq$ 164,506

Table 1.1: Statistics on the facet structure of $\mathrm{CUT}(K_n)$ for $n = 3, \ldots, 9$.

1.4.1 Selected Facet Defining Inequalities

We now introduce selected classes of valid inequalities for the cut polytope. Moreover, we specify under which conditions these inequalities define facets. The subsequent theorems are presented without the respective proofs which can be found in [BGM85, BM86, DL97].

We start with the most basic inequalities. By definition, an incidence vector is a binary vector. Consequently, each of its entries must be in the closed range $[0, 1]$.

Theorem 1.2. *The* ***trivial inequalities*** $x_e \geq 0$ *and* $x_e \leq 1$, $e \in E$, *are valid for* $\mathrm{CUT}(G)$. *They define facets of* $\mathrm{CUT}(G)$ *if and only if the edge* e *is not part of a* 3*-cycle.*

In addition, if a given vector $\boldsymbol{x}$ is indeed the incidence vector of a cut of G then the number $\boldsymbol{x}(C)$ is even for every cycle C of G. This is because a cut and a cycle can only have an even number of edges in common. The following inequalities model this property.

Theorem 1.3. *For each cycle* C *of* G *and each subset* $F \subseteq C$ *with* $|F|$ *odd, the* ***odd-cycle inequality***

$$\boldsymbol{x}(F) - \boldsymbol{x}(C \setminus F) \leq |F| - 1$$

is valid for $\mathrm{CUT}(G)$. *It defines a facet of* $\mathrm{CUT}(G)$ *if and only if* C *has no chord in* G.

We can also derive valid inequalities for $\mathrm{CUT}(G)$ from the cliques of the graph.

Theorem 1.4. *Let* (W, F) *be a clique of order* $p \geq 3$ *of* G. *Then, the* K_p***-inequality***, *or* ***clique inequality***,

$$\boldsymbol{x}(F) \leq \left\lceil \frac{p}{2} \right\rceil \left\lfloor \frac{p}{2} \right\rfloor$$

is valid for $\mathrm{CUT}(G)$. *It defines a facet of* $\mathrm{CUT}(G)$ *if and only if* p *is odd.*

Note that the above clique inequality remains valid even if we only consider a subset of the clique's edge set F. This is because we can simply add the trivial inequality $-x_e \leq 0$ for each of the missing edges.

Another structure that yields valid inequalities for $\mathrm{CUT}(G)$ is depicted in Figure 1.7. It is called a **bicycle-p-wheel** and consists of a cycle of length p as well as two additional nodes that are adjacent to each other and to every node in the cycle.

Theorem 1.5. *Let* (W, F) *be a bicycle-*p*-wheel,* $p \geq 3$, *contained in* G. *Then, the* ***bicycle-p-wheel inequality***

$$\boldsymbol{x}(F) \leq 2p$$

is valid for $\mathrm{CUT}(G)$. *It defines a facet of* $\mathrm{CUT}(G)$ *if and only if* p *is odd.*

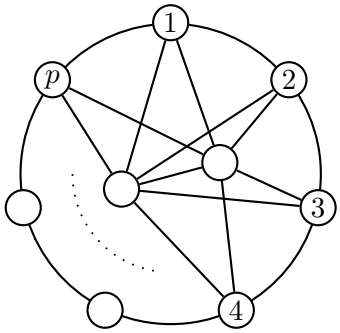


Figure 1.7: A bicycle-p-wheel.

Note that the bicycle-3-wheel inequality is identical to the K_5-inequality, but this is the only intersection of these two classes.

Finally, we introduce a class of valid homogeneous inequalities for the **cut cone** associated with K_n which is the conic hull of the incidence vectors of the cuts of K_n.

Theorem 1.6. *Let $\boldsymbol{b} = (b_1, \ldots, b_n)$, $n \geq 2$, be an integral vector satisfying $\sum_{i=1}^{n} b_i = 1$. Then, the **hypermetric inequality***

$$\sum_{1 \leq i < j \leq n} b_i b_j x_{ij} \leq 0$$

is valid for the cut cone associated with K_n.

Note that the 3-cycle inequalities are a special case of the hypermetric inequalities. A hypermetric inequality is called **pure** if $b_i \in \{-1, 0, 1\}$ for all $i = 1, \ldots, n$.

Lemma 1.7. *All pure hypermetric inequalities define facets of the cut cone associated with K_n.*

It is important to emphasize that the hypermetric inequalities are valid if and only if the associated graph is complete. Moreover, they are first and foremost valid for the cut *cone*. Of course, since the cut polytope is contained in the cut cone, this implies that the hypermetric inequalities are also valid for $\mathrm{CUT}(K_n)$. Nevertheless, it limits the use of this class of inequalities as cutting planes for a branch-and-cut method. For instance, if we want to separate a fractional LP solution that lies outside the cut polytope but inside the cut cone then hypermetric inequalities are of no use. Fortunately, this problem can be circumvented using certain transformations. We will discuss this topic in detail in Section 2.1.4.

1.4.2 Lifting Inequalities

Assume that we have found some valid or even facet defining inequalities for the cut polytope on a given graph G. Then, an obvious question is the following: Can we use the inequalities at hand to derive valid or facet defining inequalities for the cut polytope on a supergraph G' of G? This question leads to the concept of **lifting** which denotes a procedure for constructing a valid, preferably facet defining, inequality of $\mathrm{CUT}(G')$ from a given valid or facet defining inequality for the cut polytope on a subgraph G of G'.

In general, we distinguish the following two lifting operations: The first one is **node lifting** which adds a new node v as well as a nonempty subset of the set $\delta(v)$ of possible edges from v to the existing nodes. The second operation is **edge lifting** which adds edges between already existing but nonadjacent nodes. We call these missing edges the **non-edges** of the graph.

The simplest form of lifting is the so-called **trivial lifting**, or **0-lifting**. When trivially lifting an inequality $\boldsymbol{a}^T\boldsymbol{x} \leq \alpha$ for $\mathrm{CUT}(G)$ to an inequality $(\boldsymbol{a}')^T\boldsymbol{x} \leq \alpha$ for $\mathrm{CUT}(G')$, the original edges in $E(G)$ keep their left hand side coefficients while the newly added edges in $E(G') \setminus E(G)$ get a coefficient of zero. In the scope of this thesis, the use of trivial lifting is restricted to the case that G is a node-induced subgraph of G'. As a result, we will focus exclusively on trivial *node* lifting.

To begin with, we consider the complete graph $K_n = (V_n, E_n)$ of order n. We define the **0-node lifting** $\boldsymbol{a}' \in \mathbb{R}^{E_{n+1}}$ of a left hand side vector $\boldsymbol{a} \in \mathbb{R}^{E_n}$ by

$$\begin{aligned} a'_{ij} &= a_{ij} \quad \text{for } ij \in E_n, \\ a'_{i,n+1} &= 0 \quad \text{for } 1 \leq i \leq n. \end{aligned}$$

It is easy to see that 0-node lifting preserves the validity of inequalities. Suppose not and that the 0-node lifting of the valid inequality $\boldsymbol{a}^T\boldsymbol{x} \leq \alpha$ for $\mathrm{CUT}(K_n)$ results in an inequality $(\boldsymbol{a}')^T\boldsymbol{x} \leq \alpha$ which is invalid for $\mathrm{CUT}(K_{n+1})$. Consequently, there exists a subset S of V_{n+1} such that the incidence vector of the corresponding cut $\delta(S)$ violates the lifted inequality. However, since the contribution of the edges in $E_{n+1} \setminus E_n$ to the left hand side value is zero, the incidence vector of the cut $\delta(S \cap V_n)$ of K_n must violate the original inequality $\boldsymbol{a}^T\boldsymbol{x} \leq \alpha$, thereby contradicting the assumed validity.

In addition, Deza and Laurent [DL97] proved that 0-node lifting even preserves an existing facet-defining property of an inequality for the cut polytope.

Theorem 1.8 (cf. Theorem 26.5.1 in [DL97])**.** *Given $\alpha \in \mathbb{R}$, $\boldsymbol{a} \in \mathbb{R}^{E_n}$, and its 0-node lifting $\boldsymbol{a}' \in \mathbb{R}^{E_{n+1}}$, the following assertions are equivalent.*

(i) The inequality $\boldsymbol{a}^T\boldsymbol{x} \leq \alpha$ is facet defining for $\mathrm{CUT}(K_n)$.

(ii) The inequality $(\boldsymbol{a}')^T\boldsymbol{x} \leq \alpha$ is facet defining for $\mathrm{CUT}(K_{n+1})$.

To illustrate the statement of Theorem 1.8, we consider the 0-node lifting of the inequality classes introduced in Section 1.4.1. Note that the trivial inequalities do not define facets of $\mathrm{CUT}(K_n)$ since every edge in a complete graph is part of a 3-cycle. Also, the 3-cycle inequalities are the only facet defining odd-cycle inequalities for $\mathrm{CUT}(K_n)$. This is because every cycle of length greater three has a chord.

Looking at the different inequality classes, we see that 0-node lifting preserves all the relevant structures and properties. Take, for instance, the 3-cycle inequalities: Adding another node and the edges $E_{n+1} \setminus E_n$ can only create new 3-cycles, but it cannot destroy the existing ones. The same holds for cliques and bicycle-p-wheels. Finally, the 0-node lifting of a pure hypermetric inequality with the underlying vector $\boldsymbol{b}$ is again a pure hypermetric inequality with the underlying vector $\boldsymbol{b}' = (\boldsymbol{b}, 0)$.

The situation is different, though, for the cut polytope on an arbitrary graph G. While 0-node lifting still preserves the validity of an inequality—the proof is analogous to the case that G is complete—it does *not* preserve an existing facet-defining property in general. The simplest counterexample is the class of trivial inequalities. Consider, for

instance, a graph $G = (V, E)$ and an edge $e = uv$ that is not part of a 3-cycle. Assume that the 0-node lifting introduces an additional node w as well as the edges uw and vw. Then, the lifted counterparts of the trivial inequalities $x_e \geq 0$ and $x_e \leq 1$, which define facets of $\mathrm{CUT}(G)$, are still valid for the cut polytope on the supergraph G' of G. Yet, they do *not* define facets of $\mathrm{CUT}(G')$ since the edge e is now part of a 3-cycle.

In conclusion, 0-node lifting for the cut polytope on arbitrary graphs always preserves the validity of an inequality. Moreover, for certain inequalities such as odd-cycle-, clique-, and bicycle-p-wheel inequalities it even preserves an existing facet-defining property. Still, we have to keep in mind that the latter result is not true for arbitrary inequalities.

1.5 Solving Max-Cut to Optimality with Branch-and-Cut

Using the cut polytope, we can formulate the max-cut problem on a graph $G = (V, E)$ with edge weights $\boldsymbol{c} \in \mathbb{R}^{|E|}$ as the following linear program:

$$\begin{aligned} \max \ & \boldsymbol{c}^T \boldsymbol{x} \\ \text{s.t.} \ & \boldsymbol{x} \in \mathrm{CUT}(G). \end{aligned}$$

Given a complete linear description of $\mathrm{CUT}(G)$, we can solve the above LP using, for instance, the simplex method. In general, however, the required complete description is either not known or it comprises too many inequalities to be handled efficiently as indicated in Table 1.1 on page 31. For this reason, we use branch-and-cut to solve the max-cut problem to optimality. As pointed out in Section 0.4, a key advantage of branch-and-cut is that it can work with a partial and/or approximate linear description.

A suitable approximation of the cut polytope $\mathrm{CUT}(G)$ is the so-called **semimetric polytope** $\mathrm{MET}(G)$. Barahona [Bar93] proved that the semimetric polytope is described by the following linear system that comprises only trivial inequalities and odd-cycle inequalities:

$$\begin{aligned} \boldsymbol{x}(F) - \boldsymbol{x}(C \setminus F) &\leq |F| - 1 && \text{for all cycles } C \text{ of } G, \\ & && \text{for each } F \subseteq C, |F| \text{ odd}, \\ x_e &\geq 0 && \text{for all } e \in E, \\ x_e &\leq 1 && \text{for all } e \in E. \end{aligned} \tag{1.2}$$

For the complete graph K_n, however, the semimetric polytope is already completely described by the **triangle inequalities**, i. e.,

$$\begin{aligned} x_{ij} - x_{ik} - x_{jk} &\leq 0, \\ x_{ij} + x_{ik} + x_{jk} &\leq 2, \end{aligned}$$

for distinct nodes i, j, and k.

The semimetric polytope has the following two key properties:

$$\mathrm{CUT}(G) \subseteq \mathrm{MET}(G) \subseteq [0,1]^{|E|}, \qquad \mathrm{CUT}(G) \cap \{0,1\}^{|E|} = \mathrm{MET}(G) \cap \{0,1\}^{|E|}.$$

So, first of all, the semimetric polytope contains the cut polytope. Barahona and Mahjoub [BM86] showed that the two polytopes are even identical if and only if the underlying graph G is not contractible to K_5. Moreover, both polytopes share the same binary points, which are exactly the incidence vectors of the cuts of G. In other words, the semimetric polytope is a domain relaxation of the cut polytope that preserves the

set of feasible solutions. Hence, we can formulate the max-cut problem as the following binary program:

$$\begin{aligned} \max \ & \boldsymbol{c}^T \boldsymbol{x} \\ \text{s.t.} \ & \boldsymbol{x} \in \mathrm{MET}(G) \\ & \boldsymbol{x} \in \{0,1\}^{|E|}. \end{aligned}$$

In practice, however, even the linear description of $\mathrm{MET}(G)$ is too large to be handled efficiently. Initializing the branch-and-cut algorithm with the complete set of odd-cycle inequalities will considerably slow down the overall optimization process, in particular for larger instances. Nevertheless, Barahona and Mahjoub [BM86] showed that the separation problem for the system (1.2) can be solved in polynomial time.

As a consequence, we start only with the trivial inequalities and let the separation add further violated inequalities as required. Of course, we are not restricted to odd-cycle inequalities, but we can separate any class of valid inequalities. In general, however, this approach only works in conjunction with an additional **feasibility test** which is a method to determine whether or not a given binary vector defines a cut of G.

A possible feasibility test for the max-cut problem works as follows. Suppose we are given a binary vector $\boldsymbol{x}$. If $\boldsymbol{x}$ indeed defines a cut of G then it must be possible to label the nodes of G with two labels—representing the two shores of the cut—such that the ends of an edge e have the same label if and only if $x_e = 0$. This condition can easily be checked by traversing the graph in a breadth-first-search manner and labeling the nodes according to the above rule. If, at some point, the feasibility test reaches an already labeled node that would now need to be assigned the opposite label then the vector $\boldsymbol{x}$ is infeasible and we can abort the test. Otherwise, the test terminates without contradictions and thus $\boldsymbol{x}$ defines a cut of G.

In conclusion, each time the branch-and-cut algorithm finds an integral solution, we have to invoke the above feasibility test to check whether or not the solution defines a cut of G. Other than that, the workflow of a branch-and-cut algorithm for the max-cut problem conforms to the flowchart depicted in Figure 0.1 on page 14.

1.6 Short Summary of Known Results

We conclude this chapter with a survey of relevant results for the max-cut problem and its associated polytope. The following summary is mainly adapted from [GJR87].

The max-cut problem is **NP**-hard for general graphs with either arbitrary edge weights [Kar72] or all edge weights equal to 1 [GJS76]. Moreover, the problem is **NP**-hard for many important special types of graphs, including the following ones:

- cubic graphs [Yan78],
- graphs not contractible to K_6 [Bar83],
- almost planar cubic graphs [Bar83],
- three-dimensional grid graphs [Bar82],
- two layer grid graphs with weights $0, \pm 1$ [Bar82],
- planar grid graphs with weights $0, \pm 1$ and a universal node [Bar82].

However, there are certain classes of graphs and objective functions, respectively, for which the max-cut problem is polynomial.

For general graphs with exclusively nonpositive weights, the max-cut problem is equivalent to the polynomial-time solvable **min-cut problem**.

Dorfman and Orlova [DO72] as well as Hadlock [Had75] independently found a reduction of the max-cut problem in planar graphs to the so-called T**-join problem**, which can be solved in polynomial time for general graphs using an algorithm of Edmonds and Johnson [EJ73].

Barahona [Bar83] proved that the max-cut problem is polynomial for graphs not contractible to K_5. To do so, he used a theorem of Wagner [Wag37] which shows that every such graph can be decomposed into planar graphs and copies of a particular cubic graph of order eight. This ultimately allowed Barahona to reduce the overall problem to a sequence of polynomial-time solvable max-cut problems on planar and cubic graphs, respectively.

Moreover, the max-cut problem is polynomial for the following special cases:

- weakly bipartite graphs with exclusively nonnegative edge weights [GP81],
- graphs of bounded tree-width [Dre86],
- graphs with no long odd-cycles [GN84],
- graphs with bounded genus and a bounded number of different edge weights [GL99] (the special case of ± 1 weights had been addressed earlier by Barahona [Bar81]),
- graphs with bounded genus and integral edge weights bounded in absolute value by a polynomial of the size of the graph [GL99].

There have been extensive polyhedral studies of the max-cut problem on the complete graph K_n as well as the associated polytope $\mathrm{CUT}(K_n)$. Several families of valid inequalities for $\mathrm{CUT}(K_n)$, some of them facet defining, have been described (see, for instance, the surveys [DL97, PT94]). For some of these inequalities, separation procedures have been proposed (see, for instance, [BH93, DL97, DR94]). Very interesting computational results have been obtained after the introduction of a **semidefinite programming** (SDP) relaxation of the max-cut problem on K_n. The incidence vectors of the cuts of K_n are strongly related to certain symmetric positive semidefinite matrices. The set of all these matrices is in a one-to-one correspondence with a convex body H_n that, after a suitable affine transformation, contains $\mathrm{CUT}(K_n)$. In addition, the optimization of a linear function over H_n can be done efficiently via interior point techniques [HRVW96]. Using such a relaxation, Goemans and Williamson [GW95] were able to provide an approximation algorithm that delivers solutions with a guaranteed value of at least 0.87856 times the optimal value. Finally, by strengthening the SDP relaxation with linear inequalities, Helmberg and Rendl [HR95] have solved instances with up to one hundred nodes to optimality. For a self-contained introduction to semidefinite programming and its application in combinatorial optimization, we refer to [Hel00].

All the results in the previous paragraph are concerned exclusively with the max-cut problem on complete graphs. However, there are also interesting applications that require the exact solution of large max-cut problem instances on sparse graphs. A prime example is the computation of ground states of Ising spin glasses as described in Section 1.2.1. The question is whether the existing knowledge about $\mathrm{CUT}(K_n)$ can be utilized to solve the max-cut problem on an arbitrary graph G.

The trivial way to reduce the max-cut problem on an arbitrary graph G of order n to the max-cut problem on K_n is **artificial completion**. Here, we add all the missing edges

to the graph G and assign a weight of zero to them. This technique has been successfully used for other combinatorial optimization problems, where the sparsity of the original graph can be exploited to handle the completed graph efficiently. This is, for instance, the case for the traveling salesman problem [PR91]. Yet, for the max-cut problem, no such way to take advantage of a possible sparse structure of the original graph is known so far. Thus, the artificial completion will ultimately lead to the same computational complexity as in the case of a complete graph.

These observations illustrate the need for a better understanding of the cut polytope on arbitrary graphs in general as well as on sparse graphs in particular. But apart from an initial polyhedral study by Barahona and Mahjoub [BM86] and some computational results for spin glass instances on toroidal grid graphs (see, e. g., [DDJ$^+$95, DDJ$^+$96]), little effort has been devoted to this topic. The apparent lack of results motivated our investigation of a new separation approach for CUT(G) that we will present in the next chapter.

Chapter 2

Shrink Separation

In this chapter we introduce the **shrink separation** approach for the cut polytope $\mathrm{CUT}(G)$. The method uses a contracted version of the graph G, which has two main advantages: On the one hand, the separation can be performed faster due to the reduced size of the graph. On the other hand, we can introduce appropriate artificial LP values for the non-edges of the contracted graph to obtain an LP solution on the complete graph. This allows us to apply separation techniques for dense and complete graphs that we possibly would not have been able to use otherwise. In this regard, the prior contraction of the graph significantly reduces the number of non-edges and respective artificial LP values that need to be added.

The chapter is organized as follows: We start with an outline of the shrink separation to provide a basic understanding of the method as a whole. Afterwards, Section 2.1 elaborates on the single steps of the shrink separation and their respective underlying theory. Finally, in Section 2.2 we describe our implementation of the method. We also point out its differences to the theoretical conceptual design as well as some of its numerical characteristics.

We now discuss the overall workflow of the shrink separation as depicted in Figure 2.1. Suppose we are given a fractional LP solution $\boldsymbol{z} \in \mathrm{MET}(G) \setminus \mathrm{CUT}(G)$. The vector $\boldsymbol{z}$

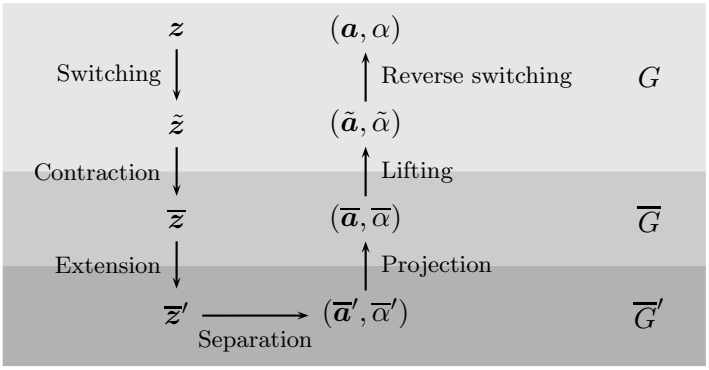


Figure 2.1: Workflow of the shrink separation.

decomposes into fractional components with an LP value in the range $(0, 1)$ as well as integral components whose LP value is either 0 or 1. The aim of the separation is to find an inequality $\boldsymbol{a}^T\boldsymbol{x} \leq \alpha$, which we abbreviate by $(\boldsymbol{a}, \alpha)$, that separates $\boldsymbol{z}$ from $\mathrm{CUT}(G)$.

In other words, we seek an inequality that is valid for the cut polytope but violated by the given LP solution at the same time.

In the first step we apply a transformation called **switching** to $\boldsymbol{z}$. As a result, all the components of the switched LP solution $\tilde{\boldsymbol{z}}$ have an LP value that is either fractional or zero. Note that the zero components of $\tilde{\boldsymbol{z}}$ correspond precisely to the integral components of $\boldsymbol{z}$.

Next, we **contract** all edges with a switched LP value of zero. This removes all the integral components from $\tilde{\boldsymbol{z}}$ and reduces it to a contracted LP solution $\overline{\boldsymbol{z}}$ with exclusively fractional components. The associated contracted graph $\overline{G}$ has at most the number of nodes and edges of the original graph G and at least its density. However, there is no guarantee that $\overline{G}$ is complete, i. e., there could still be edges missing.

To allow the use of separation techniques for dense and complete graphs, we **extend** the contracted LP solution by introducing artificial LP values for the non-edges of $\overline{G}$. These LP values are chosen in such a way that the extended LP solution $\overline{\boldsymbol{z}}'$ is an element of $\mathrm{MET}(\overline{G}')$. In other words, $\overline{\boldsymbol{z}}'$ does not violate any odd-cycle inequalities in the associated extended graph $\overline{G}'$.

The separation of the extended LP solution $\overline{\boldsymbol{z}}'$ yields an inequality $(\overline{\boldsymbol{a}}', \overline{\alpha}')$ that is valid for $\mathrm{CUT}(\overline{G}')$ but violated by $\overline{\boldsymbol{z}}'$. To make this inequality compatible with the original LP solution $\boldsymbol{z}$, we have to apply three transformations that compensate for the switching, the contraction, and the extension of $\boldsymbol{z}$.

First, we **project out** all nonzero left hand side coefficients of $(\overline{\boldsymbol{a}}', \overline{\alpha}')$ that are related to the non-edges of $\overline{G}$. To do so, we add appropriate multiples of valid inequalities to $(\overline{\boldsymbol{a}}', \overline{\alpha}')$. In the projected inequality, all the left hand side coefficients of the non-edges of $\overline{G}$ are equal to zero and can be truncated. We obtain a condensed inequality $(\overline{\boldsymbol{a}}, \overline{\alpha})$ that separates the contracted LP solution $\overline{\boldsymbol{z}}$ from $\mathrm{CUT}(\overline{G})$.

Next, we **lift** the condensed inequality. This is because the contracted LP solution $\overline{\boldsymbol{z}}$ may have less components than the switched LP solution $\tilde{\boldsymbol{z}}$. Thus, we have to adapt the dimension of the left hand side vector $\overline{\boldsymbol{a}}$ to the dimension of $\tilde{\boldsymbol{z}}$. The resulting lifted inequality $(\tilde{\boldsymbol{a}}, \tilde{\alpha})$ is valid for $\mathrm{CUT}(G)$ but violated by the switched LP solution $\tilde{\boldsymbol{z}}$. Note that $\tilde{\boldsymbol{z}}$ has the same dimension as the original LP solution.

Finally, we **switch** the lifted inequality. This adjusts the coefficients of the inequality subject to the initial switching of the LP solution $\boldsymbol{z}$. Note that we also refer to the switching of the inequality as **reverse switching** to distinguish it from the switching operation on $\boldsymbol{z}$ as well as to stress its compensating nature. Ultimately, we obtain an inequality $(\boldsymbol{a}, \alpha)$ that separates the original LP solution $\boldsymbol{z}$ from the cut polytope thus solving the initial separation problem.

In the next section we describe each of the components of the shrink separation in greater detail and provide the respective underlying theory.

2.1 Components of the Separation Procedure

As depicted in Figure 2.1, the shrink separation comprises seven major steps: switching, contraction, extension, separation, projection, lifting, and reverse switching. The first three steps modify a given LP solution, while the last three steps adjust separating inequalities subject to these previous modifications. The fourth step links the two branches by generating separating inequalities for the modified LP solution. We see that each transformation of the LP solution has its corresponding transformation of the

separating inequalities. These pairs of transformations form the horizontal layers of the diagram in Figure 2.1.

We now elaborate on the single steps of the shrink separation. For a better understanding, we group each of the LP transformations with its corresponding inequality transformation, thereby covering the horizontal layers in Figure 2.1 top down.

2.1.1 Switching and Reverse Switching

The **switching operation** is a concept related to families of subsets of a given finite set. Of particular interest are subset families that are closed under taking the symmetric difference, which we will introduce shortly. In this case, there are interesting implications regarding the structure of the convex hull of the incidence vectors of the subsets.

The switching operation has been formally introduced and studied by Deza and Laurent [DL97]. In specific scopes, however, it has been discovered independently by several other authors, for instance, by Barahona and Mahjoub [BM86] in the context of the cut polytope on an arbitrary graph.

For the max-cut problem on a graph $G = (V, E)$ we choose the edge set E as the finite set and the set $\{\delta(U) \mid U \subseteq V\}$ of cuts of G as the family of subsets of E. We define the **symmetric difference** $X \,\Delta\, Y$ of two sets X and Y by

$$X \,\Delta\, Y := (X \cup Y) \setminus (X \cap Y) = (X \setminus Y) \cup (Y \setminus X).$$

The set of cuts of G is closed under taking the symmetric difference. This property is stated more precisely in the following lemma.

Lemma 2.1. *Let $S, T \subseteq V$. Then, $\delta(S) \,\Delta\, \delta(T) = \delta(S \,\Delta\, T)$.*

Proof. First, we write the cut $\delta(S)$ in its equivalent form $(S : S^{\complement})$ and proceed analogously for the cut $\delta(T)$. Also, we note that $S = (S \setminus T) \uplus (S \cap T)$ as well as $T = (T \setminus S) \uplus (S \cap T)$. Thus, we can write $\delta(S) \cup \delta(T)$ as follows:

$$(S \setminus T : S^{\complement}) \cup (S \cap T : S^{\complement}) \cup (S \cap T : T^{\complement}) \cup (T \setminus S : T^{\complement}).$$

Next, we partition $S^{\complement}$ into $T \setminus S$ and $(S \cup T)^{\complement}$ and proceed for $T^{\complement}$ in the same manner. This results in the following representation of $\delta(S) \cup \delta(T)$:

$$\begin{aligned} &(S \setminus T : T \setminus S) \;\cup \big(S \setminus T : (S \cup T)^{\complement}\big) \\ \cup\, &(S \cap T : T \setminus S) \cup \big(S \cap T : (S \cup T)^{\complement}\big) \\ \cup\, &(S \cap T : S \setminus T) \cup \big(T \setminus S : (S \cup T)^{\complement}\big). \end{aligned} \tag{2.1}$$

Similarly, we can rewrite $\delta(S) \cap \delta(T)$ in the following form:

$$(S \setminus T : T \setminus S) \cup \big(S \cap T : (S \cup T)^{\complement}\big).$$

It comprises the first and the fourth clause of (2.1). As a result, we see that the symmetric difference of the cuts $\delta(S)$ and $\delta(T)$ equals

$$\big(S \setminus T : (S \cup T)^{\complement}\big) \cup (S \setminus T : S \cap T) \cup (T \setminus S : S \cap T) \cup \big(T \setminus S : (S \cup T)^{\complement}\big),$$

which is exactly $\delta(S \,\Delta\, T)$. □

For a vector $\boldsymbol{z} \in \mathbb{R}^E$ and a subset $B \subseteq E$ we define the vector $\boldsymbol{z}^B \in \mathbb{R}^E$ by

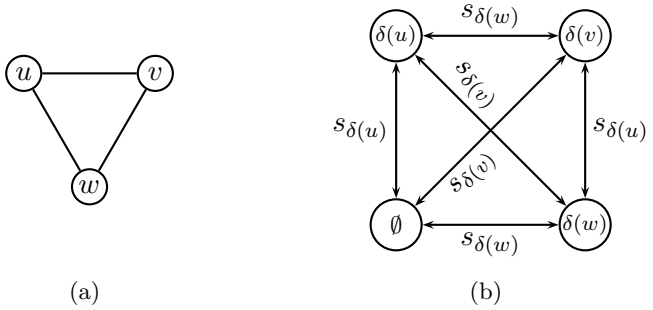
$$z_e^B := \begin{cases} -z_e & \text{if } e \in B, \\ z_e & \text{otherwise.} \end{cases}$$

Consider the mapping $s_B \colon \mathbb{R}^E \to \mathbb{R}^E$ defined by $s_B(\boldsymbol{z}) := \boldsymbol{z}^B + \boldsymbol{\chi}^B$, where $\boldsymbol{\chi}^B$ is the incidence vector of the subset B. In other words,

$$\big(s_B(\boldsymbol{z})\big)_e := \begin{cases} 1 - z_e & \text{if } e \in B, \\ z_e & \text{otherwise.} \end{cases} \tag{2.2}$$

The mapping s_B is called **switching mapping alongside** B. It is an **involution**, which means that $s_B\big(s_B(\boldsymbol{z})\big) = \boldsymbol{z}$. Furthermore, we have $s_B(\boldsymbol{\chi}^A) = \boldsymbol{\chi}^{A \,\Delta\, B}$ for two arbitrary subsets $A, B \subseteq E$. In conjunction with Lemma 2.1, this means that we can use the switching mapping to transform the incidence vectors of any two cuts into one another. Consider two arbitrary cuts $\delta(S)$ and $\delta(T)$. Then, switching the incidence vector of $\delta(S)$ alongside the cut $\delta(S \,\Delta\, T)$ yields the incidence vector of the cut $\delta\big(S \,\Delta (S \,\Delta\, T)\big)$, which is exactly $\delta(T)$. This important conclusion is illustrated in Example 2.2.

Example 2.2. *Consider the complete graph K_3 as depicted in Figure 2.2(a). The diagram in Figure 2.2(b) shows how to transform the vertices of* $\mathrm{CUT}(K_3)$ *into one another using the switching mapping.*



(a) (b)

Figure 2.2: Switching operations on the vertices of $\mathrm{CUT}(K_3)$: (a) shows the complete graph K_3; (b) shows how to transform the vertices of $\mathrm{CUT}(K_3)$ into one another using the switching mapping.

Similar to the switching mapping on vectors we can define a switching operation on inequalities. Consider an inequality $\boldsymbol{a}^T\boldsymbol{x} \leq \alpha$, with $\boldsymbol{a} \in \mathbb{R}^E$, $\alpha \in \mathbb{R}$, and a cut $\delta(U)$. We say that the inequality

$$(\boldsymbol{a}^{\delta(U)})^T\boldsymbol{x} \leq \alpha - \boldsymbol{a}\big(\delta(U)\big)$$

is obtained by **switching the inequality $\boldsymbol{a}^T\boldsymbol{x} \leq \alpha$ alongside** $\delta(U)$. Here, we used the abbreviation $\boldsymbol{a}\big(\delta(U)\big)$ for the sum $\sum_{e \in \delta(U)} a_e$. The switching operation on inequalities preserves the validity as well as a possible facet-defining property of an inequality for $\mathrm{CUT}(G)$. This fundamental result is rendered more precisely in the following theorem.

Theorem 2.3 (cf. Corollary 2.9 in [BM86])**.** *Given $\boldsymbol{a} \in \mathbb{R}^E$, $\alpha \in \mathbb{R}$, and a cut $\delta(U)$, the following assertions are equivalent.*

(i) The inequality $\boldsymbol{a}^T\boldsymbol{x} \leq \alpha$ *is valid or facet defining for* $\mathrm{CUT}(G)$*, respectively.*

(ii) The inequality $(\boldsymbol{a}^{\delta(U)})^T\boldsymbol{x} \leq \alpha - \boldsymbol{a}\big(\delta(U)\big)$ *is valid or facet defining for* $\mathrm{CUT}(G)$*, respectively.*

Moreover, if the switched inequality $(\boldsymbol{a}^{\delta(U)})^T\boldsymbol{x} \leq \alpha - \boldsymbol{a}\big(\delta(U)\big)$ is tight at a point $\boldsymbol{z}$ then the original inequality $\boldsymbol{a}^T\boldsymbol{x} \leq \alpha$ is tight at the switched point $s_{\delta(U)}(\boldsymbol{z})$.

Corollary 2.4. *Given* $\boldsymbol{a} \in \mathbb{R}^E$*,* $\alpha \in \mathbb{R}$*, and a cut* $\delta(U)$*, the inequalities* $\boldsymbol{a}^T s_{\delta(U)}(\boldsymbol{x}) \leq \alpha$ *and* $(\boldsymbol{a}^{\delta(U)})^T\boldsymbol{x} \leq \alpha - \boldsymbol{a}\big(\delta(U)\big)$ *are equivalent.*

Proof. By definition, we have $s_{\delta(U)}(\boldsymbol{x}) = \boldsymbol{x}^{\delta(U)} + \boldsymbol{\chi}^{\delta(U)}$. This allows us to rearrange the inequality $\boldsymbol{a}^T s_{\delta(U)}(\boldsymbol{x}) \leq \alpha$ in the following form:

$$\boldsymbol{a}^T\boldsymbol{x}^{\delta(U)} \leq \alpha - \boldsymbol{a}\big(\delta(U)\big).$$

Now, the assertion results from the fact that $\boldsymbol{a}^T\boldsymbol{x}^{\delta(U)}$ equals $(\boldsymbol{a}^{\delta(U)})^T\boldsymbol{x}$. □

With the help of the above corollary we can characterize an interesting symmetry of the cut polytope. First, we recall that the vertices of the cut polytope $\mathrm{CUT}(G)$ are precisely the incidence vectors of the cuts of G. We have already seen that we can transform the incidence vectors of two cuts $\delta(S)$ and $\delta(T)$ into one another using the switching mapping alongside their symmetric difference $\delta(S \,\Delta\, T)$. Now assume we are given a facet defining inequality $\boldsymbol{a}^T\boldsymbol{x} \leq \alpha$ that is tight at the vertex $\boldsymbol{\chi}^{\delta(S)}$. Due to Corollary 2.4, the inequality obtained by switching $\boldsymbol{a}^T\boldsymbol{x} \leq \alpha$ alongside the cut $\delta(S \,\Delta\, T)$ defines a facet of $\mathrm{CUT}(G)$ which is tight at the switched vertex $\boldsymbol{\chi}^{\delta(T)}$. Hence, the switching of inequalities alongside the cut $\delta(S \,\Delta\, T)$ maps the facets containing $\boldsymbol{\chi}^{\delta(S)}$ onto those containing $\boldsymbol{\chi}^{\delta(T)}$, and vice versa, in a one-to-one manner. Descriptively speaking, the local facet structure of the cut polytope looks identical at each vertex.

As a consequence, if a given inequality separates the switched point $s_{\delta(U)}(\boldsymbol{z})$ from the cut polytope then the appropriately switched inequality does the same for the point $\boldsymbol{z}$. This allows us to solve the separation problem for the switched point rather than for the original one. Any separating inequality for the switched point can then simply be switched alongside $\delta(U)$ to obtain a solution to the initial separation problem. We will also refer to this final switching of the inequality as **reverse switching**. We do so to distinguish it from the switching operation applied to $\boldsymbol{z}$ as well as to stress its compensating nature.

Suppose we are given a fractional LP solution $\boldsymbol{z} \in \mathrm{MET}(G) \setminus \mathrm{CUT}(G)$. Our goal is to find an inequality that separates $\boldsymbol{z}$ from the cut polytope. With respect to the vector $\boldsymbol{z}$, the edge set E decomposes into the following subsets:

$$\begin{aligned} E_0(\boldsymbol{z}) &:= \{e \in E \mid z_e = 0\}, \\ E_1(\boldsymbol{z}) &:= \{e \in E \mid z_e = 1\}, \\ E_f(\boldsymbol{z}) &:= \{e \in E \mid 0 < z_e < 1\}. \end{aligned}$$

The edges in $E_0(\boldsymbol{z})$ with an LP value of zero can easily be contracted, as we will explain in Section 2.1.2. The contraction of edges reduces the size of the graph and thus accelerates the separation process. Using the switching mapping we can even contract all the edges in $E_0(\boldsymbol{z})$ and $E_1(\boldsymbol{z})$ combined. To do so, we switch the LP solution $\boldsymbol{z}$ alongside a particular cut $\mathcal{C}$ of G that satisfies the following conditions:

$$E_1(\boldsymbol{z}) \subseteq \mathcal{C}, \quad E_0(\boldsymbol{z}) \cap \mathcal{C} = \emptyset. \tag{2.3}$$

According to (2.2), all components of the resulting **switched LP solution** $\tilde{\boldsymbol{z}} := s_{\mathcal{C}}(\boldsymbol{z})$ are either fractional or zero, i. e.,

$$0 \leq \tilde{z}_e < 1 \quad \text{for all } e \in E.$$

Moreover, the edge sets $E_0(\tilde{\boldsymbol{z}})$ and $E_0(\boldsymbol{z}) \cup E_1(\boldsymbol{z})$ are identical. In other words, the contraction of the edges with a switched LP value of zero is equivalent to the contraction of the edges with an integral original LP value.

The existence of a cut $\mathcal{C}$ with the desired properties (2.3) may not be instantly apparent. It is, however, directly implied by the assumption that the LP solution $\boldsymbol{z}$ is an element of the semimetric polytope.

Lemma 2.5. *Let $\boldsymbol{z} \in \mathrm{MET}(G)$. Then, there exists a cut $\mathcal{C}$ of G such that $E_1(\boldsymbol{z}) \subseteq \mathcal{C}$ and $E_0(\boldsymbol{z}) \cap \mathcal{C} = \emptyset$.*

Proof. Let $E' := E \setminus E_f(\boldsymbol{z})$ be the set of integral edges with respect to $\boldsymbol{z}$. The corresponding graph $G' := (V, E')$ features all the cycles of G except for those containing at least one fractional edge. Since the vector $\boldsymbol{z}$ is an element of the semimetric polytope $\mathrm{MET}(G)$, it satisfies in particular all odd-cycle inequalities on the cycles of G'. Thus, the binary vector $\boldsymbol{z}' := \boldsymbol{z}_{|E'}$, which is the vector $\boldsymbol{z}$ restricted to the index set E', is an element of $\mathrm{MET}(G')$. As pointed out in Section 1.5, the binary vectors inside $\mathrm{MET}(G')$ are precisely the vertices of $\mathrm{CUT}(G')$. Hence, $\boldsymbol{z}'$ is the incidence vector of a cut $\mathcal{C}' := E_1(\boldsymbol{z})$ of G'. This cut already has the desired properties:

$$E_1(\boldsymbol{z}) \subseteq \mathcal{C}', \quad E_0(\boldsymbol{z}) \cap \mathcal{C}' = \emptyset.$$

Finally, we extend $\mathcal{C}'$ with those edges of $E_f(\boldsymbol{z})$ whose ends are in different shores of $\mathcal{C}'$. This results in the desired cut $\mathcal{C}$ of G that satisfies (2.3). □

In addition, if the original LP solution $\boldsymbol{z}$ is an element of the semimetric polytope then this is also true for the switched LP solution $\tilde{\boldsymbol{z}}$. To facilitate the proof of this assertion, we first show the following lemma.

Lemma 2.6. *Any odd-cycle inequality of $\mathrm{CUT}(G)$ switched alongside a cut of G is again an odd-cycle inequality of $\mathrm{CUT}(G)$ with respect to the same cycle.*

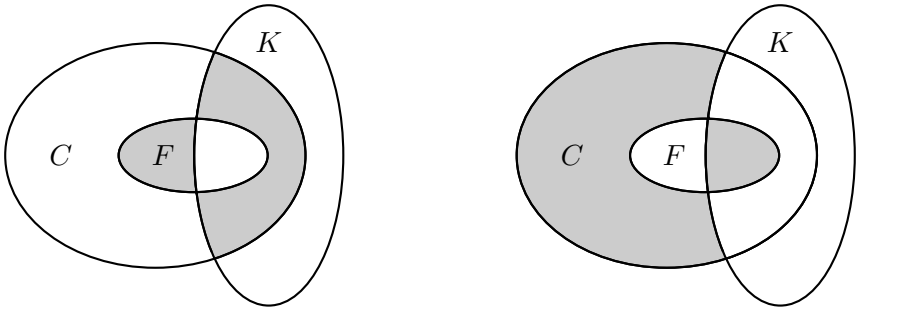
Proof. Let $\boldsymbol{x}(F) - \boldsymbol{x}(C \setminus F) \leq |F| - 1$ be an arbitrary odd-cycle inequality of $\mathrm{CUT}(G)$ and let K be an arbitrary cut of G. We partition each of the sets F and $C \setminus F$ into its intersection with K and the complement of K, respectively. As a result, we obtain the following representation of the odd-cycle inequality:

$$\boldsymbol{x}(F \setminus K) + \boldsymbol{x}(F \cap K) - \boldsymbol{x}\big((C \setminus F) \setminus K\big) - \boldsymbol{x}\big((C \setminus F) \cap K\big) \leq |F| - 1.$$

We now switch the inequality alongside the cut K. This reverses the signs of both $\boldsymbol{x}(F \cap K)$ and $\boldsymbol{x}\big((C \setminus F) \cap K\big)$. It also reduces the value of the right hand side by $|F \cap K| - |(C \setminus F) \cap K|$. Finally, to simplify the notation, we introduce F' to denote the set $(F \setminus K) \cup \big((C \setminus F) \cap K\big)$, which is highlighted in gray in Figure 2.3(a). Altogether, we obtain the following switched inequality:

$$\boldsymbol{x}(F') - \boldsymbol{x}\big((F \cap K) \cup ((C \setminus F) \setminus K)\big) \leq \big|F'\big| - 1.$$

We then use basic set calculus to show that the set $(F \cap K) \cup \big((C \setminus F) \setminus K\big)$, which is highlighted in gray in Figure 2.3(b), is actually identical to the set $C \setminus F'$. As a result,



(a) $F' := (F \setminus K) \cup \big((C \setminus F) \cap K\big)$ (b) $C \setminus F' = (F \cap K) \cup \big((C \setminus F) \setminus K\big)$

Figure 2.3: Relevant edge sets of an odd-cycle inequality switched alongside a cut K.

the switched odd-cycle inequality simplifies to $\boldsymbol{x}(F') - \boldsymbol{x}(C \setminus F') \leq |F'| - 1$, which already looks like an odd-cycle inequality. Still, we need to verify that the set F' has indeed odd cardinality, i. e., $|F'| \equiv 1 \pmod 2$.

First, we note that $(C \setminus F) \cap K$ equals $(C \cap K) \setminus F$ and thus we have

$$|F'| = |F \setminus K| + |(C \cap K) \setminus F| = |F \setminus K| + |C \cap K| - |F \cap K|.$$

In addition, we know that $|C \cap K| \equiv 0 \pmod 2$. This is due to the well-known property that a cut and a cycle can only have an even number of edges in common. Altogether, we obtain

$$|F'| \equiv |F \setminus K| - |F \cap K| \pmod 2.$$

Finally, we add the equation $|F| = |F \setminus K| + |F \cap K|$ to get

$$|F'| + |F| \equiv 2 \cdot |F \setminus K| \equiv 0 \pmod 2.$$

The odd cardinality of F implies that $|F'|$ is also odd. Otherwise, the sum $|F'| + |F|$ would be odd in contradiction to the above result. This proves the assertion. □

Using Lemma 2.6, we can now easily prove that a point (vertex) of $\mathrm{MET}(G)$ switched alongside a cut of G is again a point (vertex) of $\mathrm{MET}(G)$. This property will be a fundamental prerequisite for the subsequent steps of the shrink separation.

Theorem 2.7. *Let K be a cut of G. Then, $\boldsymbol{z} \in \mathbb{R}^E$ is a point (vertex) of $\mathrm{MET}(G)$ if and only if this is true for $s_K(\boldsymbol{z})$.*

Proof. Let $\boldsymbol{z} \in \mathrm{MET}(G)$. Consider an arbitrary cycle C of G and an arbitrary subset F of C with odd cardinality. We have to show that the switched point $\tilde{\boldsymbol{z}} := s_K(\boldsymbol{z})$ satisfies the odd-cycle inequality defined by C and F, i. e.,

$$\tilde{\boldsymbol{z}}(F) - \tilde{\boldsymbol{z}}(C \setminus F) \leq |F| - 1. \tag{2.4}$$

As stated in Corollary 2.4, switching (2.4) alongside the cut K results in an equivalent representation with respect to $\boldsymbol{z}$ instead of $\tilde{\boldsymbol{z}}$. Moreover, due to Lemma 2.6 we know that any switched odd-cycle inequality is again an odd-cycle inequality with respect to the same cycle. Hence, the switched representation of (2.4) has the form

$$\boldsymbol{z}(F') - \boldsymbol{z}(C \setminus F') \leq |F'| - 1,$$

where F' is a subset of C with odd cardinality. By assumption, however, the point $\boldsymbol{z}$ is part of the semimetric polytope and therefore clearly satisfies the above inequality. This in turn shows that the switched point indeed satisfies (2.4), which proves that $\tilde{\boldsymbol{z}}$ is a point of $\mathrm{MET}(G)$. The reverse direction works analogously by replacing $\boldsymbol{z}$ with $\tilde{\boldsymbol{z}}$ and using the fact that the switching mapping is an involution, i. e., $s_K(\tilde{\boldsymbol{z}}) = \boldsymbol{z}$.

We now prove the second assertion, namely that the switching mapping preserves the vertices of $\mathrm{MET}(G)$. First, we note that a point $\boldsymbol{z} \in \mathrm{MET}(G)$ is a vertex of the polytope if and only if there are no two distinct points $\boldsymbol{a}, \boldsymbol{b} \in \mathrm{MET}(G)$ such that $\boldsymbol{z} = \lambda\boldsymbol{a} + (1-\lambda)\boldsymbol{b}$ for a suited scalar λ in the open range $(0, 1)$. In other words, the vertices of $\mathrm{MET}(G)$ are exactly those points of the polytope that cannot be obtained as a proper convex combination of two distinct points of $\mathrm{MET}(G)$ (see, for instance, §5 and §7 of [Brø83] for further details). Now assume that $\boldsymbol{z}$ is a point but not a vertex of $\mathrm{MET}(G)$. This means that $\boldsymbol{z}$ is the proper convex combination of two distinct points $\boldsymbol{a}, \boldsymbol{b} \in \mathrm{MET}(G)$. Switching $\boldsymbol{z}$ alongside K yields the following switched point:

$$\begin{aligned} \tilde{\boldsymbol{z}} &= \big(\lambda\boldsymbol{a} + (1-\lambda)\boldsymbol{b}\big)^K + \boldsymbol{\chi}^K \\ &= \lambda(\boldsymbol{a}^K + \boldsymbol{\chi}^K) + (1-\lambda)(\boldsymbol{b}^K + \boldsymbol{\chi}^K) \\ &= \lambda s_K(\boldsymbol{a}) + (1-\lambda) s_K(\boldsymbol{b}). \end{aligned}$$

From the first part of the proof we already know that the switched points $s_K(\boldsymbol{a})$ and $s_K(\boldsymbol{b})$ are elements of $\mathrm{MET}(G)$ since this is true for $\boldsymbol{a}$ and $\boldsymbol{b}$. Hence, $\tilde{\boldsymbol{z}}$ is the proper convex combination of two distinct points of $\mathrm{MET}(G)$ and therefore cannot be a vertex.

This proves, by contrapositive, that $\boldsymbol{z}$ is a vertex of $\mathrm{MET}(G)$ provided that $\tilde{\boldsymbol{z}}$ satisfies this property. As before, we obtain the proof of the reverse direction by replacing $\boldsymbol{z}$ with $\tilde{\boldsymbol{z}}$ and using the fact that $s_K(\tilde{\boldsymbol{z}}) = \boldsymbol{z}$. □

We conclude this section with a summary of the key results so far. The switching operation allows to solve the separation problem for a switched LP solution $\tilde{\boldsymbol{z}}$ instead of the original fractional LP solution $\boldsymbol{z} \in \mathrm{MET}(G) \setminus \mathrm{CUT}(G)$. Any separating inequality for $\tilde{\boldsymbol{z}}$ can easily be transformed into a separating inequality for $\boldsymbol{z}$. This transformation preserves the validity as well as a possible facet-defining property of a given inequality for $\mathrm{CUT}(G)$.

Moreover, by switching the LP solution $\boldsymbol{z}$ alongside a particular cut, we can force the zero components of the switched LP solution $\tilde{\boldsymbol{z}}$ to correspond precisely to the integral components of $\boldsymbol{z}$. As a consequence, the removal of these zero components during the contraction reduces $\tilde{\boldsymbol{z}}$ to its fractional part and thus lowers its dimension by the number of integral components of $\boldsymbol{z}$. In comparison, if we would contract the original LP solution $\boldsymbol{z}$ instead, we could only lower its dimension by the number of zero components. Thus, it is preferable to contract the switched LP solution $\tilde{\boldsymbol{z}}$ since the additional reduction of the dimension will benefit the subsequent separation process.

2.1.2 Contraction and Lifting

The aim of the **contraction** is to lower the dimension of the LP solution $\tilde{\boldsymbol{z}}$ prior to its separation. To do so, we remove all the components of $\tilde{\boldsymbol{z}}$ with a value of zero. This is because an LP value of zero indicates that the respective edge is not part of the optimum solution. In the associated graph, the removal of the zero components of $\tilde{\boldsymbol{z}}$ corresponds to the contraction of the respective edges.

Due to the switching step described in Section 2.1.1, we can assume the components of $\tilde{\boldsymbol{z}}$ to be either fractional or zero. In other words, we have $0 \leq \tilde{z}_e < 1$ for all edges e. Hence, the removal of the zero components effectively reduces the LP solution to its fractional components. Also, by Theorem 2.7 on page 45, we can assume $\tilde{\boldsymbol{z}}$ to be an element of $\mathrm{MET}(G) \setminus \mathrm{CUT}(G)$ since we imposed this condition on the original LP solution $\boldsymbol{z}$.

Consider the set $E_0(\tilde{\boldsymbol{z}})$ of edges with a $\tilde{\boldsymbol{z}}$ value of zero. We define the auxiliary graph $G_0 := \big(V, E_0(\tilde{\boldsymbol{z}})\big)$ as a copy of the graph G with its edge set restricted to $E_0(\tilde{\boldsymbol{z}})$. Furthermore, let $\{W_i\}_{i \in I}$ be the family of node sets of the connected components of G_0. We recall that $(W_i : W_j)$ denotes the set of edges with one end in W_i and the other end in W_j. Since we assume the LP solution to be an element of the semimetric polytope, all the edges in $(W_i : W_j)$ must have the same LP value, as shown in the following lemma.

Lemma 2.8. *Let $\tilde{\boldsymbol{z}} \in \mathrm{MET}(G)$ and let W_i, W_j be the node sets of two connected components of the graph G_0. If f and g are two distinct edges in $(W_i : W_j)$ then the values $\tilde{z}_f$ and $\tilde{z}_g$ are identical.*

Proof. Suppose the assertion is wrong and that, without loss of generality, $\tilde{z}_g$ is less than $\tilde{z}_f$. Since W_i and W_j induce connected components of G_0, there exist paths that link the ends of f and g in W_i and W_j, respectively. Moreover, all the edges in these linking paths have a $\tilde{\boldsymbol{z}}$ value of zero. In conjunction with the edges f and g, the paths form a cycle C as depicted in Figure 2.4. Note that one of the paths, which are represented by dashed lines, may be empty.

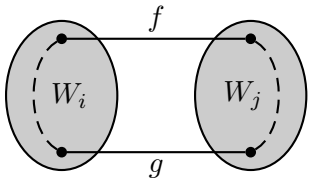


Figure 2.4: Two distinct edges in the set $(W_i : W_j)$.

We now define $F := \{f\}$ and obtain

$$\tilde{\boldsymbol{z}}(F) - \tilde{\boldsymbol{z}}(C \setminus F) = \tilde{z}_f - \tilde{z}_g > 0 = |F| - 1.$$

Thus, $\tilde{\boldsymbol{z}}$ violates the odd-cycle inequality defined by F and C. This, however, contradicts the assumption that $\tilde{\boldsymbol{z}}$ is an element of the semimetric polytope and thus proves the assertion by contrapositive. □

As already mentioned, the removal of the zero components of $\tilde{\boldsymbol{z}}$ corresponds to the contraction of the respective edges in the associated graph G. These edge-contractions are carried out as follows:

1. contract each node set W_i to a supernode w_i,
2. delete all loops,
3. replace parallel edges by a single edge with the same $\tilde{\boldsymbol{z}}$ value.

Note that the third step is well defined due to Lemma 2.8. Ultimately, we obtain a **contracted graph** $\overline{G} = (\overline{V}, \overline{E})$ as well as an associated **contracted LP solution** $\overline{\boldsymbol{z}}$ in $\mathbb{R}^{\overline{E}}$ with the following properties:

(1) $\overline{\boldsymbol{z}}$ has only fractional components,

(2) $\overline{\boldsymbol{z}}$ is a point (vertex) of $\mathrm{MET}(\overline{G})$ if and only if $\tilde{\boldsymbol{z}}$ is a point (vertex) of $\mathrm{MET}(G)$,

(3) every cut of $\overline{G}$ corresponds to one of the cuts of G that are disjoint from $E_0(\tilde{\boldsymbol{z}})$,

(4) $\overline{G}$ has at most as many nodes and edges as G.

The properties (1) and (4) are immediate consequences of the contraction. A proof for (2) was given by Laurent and Poljak (cf. Proposition 2.4 of [LP95]). In particular, this means that the contracted LP solution $\overline{\boldsymbol{z}}$ is an element of $\mathrm{MET}(\overline{G})$ since we assumed the switched LP solution $\tilde{\boldsymbol{z}}$ to be an element of $\mathrm{MET}(G)$. Finally, property (3) follows from the fact that in a given cut of $\overline{G}$ every supernode is the element of exactly one of the two shores. In addition, the ends of an arbitrary edge in $E_0(\tilde{\boldsymbol{z}})$ are both elements of the same set W_i. Thus, they are both assigned to the same shore as their corresponding supernode w_i.

As reverse transformation to the edge-contraction we introduce the so-called **node-splitting**. This operation splits up the supernodes w_i and thereby decompresses the previously contracted edges. Thus, we can use node-splitting to lift valid inequalities for $\mathrm{CUT}(\overline{G})$ to valid inequalities for $\mathrm{CUT}(G)$. For a better understanding we only consider the case of splitting a single supernode into two (super-)nodes. The results below can then be generalized by means of iteration.

Suppose we want to split a supernode w into the nodes h and t. For any partition (H, T, B) of the set of neighbors of w, we define the decompressed graph $\hat{G} = (\hat{V}, \hat{E})$, with

$$\begin{aligned} \hat{V} &:= (\overline{V} \setminus w) \cup \{h, t\}, \\ \hat{E} &:= \big(\overline{E} \setminus \delta(w)\big) \cup (h : H \cup B) \cup (t : T \cup B) \cup ht. \end{aligned}$$

We say that the graph $\hat{G}$ is obtained from the graph $\overline{G}$ by **splitting w into the nodes h and t with respect to the partition** (H, T, B). On the decompressed graph, the sets H and T correspond to the exclusive neighbors of h and t, respectively, while the set B comprises the common neighbors of both nodes. An exemplary neighborhood of a decompressed edge ht is depicted in Figure 2.5.

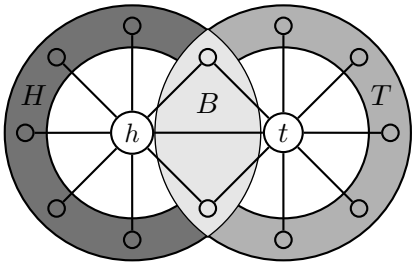


Figure 2.5: Partition of the neighborhood of a decompressed edge ht.

Consider a valid inequality $(\overline{\boldsymbol{a}}, \alpha)$, $\overline{\boldsymbol{a}} \in \mathbb{R}^{\overline{E}}$, for the cut polytope on the contracted graph $\overline{G}$. Without loss of generality, we assume that $\sum_{v \in T} |\overline{a}_{wv}| \leq \sum_{v \in H} |\overline{a}_{wv}|$. We define the inequality $(\hat{\boldsymbol{a}}, \alpha)$, $\hat{\boldsymbol{a}} \in \mathbb{R}^{\hat{E}}$, for the cut polytope on the decompressed graph $\hat{G}$

as follows:

$$\begin{array}{rcll} \hat{a}_{ht} & = & -\sum_{v\in T} |\overline{a}_{wv}|, & \\ \hat{a}_{tv} & = & 0 & \text{for all } v \in B, \\ \hat{a}_{tv} & = & \overline{a}_{wv} & \text{for all } v \in T, \\ \hat{a}_{hv} & = & \overline{a}_{wv} & \text{for all } v \in H \cup B, \\ \hat{a}_{uv} & = & \overline{a}_{uv} & \text{for all } uv \in \hat{E} \setminus \big(\delta(h) \cup \delta(t)\big). \end{array} \tag{2.5}$$

We say that the inequality $(\hat{\boldsymbol{a}}, \alpha)$ is obtained from the inequality $(\overline{\boldsymbol{a}}, \alpha)$ by **splitting** w **into the nodes** h **and** t **with respect to the partition** (H, T, B).

Descriptively speaking, splitting w into h and t replaces the end w of the edges in $(w : H)$ and $(w : T)$ with h and t, respectively. At the same time, it preserves the coefficient values of the affected edges with respect to the inequality $(\overline{\boldsymbol{a}}, \alpha)$. The edges in $(w : B)$, on the other hand, have to be duplicated to join the nodes in B to both h and t. Here, only the duplicates in $(h : B)$ inherit the coefficient values of their counterparts in $(w : B)$, whereas the **mate edges** in $(t : B)$ are assigned a coefficient of zero.

An important property of the node-splitting operation is that is preserves the validity of inequalities for the cut polytope.

Lemma 2.9. *If* $(\overline{\boldsymbol{a}}, \alpha)$ *is a valid inequality for* $\mathrm{CUT}(\overline{G})$ *then the inequality* $(\hat{\boldsymbol{a}}, \alpha)$ *obtained by node-splitting is valid for* $\mathrm{CUT}(\hat{G})$.

Proof. Let $\delta(U)$, $U \subseteq \hat{V}$, be an arbitrary cut of $\hat{G}$. We have to show that $\hat{\boldsymbol{a}}\big(\delta(U)\big) \leq \alpha$. Clearly, if the edge ht is not an element of the cut $\delta(U)$ then, according to (2.5), the values $\hat{\boldsymbol{a}}\big(\delta(U)\big)$ and $\overline{\boldsymbol{a}}\big(\delta(U)\big)$ are identical. Thus, the assertion follows directly from the validity of $(\overline{\boldsymbol{a}}, \alpha)$.

So, assume the opposite and that the edge ht is in fact part of $\delta(U)$. Without loss of generality, let $h \in U$. We define the extended shore $U' := U \uplus t$. Since the corresponding cut $\delta(U')$ does not contain the edge ht, we have

$$\hat{\boldsymbol{a}}\big(\delta(U')\big) = \overline{\boldsymbol{a}}\big(\delta(U')\big) \leq \alpha. \tag{2.6}$$

The comparison of $\delta(U)$ and $\delta(U')$ shows that shifting the node t to the shore U removes the edges ht and $(t : T \cap U)$ from $\delta(U)$ while adding the edges $(t : T \setminus U)$. Note that the edges in $(t : B)$ are omitted since they have a coefficient of zero. Altogether, we obtain

$$\hat{\boldsymbol{a}}\big(\delta(U')\big) = \hat{\boldsymbol{a}}\big(\delta(U)\big) - \hat{a}_{ht} - \sum_{v\in T\cap U} \hat{a}_{tv} + \sum_{v\in T\setminus U} \hat{a}_{tv}.$$

The value of the last two sums can be estimated as follows:

$$-\sum_{v\in T\cap U} \hat{a}_{tv} + \sum_{v\in T\setminus U} \hat{a}_{tv} \geq -\sum_{v\in T} |\hat{a}_{tv}|.$$

In addition, (2.5) states that $\hat{a}_{tv} = \overline{a}_{wv}$, for all $v \in T$. This means that the above lower estimate actually equals $\hat{a}_{ht}$, which in turn shows that $\hat{\boldsymbol{a}}\big(\delta(U)\big) \leq \hat{\boldsymbol{a}}\big(\delta(U')\big)$. Now, the assertion follows from (2.6). □

Under the conditions specified in the lemma below, the node-splitting operation even preserves an existing facet-defining property of a valid inequality for the cut polytope. Note that this lemma reformulates Theorem 2.6(a) in [BM86] to adjust its assumptions to our definition of the node-splitting in (2.5).

Lemma 2.10. *Let $(\overline{\boldsymbol{a}}, \alpha)$ define a facet of $\mathrm{CUT}(\overline{G})$ and let $(\hat{\boldsymbol{a}}, \alpha)$ be the inequality obtained from $(\overline{\boldsymbol{a}}, \alpha)$ by splitting node w with respect to the partition (H, T, B). Then, $(\hat{\boldsymbol{a}}, \alpha)$ defines a facet of $\mathrm{CUT}(\hat{G})$ if there exists a node set $S \subseteq \overline{V}$ that contains w and satisfies the following conditions:*

(i) $\overline{\boldsymbol{a}}\big(\delta(S)\big) = \alpha$,

(ii) $\overline{a}_{wv} \geq 0$ *for all* $v \in T \cap S$,

(iii) $\overline{a}_{wv} \leq 0$ *for all* $v \in T \setminus S$,

(iv) $\overline{\boldsymbol{a}}(v : S) = \overline{\boldsymbol{a}}(v : \overline{V} \setminus S)$ *for all* $v \in B$.

Proof. Let the inequality $(\hat{\boldsymbol{b}}, \beta)$ define a facet of $\mathrm{CUT}(\hat{G})$ that contains the face defined by $(\hat{\boldsymbol{a}}, \alpha)$. This means in particular that the **tight cuts** of $(\hat{\boldsymbol{a}}, \alpha)$, which are the cuts at whose incidence vectors the inequality $(\hat{\boldsymbol{a}}, \alpha)$ is tight, are a subset of the tight cuts of $(\hat{\boldsymbol{b}}, \beta)$. If we can show the following proposition:

$$\exists\, \lambda > 0 \text{ such that } \hat{b}_e = \lambda \hat{a}_e, \text{ for all } e \in \hat{E}, \tag{2.7}$$

then the assertion follows from the inclusion-maximality of facets.

First, we note that, due to (2.5), any cut $\delta(U)$ not containing the edge ht satisfies the following equation:

$$\hat{\boldsymbol{a}}\big(\delta(U)\big) = \overline{\boldsymbol{a}}\big(\delta(U)\big).$$

As a consequence, each tight cut of $(\overline{\boldsymbol{a}}, \alpha)$ corresponds to a tight cut of $(\hat{\boldsymbol{a}}, \alpha)$ that does not contain the edge ht. Also, if the edge ht is not part of the cut $\delta(U)$ then we know for all $v \in B$ that $\delta(U)$ contains either both edges hv and tv or none of them. Altogether, we obtain a one-to-one correspondence between the edges in $\overline{E} \setminus (w : B)$ and those in $\hat{E} \setminus \big((\{h,t\} : B) \cup ht\big)$ as well as between the edges in $(w : B)$ and the edge pairs $\{hv, tv\}$ for all $v \in B$. Since, by assumption, $(\overline{\boldsymbol{a}}, \alpha)$ defines a facet, the above results imply the existence of a scalar $\lambda > 0$ such that:

$$\hat{b}_e = \lambda \hat{a}_e \quad \text{for all } e \in \hat{E} \setminus \big((\{h,t\} : B) \cup ht\big), \tag{2.8a}$$

$$\hat{b}_{hv} + \hat{b}_{tv} = \lambda \hat{a}_{hv} \quad \text{for all } v \in B. \tag{2.8b}$$

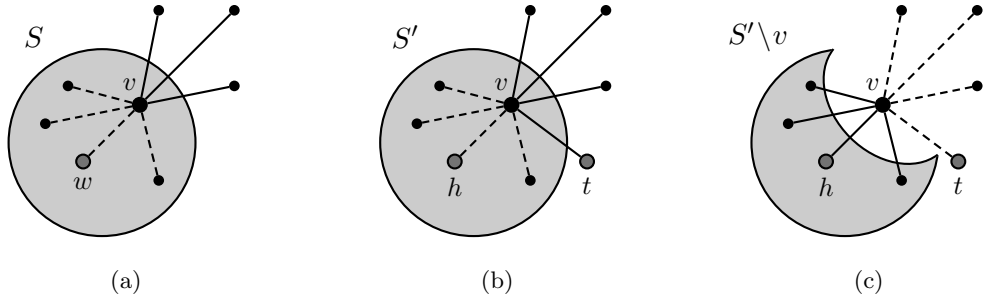
Consequently, the proposition (2.7) is correct for all edges in $\hat{E} \setminus \big((\{h,t\} : B) \cup ht\big)$.

In the next step we will prove the proposition for the edges in $(\{h,t\} : B)$. Note that, due to (2.8b), it is sufficient to show that $\hat{b}_{tv} = \lambda \hat{a}_{tv} = 0$ for all $v \in B$. To do so, we use an auxiliary cut that is induced by the set $S' := (S \setminus w) \cup h \subseteq \hat{V}$ as depicted in Figure 2.6(b). This cut $\delta(S')$ is tight for $(\hat{\boldsymbol{a}}, \alpha)$, as we will see shortly. A comparison of $\delta(S')$ with $\delta(S)$ shows that we gain the edge ht as well as the edges in $(w : T \cap S)$ while losing those in $(w : T \setminus S)$ (compare Figures 2.6(a) and (b)). As a consequence, we obtain that

$$\hat{\boldsymbol{a}}\big(\delta(S')\big) = \overline{\boldsymbol{a}}\big(\delta(S)\big) + \hat{a}_{ht} + \overline{\boldsymbol{a}}(w : T \cap S) - \overline{\boldsymbol{a}}(w : T \setminus S).$$

Yet, due to conditions (ii) and (iii), we have

$$\overline{\boldsymbol{a}}(w : T \cap S) - \overline{\boldsymbol{a}}(w : T \setminus S) = \sum_{v \in T} |\overline{a}_{wv}| \overset{(2.5)}{=} -\hat{a}_{ht}. \tag{2.9}$$



(a) (b) (c)

Figure 2.6: Contributions of a star $\delta(v)$ to the cuts induced by different node sets.

Thus, we see that

$$\hat{\boldsymbol{a}}\big(\delta(S')\big) = \overline{\boldsymbol{a}}\big(\delta(S)\big) \overset{\text{(i)}}{=} \alpha,$$

which verifies that the auxiliary cut $\delta(S')$ is tight for $(\hat{\boldsymbol{a}}, \alpha)$. With the help of $\delta(S')$ we can now show that $\hat{b}_{tv} = \lambda \hat{a}_{tv} = 0$ for all $v \in B$. In the following, we distinguish two cases, namely $v \in B \cap S$ and $v \in B \setminus S$, respectively.

For the first case '$v \in B \cap S$', we take the cut $\delta(S')$ and let the node v switch shores. We claim that the resulting cut $\delta(S' \setminus v)$ is also tight for $(\hat{\boldsymbol{a}}, \alpha)$. To see this, we recall (2.5), which states that $\hat{a}_{hv} = \overline{a}_{wv}$ as well as $\hat{a}_{tv} = 0$ for all $v \in B$. Therefore, condition (iv) implies that $\hat{\boldsymbol{a}}(v : S') = \hat{\boldsymbol{a}}(v : \hat{V} \setminus S')$ for all $v \in B$. In fact, since $\hat{a}_{tv} = 0$, we even have

$$\hat{\boldsymbol{a}}(v : S') = \hat{\boldsymbol{a}}\big(v : \hat{V} \setminus (S' \cup t)\big), \text{ for all } v \in B. \tag{2.10}$$

As a result, the values $\hat{\boldsymbol{a}}\big(\delta(S')\big)$ and $\hat{\boldsymbol{a}}\big(\delta(S' \setminus v)\big)$ are identical, which proves that $(\hat{\boldsymbol{a}}, \alpha)$ is indeed tight at $\delta(S' \setminus v)$.

Let $S'^{\complement}$ denote the complement of S' in $\hat{V}$. As we have seen, the two cuts $\delta(S')$ and $\delta(S' \setminus v)$ are tight for $(\hat{\boldsymbol{a}}, \alpha)$ and thus also for $(\hat{\boldsymbol{b}}, \beta)$. By removing v from S', we lose the edge tv as well as the edges in $\big(v : S'^{\complement} \setminus t\big)$ while gaining those in $\big(v : S'\big)$ (compare Figures 2.6(b) and (c)). Consequently, we have

$$0 = \hat{\boldsymbol{b}}\big(\delta(S')\big) - \hat{\boldsymbol{b}}\big(\delta(S' \setminus v)\big) = \hat{b}_{tv} + \hat{\boldsymbol{b}}\big(v : S'^{\complement} \setminus t\big) - \hat{\boldsymbol{b}}\big(v : S'\big).$$

We now show that the difference $\hat{\boldsymbol{b}}\big(v : S'^{\complement} \setminus t\big) - \hat{\boldsymbol{b}}\big(v : S'\big)$ actually equals $\hat{b}_{tv}$. Consider the equation (2.10). First, we write $\hat{\boldsymbol{a}}\big(v : S'\big)$ as the sum $\hat{\boldsymbol{a}}_{hv} + \hat{\boldsymbol{a}}\big(v : S' \setminus h\big)$. Next, we multiply the whole equation by the scalar λ from (2.8):

$$\lambda \hat{\boldsymbol{a}}_{hv} + \lambda \hat{\boldsymbol{a}}\big(v : S' \setminus h\big) = \lambda \hat{\boldsymbol{a}}\big(v : S'^{\complement} \setminus t\big).$$

Due to (2.8), the above equation is equivalent to the following one:

$$\hat{\boldsymbol{b}}_{hv} + \hat{\boldsymbol{b}}_{tv} + \hat{\boldsymbol{b}}\big(v : S' \setminus h\big) = \hat{\boldsymbol{b}}\big(v : S'^{\complement} \setminus t\big).$$

After rearranging the terms, we obtain the desired equality $\hat{\boldsymbol{b}}\big(v : S'^{\complement} \setminus t\big) - \hat{\boldsymbol{b}}\big(v : S'\big) = \hat{b}_{tv}$. As a result, we get

$$0 = \hat{\boldsymbol{b}}\big(\delta(S')\big) - \hat{\boldsymbol{b}}\big(\delta(S' \setminus v)\big) = 2\hat{b}_{tv},$$

which proves that $\hat{b}_{tv} = 0$ for all $v \in B \cap S$. The proof for the second case '$v \in B \setminus S$' works analogously by using the tight cuts $\delta(S' \cup v)$ and $\delta(S')$.

The only thing left to show is that $\hat{b}_{ht} = \lambda \hat{a}_{ht}$. To do so, we consider the cut $\delta(S' \uplus t)$, which is tight for the inequality $(\hat{\boldsymbol{a}}, \alpha)$. This follows from condition (i) since the cut does not contain the edge ht. In conjunction with the tightness of $\delta(S')$, we obtain that

$$0 = \hat{\boldsymbol{b}}\big(\delta(S')\big) - \hat{\boldsymbol{b}}\big(\delta(S' \uplus t)\big) = \hat{b}_{ht} + \hat{\boldsymbol{b}}(t : S' \setminus h) - \hat{\boldsymbol{b}}\big(t : S'^{\complement}\big).$$

However, we know that $\hat{b}_{tv} = 0$ for all $v \in B$. Thus, the difference $\hat{\boldsymbol{b}}(t : S' \setminus h) - \hat{\boldsymbol{b}}\big(t : S'^{\complement}\big)$ simplifies to $\hat{\boldsymbol{b}}(t : T \cap S') - \hat{\boldsymbol{b}}(t : T \setminus S')$, which in turn equals $\lambda\big(\hat{\boldsymbol{a}}(t : T \cap S') - \hat{\boldsymbol{a}}(t : T \setminus S')\big)$ due to (2.8a). Moreover, according to (2.5) we have $\hat{a}_{tv} = \overline{a}_{wv}$ for all $v \in T$. Therefore, the term $\lambda\big(\hat{\boldsymbol{a}}(t : T \cap S') - \hat{\boldsymbol{a}}(t : T \setminus S')\big)$ actually equals $\lambda\big(\overline{\boldsymbol{a}}(w : T \cap S) - \overline{\boldsymbol{a}}(w : T \setminus S)\big)$, which is exactly $-\lambda \hat{a}_{ht}$ according to (2.9). Ultimately, we obtain the following equation:

$$0 = \hat{b}_{ht} + \hat{\boldsymbol{b}}(t : S' \setminus h) - \hat{\boldsymbol{b}}\big(t : S'^{\complement}\big) = \hat{b}_{ht} - \lambda \hat{a}_{ht}.$$

In summary, we have shown that $\hat{\boldsymbol{b}} = \lambda \hat{\boldsymbol{a}}$ thus proving that $(\hat{\boldsymbol{a}}, \alpha)$ defines a facet of the cut polytope $\mathrm{CUT}(\hat{G})$. □

Finally, the node-splitting operation also preserves an existing odd-cycle structure of a given inequality.

Lemma 2.11. *An inequality obtained from an odd-cycle inequality of* $\mathrm{CUT}(\overline{G})$ *by node-splitting is an odd-cycle inequality of* $\mathrm{CUT}(\hat{G})$.

Proof. Let $\overline{C}$ and $\overline{F}$ be the cycle and its odd subset, respectively, that define the initial odd-cycle inequality $(\overline{\boldsymbol{a}}, \alpha)$ of $\mathrm{CUT}(\overline{G})$. For ease of notation, we introduce the abbreviation

$$|\overline{\boldsymbol{a}}|(S) := \sum_{e \in S} |\overline{a}_e|.$$

Let $w \in \overline{C}$ be the node to be split into the nodes h and t with respect to the partition (H, T, B). In order to be consistent with (2.5) on page 49, we assume, without loss of generality, that $|\overline{\boldsymbol{a}}|(w : T) \leq |\overline{\boldsymbol{a}}|(w : H)$. Also, we denote the two neighbors of w in the cycle $\overline{C}$ by u and v, respectively.

First, we assume that at least one of the neighbors u, v lies outside B. Without loss of generality, let $u \in H$. Then, the remaining neighbor v can either be an element of T or of B. If $v \in T$ then both $|\overline{\boldsymbol{a}}|(w : H)$ and $|\overline{\boldsymbol{a}}|(w : T)$ have a value of one. Hence, the coefficient of ht is set to -1. The node sequence uwv of the supporting cycle $\overline{C}$ is replaced by $uhtv$, thereby extending $\overline{C}$ to the cycle $\hat{C} := \overline{C} \uplus ht$. However, since ht has a coefficient of -1, the odd subset $\overline{F}$ remains unchanged and the odd-cycle structure of the inequality is preserved. If, on the other hand, $v \in B$ then we have $|\overline{\boldsymbol{a}}|(w : T) = 0$. Consequently, the decompressed edge ht gets a coefficient of zero and the node sequence uwv of $\overline{C}$ is simply replaced by uhv. Altogether, the odd-cycle inequality remains unaltered.

Finally, if both u and v are elements of B, we proceed analogously to the latter one of the previous cases. □

In conjunction with two earlier results to be specified shortly, the above Lemma 2.11 allows the following important conclusion: If necessary, we can perform the separation of odd-cycle inequalities on the contracted graph $\overline{G}$ rather than on the initial graph G. In general, this will be more efficient due to the reduced size of the contracted graph. To see the correctness of this approach, we recall property (2) on page 48 concerning

the semimetric polytope. The property states that a given contracted LP solution $\overline{\boldsymbol{z}}$ is an element of $\mathrm{MET}(\overline{G})$ if and only if the corresponding switched LP solution $\tilde{\boldsymbol{z}}$ is an element of $\mathrm{MET}(G)$. In other words, if the switched LP solution violates at least one odd-cycle inequality then this is also true for the contracted LP solution. Moreover, any violated odd-cycle inequality on the contracted graph can be transformed into a separating odd-cycle inequality on the initial graph using lifting and switching. This is because both steps of the transformation preserve the odd-cycle structure as stated in the above Lemma 2.11 for the lifting and in Lemma 2.6 on page 44 for the switching, respectively.

In conclusion, the contraction procedure described in this section projects the switched LP solution $\tilde{\boldsymbol{z}}$ onto the subspace $\{\boldsymbol{x} \in \mathbb{R}^E \mid x_e = 0, \text{ for all } e \in E_0(\tilde{\boldsymbol{z}})\}$. Since the components of $\tilde{\boldsymbol{z}}$ are either fractional or zero, the contracted LP solution $\overline{\boldsymbol{z}}$ corresponds precisely to the fractional part of the switched LP solution. Thus, $\overline{\boldsymbol{z}}$ has at most the dimension of $\tilde{\boldsymbol{z}}$. Accordingly, the related contracted graph $\overline{G}$ has at least the density of the initial graph G. Any separating inequality for the contracted LP solution can be lifted to a separating inequality for the switched LP solution. The respective lifting procedure always preserves the validity of an inequality for the cut polytope. Moreover, it even preserves an existing facet-defining property of an inequality provided that the conditions of Lemma 2.10 on page 50 are met.

Yet, the contraction procedure does not guarantee that the contracted graph $\overline{G}$ is complete. This means that in order to allow the use of separation techniques for complete graphs, we possibly have to take further steps, which we will elaborate on in the next section.

2.1.3 Extension and Projection

In the previous section we have seen that the contracted graph $\overline{G}$ has at least the density of the initial graph G. Still, it is possible that $\overline{G}$ is not complete. In this case, an associated contracted LP solution $\overline{\boldsymbol{z}}$ does not specify LP values for the non-edges of $\overline{G}$. In order to apply separation procedures for complete graphs, we have to introduce artificial variables for the non-edges and assign suitable LP values to them. This results in an extended LP solution $\overline{\boldsymbol{z}}'$. Once we have found a separating inequality for $\overline{\boldsymbol{z}}'$, we have to project out the artificial variables to obtain a separating inequality for the contracted LP solution $\overline{\boldsymbol{z}}$. For reasons of clarity we will only consider the extension by a single variable. The general case can be derived by means of iteration.

Assume we are given a contracted yet non-complete graph $\overline{G} = (\overline{V}, \overline{E})$ as well as an LP solution $\overline{\boldsymbol{z}} \in \mathrm{MET}(\overline{G}) \setminus \mathrm{CUT}(\overline{G})$. Let $e = uv$ be a non-edge of $\overline{G}$. We define the extended edge set $\overline{E}' := \overline{E} \cup e$ and the respective **extended graph** $\overline{G}' = (\overline{V}, \overline{E}')$. Our goal is to obtain an **extended LP solution** $\overline{\boldsymbol{z}}'^T = (\overline{\boldsymbol{z}}^T, \xi)$ that is an element of $\mathrm{MET}(\overline{G}')$. Note that the corresponding set $\{\xi \mid (\overline{\boldsymbol{z}}^T, \xi) \in \mathrm{MET}(\overline{G}')\}$ of feasible values of ξ is nonempty. This is because $\overline{\boldsymbol{z}}$ is an element of $\mathrm{MET}(\overline{G})$, which is a projection of $\mathrm{MET}(\overline{G}')$ along the variable x_e (cf. Remark 6.1 in [Bar93]). In fact, the feasible set of artificial LP values is precisely the interval $[\xi_l, \xi_u]$ with the limits $\xi_l := \max\{0, L\}$ and $\xi_u := \min\{U, 1\}$, where L and U are defined as follows:

$$\begin{aligned} L &:= \max\{\overline{\boldsymbol{z}}(F) - \overline{\boldsymbol{z}}(P \setminus F) - |F| + 1 \mid P \ (u,v)\text{-path of } \overline{G},\ F \subseteq P,\ |F| \text{ odd}\}, \\ U &:= \min\{-\overline{\boldsymbol{z}}(F) + \overline{\boldsymbol{z}}(P \setminus F) + |F| \ \mid \ P \ (u,v)\text{-path of } \overline{G},\ F \subseteq P,\ |F| \text{ even}\}. \end{aligned} \tag{2.11}$$

To simplify matters, we will refer to the feasible range of artificial LP values simply as **feasible LP range**, the limits of which can be deduced as follows: First, we note that each cycle of the extended graph $\overline{G}'$ is either a cycle of the contracted graph $\overline{G}$ or it contains the artificial edge e. The cycles of $\overline{G}$ are negligible, though, since we already know that $\overline{\boldsymbol{z}} \in \mathrm{MET}(\overline{G})$. So, let C be a cycle of $\overline{G}'$ that contains the artificial edge e. As depicted in Figure 2.7, such a cycle decomposes into the edge e and a path $P = C \setminus e$ that links the ends u and v of e.

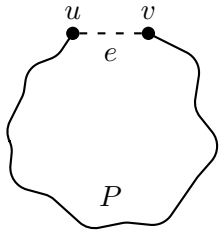


Figure 2.7: A cycle of the extended graph $\overline{G}'$ containing the artificial edge e.

Let $\boldsymbol{x}(F) - \boldsymbol{x}(C \setminus F) \leq |F| - 1$ be an arbitrary odd-cycle inequality on C. We solve the inequality for the artificial variable x_e. Depending on whether e is part of F or $C \setminus F$, we obtain one of the following inequalities:

$$\begin{aligned} x_e &\leq -\boldsymbol{x}(\hat{F}) + \boldsymbol{x}(P \setminus \hat{F}) + |\hat{F}|, \\ x_e &\geq \boldsymbol{x}(F) - \boldsymbol{x}(P \setminus F) - |F| + 1, \end{aligned}$$

where $\hat{F} := F \setminus e$ and thus $|\hat{F}|$ is even. Setting $\boldsymbol{x}$ to $\overline{\boldsymbol{z}}'$ provides an upper and lower bound on $\overline{z}'_e$, respectively. Finally, we get the preliminary limits L and U as specified in (2.11) by maximizing the lower bound and minimizing the upper bound over all (u, v)-paths P and all possible subsets F and $\hat{F}$, respectively, of P with appropriate parity.

The above construction makes sure that an extended LP solution $(\overline{\boldsymbol{z}}^T, \xi)$ satisfies all odd-cycle inequalities of the extended graph $\overline{G}'$ if and only if ξ lies within the range $[L, U]$. Note that the preliminary limits can be computed efficiently by solving a shortest-path problem on an auxiliary graph proposed by Barahona and Mahjoub [BM86]. A detailed description of this graph will be provided in the proof of Lemma 2.14 on page 57.

However, apart from the odd-cycle inequalities, the linear description of the semimetric polytope also comprises the trivial inequalities, which we haven't taken into account yet. In fact, we will see in Example 2.12 that the preliminary limits L and U are not restricted to the range $[0, 1]$.

Example 2.12. *Consider the graph* $\overline{G} = (\{s, t, u, v\}, \{f, g, h\})$ *as depicted in Figure 2.8. Since* $\overline{G}$ *is acyclic, we have* $\mathrm{MET}(\overline{G}) = [0, 1]^3$. *Hence, the point* $\overline{\boldsymbol{z}}^T := (0.5, 0.5, 0.5)$ *is an element of* $\mathrm{MET}(\overline{G})$, *but it does not represent a cut of* $\overline{G}$. *We introduce the artificial variable* x_e *for the non-edge* e. *For the computation of the preliminary limits* L *and* U *there is only one* (u, v)*-path to consider, namely* $P = \{f, g, h\}$. *Ultimately, we obtain the following limits:*

$$\begin{aligned} L &= \max_{F \subseteq P,\, |F| \text{ odd}} \{\, \overline{\boldsymbol{z}}(F) - \overline{\boldsymbol{z}}(P \setminus F) - |F| + 1 \,\} = -0.5, \\ U &= \min_{F \subseteq P,\, |F| \text{ even}} \{\, -\overline{\boldsymbol{z}}(F) + \overline{\boldsymbol{z}}(P \setminus F) + |F| \,\} = 1.5. \end{aligned}$$

This clearly shows that the values of L *and* U *are not restricted to the range* $[0, 1]$.

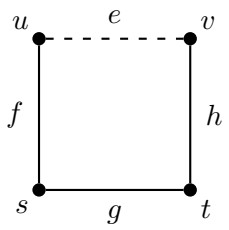


Figure 2.8: Graph of order four with artificial edge e.

As a consequence, to ensure that the extended LP solution is an element of the semimetric polytope, we introduce the revised limits $\xi_l := \max\{0, L\}$ and $\xi_u := \min\{U, 1\}$.

If lower and upper limit coincide, in which case the artificial LP value is uniquely defined, we say that the artificial edge e is **rigid**. Furthermore, we call an odd-cycle- and a trivial inequality, respectively, a **lower inequality** for e if it is tight at $(\overline{\boldsymbol{z}}^T, \xi_l)$. Analogously, an **upper inequality** for e is an odd-cycle- or trivial inequality that is tight at $(\overline{\boldsymbol{z}}^T, \xi_u)$. To be more precise, the trivial inequality $-x_e \leq 0$ is a lower inequality for e if $L \leq 0$. Otherwise, a lower inequality is given by the odd-cycle inequality derived from the argument of the maximum of (2.11). Similarly, the trivial inequality $x_e \leq 1$ is an upper inequality for e if $U \geq 1$. Otherwise, we can derive an upper inequality from the argument of the minimum of (2.11). When dealing with odd-cycle inequalities, we always assume that they are written in the form $\boldsymbol{x}(F) - \boldsymbol{x}(C \setminus F) \leq |F| - 1$. Then, the coefficient of the artificial variable x_e is -1 in a lower inequality and $+1$ in an upper inequality, respectively. We will come back to this property shortly.

For the moment, let us assume that we have successfully extended the LP solution to $\overline{\boldsymbol{z}}'$. Let $(\overline{\boldsymbol{a}}', \overline{\alpha}')$ be a corresponding separating inequality. Then, $\eta := \overline{\boldsymbol{a}}'^T \overline{\boldsymbol{z}}' - \overline{\alpha}' > 0$ is the amount by which $\overline{\boldsymbol{z}}'$ violates the inequality. Our next task is to derive a separating inequality for the contracted LP solution $\overline{\boldsymbol{z}}$. If the coefficient $\overline{a}'_e$ of the artificial edge is zero, we can simply truncate the left hand side vector $\overline{\boldsymbol{a}}'$ to the index set $\overline{E}$. Note that in this case $\overline{\boldsymbol{z}}$ violates the truncated inequality by the same amount η. Otherwise, we have to **project out** the artificial variable x_e. To do so, we need a valid inequality $(\overline{\boldsymbol{b}}', \overline{\beta}')$ for $\mathrm{CUT}(\overline{G}')$ such that $\overline{b}'_e$ is nonzero and has the opposite sign of $\overline{a}'_e$. Then,

$$(\overline{\boldsymbol{a}}', \overline{\alpha}') - \tfrac{\overline{a}'_e}{\overline{b}'_e}(\overline{\boldsymbol{b}}', \overline{\beta}') \tag{2.12}$$

is a valid inequality for $\mathrm{CUT}(\overline{G}')$ that has a zero coefficient for x_e. Thus, it can be truncated in the manner just mentioned. Moreover, if $(\overline{\boldsymbol{b}}', \overline{\beta}')$ is tight at $\overline{\boldsymbol{z}}'$ then (2.12) even preserves the original violation η. Otherwise, the violation can reduce and may even become negative.

At this point we recall that the coefficient of the artificial variable x_e is always -1 in a lower inequality for e and $+1$ in an upper inequality for e, respectively. As a consequence, a lower inequality is a natural choice for the auxiliary inequality $(\overline{\boldsymbol{b}}', \overline{\beta}')$ if the artificial coefficient $\overline{a}'_e$ is positive. Analogously, we can use an upper inequality if $\overline{a}'_e$ is negative. Note that if the edge e is rigid, which means that both the upper and the lower inequality are tight at $\overline{\boldsymbol{z}}'$, the projection preserves the amount of violation in any case. Otherwise, the actual loss of violation depends on the choice of the artificial LP value ξ within the feasible LP range $[\xi_l, \xi_u]$. However, this loss is at most $|\overline{a}'_e|(\xi_u - \xi_l)$.

Due to their favorable characteristics regarding the projection, rigid artificial edges are particularly interesting. Therefore, we now specify a sufficient condition for the

rigidity of a non-edge. It requires the notion of a tight cycle. We say that a cycle is **tight** at a given point if this is true for at least one odd-cycle inequality with respect to this cycle.

Lemma 2.13. *A non-edge $e \notin \overline{E}$ is rigid if it is the chord of a cycle that is tight at $\overline{\boldsymbol{z}} \in \mathrm{MET}(\overline{G})$.*

Proof. First, we note that the feasible LP range $[\xi_l, \xi_u]$ of the non-edge e is nonempty. This is because $\overline{\boldsymbol{z}}$ is an element of $\mathrm{MET}(\overline{G})$, which in turn is a projection of $\mathrm{MET}(\overline{G}')$ along the variable x_e. Hence, there must exist an element $(\overline{\boldsymbol{z}}^T, \xi)$ of $\mathrm{MET}(\overline{G}')$ that is projected onto $\overline{\boldsymbol{z}}$. As a result, we have $\xi_l \leq \xi_u$.

Let $\boldsymbol{x}(F) - \boldsymbol{x}(C \setminus F) \leq |F| - 1$ be an odd-cycle inequality that is tight at $\overline{\boldsymbol{z}}$. If $e = uv$ is a chord of C then it partitions the cycle into two (u, v)-paths with the respective edge sets C_1 and C_2. Let $F_1 := F \cap C_1$ and $F_2 := F \cap C_2$. Without loss of generality, we assume that $|F_1|$ is odd. Accordingly, the set F_2 has even cardinality. In conjunction with (2.11) on page 53, this leads to the following estimates:

$$\begin{aligned} L &\geq \overline{\boldsymbol{z}}(F_1) - \overline{\boldsymbol{z}}(C_1 \setminus F_1) - |F_1| + 1, \\ U &\leq -\overline{\boldsymbol{z}}(F_2) + \overline{\boldsymbol{z}}(C_2 \setminus F_2) + |F_2|. \end{aligned}$$

Since the odd-cycle inequality is tight at $\overline{\boldsymbol{z}}$, we have

$$\overline{\boldsymbol{z}}(F_1) + \overline{\boldsymbol{z}}(F_2) - \overline{\boldsymbol{z}}(C_1 \setminus F_1) - \overline{\boldsymbol{z}}(C_2 \setminus F_2) = |F_1| + |F_2| - 1.$$

This, however, implies that $U \leq L$. Finally, since $\xi_u \leq U$ and $L \leq \xi_l$ by definition, we obtain $\xi_u \leq \xi_l$. Together with the earlier result $\xi_l \leq \xi_u$, this proves the assertion. □

Finally, we briefly discuss under which conditions the projection of a single variable preserves an existing facet-defining property of a valid inequality for the cut polytope. Geometrically speaking, this is the case if and only if the projected inequality defines a ridge of $\mathrm{CUT}(\overline{G}')$. Let us assume that the separating inequality $(\overline{\boldsymbol{a}}', \overline{\alpha}')$ defines a facet Q of $\mathrm{CUT}(\overline{G}')$. Then, the projected inequality defines a ridge of $\mathrm{CUT}(\overline{G}')$ if and only if the face defined by the auxiliary inequality $(\overline{\boldsymbol{b}}', \overline{\beta}')$ is either a facet of Q itself or a facet of $\mathrm{CUT}(\overline{G}')$ that shares a common ridge with Q. There is also the special case that the projection transforms a ridge of $\mathrm{CUT}(\overline{G}')$ into a facet of $\mathrm{CUT}(\overline{G})$. Since a ridge is contained in exactly two facets, this can only happen if $(\overline{\boldsymbol{a}}', \overline{\alpha}')$ defines a ridge R of $\mathrm{CUT}(\overline{G}')$ and $(\overline{\boldsymbol{b}}', \overline{\beta}')$ defines one of the two facets containing R.

In conclusion, when using **static extension**, i. e., when assigning a fixed LP value to the artificial variable x_e, we can directly apply standard separation procedures to the extended LP solution $\overline{\boldsymbol{z}}'$. However, it is possible that the projection later on reduces the amount by which $\overline{\boldsymbol{z}}'$ violates an obtained separating inequality.

Adaptive Extension

We now introduce an alternative extension approach which we refer to as **adaptive extension**. Here, we do not fix the LP value of the artificial variable x_e. Instead, we store the limits ξ_l and ξ_u of the feasible LP range and leave the artificial LP value itself undetermined. We assume that it can take the value of either of the two limits depending on the sign of the respective coefficient $\overline{a}'_e$ in a given inequality $(\overline{\boldsymbol{a}}', \overline{\alpha}')$. To be

more precise, if, for instance, the artificial coefficient $\overline{a}'_e$ is positive then we assume that x_e takes the value of the lower limit ξ_l. This is because we would use a lower inequality to project out a positive coefficient. Accordingly, if $\overline{a}'_e$ is negative then we assume that the artificial LP value equals the upper limit ξ_u since we would use an upper inequality for the projection. As a result, the lower/upper inequality used to project out the artificial coefficient $\overline{a}'_e$ is guaranteed to be tight at the adaptively extended LP solution. Thus, the projection will not alter the amount of violation.

At this point we need to clarify that the term "violation" has a slightly different meaning in the context of adaptive extension. Consider an extended LP solution $\overline{\boldsymbol{z}}'$ and an inequality $(\overline{\boldsymbol{a}}', \overline{\alpha}')$. Let $\eta := \overline{\boldsymbol{a}}'^T \overline{\boldsymbol{z}}' - \overline{\alpha}'$ be the respective violation. For a statically extended LP solution, η measures the amount by which $\overline{\boldsymbol{z}}'$ violates the inequality $(\overline{\boldsymbol{a}}', \overline{\alpha}')$. In contrast, when using adaptive extension, η specifies the amount by which the contracted LP solution $\overline{\boldsymbol{z}}$ will violate the already projected inequality $(\overline{\boldsymbol{a}}, \overline{\alpha})$. Clearly, the latter value is invariant under the projection.

Another key advantage of the adaptive extension is that the limits of the feasible LP ranges of different artificial variables are mutually independent. Therefore, we can exploit an existing sparse structure of the contracted graph. This accelerates the computation of the limits and thus the overall shrink separation.

Lemma 2.14. *Let $\overline{\boldsymbol{z}} \in \mathrm{MET}(\overline{G})$ be a contracted LP solution. Then, the respective limits of the feasible LP ranges of different non-edges are mutually independent. In particular, the values of the limits are only determined by the LP values of existing edges.*

Proof. Let $\overline{G} = (\overline{V}, \overline{E})$ be the underlying graph of the contracted LP solution $\overline{\boldsymbol{z}}$. We introduce two copies $G^1 = (V^1, E^1)$ and $G^2 = (V^2, E^2)$ of the graph $\overline{G}$. In these copies we denote the duplicate of a node $v \in \overline{V}$ by v^1 and v^2, respectively. We now define the auxiliary graph $G^a = (V^a, E^a)$ with the node set $V^a := V^1 \dot\cup V^2$ and the edge set $E^a := E^1 \dot\cup E^2 \dot\cup \{u^1v^2, u^2v^1 \mid uv \in \overline{E}\}$. Each edge $e \in E^a$ receives an edge weight w_e as follows:

$$w_e := \begin{cases} \overline{z}_{uv} & \text{if } e \in \{u^1v^1, u^2v^2\}, \\ 1 - \overline{z}_{uv} & \text{otherwise.} \end{cases}$$

Note that all edge weights in G^a have a value between zero and one. This is because the contraction ensures that $0 < \overline{z}_{uv} < 1$, for all $uv \in \overline{E}$. The auxiliary graph G^a was initially proposed by Barahona and Mahjoub [BM86] for the exact separation of odd-cycle inequalities in polynomial time. For a better understanding, Figure 2.9 depicts the subgraph of G^a that is induced by a single edge uv of the contracted graph $\overline{G}$.

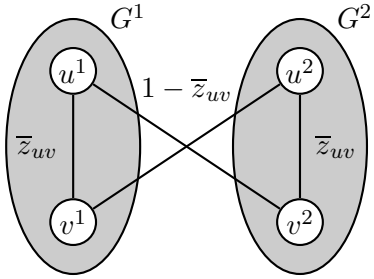


Figure 2.9: Subgraph of the auxiliary graph G^a induced by a single edge uv of $\overline{G}$.

Our goal is to show that we can derive the limits L and U, as given in (2.11) on page 53, from the lengths of certain shortest paths in G^a. To be more precise: For a

given non-edge $f = st$ of $\overline{G}$, we obtain the respective limits L_f and U_f as

$$\begin{aligned} L_f &:= \max_{\substack{P\ (s,t)\text{-path of } \overline{G},\\ F \subseteq P, |F| \text{ odd}}} \{\, \overline{\boldsymbol{z}}(F) - \overline{\boldsymbol{z}}(P \setminus F) - |F| + 1 \,\} = 1 - \boldsymbol{w}(P^a_{s^1t^2}), \\ U_f &:= \min_{\substack{P\ (s,t)\text{-path of } \overline{G},\\ F \subseteq P, |F| \text{ even}}} \{\, -\overline{\boldsymbol{z}}(F) + \overline{\boldsymbol{z}}(P \setminus F) + |F| \,\} \quad = \boldsymbol{w}(P^a_{s^1t^1}). \end{aligned} \tag{2.13}$$

Here, $P^a_{s^1t^2}$ and $P^a_{s^1t^1}$ denote a shortest (s^1, t^2)- and (s^1, t^1)-path in G^a, respectively. Note that, due to the symmetry of G^a, we can always assume the first node of a path to be an element of V^1 without loss of generality.

Before we can prove the above proposition, we first need to characterize some important properties of paths in the auxiliary graph. We start by examining how shortest paths in G^a relate to paths in $\overline{G}$. For a given path in G^a, we obtain its counterpart in $\overline{G}$ by identifying each node duplicate v^1 and v^2, respectively, with the corresponding original node v. In general, however, the resulting structure is not a path but merely a chain. This is because paths in G^a may contain nodes from both V^1 and V^2 at the same time. The situation is different for *shortest* paths in G^a. Here, the respective counterpart in $\overline{G}$ is indeed a path, as we will show now. Let P^a be an arbitrary shortest path in G^a. If the path contains no recurring duplicates, i. e., if P^a does not contain both v^1 and v^2 for any of the nodes $v \in \overline{V}$, then its counterpart in $\overline{G}$ is obviously a path. So, assume that P^a contains at least one recurring duplicate. Let (u^1, u^2) be the earliest such recurrence in P^a. In other words, u^2 is the first node of P^a for which there has already been another duplicate, namely u^1, of the same original node u before. We denote the corresponding (u^1, u^2)-subpath of P^a by $P^a_{u^1u^2}$. The pair (u^1, u^2) is, by construction, the only recurring duplicate in $P^a_{u^1u^2}$ and thus the subpath corresponds to a cycle C in $\overline{G}$. Next, we consider the edge set F^a which consists of the common edges shared by $P^a_{u^1u^2}$ and the cut $(V^1 : V^2)$. Since the subpath starts in V^1 and ends in V^2, the set F^a must have odd cardinality. Moreover, F^a corresponds to an odd subset F of the cycle C in $\overline{G}$. Finally, the length of $P^a_{u^1u^2}$ is given by

$$\sum_{e \in F} (1 - \overline{z}_e) + \sum_{e \in C \setminus F} \overline{z}_e \;=\; |F| - \overline{\boldsymbol{z}}(F) + \overline{\boldsymbol{z}}(C \setminus F).$$

Since $\overline{\boldsymbol{z}}$ is an element of $\mathrm{MET}(\overline{G})$, we have $\overline{\boldsymbol{z}}(F) - \overline{\boldsymbol{z}}(C \setminus F) \leq |F| - 1$ and thus the length of $P^a_{u^1u^2}$ is greater or equal one. At this point we recall that all edge weights in G^a have a value between zero and one. Consequently, we can reduce the length of the path P^a as follows: We omit the (u^1, u^2)-subpath entirely; instead, we traverse the edge that joins u^1 to the successor of u^2 in P^a. This, however, contradicts the assumption that P^a is a shortest path. In summary, we have shown that shortest paths in G^a cannot contain recurring duplicates and thus correspond to paths in $\overline{G}$.

The next important property of paths in G^a has already been mentioned above: Any (u^1, v^2)-path shares an odd number of edges with the cut $(V^1 : V^2)$. Analogously, the number of edges in the intersection of $(V^1 : V^2)$ with an arbitrary (u^1, v^1)-path is always even.

With the above results we can finally show that the limits L_f and U_f in (2.13) are indeed given by the lengths of appropriate shortest paths in G^a. Yet, we will only provide the detailed proof for U_f. The result for L_f can be derived analogously. So, our task is to minimize the objective $-\overline{\boldsymbol{z}}(F) + \overline{\boldsymbol{z}}(P \setminus F) + |F|$ over all (s,t)-paths P of $\overline{G}$ and all

subsets F of P with even cardinality. First, we recall the relationship between paths in $\overline{G}$ and shortest paths in G^a. Since we are only interested in the (s,t)-paths of $\overline{G}$, it suffices to consider the shortest paths in G^a from the duplicate s^1 to either of the duplicates t^1 and t^2 of t, respectively. Remember that, without loss of generality, we can assume the shortest path to start in V^1 due to the symmetry of G^a. In the next step we rewrite the objective $-\overline{\boldsymbol{z}}(F) + \overline{\boldsymbol{z}}(P \setminus F) + |F|$ as follows:

$$\sum_{uv \in F} (1 - \overline{z}_{uv}) + \sum_{uv \in P \setminus F} \overline{z}_{uv}.$$

Note that the respective summands of the edges in F resemble the weights of the edges in the cut $(V^1 : V^2)$. Moreover, F is supposed to have even cardinality. As a result, we only need to consider the shortest (s^1, t^1)-paths since these are the ones that cross the cut $(V^1 : V^2)$ an even number of times. Altogether, we see that the value of the upper limit U_f equals the length of a shortest (s^1, t^1)-path in G^a.

As a direct consequence of (2.13), we obtain that the limits L and U of different non-edges are mutually independent. This is because the auxiliary graph G^a is constructed from the contracted graph $\overline{G}$, which does not contain any artificial edges. To see that this is indeed correct, let us consider the following scenario: We want to determine the limits L_f and U_f of a given non-edge $f = st$ of $\overline{G}$. In this case, however, we also use the information on another non-edge $g = ij \neq st$ and its artificial LP limits L_g and U_g. To do so, we add the artificial edges i^1j^1, i^2j^2, i^1j^2, and i^2j^1 to G^a. Yet, since we use adaptive extension, the artificial LP value of the non-edge g is a priori undetermined. Therefore, the actual weights of the newly introduced edges depend on the signs of their respective coefficients in a given inequality. In this scenario, the relevant inequalities are the tight odd-cycle inequalities that can be derived from the argument of the maximum and the minimum, respectively, in (2.13). These tight odd-cycle inequalities have the general form

$$\boldsymbol{x}(F) - \boldsymbol{x}(C \setminus F) \leq |F| - 1,$$

where F is a subset of the cycle C with odd cardinality. Looking at the following equivalent formulation

$$\sum_{e \in F} (1 - x_e) + \sum_{e \in C \setminus F} x_e \geq 1,$$

we see that the edges in F correspond to edges in the cut $(V^1 : V^2)$. Thus, in the general form of the odd-cycle inequalities, the edges i^1j^2 and j^1i^2 have a respective coefficient value of 1. Accordingly, the artificial LP value of g is set to the lower limit L_g, which results in an edge weight of $1 - L_g$. Analogously, the edges i^1j^1 and i^2j^2 correspond to edges in $C \setminus F$ with a coefficient of -1. Hence, we obtain an edge weight of U_g.

Now, according to (2.13) we can replace each occurrence of an edge i^1j^2 or j^1i^2 with a shortest (i^1, j^2)- or (j^1, i^2)-path that does not contain any of the newly introduced edges. The weight of such a shortest path is precisely the weight of the replaced edge, namely $1 - L_g$. Moreover, this substitution does not change the parity of the number of times the overall path crosses the cut $(V^1 : V^2)$. This is because the intersection of a shortest (i^1, j^2)- or (j^1, i^2)-path with $(V^1 : V^2)$ has odd cardinality. Analogously, each occurrence of an edge i^1j^1 or i^2j^2 can be replaced with a shortest (i^1, j^1)- or (i^2, j^2)-path. As before, this neither changes the length of the overall path nor the parity of the number of times it crosses $(V^1 : V^2)$.

In conclusion, we can eliminate any occurrence of a newly introduced edge without loss of information. This proves the mutual independence of the limits of different feasible LP ranges. □

However, there is the following downside to adaptive extension: In general, the available separation procedures cannot handle the a priori undetermined LP values. This means that we have to make certain adjustments, which in turn can increase the computational complexity. The exception are separation procedures for classes of inequalities in which all nonzero coefficients have the same sign. Here, we can simply fix the artificial LP value to the appropriate limit of the feasible LP range and then apply a standard separation procedure. This is, for instance, the case for bicycle-p-wheel- or clique inequalities, which have exclusively nonnegative coefficients.

In summary, the key benefit of the extension—especially when dealing with sparse graphs—is that it allows to apply separation techniques for dense and complete graphs that we possibly would not have been able to use on a contracted LP solution otherwise. For this purpose, we introduce artificial LP values for the non-edges. In doing so, we can choose between the following two approaches: The first one is static extension, which works with fixed artificial LP values. This allows us to use unmodified separation procedures on the extended LP solution, but it also carries the risk of losing some of the obtained separating inequalities during projection. The second approach is adaptive extension, which uses a priori undetermined artificial LP values. Although it may require extensive modifications of the separation procedures, this alternative method guarantees that the projection does not affect the amount of violation of the separated inequalities.

Either way, each separating inequality has to be projected to remove all nonzero left hand side coefficients related to non-edges. However, the necessary information for this transformation is obtained as a byproduct of the extension without additional effort.

2.1.4 Separation

We now take a closer look at the separation of valid inequalities for the cut polytope. Here, we focus on four specific classes of inequalities, namely bicycle-p-wheel-, clique-, and hypermetric inequalities as well as target cuts. This is because these are the inequality classes that we actually separate in our implementation of the shrink separation. In addition, we pay special attention to the respective characteristics of the separation procedures with regard to adaptive extension.

Bicycle-p-wheel- and Clique Inequalities

As pointed out in Section 1.4, bicycle-p-wheel- and clique inequalities are valid for the cut polytope. We recall that bicycle-p-wheel inequalities have the following form:

$$\boldsymbol{x}(B) \leq 2p,$$

where B denotes the edge set of a bicycle-p-wheel as depicted in Figure 1.7 on page 32. Clique inequalities, on the other hand, look as follows:

$$\boldsymbol{x}(F) \leq \left\lceil \frac{k}{2} \right\rceil \left\lfloor \frac{k}{2} \right\rfloor,$$

where F denotes the edge set of a clique of order k. Note that the left hand side coefficients of both types of inequalities are all either zero or one. In particular, all nonzero left

hand side coefficients have the same sign. As a result, these two inequality classes are convenient special cases regarding adaptive extension.

As explained in Section 2.1.3, the adaptive extension approach uses a priori undetermined artificial LP values. We assume, however, that the possible values are restricted to either of the limits of their respective feasible LP range. The decision, which of the limits is actually taken, is made during the course of the separation. It depends on the sign of the corresponding coefficient in a given inequality. However, things are different for inequality classes in which for all contained inequalities the nonzero left hand side coefficients have the same sign. In this case, the adaptive extension either fixes all artificial LP values to the lower limit of their respective feasible LP range or it fixes them all to the upper one. Thus, under these circumstances, the adaptive extension is identical to the static extension and therefore does not require any modifications to the separation procedure.

As a result, we can separate both bicycle-p-wheel- and clique inequalities using standard procedures regardless of the chosen extension approach. In our implementation we use the algorithm of Gerards [Ger85] for the bicycle-p-wheel inequalities and a greedy heuristic for the clique inequalities, respectively. We now proceed with the class of hypermetric inequalities.

Hypermetric Inequalities

In Section 1.4 we have seen that hypermetric inequalities have the following form:

$$\sum_{1 \le i < j \le n} b_i b_j x_{ij} \le 0,$$

where $\boldsymbol{b} = (b_1, \ldots, b_n)$, $n \ge 2$, is an integral vector that satisfies $\sum_{i=1}^{n} b_i = 1$. These inequalities are valid for the cut cone associated with the complete graph $K_n = (V_n, E_n)$. Since the cut cone contains the cut polytope $\mathrm{CUT}(K_n)$, the hypermetric inequalities are also valid for $\mathrm{CUT}(K_n)$. However, if the LP solution $\boldsymbol{z}$ to be separated lies in the interior of the cut cone then the hypermetric inequalities are of no use. Fortunately, we can use the switching operation from Section 2.1.1 to circumvent this problem in the following way: First, we locate a vertex $\boldsymbol{v}$ of $\mathrm{CUT}(K_n)$ in the vicinity of the LP solution. This can be accomplished, for instance, by using a rounding heuristic like Algorithm 1.2 on page 28. Next, we apply the switching mapping alongside the cut that is associated with the vertex $\boldsymbol{v}$. This maps $\boldsymbol{v}$ to the origin $\mathbf{0}$, which is the apex of the cut cone. At the same time, the LP solution $\boldsymbol{z}$ is mapped to a fractional point $\boldsymbol{y}$ near the apex. Note that, due to Corollary 2.4 on page 43, the point $\boldsymbol{y}$ is guaranteed to lie outside the cut cone. We can now separate $\boldsymbol{y}$ from the cut cone using separation procedures for hypermetric inequalities. Finally, the resulting separating inequalities are switched alongside the cut associated with $\boldsymbol{v}$ to obtain their respective counterparts for the original LP solution $\boldsymbol{z}$.

In our implementation we use a greedy heuristic to separate the hypermetric inequalities. For a given fractional LP solution $\boldsymbol{z}$ we first initialize the integral vector $\boldsymbol{b}$ in such a way that it already satisfies $\sum_{i=1}^{n} b_i = 1$. The goal is to increase the amount of violation $\sum_{1 \le i < j \le n} b_i b_j z_{ij}$. However, for the considerations below it is more convenient to refer to the edges as "$uv \in E_n$" rather than "ij, $1 \le i < j \le n$". So, let

$$viol(\boldsymbol{z}, \boldsymbol{b}) := \sum_{uv \in E_n} b_u b_v z_{uv}$$

denote the amount by which the LP solution $\boldsymbol{z}$ violates the hypermetric inequality induced by the vector $\boldsymbol{b}$. In each iteration we perform a local change on $\boldsymbol{b}$ to obtain a new integral vector $\hat{\boldsymbol{b}}$. To do so, we determine two node indices k, l and define $\hat{\boldsymbol{b}}$ as follows:

$$\hat{b}_u := \begin{cases} b_u + 1 & \text{if } u = k, \\ b_u - 1 & \text{if } u = l, \\ b_u & \text{otherwise.} \end{cases}$$

By construction, the new vector also satisfies $\sum_{v \in V_n} \hat{b}_v = 1$. Depending on the setting, the indices k and l are either chosen at random or in such a way that the resulting increase in violation is maximized over all possible choices. We obtain the amount of violation with respect to the new vector $\hat{\boldsymbol{b}}$ via the following update formula:

$$viol(\boldsymbol{z}, \hat{\boldsymbol{b}}) \;=\; viol(\boldsymbol{z}, \boldsymbol{b}) \;+ \sum_{v \in V_n \setminus \{k,l\}} b_v(z_{kv} - z_{lv}) + (b_l - b_k - 1)z_{kl}.$$

If the local change on $\boldsymbol{b}$ increases the amount of violation then we accept it, which means that we set $\boldsymbol{b} = \hat{\boldsymbol{b}}$ and iterate.

When using adaptive extension, though, the update formula becomes more involved. This is because the artificial LP value of a non-edge uv changes with the sign of the product $b_u b_v$. For a better understanding, consider the following example: Let z_{uv} be an artificial LP value with lower limit ξ_l and upper limit ξ_u. Furthermore, let the corresponding coefficients in the hypermetric inequalities induced by $\boldsymbol{b}$ and $\hat{\boldsymbol{b}}$ be $b_u b_v > 0$ and $\hat{b}_u \hat{b}_v < 0$, respectively. This means that z_{uv} takes the lower limit in $viol(\boldsymbol{z}, \boldsymbol{b})$ while taking the upper one in $viol(\boldsymbol{z}, \hat{\boldsymbol{b}})$. We see that a change to the vector $\boldsymbol{b}$ can affect the artificial LP values that we take as a basis for the computation of the violation. Consequently, the update formula has to take this effect into account.

To simplify the notation below, we denote the LP solutions subject to the old vector $\boldsymbol{b}$ and the new vector $\hat{\boldsymbol{b}}$ simply by $\boldsymbol{z}$ and $\hat{\boldsymbol{z}}$, respectively. Then, the new amount of violation is as follows:

$$\begin{aligned} viol(\hat{\boldsymbol{z}}, \hat{\boldsymbol{b}}) \;= \sum_{uv \in E_n \setminus (\delta(k) \cup \delta(l))} b_u b_v z_{uv} + \sum_{v \in V_n \setminus \{k,l\}} b_v\big((b_k + 1)\hat{z}_{kv} + (b_l - 1)\hat{z}_{lv}\big) \\ + (b_k + 1)(b_l - 1)\hat{z}_{kl}. \end{aligned}$$

Finally, we define $d_{uv} := (\hat{z}_{uv} - z_{uv})$ and substitute the term $b_u b_v z_{uv} + b_u b_v d_{uv}$ for each occurrence of $b_u b_v \hat{z}_{uv}$ where at least one of the indices u and v is in $\{k, l\}$. As a result, we obtain the following update formula for the adaptive extension case:

$$\begin{aligned} viol(\hat{\boldsymbol{z}}, \hat{\boldsymbol{b}}) \;= \; viol(\boldsymbol{z}, \boldsymbol{b}) + \sum_{v \in V_n \setminus \{k,l\}} b_v\big(b_k d_{kv} + b_l d_{lv} + (\hat{z}_{kv} - \hat{z}_{lv})\big) \\ + b_k b_l d_{kl} + (b_l - b_k - 1)\hat{z}_{kl}. \end{aligned}$$

In conclusion, compared to bicycle-p-wheel- or clique inequalities, it is considerably more complex to adjust the respective separation procedure for hypermetric inequalities to the use of adaptive extension. Still, the modifications are relatively easy to accomplish since we wrote the procedure ourselves and therefore have full access to the source code.

The final part of this section discusses the separation of target cuts. In particular, we will give an example of how to use adaptive extension in conjunction with a target cut application programming interface.

Target Cuts

As explained in Section 0.4.5, target cuts are inequalities that do not put any restrictions on their actual structure. In particular, they do not require a complete underlying graph. As a result, we can separate target cuts directly on a contracted LP solution. At this point we would like to thank Dr. Christoph Buchheim, Dr. Frauke Liers, and Dr. Marcus Oswald for kindly providing their target cut software framework. This framework implements an **application programming interface** (API) for the separation of target cuts. We extended the framework to use it in our max-cut solver. In particular, we implemented the virtual oracle function, which is necessary for the delayed column generation.

The framework expects two sets of vectors $\overline{\boldsymbol{x}}_1, \ldots, \overline{\boldsymbol{x}}_s$ and $\overline{\boldsymbol{y}}_1, \ldots, \overline{\boldsymbol{y}}_t$ as well as an additional pair of vectors $\overline{\boldsymbol{q}}$ and $\overline{\boldsymbol{x}}^*$ as input. All vectors must have the same dimension. The two vector sets, the second of which may be empty, define a polyhedron $Q = \operatorname{conv}\{\overline{\boldsymbol{x}}_1, \ldots, \overline{\boldsymbol{x}}_s\} + \operatorname{cone}\{\overline{\boldsymbol{y}}_1, \ldots, \overline{\boldsymbol{y}}_t\}$. The remaining two vectors specify an interior point $\overline{\boldsymbol{q}} \in Q$ and an infeasible point $\overline{\boldsymbol{x}}^*$ to be separated from Q, respectively. Given this input data, the framework solves the following generalized version of the LP (0.5) on page 17:

$$\begin{aligned} \max\ & \boldsymbol{a}^T(\overline{\boldsymbol{x}}^* - \overline{\boldsymbol{q}}) \\ \text{s.t.}\ & \boldsymbol{a}^T(\overline{\boldsymbol{x}}_i - \overline{\boldsymbol{q}}) \leq 1 \ \text{ for all } i = 1, \ldots, s \\ & \boldsymbol{a}^T\overline{\boldsymbol{y}}_j \qquad \leq 0 \ \text{ for all } j = 1, \ldots, t \\ & \boldsymbol{a} \in \mathbb{R}^m \end{aligned} \tag{2.14}$$

The above LP is more general than (0.5) since it optimizes over a polyhedron rather than a polytope. Yet, the geometrical interpretation of the target cut separation is the same in both cases.

For the separation of a contracted LP solution $\overline{\boldsymbol{z}}$, we choose Q to be the cut polytope $\operatorname{CUT}(\overline{G})$. As first set of vectors $\overline{\boldsymbol{x}}_1, \ldots, \overline{\boldsymbol{x}}_s$ we provide the incidence vectors of all the cuts of $\overline{G}$. Alternatively, if we enable delayed column generation then we only need a suitable subset of the incidence vectors. In either case, the second set of vectors $\overline{\boldsymbol{y}}_1, \ldots, \overline{\boldsymbol{y}}_t$ is left empty. The interior point $\overline{\boldsymbol{q}}$ can be, for instance, the vertex barycenter of the cut polytope or, even simpler, the vector $1/2 \cdot \mathbf{1}$ of all one-halves. Finally, we set the infeasible point $\overline{\boldsymbol{x}}^*$ to the contracted LP solution and call the target cut separation procedure.

Our preliminary computational experiments showed that the target cut software framework can only handle graphs with up to twenty nodes within a reasonable time. As a rule of thumb, for graphs with at most twelve nodes it is typically faster to provide the incidence vectors of all the cuts at the very beginning than using the delayed column generation. For larger graphs, though, the total enumeration of all incidence vectors is out of the question.

However, for problem sizes beyond the limits just mentioned we have to restrict the target cut separation to reasonably sized node-induced subgraphs of $\overline{G}$. These subgraphs are chosen as follows: We start with a random node and sequentially add more nodes using a greedy criterion with respect to the fractionality of the involved edges. For a given node set W, let $N(W)$ denote the set of nodes outside W that are adjacent to at least one element of W. Then, the node set W_k of a subgraph of order k is built as

follows:

$$W_1 = \text{random node } u,$$
$$W_i = W_{i-1} \cup \operatorname*{arg\,max}_{v \in N(W_{i-1})} \sum_{e \in (v:W_{i-1})} (\tfrac{1}{2} - |\overline{z}_e - \tfrac{1}{2}|), \text{ for } i = 2, \ldots, k.$$

Once we have chosen a subgraph G' of $\overline{G}$, we project the separation problem onto the affine hull of $\mathrm{CUT}(G')$ and apply the target cut separation. Note that the projection may map the contracted LP solution to a point inside $\mathrm{CUT}(G')$, in which case the separation will fail. Yet, if we do find a facet defining inequality for the cut polytope on the subgraph then we can use 0-node lifting to make it applicable to the contracted LP solution $\overline{z}$. As pointed out in Section 1.4.2, the lifted inequality is still valid for the cut polytope whereas the facet-defining property of the original inequality may be lost.

In principle, it is also possible to separate target cuts on an extended LP solution. We now explain how to use the target cut API in conjunction with adaptive extension. However, these considerations are to be understood as a proof of concept and have no practical relevance. For the separation of the extended LP solution $\overline{\boldsymbol{z}}'$, we choose Q to be the cut polytope $\mathrm{CUT}(K_p)$, where $p := |\overline{V}|$ is the order of the contracted graph $\overline{G}$. Let $m := {}^{p(p-1)}/_2$ be the number of edges of K_p. As first set of vectors $\overline{\boldsymbol{x}}_1, \ldots, \overline{\boldsymbol{x}}_s$ we provide the m-dimensional incidence vectors of the cuts of K_p. As before, we leave the second set of vectors $\overline{\boldsymbol{y}}_1, \ldots, \overline{\boldsymbol{y}}_t$ empty and choose the interior point $\overline{\boldsymbol{q}}$ to be either the vertex barycenter of the cut polytope or the vector ${}^1/_2 \cdot \mathbf{1}$. Let l denote the number of non-rigid non-edges of $\overline{G}$. Then, the infeasible point $\overline{\boldsymbol{x}}^*$ is composed of the following three parts: the contracted LP solution $\overline{\boldsymbol{z}}$, the unique artificial LP values of all rigid non-edges, and l yet undetermined LP values of the non-rigid non-edges. Without loss of generality, we assume that the last l entries of the m-dimensional vectors correspond to the non-rigid non-edges. We now define the $(m+l)$-dimensional vectors $\overline{\boldsymbol{x}}'_i$, $\overline{\boldsymbol{x}}'^*$ and $\overline{\boldsymbol{q}}'$. In these extended vectors, the k-th non-rigid non-edge corresponds to the pair of entries at the positions $m-l+k$ and $m+k$. We obtain the vector $\overline{\boldsymbol{q}}'$ by taking $\overline{\boldsymbol{q}}$ and appending copies of the last l entries to it. The vectors $\overline{\boldsymbol{x}}'_i$, $i = 1, \ldots, s$, are constructed analogously. For the extended infeasible vector $\overline{\boldsymbol{x}}'^*$ we carry over the first $m-l$ entries of $\overline{\boldsymbol{x}}^*$. The remaining entries $m-l+k$ and $m+k$, for $k = 1, \ldots, l$, are set to the lower and upper limit, respectively, of the feasible LP range of the k-th non-rigid non-edge. Finally, we define the polyhedron

$$Q' = \mathrm{conv}\,\{\overline{\boldsymbol{x}}'_1, \ldots, \overline{\boldsymbol{x}}'_s\} + \mathrm{cone}\,\{-\boldsymbol{e}_{m-l+k}, \boldsymbol{e}_{m+k} \mid k = 1, \ldots, l\},$$

where $\boldsymbol{e}_j$ denotes the j-th canonical unit vector of dimension $m+l$.

Given Q', $\overline{\boldsymbol{q}}'$, and $\overline{\boldsymbol{x}}'^*$ as input, the target cut separation solves the following linear program:

$$\begin{aligned} \max\ & \boldsymbol{a}'^T(\overline{\boldsymbol{x}}'^* - \overline{\boldsymbol{q}}') & & \\ \text{s.t.}\ & \boldsymbol{a}'^T(\overline{\boldsymbol{x}}'_i - \overline{\boldsymbol{q}}') & \le 1 & \text{ for all } i = 1, \ldots, s \\ & a'_{m-l+k} & \ge 0 & \text{ for all } k = 1, \ldots, l \\ & a'_{m+k} & \le 0 & \text{ for all } k = 1, \ldots, l \\ & \boldsymbol{a}' \in \mathbb{R}^{m+l} & & \end{aligned} \tag{2.15}$$

In each of the vertices $\overline{\boldsymbol{x}}'_i$ of Q' as well as in the interior point $\overline{\boldsymbol{q}}'$, the two respective entries of a given non-rigid non-edge share the same value by construction. Moreover,

the infeasible point $\overline{\boldsymbol{x}}'^*$ provides for each non-rigid non-edge the two possible LP values, namely the respective lower and upper interval limits ξ_l and ξ_u. Finally, by introducing the directions $-\boldsymbol{e}_{m-l+k}$ and $\boldsymbol{e}_{m+k}$, $k = 1, \ldots, l$, we force the $\boldsymbol{a}'$ coefficients of the lower and upper limits to have positive and negative sign, respectively.

Let $\boldsymbol{a}'^*$ denote an optimum solution of the LP (2.15). Then, $\boldsymbol{a}'^*$ induces an optimum solution $\boldsymbol{a}^*$ of the LP (2.14) if for each non-rigid non-edge at most one of the two respective solution entries a'^*_{m-l+k} and a'^*_{m+k} is nonzero. This property, however, is already implied by the LP formulation. Suppose not and that there exists a $\kappa \in \{1, \ldots, l\}$ such that $a'^*_{m-l+\kappa}$ is positive and $a'^*_{m+\kappa}$ is negative. To simplify matters, we substitute the indices $m-l+\kappa$ and $m+\kappa$ with the specifiers '*pos*' and '*neg*', respectively. Let ξ_l and ξ_u be the limits of the feasible LP range of the κ-th non-rigid non-edge. We recall that the entries $\overline{q}'_{pos}$ and $\overline{q}'_{neg}$ have the same value by construction. Thus, the corresponding summands of the objective function are as follows:

$$a'^*_{pos}(\overline{x}'^*_{pos} - \overline{q}'_{pos}) + a'^*_{neg}(\overline{x}'^*_{neg} - \overline{q}'_{neg}) = a'^*_{pos}\xi_l + a'^*_{neg}\xi_u - (a'^*_{pos} + a'^*_{neg})\overline{q}'_{pos}.$$

Let $\Delta := \min\{a'^*_{pos}, -a'^*_{neg}\}$. We note that $\xi_l < \xi_u$ and that a'^*_{pos} and a'^*_{neg} have opposite signs. This means that we can increase the objective value by replacing a'^*_{pos} with $a'^*_{pos} - \Delta$ as well as a'^*_{neg} with $a'^*_{neg} + \Delta$. The result is also feasible since the sum of the coefficients remains unchanged and thus all the constraints are still satisfied. This, however, contradicts the optimality of $\boldsymbol{a}'^*$ and thus proves by contrapositive that at most one of the two values a'^*_{m-l+k} and a'^*_{m+k} can be nonzero.

We conclude this section with some recapitulatory remarks on adaptive extension. Except for some special cases like, for instance, bicycle-p-wheel- and clique inequalities, the use of adaptive extension requires the modification of the applied separation procedures. The difficulty of this task varies with the complexity of these procedures and the level of access to their source code. However, in case of a **black box** procedure, i. e., a program with fixed input/output interface and inaccessible source code, it may not be possible to use adaptive extension at all.

2.2 Implementation

Based on the theory presented in Section 2.1 we implemented the shrink separation in C++ and embedded it in the branch-and-cut framework ABACUS ("A Branch-And-CUt System"). For more information regarding ABACUS we refer to [EGJR01, JT98, JT00, Thi95].

However, parts of the theory turned out to be computationally inconvenient. In Section 2.2.1 we discuss the workflow of the actual implementation and highlight the differences to the underlying theory. Afterwards, we examine certain numerical aspects of the procedure in Section 2.2.2.

2.2.1 Workflow

To provide a general idea of the implemented shrink separation, Figure 2.10 depicts a flowchart of the overall workflow. The initial input is a fractional LP solution $\boldsymbol{z}$ that we want to separate from the cut polytope. In the theory we simply assume that $\boldsymbol{z}$ is an element of the semimetric polytope. In the procedure, on the other hand, we have to verify this property using exact odd-cycle separation. Though being polynomial, the

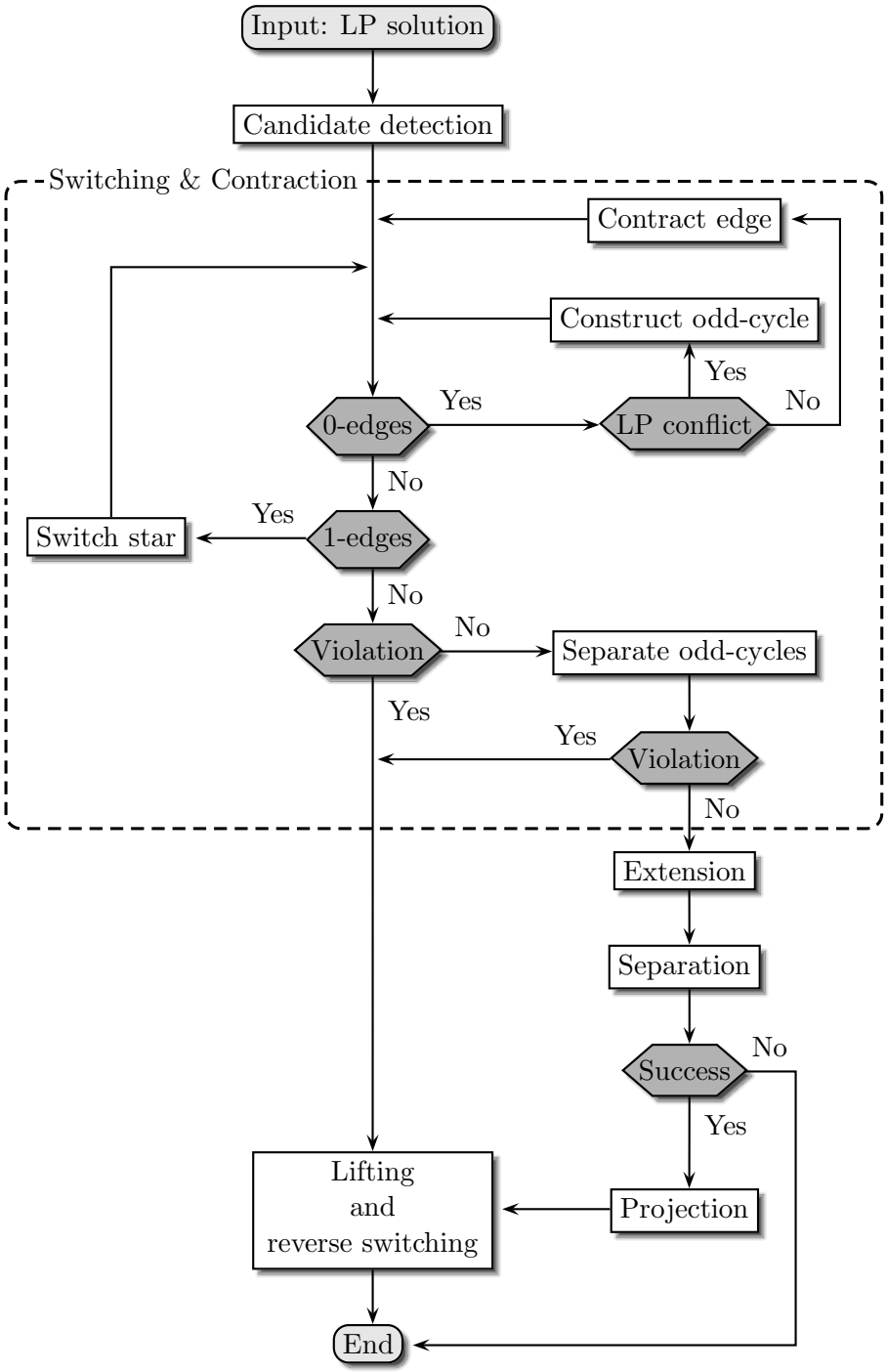


Figure 2.10: Workflow of the implemented shrink separation.

exact separation of odd-cycle inequalities is still computationally expensive. Therefore, we apply the following hierarchical approach: First, we perform a fast heuristic check. As we will see shortly, this can be done efficiently during the combined switching and contraction phase. If we detect any violated odd-cycle inequalities then we abort the shrink separation and simply return the inequalities found so far. Otherwise, there could still be violated odd-cycle inequalities that were missed by the heuristic. Thus, we have to apply exact odd-cycle separation but with one important difference: At this point, the LP solution has already been contracted, i. e., its dimension has been reduced. As a result, the exact check can now be performed much faster than in case of the original LP solution.

So, for the moment we simply accept the given fractional LP solution $\boldsymbol{z}$ as input regardless of whether or not it is an element of the semimetric polytope.

Candidate Detection

Our first task is to detect the candidate edges for contraction. These are precisely the edges with an LP value of either zero or one. However, the variables which store the LP values are of floating point type. As a consequence, we can only detect the candidates with a certain precision ε, where ε is a positive but sufficiently small real number. The actual choice of ε can have a significant impact on the lifting procedure later on. We will elaborate on this topic in Section 2.2.2.

Eventually we obtain a list of candidate edges for the contraction. To simplify matters, we will refer to the candidates with an LP value near zero and one as 0-edges and 1-edges, respectively.

Switching and Contraction

At this point we have the first major difference between theory and implementation. In the theory, we switch the LP solution alongside a specific cut K (see Lemma 2.5 on page 44 for more details). This results in a switched LP solution $\tilde{\boldsymbol{z}}$ whose 0-edges correspond exactly to the combined 0- and 1-edges of the original LP solution $\boldsymbol{z}$. In the implementation, however, we do not determine the actual cut K since this is computationally expensive. Instead, we successively contract single candidate edges.

Any 0-edge can be contracted directly. A 1-edge, on the other hand, needs to be switched first. To be more precise, we have to switch the partially contracted LP solution alongside the star induced by either of the 1-edge's ends. Yet, apart from transforming the 1-edge into a 0-edge, this will also affect the LP values of the remaining edges in the star. In particular, all the star's 0-edges will be transformed into 1-edges, which can increase the total number of necessary switching operations. This is even more of a problem considering that for each switching operation applied at this stage we will have to switch each of the obtained separating inequalities accordingly later on.

Evidently, the order of the contractions is crucial to this approach. Our procedure uses the following strategy: First, we contract all original 0-edges. Then, we select a 1-edge and switch the partially contracted LP solution alongside the star induced by either of the 1-edge's ends. Afterwards, we contract all 0-edges that emerged from this switching. We repeat the last two steps until all candidate edges have been processed.

Each time we contract a 0-edge $e = ht$, we also store the partition (H, T, B) of its neighborhood (cf. Figure 2.5 on page 48). This information is required for the lifting later

on. Note that contracting the edge ht merges each pair of edges (hv, tv) with $v \in B$ into a multiple edge. If the LP solution lies inside the semimetric polytope then the edges hv and tv have the same LP value according to Lemma 2.8 on page 47. In this case, the resulting multiple edge simply inherits this unique value. Otherwise, we can construct a violated odd-cycle inequality as explained in the proof of the lemma just mentioned. This is referred to as "LP conflict" in the flowchart in Figure 2.10. In particular, we see that the above combined switching and contraction can be used as an efficient heuristic to separate odd-cycle inequalities.

If we detect any violated odd-cycle inequalities, we cannot proceed with the shrink separation. This is because the remainder of the procedure only works correctly if the LP solution is an element of the semimetric polytope. The inequalities found so far are lifted, reverse-switched, and then returned as separating inequalities for the original LP solution $\boldsymbol{z}$. Otherwise, we eventually obtain a contracted LP solution $\overline{\boldsymbol{z}}$ that only contains fractional values. Still, it is not clear whether $\overline{\boldsymbol{z}}$ is an element of the semimetric polytope since the previous heuristic check may have missed some violated odd-cycle inequalities. Consequently, we have to apply exact odd-cycle separation. However, since we are now working on the contracted LP solution, the exact check can be performed much faster than for the original LP solution $\boldsymbol{z}$. Either we verify that $\overline{\boldsymbol{z}}$ is indeed part of the semimetric polytope or we find at least one violated odd-cycle inequality. In the latter case, we transform the separating inequalities as mentioned above and then terminate. Otherwise, we proceed with the extension of the contracted LP solution.

Extension

In the extension phase we introduce artificial LP values for all non-edges of the contracted graph $\overline{G}$. This allows us to apply separation techniques for dense and complete graphs that we couldn't have used otherwise. The artificial LP values are chosen in such a way that the extended LP solution $\overline{\boldsymbol{z}}'$ is an element of the semimetric polytope. To do so, we determine for each non-edge its feasible LP range via a series of shortest-path computations on a specific auxiliary graph. At the same time, we obtain the respective lower and upper inequalities for the non-edges as a byproduct. These inequalities will be crucial to the projection later on. Once we have computed the feasible LP ranges, there are two possible ways to introduce the artificial LP values. The first one is the static extension in which we choose for each non-edge a fixed element of its respective feasible LP range. In the second approach, the so-called adaptive extension, we leave the artificial LP values a priori undetermined, but we restrict them to either of the limits of their feasible LP range. For the details, we refer to Section 2.1.3.

Our procedure uses the adaptive extension approach for the following reasons: First and foremost, it preserves the violation of the separating inequalities during the projection. Furthermore, the limits of different feasible LP ranges are mutually independent (cf. Lemma 2.14 on page 57). This allows us to exploit an existing sparse structure of the auxiliary graph during the shortest-path computations. We use a version of Dijkstra's algorithm (see, for instance, [CLRS09]) for sparse graphs with a heap as priority queue and some additional adjustments to account for the specific structure of the auxiliary graph. We also experimented with a bidirectional search approach. Yet, the impact on the performance was negligible at best.

Once we have obtained the extended LP solution $\overline{\boldsymbol{z}}'$, we proceed with the separation.

Separation

We implemented separation procedures for bicycle-p-wheel-, clique-, and hypermetric inequalities as well as target cuts. For more information regarding these inequality classes we refer to Sections 1.4 and 0.4.5.

As a consequence of using adaptive extension, we now have to deal with the problem that separation procedures are generally unable to handle partially undetermined LP solutions. Bicycle-p-wheel- and clique inequalities are exceptions, though, since their nonzero left hand side coefficients all have the same sign. As a result, static and adaptive extension are equivalent. To be more precise, since the nonzero left hand side coefficients are all positive, we can simply fix each undetermined LP value to its respective lower limit and apply a standard separation procedure. In case of hypermetric inequalities and target cuts, on the other hand, we have to adjust the respective separation procedures to make them compatible with adaptive extension.

However, at least the target cuts can be separated directly on the contracted LP solution and thus the extension per se is not advisable in this case. We implemented a target cut separation procedure using a framework provided by Buchheim, Liers, and Oswald [BLO08]. As an oracle for the delayed column generation we used our own max-cut solver but with more generic parameter settings. In particular, we deactivated delayed column generation inside the oracle calls to prevent recursion.

If the separation succeeds, we obtain a set of violated inequalities. Yet, the left hand sides of these inequalities possibly contain nonzero coefficients that are related to non-edges. Thus, the next step is to project out the corresponding artificial variables.

Projection

For every nonzero coefficient related to a non-edge, we add the respective lower or upper inequality to each of the violated inequalities found by the separation. As mentioned earlier, the lower and upper inequalities are a byproduct of the computation of the feasible LP ranges. In addition, since the limits of the different ranges are mutually independent, the non-edges' variables can be projected out in an arbitrary order.

Eventually we get a set of projected violated inequalities in which all artificial coefficients are zero. By truncating these zero coefficients we obtain violated inequalities for the contracted LP solution. Moreover, our use of adaptive extension guarantees that the projection preserves the amount of violation of each inequality.

In a final step, we have to lift the separating inequalities for the contracted LP solution to make them applicable to the higher dimensional switched LP solution.

Lifting and Reverse Switching

We lift the inequalities via a series of node-splitting operations. In this context, our combined switching and contraction approach on single edges has the following consequences:

1) We have to perform the node-splitting operations in the exact reverse order of the corresponding edge-contractions.

2) Since the initial switching operations have been applied to the partially contracted LP solution, we cannot carry out the lifting and reverse switching sequentially. Instead, we have to switch the partially lifted inequality directly after each node-splitting of a former 1-edge.

Note that the necessary information on the partitions (H, T, B) of the neighborhoods of the contracted edges has already been gathered during the contraction.

Ultimately, we obtain a set of separating inequalities for the original LP solution $\boldsymbol{z}$ and the shrink separation terminates. However, preliminary testing of the above implementation showed that specific parts of it are sensitive to numerical errors. Thus, we now take a closer look at the numerical behavior of the shrink separation.

2.2.2 Numerical Behavior

In the previous section we have seen that some of the tasks in the course of the shrink separation can only be performed with a certain precision. One such example is the decision whether or not a given floating point value is integral. Yet, to what extend do these numerical errors affect the outcome of our computations? Of particular interest are the effects on the amount of violation during the projection, the lifting, and the reverse switching of a separating inequality.

First, we note that the violation is invariant under reverse switching. Furthermore, due to our use of adaptive extension, also the projection preserves the amount of violation. As a result, we can focus our analysis entirely on the lifting of the already projected inequalities. Here, we have the following two numerically critical operations to consider: The first one is to check whether a floating point value is integral. The second critical operation is to detect whether two floating point values are identical. We now discuss each of these tasks in greater detail.

Integrality of a Floating Point Value

We consider a floating point value to be integral if its absolute difference to an integer is at most the **integrality precision** ε, where $0 < \varepsilon \ll 1$ is a positive but sufficiently small real number. We check for integral values to detect 0- and 1-edges as candidates for the contraction.

Assume we are given a switched LP solution $\tilde{\boldsymbol{z}}$ with $0 \leq \tilde{z}_e < 1$, for all $e \in E$. In the theory, the candidate edges for contraction are those which have a switched LP value of zero. However, we can detect the integral floating point values only with a certain accuracy, namely the integrality precision ε. So, from a computational point of view, the candidate edges for contraction are in fact those which have a switched LP value of at most ε.

During the lifting of a separating inequality $(\overline{\boldsymbol{a}}, \overline{\alpha})$ we reinsert each contracted edge ht by splitting its corresponding supernode w. According to (2.5) on page 49, the edge ht gets a coefficient of $\hat{a}_{ht} := -\min\{\sum_{v \in H} |\overline{a}_{wv}|, \sum_{v \in T} |\overline{a}_{wv}|\}$. If the switched LP value of ht is indeed zero, as assumed by the theory, then this does not affect the violation of the inequality. Yet, we have just seen that the switched LP value of the edge ht can be up to ε. So, if both the coefficient $\hat{a}_{ht}$ and the value $\tilde{z}_{ht}$ are nonzero then the node-splitting alters the violation of the inequality. We will denote this **integrality error** by ρ_{int}.

Since we perform the lifting in an iterative manner, we introduce the following additional notations: We denote the intermediate states of the left hand side during lifting by $\hat{\boldsymbol{a}}$ and its final state by $\tilde{\boldsymbol{a}}$. Note that the transformation leaves the right hand side $\overline{\alpha}$ unchanged. Let $\{e_1, \ldots, e_m\}$ be the sequence of contracted edges and let (H_i, T_i, B_i) be the partition of the neighborhood of edge $e_i = h_i t_i$, $i = 1, \ldots, m$, as depicted in Figure 2.5 on page 48. Finally, we introduce $\tilde{\boldsymbol{z}}^*$ to denote the version of the switched LP solution $\tilde{\boldsymbol{z}}$

in which all the LP values of contracted edges are set to zero. This allows us to write the initial violation $\overline{\boldsymbol{a}}^T\overline{\boldsymbol{z}} - \overline{\alpha}$ in the form $\tilde{\boldsymbol{a}}^T\tilde{\boldsymbol{z}}^* - \overline{\alpha}$. To simplify matters, we assume without loss of generality that $\sum_{v\in T_i} |\hat{a}_{w_i v}| \leq \sum_{v\in H_i} |\hat{a}_{w_i v}|$ for all $i = 1, \ldots, m$. Then, we get an absolute error of

$$\rho_{int} = |\tilde{\boldsymbol{a}}^T(\tilde{\boldsymbol{z}} - \tilde{\boldsymbol{z}}^*)| = \Big|\sum_{i=1}^{m} \tilde{z}_{e_i} \sum_{v\in T_i} |\hat{a}_{w_i v}|\Big| \leq \varepsilon \sum_{i=1}^{m} \sum_{v\in T_i} |\hat{a}_{w_i v}|.$$

The following example describes an ill-conditioned scenario for the integrality error.

Example 2.15. *Consider a graph as depicted in Figure 2.11. It is the union of* $2l$ *disjoint*

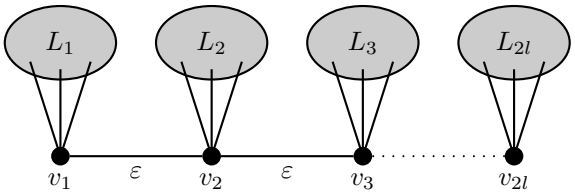


Figure 2.11: An ill-conditioned example for the integrality error ρ_{int}.

stars. Each star has a center node v_i *and a set* L_i *of leaves that are exclusively adjacent to* v_i*. Moreover, the number of leaves* $|L_i| =: c$ *is constant for all* $i = 1, \ldots, 2l$*. Finally, the pairs of consecutive center nodes are joined by the edges* $v_i v_{i+1}$*,* $i = 1, \ldots, 2l-1$*, each with an LP value of* ε*. By sequentially contracting the edges* $v_i v_{i+1}$*, we obtain the following supernodes:*

$$\begin{aligned} w_0 &= v_1, \\ w_i &= \{w_{i-1}, v_{i+1}\}, \text{ for } i = 1, \ldots, 2l-1. \end{aligned}$$

When contracting the edge $e_i = w_{i-1}v_{i+1}$*,* $i = 1, \ldots, 2l-2$*, the partition* (H_i, T_i, B_i) *of its neighborhood looks as follows:*

$$\begin{aligned} h_i &= w_{i-1}, & t_i &= v_{i+1}, \\ H_i &= \uplus_{j=1}^{i} L_j, & T_i &= L_{i+1} \uplus v_{i+2}, & B_i &= \emptyset. \end{aligned}$$

In case of the last edge e_{2l-1}*, the partition is analogous except that the set* T_{2l-1} *is simply* L_{2l}*. Eventually we obtain the contracted graph, which is a star with the center node* w_{2l-1} *and the set of leaves* $\uplus_{j=1}^{2l} L_j$*.*

Now, consider an already projected violated inequality $(\overline{\boldsymbol{a}}, \overline{\alpha})$*. To simplify matters, we assume that all left hand side coefficients have the value 1. When splitting the supernode* w_i*, we reinsert the edge* $w_{i-1}v_{i+1}$ *with the following coefficient:*

$$-\min\Big\{\sum_{u\in H_i} |\hat{a}_{w_i u}|, \sum_{u\in T_i} |\hat{a}_{w_i u}|\Big\} = -\min\{ic, (2l-i)c\}.$$

Consequently, we set the coefficients of the edges $v_i v_{i+1}$ *with* $i = 1, \ldots, l$ *to* $-ic$*. The remaining edges with* $i = l+1, \ldots, 2l-1$ *get the coefficients* $-(2l-i)c$*. This results in the following absolute error:*

$$\rho_{int} = \varepsilon c \cdot \Big(l + 2\sum_{i=1}^{l-1} i\Big) = \varepsilon c l^2.$$

Obviously, the absolute integrality error grows arbitrarily large for increasing values of l and c, respectively. However, if we fix the order n of the graph then we can give an upper bound on the error. In this example we have $n = 2l(c+1)$. By solving this term for l and inserting it into the formula of the absolute integrality error, we obtain:

$$\rho_{int}(c) = \tfrac{\varepsilon n^2}{4} \cdot \tfrac{c}{(c+1)^2}.$$

It is easy to show that the above term is maximal if c equals one. Hence, the maximum absolute integrality error for fixed order n is $\varepsilon n^2/16$.

Identity of two Floating Point Values

We consider two floating point values to be identical if their absolute difference is at most the **identity precision** σ, where $0 < \sigma \ll 1$ is a positive but sufficiently small real number. We check the identity of a given pair of values during the contraction of 0-edges.

Suppose we contract an edge ht. This merges the pair of edges hv and tv for each node v that is adjacent to both h and t. According to Lemma 2.8 on page 47, the LP values of hv and tv are identical if the switched LP solution $\tilde{\boldsymbol{z}}$ is an element of the semimetric polytope. Otherwise, the values may differ, in which case we can construct a violated odd-cycle inequality. Therefore, the identity precision marks the threshold above which we actually recognize a violated odd-cycle inequality as such. Moreover, the choice of σ affects the lifting of inequalities as we will see shortly.

Let us assume that the LP values of the edges hv and tv differ by at most σ. Then, we can merge them into a multiple edge without any problems. However, if the two LP values are indeed different then we do not have a uniquely defined LP value for the resulting multiple edge. In this case, we set the LP value of the multiple edge to the arithmetic mean of the LP values of all the edges merged into it.

When splitting the supernode $\{h, t\}$ later on, we separate the merged edges hv and tv again. Here, only one of the two edges inherits the coefficient of the former multiple edge whereas the coefficient of the remaining edge is set to zero. Without loss of generality, we assume tv to be the inheriting edge. However, the coefficient of tv has been determined with respect to the LP value of the multiple edge. So, if the LP values of tv and the multiple edge are different then the lifting process alters the amount of violation of the given inequality. We will denote this **identity error** by ρ_{id}.

Now, consider a contracted graph $\overline{G}$. Let $\mathcal{M}$ be its set of multiple edges. For a given multiple edge p, let $\iota(p)$ denote the one edge merged into p that eventually inherits the coefficient of p after the lifting. Finally, we introduce $\tilde{\boldsymbol{z}}^*$ to denote the version of the switched LP solution $\tilde{\boldsymbol{z}}$ in which the LP values of all the edges merged into a multiple edge p are set to $\overline{z}_p$, i. e., the value of p in the contracted LP solution. This allows us to write the initial violation $\overline{\boldsymbol{a}}^T\overline{\boldsymbol{z}} - \overline{\alpha}$ in the form $\tilde{\boldsymbol{a}}^T\tilde{\boldsymbol{z}}^* - \overline{\alpha}$. We thus obtain the following absolute error:

$$\rho_{id} = |\tilde{\boldsymbol{a}}^T(\tilde{\boldsymbol{z}} - \tilde{\boldsymbol{z}}^*)| = \Big|\sum_{p\in\mathcal{M}} \tilde{a}_{\iota(p)}(\tilde{z}_{\iota(p)} - \tilde{z}^*_{\iota(p)})\Big|.$$

Note that in the scope of the max-cut problem, the LP values are restricted to the range $[0, 1]$. This is in particular true for any of the multiple edges. Thus, the above difference $(\tilde{z}_{\iota(p)} - \tilde{z}^*_{\iota(p)})$ has an absolute value of at most 1. The following example discusses an ill-conditioned scenario in which this maximum LP discrepancy is indeed attained.

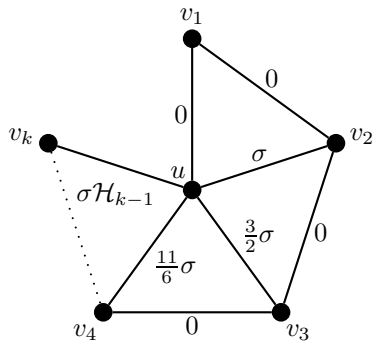


Figure 2.12: An ill-conditioned example for the identity error ρ_{id}.

Example 2.16. *Consider a fan-shaped graph as depicted in Figure 2.12. The edges v_iv_{i+1}, for $i = 1, \dots, k-1$, have an LP value of zero and are contracted sequentially. We assume that the LP value of a multiple edge is set to the arithmetic mean of the LP values of the edges that were merged into it. Moreover, we choose the LP values of the edges uv_i in the following way:*

$$\tilde{z}_{uv_1} = 0, \qquad \tilde{z}_{uv_{i+1}} = \sigma + \tfrac{1}{i}\sum_{j=1}^{i} \tilde{z}_{uv_j}, \text{ for } i = 1, \dots, k-1. \tag{2.16}$$

In other words, the LP value of the next edge to be merged into the multiple edge always exceeds the current LP value of the latter by the identity precision σ. To derive the corresponding recursion formula, we consider the following difference:

$$\tilde{z}_{uv_{i+1}} - \tilde{z}_{uv_i} = \tfrac{1}{i}\sum_{j=1}^{i} \tilde{z}_{uv_j} - \tfrac{1}{i-1}\sum_{j=1}^{i-1} \tilde{z}_{uv_j}.$$

By factoring out $\frac{1}{i}$, we obtain

$$\tilde{z}_{uv_{i+1}} - \tilde{z}_{uv_i} = \tfrac{1}{i}\Big(\sum_{j=1}^{i} \tilde{z}_{uv_j} - \tfrac{i}{i-1}\sum_{j=1}^{i-1} \tilde{z}_{uv_j}\Big).$$

As $\frac{i}{i-1} = 1 + \frac{1}{i-1}$, we can simplify the above term as follows:

$$\tilde{z}_{uv_{i+1}} - \tilde{z}_{uv_i} = \tfrac{1}{i}\Big(\tilde{z}_{uv_i} - \tfrac{1}{i-1}\sum_{j=1}^{i-1} \tilde{z}_{uv_j}\Big) \overset{(2.16)}{=} \tfrac{1}{i}\sigma.$$

Thus, we get the following recursion formula:

$$\tilde{z}_{uv_1} = 0, \qquad \tilde{z}_{uv_{i+1}} = \tilde{z}_{uv_i} + \tfrac{\sigma}{i}, \text{ for } i = 1, \dots, k-1.$$

*After the contraction of the edges v_iv_{i+1}, $i = 1, \dots, k-1$, we obtain a single multiple edge p with an LP value of $\sigma(\mathcal{H}_k - 1)$. Here, $\mathcal{H}_k$ denotes the **k-th harmonic number** which is the k-th partial sum of the diverging harmonic series $\sum_{j=1}^{\infty} \frac{1}{j}$. Note that $\mathcal{H}_k \approx \ln(k) + \gamma$, where γ denotes the **Euler-Mascheroni constant** with an approximate value of* 0.5772. *Let us assume that the edge uv_1, which has an LP value of zero, is the inheriting edge $\iota(p)$*

of p. This means that in order to attain an LP discrepancy of $|\tilde{z}_{\iota(p)} - \tilde{z}^*_{\iota(p)}| \approx \tau$, $\tau \in [0,1]$, we need to merge $k \approx \lceil \exp(\frac{\tau}{\sigma} + 1 - \gamma) \rceil$ edges. In other words, to increase the LP discrepancy by one order of magnitude, we have to raise the number of merged edges to the tenth power.

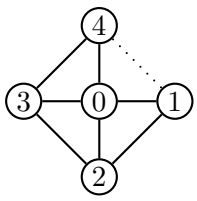
Note that the above example is purely academic. For instance, assuming an identity precision of 0.01—which is far too large for practical use—we would need to merge more than $7 \cdot 10^{43}$ edges in order to attain the maximum LP discrepancy of $\tau = 1$ on a single multiple edge. Simply storing a graph of this size would already require approximately 10^{33} terabytes of disk space.

Further Problems

Another possible problem is related to the so-called **violation threshold** ε_{vio}. This parameter is common in branch-and-cut solvers. The idea is to only add a separating inequality to the constraint pool if it is violated by the current LP solution by at least ε_{vio}. Aside from numerical considerations, this is mainly due to the empirical result that a marginally violated inequality is unlikely to substantially improve subsequent iterations of the LP solver.

However, if we choose the value of ε_{vio} too large then we may encounter empty feasible LP ranges for some of the non-edges.

Example 2.17. *Consider the contracted LP solution $\overline{\boldsymbol{z}}$ and its underlying graph $\overline{G}$ as specified in Figure 2.13. We obtain the following limits on the feasible LP range of the*



$\overline{z}_{01} = 0.179656$
$\overline{z}_{02} = 0.389984$
$\overline{z}_{03} = 0.272336$
$\overline{z}_{04} = 0.007065$
$\overline{z}_{12} = 0.209948$
$\overline{z}_{23} = 0.66232$
$\overline{z}_{34} = 0.265271$

Figure 2.13: A contracted LP solution and its underlying graph.

non-edge from 1 to 4:

$$\begin{aligned} L &:= \max\{\, \overline{\boldsymbol{z}}(F) - \overline{\boldsymbol{z}}(P \setminus F) - |F| + 1 \mid P \text{ } (1,4)\text{-path of } \overline{G},\, F \subseteq P,\, |F| \text{ odd}\} \\ &= \overline{z}_{23} - \overline{z}_{12} - \overline{z}_{34} = 0.187101, \\ U &:= \min\{\, -\overline{\boldsymbol{z}}(F) + \overline{\boldsymbol{z}}(P \setminus F) + |F| \mid P \text{ } (1,4)\text{-path of } \overline{G},\, F \subseteq P,\, |F| \text{ even}\} \\ &= \overline{z}_{01} + \overline{z}_{04} = 0.186721. \end{aligned}$$

Note that we have $U = L - 3.8 \cdot 10^{-4}$. In other words, the upper limit is less than the lower one and thus the feasible LP range is empty. As a consequence, there has to be a violated odd-cycle inequality. In this example, the contracted LP solution violates the inequality $\overline{z}_{02} - \overline{z}_{01} - \overline{z}_{12} \leq 0$ by precisely $3.8 \cdot 10^{-4}$. However, if we chose a violation threshold of, for example, 10^{-3} then an exact odd-cycle separation would not report the above inequality, thus leading us to believe that $\overline{\boldsymbol{z}}$ is an element of the semimetric polytope.

Consequences in Practice

The complexity of the ill-conditioned scenarios in Examples 2.15 and 2.16 is due to their specific combination of graph structure and contraction order, which is rather unlikely to be encountered in practice. Nonetheless, these examples demonstrate the need to pay attention to both the integrality- and the identity precision. We combined these two parameters into a single one denoted by ε_{int}. Preliminary computational experiments, in which we chose ε_{int} in the same order of magnitude as the violation threshold ε_{vio}, indeed showed cases of violation reduction during lifting. Thus, we revisited Example 2.15 to find a better setting for ε_{int}. In the example, the maximum absolute integrality error for fixed order n is $\varepsilon n^2/16$. Consider a separating inequality with violation at least ε_{vio}. Then, we need an integrality precision below $16\varepsilon_{vio}/n^2$ to assure that the lifted inequality is still violated. For instance, given a violation threshold $\varepsilon_{vio} = 10^{-3}$ and an order $n = 100$, we need ε to be less than $1.6 \cdot 10^{-6}$. After further experiments we finally chose the value of ε_{int} approximately three orders of magnitude below the violation threshold ε_{vio}. This proved to be sufficient throughout the course of our computations. In future work one could also determine ε_{int} adaptively with respect to the given problem instance.

Finally, we addressed the problem related to the violation threshold (cf. Example 2.17) by simply omitting the non-edges with an empty feasible LP range. As a result, the contracted and extended graph may not be complete. However, except for the separation of hypermetric inequalities, none of the involved separation procedures explicitly requires a complete underlying graph. Therefore, omitting the non-edges with empty feasible LP ranges is no major limitation.

Chapter 3

Computational Results

In the following we describe the computational experiments that we carried out to test the shrink separation presented in Chapter 2. We start with an introduction of the considered problem instances in Section 3.1. In Section 3.2 we look at the computational setup, including the hard- and software used, chosen parameters, and tested separation scenarios. Section 3.3 compares the performance of the examined scenarios for the different classes of test instances. Finally, in Section 3.4 we take a closer look at a particular set of test instances generated from real-world data, namely the Mannino instances.

3.1 Test Instances

As a basis for our computational experiments we took several classes of test instances for both the max-cut problem and unconstrained quadratic 0/1 optimization. Below, we provide the details on how these instances were generated and where they can be obtained. In addition, we point out the respective tables in Appendix A with exhaustive data on the individual test problems such as sizes and densities of the underlying graphs, known optimum values et cetera.

3.1.1 Ising Spin Glass Problems

In Section 1.2.1 we briefly introduced spin glasses and the corresponding Ising model. For our computational experiments we considered two- and three-dimensional Ising spin glass problems with nearest neighbor interactions, periodic boundary conditions, and without external magnetic field. The respective underlying interaction graphs are toroidal grids whose general structure is as follows: A two-dimensional toroidal grid is a square grid with additional edges that join the outermost nodes of each row and column, respectively. Consequently, a toroidal $(k \times k)$ grid graph has $n = k^2$ nodes and $2n$ edges. An exemplary toroidal (3×3) grid is depicted in Figure 3.1. Similarly, a three-dimensional toroidal grid is a cubic grid with additional edges such that each horizontal and vertical layer, respectively, is a two-dimensional toroidal grid. As a result, a toroidal $(k \times k \times k)$ grid graph has $n = k^3$ nodes and $3n$ edges.

Any two-dimensional toroidal grid graph has genus 1, which means that it is embeddable into the torus S_1. Galluccio and Loebl [GL99] showed that the max-cut problem is polynomial on graphs with bounded genus if there is only a bounded number of different edge weights, or if the edge weights are integers bounded in absolute value by a polynomial of the order of the graph. As a consequence, the max-cut problem is

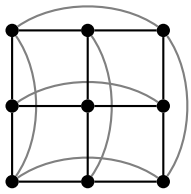


Figure 3.1: A toroidal (3×3) grid.

polynomial on two-dimensional toroidal grids whose edge weights satisfy the restrictions just mentioned. However, the result of Galluccio and Loebl cannot be applied to three-dimensional toroidal grids since their genus increases with the number of grid points, as was shown by Regge and Zecchina [RZ00]. In fact, Barahona [Bar82] proved that the max-cut problem on three-dimensional grid graphs is **NP**-hard.

We used a custom generator to produce toroidal grids with two types of random edge weights. Either the weights have uniformly distributed ± 1 values or Gaussian distributed integral values. Moreover, to allow the efficient enumeration of the 4-cycles in the toroidal grid, the nodes and edges are numbered consecutively in a specific way. Below, when speaking about the layers of a grid we always refer to its *horizontal* layers.

We number the nodes of a given two-dimensional toroidal grid by traversing each of the layers from left to right, starting at the topmost one. The numbering of the edges, on the other hand, is done in two steps. First, we number the **intralayer edges** which have both ends in the same layer. To do so, we process the layers in order from top to bottom. Within each layer we number the edges from left to right, the last edge being the one that joins the outermost nodes. In the second step we number the **interlayer edges** which have ends in different layers. Here, we consider the pairs of adjacent layers in order from top to bottom and number the edges in the respective interlayers from left to right. The final interlayer in this process is the one that connects the bottommost to the topmost layer. An exemplary numbering of the nodes and edges in a toroidal (3×3) grid is depicted in Figure 3.2.

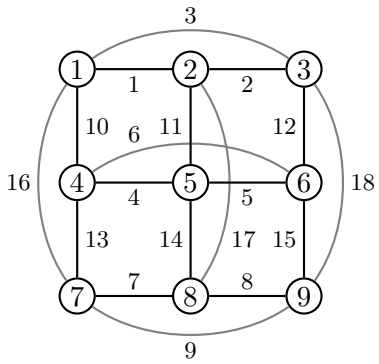


Figure 3.2: Numbering of the nodes and edges in a toroidal (3×3) grid.

In a three-dimensional toroidal grid, each layer in itself is a two-dimensional toroidal grid. As before, the term 'layer' is used synonymously for 'horizontal layer'. We number the nodes by processing the layers in order from top to bottom, handling each layer as described above. For the numbering of the edges we process layers and interlayers in an alternating fashion. We start with the topmost layer and number its intra-edges using the

rule for two-dimensional toroidal grids. Afterwards, we number the interlayer edges that join the nodes in the topmost layer to those in the subjacent one. To do so, we traverse the edges in ascending order of the indices of their respective ends inside the upper layer. Then, we proceed with the intra-edges of the subjacent layer and so forth. As in the two-dimensional case, the final interlayer to be processed is the one that connects the bottommost to the topmost layer.

We now describe how we generated the two different types of random edge weights.

Uniformly Distributed ±1 Weights

We initialized the first and the second half of the edge weights with -1 and $+1$, respectively. Then, we computed a random permutation of the edge weights by using the so-called **Knuth shuffle**. This algorithm guarantees that every permutation is equally likely. Its underlying method was first described in 1938 by Fisher and Yates [FY38]. A modern version, designed for computer use, was introduced by Durstenfeld [Dur64] in 1964 and was later popularized by Knuth [Knu97].

We considered toroidal $(k \times k)$ grids with $k = 30, 35, \ldots, 85$ and three-dimensional toroidal grids with $k = 5, \ldots, 8$. For each grid size we generated ten random instances. The respective optimum values or lower/upper bounds are provided in Table A.8 on page 101 for the two-dimensional grids and in Table A.11 on page 104 for the three-dimensional grids.

The Instance 'toruspm-8-50'. In addition to our self-generated ±1-instances, we looked at a toroidal $(8 \times 8 \times 8)$ grid instance named 'toruspm-8-50' from the "DIMACS library of mixed semidefinite-quadratic-linear programs" [PS00]. The challenge in this case was to find the optimum value at all; the DIMACS library itself only provides an upper bound of 527.808663 computed by a primal-dual interior point code. Therefore, this instance is noncompetitive with regard to our remaining computational experiments and is not considered in any performance evaluations. Krishnan and Mitchell [KM06] used 'toruspm-8-50' to test their polyhedral cut and price approach for the max-cut problem. In the corresponding article the authors state an optimum value of 456. However, we were able to compute a provably optimal solution with an objective value of 458. The smaller of the shores is given below. The numbering of the nodes starts at 1. To simplify matters, we specify sequences of more than two consecutive node indices as ranges.

{4–6, 13, 16, 19, 20, 23, 26, 30, 33, 36–38, 40–43, 52, 53, 58, 60, 61, 65–67, 69, 71–73, 75–78, 81–85, 87, 90, 93, 97, 98, 102–105, 109–116, 118, 119, 124, 126–128, 130–132, 137, 139–142, 148, 150, 153, 154, 156, 157, 159, 160, 163, 164, 167, 169, 171, 174, 175, 178–180, 185, 191, 192, 197, 199, 200, 203, 204, 206, 207, 209–213, 215, 217, 219–222, 224–226, 228, 229, 233, 234, 237, 238, 240, 243, 245, 247, 250, 253–257, 259, 266, 270, 271, 273, 278, 286, 288, 289, 291–293, 295–298, 300, 301, 303–306, 308, 310, 312, 313, 316, 319, 323, 331, 334–336, 339, 341, 342, 346, 351, 354–356, 362–367, 369–373, 378, 380, 381, 384, 386, 387, 392–394, 397, 399, 400, 402, 404, 406, 408, 409, 413, 414, 417, 420, 421, 425, 428, 430, 431, 433, 434, 436–438, 441, 442, 445, 446, 448, 452, 454, 456, 459, 461, 462, 464, 466, 469, 472, 474, 476, 477, 480, 482, 485, 488, 490, 495–498, 500–502, 504–507, 509, 512}.

Gaussian Distributed Integral Weights

We initialized each edge weight with the value of a standard normal random variable. These initial values were then multiplied by a scaling factor of 10^5 and typecast to an integer. After that, we applied the Knuth shuffle to produce a random permutation of the edge weights.

We generated the standard normal random variables using the **Marsaglia polar method**. It transforms a pair $\boldsymbol{u} = (u_1, u_2)$ of independent uniformly distributed random variables in the range $[0, 1]$ into a pair $\boldsymbol{s} = (s_1, s_2)$ of independent standard normal random variables. The polar method traces back to an algorithm by Box and Muller [BM58] which uses Euclidean coordinates instead of polar ones. However, the use of polar coordinates avoids the costly evaluation of trigonometric functions.

We now briefly outline the polar method. It starts with a pair $\boldsymbol{u}$ of independent uniformly distributed random variables in the range $[0, 1]$. First, it computes $\boldsymbol{v} = 2\boldsymbol{u} - \mathbf{1}$. This results in a pair $\boldsymbol{v} = (v_1, v_2)$ of independent uniformly distributed random variables in the range $[-1, 1]$. However, in order for the method to work, the point $\boldsymbol{v}$ has to lie in the interior of the unit circle. In other words, the value $q = v_1^2 + v_2^2$ has to be less than 1. Also, q has to be nonzero, as we will see shortly. If $\boldsymbol{v}$ does not meet the requirements just mentioned then the method starts over with a new pair $\boldsymbol{u}$. Otherwise, it continues by computing the following scalar:

$$p = \sqrt{\frac{-2\ln(q)}{q}}.$$

Note that p is not defined for q equal to zero. Finally, the desired pair $\boldsymbol{s}$ of independent standard normal random variables is obtained as $\boldsymbol{s} = p\boldsymbol{v}$.

We considered toroidal $(k \times k)$ grids with $k = 30, 35, \ldots, 185$ as well as three-dimensional toroidal grids with $k = 5, \ldots, 11$. For each grid size we generated ten random instances. The respective optimum values or lower/upper bounds are provided in Table A.9 on page 102 and Table A.10 on page 103 for the two-dimensional grids as well as in Table A.12 on page 104 for the three-dimensional grids.

3.1.2 Biq Mac Library

In addition to the Ising spin glass problems we used test instances from the Biq Mac Library [Wie07]. As stated on the corresponding website, this library is a "collection of Max-Cut instances and quadratic 0-1 programming problems of medium size. Most of the instances were collected while developing Biq Mac, an SDP based Branch-and-Bound code (see, e. g., [Wie06]). The dimension of the problems [...] ranges from 20 to 500." We now provide detailed information on the problem classes that we considered for our computational experiments.

Quadratic 0/1 Optimization Problems

As discussed in Section 1.1.2, the unconstrained quadratic 0/1 optimization problem is equivalent to the max-cut problem. The Biq Mac Library features three different classes of quadratic 0/1 optimization problem instances.

Beasley Instances. These data sets are due to Beasley [Bea98]. All problems have a density of 0.1 and integer uniform coefficients in the range $[-100, 100]$. The respective

sizes, optimum values or lower/upper bounds of the data sets are given in Table A.3 on page 98.

Billionnet and Elloumi Instances. Billionnet and Elloumi [BE07] used the generator introduced in [PR90] to generate instances of size $n = 100, 120, 150, 200$ and varying density $d = 0.3, 0.8, 1.0$. For each class of problems, ten instances have been generated, one for each seed in $1, \ldots, 10$. The coefficients of the instances are integral uniform values in the range $[-100, 100]$ for the diagonal entries and in the range $[-50, 50]$ for the off-diagonal ones. In the Biq Mac Library, these instances have been extended by an additional ten instances of size 250 with density 0.1. The respective sizes, densities, and optimum values of the different instances can be found in Table A.1 on page 97.

Glover, Kochenberger, and Alidaee Instances. These data sets are due to Glover, Kochenberger, and Alidaee [GKA98]. Table 3.1 lists the different settings that were used for the coefficient elements. The respective sizes, densities, and optimum values or lower/upper bounds of the data sets are given in Table A.4 on page 99.

	Integer uniform range	
Set	Diagonal	Off-diagonal
a	$[-100, 100]$	$[-100, 100]$
b	$[-63, 0]$	$[0, 100]$
c, e	$[-100, 100]$	$[-50, 50]$
d, f	$[-75, 75]$	$[-50, 50]$

Table 3.1: Coefficient ranges of the Glover, Kochenberger, and Alidaee instances.

Max-Cut Problem Instances Generated With `rudy`

The Biq Mac Library contains several classes of max-cut problem instances whose underlying graphs have been generated using `rudy` [Rin98]. For each category, ten instances have been generated which are numbered consecutively with $i = 0, \ldots, 9$.

- $G_{0.5}$ (`g05_`n`.`i)
  Unweighted graphs with edge probability 1/2 and order $n = 60, 80, 100$.
- $G_{-1/0/1}$ (`pm1s_`n`.`i`, pm1d_`n`.`i)
  Weighted graphs of order $n = 80, 100$ and a density of either 0.1 (`pm1s`) or 0.99 (`pm1d`). The edge weights were chosen uniformly from $\{-1, 0, 1\}$.
- $G_{[-10,10]}$ (`w`d`_`n`.`i)
  Graphs of order 100 and varying density $d = 0.1, 0.5, 0.9$. The integral edge weights were chosen from the range $[-10, 10]$.
- $G_{[0,10]}$ (`pw`d`_`n`.`i)
  Graphs of order 100 and varying density $d = 0.1, 0.5, 0.9$. The integral edge weights were chosen from the range $[0, 10]$.

The respective optimum values of the instances are listed in Table A.5 on page 99 for the $G_{0.5}$ graphs, in Table A.6 on page 100 for the $G_{-1/0/1}$ graphs, and in Table A.7 on page 100 for the $G_{[-10,10]}$ and $G_{[0,10]}$ graphs.

3.1.3 Mannino Instances

The Mannino instances were provided by Rinaldi [Rin10]. They were generated from real-world data on radio frequency interferences between major Italian cities in the context of a frequency allocation problem. Table A.2 on page 98 provides the details on the instances such as respective sizes, densities, ranges of weight values, and optimum values.

3.2 Computational Setup

We implemented the algorithm in C++ and embedded it in the branch-and-cut framework ABACUS ("A Branch-And-CUt System", see [EGJR01, JT98, JT00, Thi95]). The computational experiments were carried out on an Intel® Xeon® E5450 processor with 2.5 GHz clock rate, 2×6MB shared L2-Cache and 8GB RAM. The operating system was Debian GNU/Linux 4.0. We used ABACUS 2.3 in conjunction with the LP solver CPLEX 8.1 [Ilo02].

In addition to the new shrink separation, our branch-and-cut solver comprised four separation procedures for odd-cycle inequalities, namely GEN3CYC, GEN4CYC, SHOC, and OC. The latter three are almost identical to the procedures of the same name described in [BGJR88], except for some minor modifications to meet the specifications of the ABACUS framework. Yet, we only elaborate on GEN4CYC and OC since we did not apply the remaining procedures in our experiments. Also, to simplify the $\mathcal{O}$-notation, we will use V and E instead of $|V|$ and $|E|$, respectively, where necessary.

GEN4CYC is a heuristic for the separation of 4-cycle inequalities. It is specifically designed for toroidal grid graphs, in which the chordless 4-cycles correspond precisely to the grid squares. The algorithm requires a certain numbering of the nodes and edges, which is described in Section 3.1.1. By exploiting this structure, we can enumerate the chordless 4-cycles in time $\mathcal{O}(V)$.

OC is an implementation of the exact odd-cycle separation as proposed by Barahona and Mahjoub [BM86]. It uses an auxiliary graph consisting of two copies of the original graph that are joined in a specific way. A detailed description can be found in the first paragraph of the proof of Lemma 2.14 on page 57. In essence, a cycle in the original graph corresponds to a shortest path in the auxiliary graph that links a node in the one graph copy with its counterpart in the other copy. As a result, the odd-cycle separation reduces to a shortest-path problem on the auxiliary graph. Assuming that the shortest-path computation requires $\mathcal{O}(V^2)$ calculations, Barahona and Mahjoub stated a time complexity of $\mathcal{O}(V^3)$ for their algorithm. Our implementation uses Dijkstra's algorithm (see, for instance, [CLRS09]) with heaps for the shortest-path computation, which has a time complexity of $\mathcal{O}(E \log V)$. Thus, OC runs in time $\mathcal{O}(VE \log V)$.

As a primal heuristic we used a combined rounding and improvement approach. We start with the spanning tree rounding heuristic described in Algorithm 1.2 on page 28 to obtain a feasible solution in the vicinity of the current fractional LP solution. Then, we try to improve upon this solution by applying the Kernighan-Lin heuristic given in Algorithm 1.3 on page 29. However, we did not invoke the primal heuristic for every LP solution in the course of the optimization; instead, we let it pass through alternating active and idle phases. In an active phase, the heuristic is called for every new LP solution until it failed five consecutive times to improve the value of the best known solution. Then, it enters an idle phase of a certain length. This idle length is initially set to one hundred, which means that the heuristic ignores the next one hundred LP

solutions before switching to an active phase again. Moreover, the idle length doubles each time the preceding active phase failed at each of its improvement attempts. Yet, if at least one of the attempts was successful then the idle length is reset to the initial value of one hundred.

We proceed with the details on how we configured the ABACUS framework itself. The list of settings that we used throughout the experiments is provided in Table 3.2. Note that '<auto-detect>' is not an actual setting for the option 'Objective function values all integers'. It means that we check for integral coefficients while reading a given input file and then adjust the setting accordingly at runtime.

Enumeration strategy	: BestFirst
Branching strategy	: CloseHalfExpensive
Tested candidates for branching variables	: 9
Maximal enumeration level	: 99999
CPU time limit	: 10:00:00
Wall-clock time limit	: 99999:59:59
Objective function values all integers	: <auto-detect>
Tailing-off control	: No
Delayed branching threshold	: 1
Minimal number of rounds a subproblem stays dormant	: 1
Primal bound initialization	: None
Frequency of additional pricing	: 0 LPs
Cutting skip factor	: 1
Skipping mode	: SkipByNode
Fix/set by reduced costs	: Yes
Maximal number of constraints added per iteration	: 10000
Maximal number of constraints buffered per iteration	: 100000
Maximal number of variables added per iteration	: 100000
Maximal number of variables buffered per iteration	: 1000000
Maximal number of iterations in cutting plane phase	: -1
Elimination of fixed and set variables	: No
Reoptimization after a root change	: No
Show average distance of added cuts	: No
Elimination of constraints	: NonBinding
Elimination of variables	: ReducedCost
Tolerance for constraint elimination	: 0.001
Tolerance for variable elimination	: 0.001
Age for constraint elimination	: 1
Age for variable elimination	: 1
Default LP-solver	: Cplex
Machine accuracy	: 0.0001

Table 3.2: List of ABACUS settings used for the computational experiments.

We compared the following four separation scenarios:

(i) **CYC** exclusively uses OC. The only exception are instances on toroidal grid graphs. Here, we call GEN4CYC first and only invoke OC if no violated 4-cycle inequalities could be found.

(ii) **CON** uses the graph contraction as a heuristic to detect violated odd-cycle inequalities. If none are found then we perform an exact odd-cycle separation on the successfully contracted LP solution.

(iii) **CLQ** is initially identical to CON. However, if the contracted LP solution does not

violate any odd-cycle inequalities then we extend it and separate cliques of order five and seven, respectively.

(iv) **TC** only differs from CLQ in that it separates target cuts on the contracted LP solution instead of clique inequalities on the extended one. For the Mannino instances and those on toroidal grid graphs, respectively, the target cut separation uses three hundred subgraphs of order ten of the contracted graph. For the remaining instances it uses six hundred subgraphs of order eight. Each subgraph is initialized with a random node and then expanded by successively adding nodes in a greedy manner. For more information, see the section about target cut separation on page 63.

Note that we deliberately omitted bicycle-p-wheel- and hypermetric inequalities due to poor results in preliminary computational experiments. The separation of bicycle-p-wheel inequalities was simply too time-consuming. Moreover, it exclusively detected violated bicycle-3-wheel inequalities. These are, however, identical to K_5-inequalities and are thus also covered by the CLQ scenario. Similarly, the separation of hypermetric inequalities took too much time considering its overall poor success rate.

In a given iteration of the separation phase, we allowed each active separation procedure to add up to six hundred cutting planes to an individual preliminary constraint pool. If the size of such a pool exceeded three hundred, we first sorted its elements in descending order of their ranks. The rank of a cutting plane increases with the similarity of its slope to that of the problem's iso-value planes. After that, we discarded all but the best three hundred elements. Finally, the remaining contents of the preliminary pools were added to the overall constraint pool of the branch-and-cut framework.

We tested each of the four separation scenarios with the instances introduced in Section 3.1. The CPU time to be spent per instance was limited to ten hours. Finally, note that we facilitated the solution process of the Ising spin glass problems with ± 1 weights by perturbing the edge weights prior to the optimization. This was necessary because the ± 1-instances with their limited diversity of edge weights are considerably harder to solve than Gaussian weighted instances of equal size.

3.3 Performance Comparison

Before we evaluate the performance of the tested separation scenarios for the different problem classes, we point out the respective tables with the detailed data on the results. The combined tables can be found in Appendix B.

Ising Spin Glass Problems. For the two-dimensional grids, the Tables B.2 to B.6 on the pages 106 to 110 list the CPU times required by the different scenarios. The results for the three-dimensional grids are provided in the Tables B.1 and B.3 on the pages 105 and 107, respectively.

There are two notable observations regarding the two-dimensional instances. If an instance could be solved to optimality within the ten hour CPU time limit then this was achieved in the root node. Moreover, regardless of the separation scenario only odd-cycle inequalities were added to the constraint pool in the course of the optimization. In other words, the scenarios CLQ and TC are equivalent to CON for all two-dimensional grids.

Max-Cut Problem Instances Generated With `rudy`. In certain subsets of the `rudy`-generated problem classes we could not solve any of the instances to optimality within the ten hour time limit regardless of the separation scenario. For these subsets we provide the relative gap at the end of the computations. Consequently, the results are split into the comparison of the required CPU times in Table B.9 on page 113 as well as the relative gaps in Table B.11 on page 114.

Note that the CON scenario ran out of memory for every $G_{0.5}$ graph of order sixty and eighty. This is because the odd-cycle inequalities give very poor bounds for these instances. As a result, the solver is forced to branch constantly and thus the branch-and-cut tree quickly grows in size. Technically, this would also have happened in the CYC scenario. However, since CYC is generally slower than CON, the time limit was reached before the algorithm could run out of memory.

Quadratic 0/1 Optimization Problems. Similar to the above `rudy`-generated instances, the results are split into the comparison of the CPU times in Table B.7 on page 111 and the relative gaps in Table B.8 on page 112, respectively.

Mannino Instances. The CPU times for the different scenarios are listed in Table B.10 on page 113.

In all the tables just mentioned, the minimum CPU time resp. relative gap over all separation scenarios is typeset bold for each instance.

We now compare the performance of the four examined separation scenarios for the different classes of test instances. As mentioned before, there are subsets of these classes in which we could solve none of the instances to optimality within the ten hour time limit regardless of the used separation scenario. Therefore, we split our overview into two parts. In Section 3.3.1 we focus on the classes in which we could solve most or even all of the instances to optimality. Here, we use the average CPU time reduction with respect to the standard scenario CYC to measure the performance of the remaining ones. Section 3.3.2 covers the instances which none of the scenarios could solve to optimality within the ten hour time limit. In this case we use the gap closure with respect to CYC as measure.

3.3.1 CPU Time Reduction

The results in terms of CPU time reduction are listed in Table 3.3 on page 90. The rows refer to the different classes of test instances. The columns are organized as follows:

- **Class** specifies the class of test instances. We considered Beasley instances of size $n = 50, 100, 250$ (beasley), Billionnet and Elloumi instances of size $n = 120, 250$ (be), Glover, Kochenberger, and Alidaee instances with the settings a, b, c, and d (gka), two- and three-dimensional toroidal grids with uniformly distributed ± 1 weights (tpm) resp. Gaussian distributed integral weights (tg), unweighted graphs with edge probability $1/2$ and order $n = 60$ (g05_60), ± 1-weighted graphs of order $n = 80, 100$ and density $d = 0.1$ (pm1s), graphs of order $n = 100$ and density $d = 0.1$ with weights chosen from the ranges $[-10, 10]$ (w01) resp. $[0, 10]$ (pw01), and finally the Mannino instances (mannino).
- **#Files** lists the total number of instances in each class.

- **#Limit** gives the number of instances per class that could not be solved to optimality within ten hours by any of the separation scenarios.
- **#Wins** lists for each separation scenario the number of instances per class for which this scenario took the least time to solve them to optimality. Note that the instances counted by '#Limit' were excluded from this evaluation. Also, the line total may exceed the number of remaining instances since different scenarios can have identical CPU times. The maximum value over all scenarios is typeset bold for each class.
- **#Add. rejects** gives for each of the scenarios CON, CLQ, and TC the number of instances per class that were excluded from the evaluation of the respective average CPU time reduction in addition to those already counted by '#Limit'. We rejected an instance if CYC or the respective scenario exceeded the time limit. Nonzero entries consist of two values. The first one is the total number of additionally rejected instances. The second value, given in parentheses, specifies how many of these rejections were caused by a time limit exceedance of the respective scenario itself rather than of CYC. Consequently, the difference between the first and the second value gives the number of instances that could be solved to optimality by the respective scenario while CYC exceeded the time limit. The nonzero entries with maximum difference between first and second value are typeset bold for each class.
- **Avg. CPU time red.** lists for each of the scenarios CON, CLQ, and TC its respective average CPU time reduction compared to CYC. The entries specify the reduction averaged over all instances in a given class except for those counted by '#Limit' and the respective entry of '#Add. rejects'. The maximum average reduction over all scenarios is typeset bold for each class.

Note that in case of the class 'g05_60', the scenario CYC always exceeded the time limit. As a result, we could not evaluate the actual average time reduction for the scenarios CON, CLQ, and TC. However, rather than omitting the entire class, we used the ten hour time limit as a lower bound on the computation time of CYC. Thus, the average reduction values for the class 'g05_60' are to be understood as lower bounds as well.

We see that the CON scenario clearly dominates with average time reductions ranging from 51 to 95% for most of the classes and an average reduction over all the classes of approximately 50%. Particularly noteworthy are the good results for the two-dimensional Ising spin glass problems. This is because for these instances the CYC scenario uses a very efficient linear time heuristic to separate 4-cycle inequalities. It enumerates the squares in a toroidal grid, which are chordless cycles and thus induce facets. It is quite remarkable that a rather general approach like the shrink separation is able to outclass this highly specialized heuristic—in particular since, unlike the heuristic, the shrink separation does not require a specific numbering of the grid's nodes and edges.

However, the results for the scenarios CLQ and TC are less distinct. Both perform well on some classes, poorly on others, and are sometimes even outperformed by CYC. Take, for instance, the CLQ scenario: Despite partly good time reductions of up to 95%, this scenario is on average approximately 40% slower than CYC. The results for TC are even worse—at least at first glance—with a time increase of 2014% averaged over all classes. Yet, this is mainly due to the extremely bad performance on the Glover, Kochenberger, and Alidaee instances of setting b (gka.b). If we omit this class in the

evaluation, we see that TC features a solid time reduction of almost 32% on average for the remaining classes.

3.3.2 Gap Closure

In general, the **gap closure** of a maximization problem is the percentage by which the gap between the lower bound c_{best} and an old upper bound c as well as a new upper bound c' could be closed, i. e.,

$$\text{gap closure} = \left(1 - \frac{c' - c_{\text{best}}}{c - c_{\text{best}}}\right) \cdot 100\%.$$

Thus, we obtain the gap closure with respect to CYC by setting c to the upper bound given by the CYC scenario at the end of the computations and c' to the one given by the respective remaining scenario whose performance we want to measure. The results are provided in Table 3.4 on page 91. As before, the rows refer to the different classes of test instances. The columns are organized as follows:

- **Class** specifies the class of test instances. We considered Beasley instances of size $n = 500$ (beasley500), Billionnet and Elloumi instances of size $n = 100, 120, 150, 200$ (be), Glover, Kochenberger, and Alidaee instances with the settings e and f (gka), unweighted graphs with an edge probability of $1/2$ and order $n = 80, 100$ (g05), ± 1-weighted graphs of order $n = 80, 100$ and density $d = 0.99$ (pm1d), and graphs of order $n = 100$ and varying density $d = 0.5, 0.9$ with weights chosen from the ranges $[-10, 10]$ (w) resp. $[0, 10]$ (pw).
- **#Files** lists the total number of instances in each class.
- **#Wins** specifies for each separation scenario the number of instances per class for which this scenario gave the smallest relative gap after ten hours of computation. Note that the line total may exceed the number of instances since different scenarios can result in the same relative gap. The maximum value over all scenarios is typeset bold for each class.
- **Avg. gap cl.** lists for each of the scenarios CON, CLQ, and TC, respectively, its average gap closure with respect to the CYC scenario after ten hours of computation. The maximum average closure over all scenarios is typeset bold for each class.

Although CON delivered the smallest gaps in most cases, its average gap closure over all classes is only about 15% with a peak value of 68% on single classes. The scenarios CLQ and TC, on the other hand, both feature a gap closure of approximately 19% averaged over all classes and peak values of 85 and 77%, respectively. Still, all three scenarios only work well for about one third of the tested classes. For the remaining ones, the results are mediocre at best. In some cases, the scenarios are even outperformed by CYC.

In conclusion, for most of the considered classes the scenarios CON, CLQ, and TC are superior to CYC. Yet, apart from CLQ and TC performing slightly better on average than CON, there is no clearly dominant scenario.

3.4 A Case Study: The Mannino Instances

Unlike the remaining test problems, the Mannino instances introduced in Section 3.1.3 were generated from real-world data. With the exception of '`mannino_k48`', the respective underlying graphs are extremely sparse with densities ranging from 1 to 7%. The instance '`mannino_k48`', on the other hand, is based on a complete graph. However, it is extremely easy to solve and will thus be omitted. In summary, the combination of real-world data and low densities makes these problems particularly interesting for testing the shrink separation. This is why we performed additional experiments using a variation of the CYC scenario which we will refer to as CYC′. Here, we removed the CPU time limit and also activated the tailing-off control to accelerate the optimization. This control enforced a branching step if the minimal change of the objective function value between the solution of one hundred successive linear programming relaxations in the subproblem optimization was below 0.0015%. The remaining scenarios were identical to the ones introduced in Section 3.2. In particular, they did not use the tailing-off control. However, like the original CYC scenario, its variation could not solve the largest instance '`mannino_k487c`' to optimality. Due to the steadily growing branch-and-cut tree, the algorithm eventually ran out of memory. We will discuss this particular problem in greater detail shortly, but first we look at the results for the remaining two instances.

In terms of CPU times, Table 3.5 on page 91 clearly shows that CON was the dominant scenario with an average time reduction of 98% compared to CYC′. The scenarios CLQ and TC performed slightly worse. However, with average reductions of 81 and 75%, respectively, they were still far superior to CYC′.

Next, we look at the size of the branch-and-cut tree at the end of each optimization run. Table 3.6 on page 91 shows that CON, despite a slight increase in case of '`mannino_k487a`', reduced the number of nodes in the branch-and-cut tree by almost 12% on average. Still, the CON scenario needed to branch, which indicates that odd-cycle inequalities alone are not sufficient to solve the Mannino instances in the root node. In this regard, the remaining two scenarios are clearly preferable. CLQ reduced the tree size by 69% on average while TC was even able to solve both instances in the root node. A check of the final constraint pools showed that the TC scenario achieved this with only 179 resp. 134 target cuts in addition to the odd-cycle inequalities.

The instance '`mannino_k487c`', however, was a greater challenge. As mentioned earlier, CYC′ ran out of memory during the optimization while the remaining scenarios exceeded the ten hour CPU time limit. By dropping this limit also for CON, CLQ, and TC, we were finally able to solve the largest instance to optimality. In fact, only the TC scenario achieved this goal. The optimization took 17.5 hours and required 45 nodes in the branch-and-cut tree. The number of target cuts in addition to odd-cycle inequalities in the final constraint pool was 766. By increasing the size of the subgraphs for the target cut separation from ten to eleven, we managed to solve the problem to optimality within a reduced running time of 10.2 hours and with only eleven nodes in the branch-and-cut tree. In this case, the final constraint pool contained 758 target cuts. Regarding the remaining two scenarios: CON ran out of memory due to the steadily growing branch-and-cut tree while CLQ was manually aborted after three days, at which point the relative gap was slightly below 0.015%.

In conclusion, we see that the CON scenario, though capable of accelerating the optimization significantly, may not be sufficient to solve certain problems to optimality. The more difficult problems can be dealt with by separating additional clique inequalities or

target cuts. If a slight increase in CPU time is not an issue, one can even apply the target cut separation to the easier problem instances, which allows to solve them in the root node instead of resorting to branching. Finally, we saw that an additional fine tuning of the target cut parameters can improve the already good results even further.

			#Wins				#Add. rejects			Avg. CPU time red. [%]		
Class	#Files	#Limit	CYC	CON	CLQ	TC	CON	CLQ	TC	CON	CLQ	TC
beasley50	10	0	0	**7**	5	5	0	0	0	**93**	88	90
beasley100	10	0	0	**8**	6	7	0	0	0	**95**	**95**	94
beasley250	10	5	0	**3**	0	2	**2(0)**	3(3)	**2(0)**	**58**	28	55
be120.3	10	0	0	**10**	0	0	**1(0)**	1(1)	1(1)	**53**	−75	13
be250	10	6	0	**3**	0	1	0	1(1)	0	**52**	−22	44
gka.a	8	0	0	5	**7**	6	0	0	0	88	**90**	88
gka.b	10	0	0	**10**	0	0	0	0	3(3)	**63**	−14	−36786
gka.c	7	0	0	3	**4**	2	0	0	0	**95**	**95**	**95**
gka.d	10	5	0	**4**	1	1	0	1(1)	1(1)	**51**	−53	29
tpm.2d	120	23	10	**87**	—[b]	—[b]	**9(6)**	—[b]	—[b]	**76**	—[b]	—[b]
tg.2d	320	0	36	**284**	—[b]	—[b]	**4(3)**	—[b]	—[b]	**51**	—[b]	—[b]
tpm.3d	40	6	**14**	9	4	7	0	7(7)	0	**−10**	−218	−56
tg.3d	70	8	7	**34**	7	18	2(1)	4(4)	**2(0)**	**27**	−69	6
g05_60[c]	10	1	0	—[a]	**9**	0	—[a]	0	3(3)	—[a]	**≥ 70**	≥ 46
pm1s	20	0	5	**14**	1	1	0	0	0	**6**	−187	−49
w01	10	0	**6**	3	0	1	0	0	0	**−4**	−337	−44
pw01	10	0	**6**	3	0	1	0	0	0	**−7**	−398	−50
mannino	4	1	0	**3**	0	0	**1(0)**	**1(0)**	**1(0)**	**70**	59	52

[a] The CON scenario ran out of memory for every instance. See page 85 for more information.
[b] Equivalent to CON scenario.
[c] The CYC scenario exceeded the ten hour time limit on all instances. Instead of omitting the entire class, we used the limit as a lower bound on the computation time of CYC.

Table 3.3: Statistics on the CPU times of different separation scenarios.

		#Wins				Avg. gap cl. [%]		
Class	#Files	CYC	CON	CLQ	TC	CON	CLQ	TC
beasley500	10	0	2	0	**8**	68	**69**	**69**
be100	10	0	**10**	0	0	**13**	8	−11
be120.8	10	0	**10**	0	0	**10**	4	−1
be150.3	10	0	**10**	0	0	**15**	−26	−1
be150.8	10	**4**	1	3	3	2	**3**	**3**
be200.3	10	0	1	0	**9**	9	0	**11**
be200.8	10	**9**	0	0	1	**−21**	−25	−22
gka.e	5	1	**3**	1	**3**	19	10	**20**
gka.f	5	**3**	0	2	0	−5	−5	**−4**
g05_80	10	0	—[a]	**10**	0	—[a]	**85**	77
g05_100	10	0	0	**9**	1	26	**65**	57
pm1d	20	0	**20**	1	0	**19**	10	6
w05	10	0	**10**	0	0	**43**	23	37
w09	10	0	**10**	0	0	**19**	14	9
pw05	10	0	0	**9**	1	12	**56**	50
pw09	10	0	0	**8**	2	6	**13**	8

[a] The CON scenario ran out of memory for every instance. See page 85 for more information.

Table 3.4: Statistics on the gap closure with respect to CYC after ten hours of computation for different separation scenarios.

	CPU time [sec]			
Instance	CYC′	CON	CLQ	TC
mannino_k487a	209	**8**	71	99
mannino_k487b	38167	**68**	1262	1232
mannino_k487c	—[a]	—[a]	>259200[b]	**63022**[b]

[a] 'Out of memory' error.
[b] No limit on CPU time.

Table 3.5: CPU times for Mannino instances.

	#Nodes in B&C-tree			
Instance	CYC′	CON	CLQ	TC
mannino_k487a	31	45	11	**1**
mannino_k487b	41	13	11	**1**
mannino_k487c	—[a]	—[a]	≥27[b]	**45**[b]

[a] 'Out of memory' error.
[b] No limit on CPU time.

Table 3.6: Number of nodes in the branch-and-cut tree for Mannino instances.

Chapter 4

Discussion and Conclusions

We have presented a new shrink separation approach that can be used within a branch-and-cut algorithm for solving max-cut problems to optimality. The key idea is to contract all edges with a respective integral LP value. In its simplest form, this graph contraction allows the efficient separation of odd-cycle inequalities. We described how to lift cutting planes for the contracted LP solution to cutting planes for the original one. Moreover, we specified under which conditions the lifting preserves existing facet-defining properties of valid inequalities for the cut polytope.

Building upon the contraction approach, we contributed means to make methods intended for max-cut problems on complete graphs applicable to problems on sparse graphs. We proposed two techniques to add missing edges and to efficiently compute suitable LP values in order to extend the contracted LP solution accordingly. The static extension assigns fixed LP values to the artificial edges while the adaptive extension uses adjustable LP values that are a priori undetermined. In this regard, the prior contraction of the graph has the benefit of lowering the number of edges that need to be added and thus reducing the subsequent computational effort. We described how to project out the variables associated with artificial edges from the separating inequalities for the extended LP solution. The properties of this projection differ depending on the choice of the extension technique. In summary, the static extension allows to use standard separation procedures for the max-cut problem, but the projection can alter the amount by which the extended LP solution violates the obtained separating inequalities. The adaptive extension, on the other hand, always preserves the amount of violation during the projection. However, standard separation procedures are typically unable to handle the adaptive LP values and may need adjustment. In this context we presented a proof of concept by customizing an existing target cut separation procedure to the use of adaptive extension.

Finally, we implemented the shrink separation in C++ and embedded it in an exact branch-and-cut solver based on the ABACUS framework. After pointing out some of the realization's numerical aspects, we evaluated its performance on a variety of test instances. First, we looked at the CPU times, with the focus on those instances that could be solved to optimality within a time limit of ten hours. Here, using the graph contraction as means to separate odd-cycle inequalities proved to be the dominant strategy. Compared to standard odd-cycle separation techniques, the contraction approach on average halved the time required to solve the problems to optimality, with peak time reductions of up to 95% on selected problem classes. Particularly noteworthy is the sig-

nificant acceleration when optimizing two-dimensional Ising spin glass problems. This is because the standard techniques involved an efficient linear time separation procedure specifically designed for toroidal grid instances. It enumerates the grid squares, which are chordless cycles. Consequently, the respective 4-cycle inequalities define facets of the cut polytope. Still, the more general contraction-based odd-cycle separation was able to outclass this highly specialized method. These results are even more stunning considering that, unlike the enumeration method, the contraction approach does not require a specific numbering of the grid's nodes and edges. We also experimented with separating additional clique inequalities and target cuts on the already contracted LP solutions but obtained mixed results. While the computation of these extra cutting planes can be worthwhile for some problem classes, it generally only produces an overhead. This can slow down the entire shrink separation up to the point where it is even outperformed by the standard odd-cycle separation techniques.

Next, we considered the test instances that could not be solved to optimality within the time limit. Here, we evaluated the gap closure that the shrink separation could achieve compared to the standard odd-cycle separation. The contraction-based odd-cycle separation could close the gap by 15% on average. The addition of clique inequalities and target cuts slightly improved this value to 19%. Still, the results were mixed and the standard odd-cycle separation partly outperformed the shrink separation.

As a final part of our computational experiments we presented a case study in which we looked closer at the Mannino instances—a set of problems generated from real-world data in the context of a frequency allocation problem for major Italian cities. Here, the contraction-based odd-cycle separation was extremely effective and reduced the CPU time required on average by 98%. The only exception was the largest instance, for which the approach ran out of memory due to the steady growth of the branch-and-cut tree. However, with the addition of target cuts we were able to also solve the largest problem to optimality. Moreover, the target cuts, though slightly slowing down the overall optimization, even allowed to solve the remaining instances in the root node while all the other separation scenarios required branching.

In conclusion, the presented shrink separation shows great potential to facilitate the optimization of max-cut problems. As a rule of thumb, the approach seems to be most effective on problems whose underlying graphs are sparse. Its use for problems on dense or complete graphs, on the other hand, is limited and dwindles with increasing problem size. This is partly due to the fact that we use a formulation of the max-cut problem which introduces a decision variable for each edge in the graph. So, for problem instances based on a dense or even complete graph, an SDP based solver like, for instance, Biq Mac [Wie06] is preferable. However, this type of solver tends to easily run out of memory for large sparse problems. To give an example, we experimented with a version of Biq Mac [RRW10] that is made available through a web interface[1]. We used two-dimensional Ising spin glass problems as test instances. In our experience, the solver returned a storage allocation failure for grid sizes beyond (65×65). Using the shrink separation, on the other hand, our branch-and-cut algorithm successfully solved problems with sizes up to (185×185) to optimality within ten hours and, by dropping the time limit, can handle even larger instances.

Concerning future work, there are several aspects we did not explore that could improve our results even further. First of all, the use of target cuts shows great potential

[1]Biq Mac ("Binary quadratic and Max cut") Solver, http://biqmac.uni-klu.ac.at.

in principle. However, in its current form it often slows down the overall optimization significantly. Thus, investigating efficient methods to generate stronger target cuts may be worthwhile, in particular the development of an advanced strategy to select the subgraphs for the target cut separation. One could also think of maintaining a repository of promising subgraphs—or subgraph seeds—from prior iterations, which could be exploited in the further course of the optimization. It may also improve the results to use local cuts instead of target cuts. Although local cuts in general do not produce facet defining inequalities, their computation requires less effort, which could accelerate the overall optimization. Lastly, the use of custom branching strategies that aim at maximizing the number of contractible edges could be worth investigating.

Ultimately, the shrink separation with its unique combination of graph contraction and extension provides the means to apply arbitrary separation methods for the well-studied max-cut problem on complete graphs to problems on sparse graphs and thus could prove to be a solid base for further research.

Appendix A

Data on Test Instances

In the tables below we provide exhaustive data on the individual test instances that we used for our computational experiments. The data comprises sizes and densities of the underlying graphs as well as optimum values, if known, or bounds on the optimum otherwise.

$n = 100, d = 1.0$	
Name	Opt
be100.1	−19412
be100.2	−17290
be100.3	−17565
be100.4	−19125
be100.5	−15868
be100.6	−17368
be100.7	−18629
be100.8	−18649
be100.9	−13294
be100.10	−15352

$n = 120, d = 0.3$	
Name	Opt
be120.3.1	−13067
be120.3.2	−13046
be120.3.3	−12418
be120.3.4	−13867
be120.3.5	−11403
be120.3.6	−12915
be120.3.7	−14068
be120.3.8	−14701
be120.3.9	−10458
be120.3.10	−12201

$n = 120, d = 0.8$	
Name	Opt
be120.8.1	−18691
be120.8.2	−18827
be120.8.3	−19302
be120.8.4	−20765
be120.8.5	−20417
be120.8.6	−18482
be120.8.7	−22194
be120.8.8	−19534
be120.8.9	−18195
be120.8.10	−19049

$n = 150, d = 0.3$	
Name	Opt
be150.3.1	−18889
be150.3.2	−17816
be150.3.3	−17314
be150.3.4	−19884
be150.3.5	−16817
be150.3.6	−16780
be150.3.7	−18001
be150.3.8	−18303
be150.3.9	−12838
be150.3.10	−17963

$n = 150, d = 0.8$	
Name	Opt
be150.8.1	−27089
be150.8.2	−26779
be150.8.3	−29438
be150.8.4	−26911
be150.8.5	−28017
be150.8.6	−29221
be150.8.7	−31209
be150.8.8	−29730
be150.8.9	−25388
be150.8.10	−28374

$n = 200, d = 0.3$	
Name	Opt
be200.3.1	−25453
be200.3.2	−25027
be200.3.3	−28023
be200.3.4	−27434
be200.3.5	−26355
be200.3.6	−26146
be200.3.7	−30483
be200.3.8	−27355
be200.3.9	−24683
be200.3.10	−23842

$n = 200, d = 0.8$	
Name	Opt
be200.8.1	−48534
be200.8.2	−40821
be200.8.3	−43207
be200.8.4	−43757
be200.8.5	−41482
be200.8.6	−49492
be200.8.7	−46828
be200.8.8	−44502
be200.8.9	−43241
be200.8.10	−42832

$n = 250, d = 0.1$	
Name	Opt
be250.1	−24076
be250.2	−22540
be250.3	−22923
be250.4	−24649
be250.5	−21057
be250.6	−22735
be250.7	−24095
be250.8	−23801
be250.9	−20051
be250.10	−23159

Table A.1: Data on the Billionnet and Elloumi instances.

Name	#Nodes	#Edges	Density	Range of weights	Opt
mannino_k48	48	1128	1.0	$[13146, 841699]$	252518838
mannino_k487a	487	1435	≈ 0.01	$[101, 33631]$	1110926
mannino_k487b	487	5391	≈ 0.05	$[5, 176030]$	3655475
mannino_k487c	487	8511	≈ 0.07	$[5, 203785]$	8640860

Table A.2: Data on the Mannino instances.

$n = 50$		$n = 100$		$n = 250$	
Name	Opt	Name	Opt	Name	Opt
beasley50.1	−2098	beasley100.1	−7970	beasley250.1	−45607
beasley50.2	−3702	beasley100.2	−11036	beasley250.2	−44810
beasley50.3	−4626	beasley100.3	−12723	beasley250.3	−49037
beasley50.4	−3544	beasley100.4	−10368	beasley250.4	−41274
beasley50.5	−4012	beasley100.5	−9083	beasley250.5	−47961
beasley50.6	−3693	beasley100.6	−10210	beasley250.6	−41014
beasley50.7	−4520	beasley100.7	−10125	beasley250.7	−46757
beasley50.8	−4216	beasley100.8	−11435	beasley250.8	−35726
beasley50.9	−3780	beasley100.9	−11455	beasley250.9	−48916
beasley50.10	−3507	beasley100.10	−12565	beasley250.10	−40442

$n = 500$	
Name	Opt
beasley500.1	[−121588.41, −116586]
beasley500.2	[−132216.45, −128223]
beasley500.3	[−134214.12, −130812]
beasley500.4	[−134781.02, −130097]
beasley500.5	[−129572.87, −125487]
beasley500.6	[−126429.50, −121772]
beasley500.7	[−127136.37, −122201]
beasley500.8	[−128574.61, −123559]
beasley500.9	[−125821.63, −120798]
beasley500.10	[−134352.34, −130619]

Table A.3: Data on the Beasley instances. All problems have a density of 0.1 and integer uniform coefficients in the range $[-100, 100]$.

Set a			
Name	n	d	Opt
gka1a	50	0.1	−3414
gka2a	60	0.1	−6063
gka3a	70	0.1	−6037
gka4a	80	0.1	−8598
gka5a	50	0.2	−5737
gka6a	30	0.4	−3980
gka7a	30	0.5	−4541
gka8a	100	0.0625	−11109

Set c			
Name	n	d	Opt
gka1c	40	0.8	−5058
gka2c	50	0.6	−6213
gka3c	60	0.4	−6665
gka4c	70	0.3	−7398
gka5c	80	0.2	−7362
gka6c	90	0.1	−5824
gka7c	100	0.1	−7225

Set b ($d = 1.0$)		
Name	n	Opt
gka1b	20	−133
gka2b	30	−121
gka3b	40	−118
gka4b	50	−129
gka5b	60	−150
gka6b	70	−146
gka7b	80	−160
gka8b	90	−145
gka9b	100	−137
gka10b	125	−154

Set d ($n = 100$)		
Name	d	Opt
gka1d	0.1	−6333
gka2d	0.2	−6579
gka3d	0.3	−9261
gka4d	0.4	−10727
gka5d	0.5	−11626
gka6d	0.6	−14207
gka7d	0.7	−14476
gka8d	0.8	−16352
gka9d	0.9	−15656
gka10d	1.0	−19102

Set e ($n = 200$)		
Name	d	Opt
gka1e	0.1	−16464
gka2e	0.2	−23395
gka3e	0.3	−25243
gka4e	0.4	−35594
gka5e	0.5	−35154

Set f ($n = 500$)		
Name	d	Opt
gka1f	0.1	[−63400.98, −61194]
gka2f	0.25	[−104868.34, −100161]
gka3f	0.5	[−145420.14, −138035]
gka4f	0.75	[−181507.74, −172771]
gka5f	1.0	[−201130.98, −190507]

Table A.4: Data on the Glover, Kochenberger, and Alidaee instances.

$n = 60$		$n = 80$		$n = 100$	
Name	Opt	Name	Opt	Name	Opt
g05_60.0	536	g05_80.0	929	g05_100.0	1430
g05_60.1	532	g05_80.1	941	g05_100.1	1425
g05_60.2	529	g05_80.2	934	g05_100.2	1432
g05_60.3	538	g05_80.3	923	g05_100.3	1424
g05_60.4	527	g05_80.4	932	g05_100.4	1440
g05_60.5	533	g05_80.5	926	g05_100.5	1436
g05_60.6	531	g05_80.6	929	g05_100.6	1434
g05_60.7	535	g05_80.7	929	g05_100.7	1431
g05_60.8	530	g05_80.8	925	g05_100.8	1432
g05_60.9	533	g05_80.9	923	g05_100.9	1430

Table A.5: Data on $G_{0.5}$ instances. The underlying graphs are unweighted with edge probability $1/2$.

$n = 80, d = 0.1$	
Name	Opt
pm1s_80.0	79
pm1s_80.1	85
pm1s_80.2	82
pm1s_80.3	81
pm1s_80.4	70
pm1s_80.5	87
pm1s_80.6	73
pm1s_80.7	83
pm1s_80.8	81
pm1s_80.9	70

$n = 100, d = 0.1$	
Name	Opt
pm1s_100.0	127
pm1s_100.1	126
pm1s_100.2	125
pm1s_100.3	111
pm1s_100.4	128
pm1s_100.5	128
pm1s_100.6	122
pm1s_100.7	112
pm1s_100.8	120
pm1s_100.9	127

$n = 80, d = 0.99$	
Name	Opt
pm1d_80.0	227
pm1d_80.1	245
pm1d_80.2	284
pm1d_80.3	291
pm1d_80.4	251
pm1d_80.5	242
pm1d_80.6	205
pm1d_80.7	249
pm1d_80.8	293
pm1d_80.9	258

$n = 100, d = 0.99$	
Name	Opt
pm1d_100.0	340
pm1d_100.1	324
pm1d_100.2	389
pm1d_100.3	400
pm1d_100.4	363
pm1d_100.5	441
pm1d_100.6	367
pm1d_100.7	361
pm1d_100.8	385
pm1d_100.9	405

Table A.6: Data on $G_{-1/0/1}$ instances. The underlying graphs are weighted with edge weights chosen uniformly from $\{-1, 0, 1\}$.

$n = 100, d = 0.1$	
Name	Opt
w01_100.0	651
w01_100.1	719
w01_100.2	676
w01_100.3	813
w01_100.4	668
w01_100.5	643
w01_100.6	654
w01_100.7	725
w01_100.8	721
w01_100.9	729

$n = 100, d = 0.5$	
Name	Opt
w05_100.0	1646
w05_100.1	1606
w05_100.2	1902
w05_100.3	1627
w05_100.4	1546
w05_100.5	1581
w05_100.6	1479
w05_100.7	1987
w05_100.8	1311
w05_100.9	1752

$n = 100, d = 0.9$	
Name	Opt
w09_100.0	2121
w09_100.1	2096
w09_100.2	2738
w09_100.3	1990
w09_100.4	2033
w09_100.5	2433
w09_100.6	2220
w09_100.7	2252
w09_100.8	1843
w09_100.9	2043

$n = 100, d = 0.1$	
Name	Opt
pw01_100.0	2019
pw01_100.1	2060
pw01_100.2	2032
pw01_100.3	2067
pw01_100.4	2039
pw01_100.5	2108
pw01_100.6	2032
pw01_100.7	2074
pw01_100.8	2022
pw01_100.9	2005

$n = 100, d = 0.5$	
Name	Opt
pw05_100.0	8190
pw05_100.1	8045
pw05_100.2	8039
pw05_100.3	8139
pw05_100.4	8125
pw05_100.5	8169
pw05_100.6	8217
pw05_100.7	8249
pw05_100.8	8199
pw05_100.9	8099

$n = 100, d = 0.9$	
Name	Opt
pw09_100.0	13585
pw09_100.1	13417
pw09_100.2	13461
pw09_100.3	13656
pw09_100.4	13514
pw09_100.5	13574
pw09_100.6	13640
pw09_100.7	13501
pw09_100.8	13593
pw09_100.9	13658

Table A.7: Data on $G_{[-10,10]}$ (w) and $G_{[0,10]}$ (pw) instances. The underlying graphs are weighted with integral edge weights chosen from the specified ranges.

$k = 30$	
Seed	Opt
11653	636
13441	620
14579	624
16241	626
22000	630
23307	626
30533	640
32642	626
4527	626
7903	646

$k = 35$	
Seed	Opt
14423	858
15415	840
15846	836
20849	848
23778	858
23798	856
27102	858
28373	852
28708	868
457	864

$k = 40$	
Seed	Opt
12504	1126
13024	1116
17666	1118
21887	1124
24026	1144
2566	1128
31352	1108
3336	1102
8193	1132
8614	1114

$k = 45$	
Seed	Opt
12027	1408
12898	1402
14407	1400
16600	1406
16710	1428
19930	1416
27477	1418
4946	1432
7500	1418
965	1410

$k = 50$	
Seed	Opt
12205	1740
14092	1740
14211	1752
17695	1764
18152	1754
24084	1750
31133	1770
3235	1726
5102	1744
8545	1746

$k = 55$	
Seed	Opt
11787	2112
13203	2102
13295	2110
15123	2126
21910	2130
23960	2094
30290	2142
3697	2112
6263	2128
6808	2112

$k = 60$	
Seed	Opt
17049	2524
19697	2502
22181	2514
27127	2524
28147	2518
2987	2500
4211	2532
5022	2524
7774	2534
8278	2500

$k = 65$	
Seed	Opt
10959	2990
17716	2956
18062	2950
21384	2954
21907	2970
23164	2946
26400	2962
32274	2966
7291	2982
8051	2970

$k = 70$	
Seed	Opt
25063	3412
26486	3424
28924	3422
2941	3442
2964	3428
30920	3432
32012	3430
7014	3430
9204	3394
9939	3430

$k = 75$	
Seed	Opt
10738	3934
11748	3928
151	3928
20442	3918
241	3916
27279	3934
28720	3982
28797	3972
31708	3964
6726	3924

$k = 80$	
Seed	Opt
11489	4478
18055	4464
18637	4500
22148	4482
22267	4534
28635	4474
29439	4470
3438	4478
3932	4490
5470	4506

$k = 85$	
Seed	Opt
10733	5086
11774	5078
13674	5078
14944	5054
15307	5074
18788	[5060, 5062]
22879	5068
5367	5100
7215	5072
7602	5040

Table A.8: Data on toroidal ($k \times k$) grids with uniformly distributed ± 1 weights. 'Seed' lists the random seeds used for generating the instances.

$k = 30$	
Seed	Opt
11495	58661632
13235	57521425
13605	57519669
13937	62318169
14964	59736718
20209	62734462
22312	62001902
3934	61760927
7382	58959125
8289	62797247

$k = 35$	
Seed	Opt
10018	77903675
18452	82627972
23988	83242094
278	80942210
30846	81373652
3294	82910165
4430	74894853
4948	82055249
7040	81448751
8883	81816684

$k = 40$	
Seed	Opt
11761	108114447
1312	104346958
14029	102247437
21074	104539379
2480	104579088
30667	107862915
5273	102345107
7115	103566694
7974	103894527
9512	106727304

$k = 45$	
Seed	Opt
16083	133047085
20652	134083067
21007	132511460
21053	135233420
22054	130039421
24987	139060536
2848	140104818
31108	134968926
508	134683245
8448	134798284

$k = 50$	
Seed	Opt
10929	171848787
11184	170781311
16132	159143501
16361	164406485
1813	170718815
25245	165605411
32072	168784903
3735	160158697
7352	161761289
7890	161235815

$k = 55$	
Seed	Opt
10166	193942133
12625	198025786
18140	209770015
20590	205590122
22691	197378402
25255	205204723
26654	196886913
29134	192367236
8853	202393689
8897	206313116

$k = 60$	
Seed	Opt
13540	242986106
15097	238317681
15359	236994157
18207	228999999
18409	237511199
27474	234495440
4383	242539481
5759	242683201
6267	232971654
6648	240280455

$k = 65$	
Seed	Opt
10361	283876459
14157	282183788
1523	273163929
17884	279329643
21291	271073381
25341	271730791
3687	282992077
7422	278638796
8705	277178635
9236	277928660

$k = 70$	
Seed	Opt
11214	317363325
13627	320960806
16057	319932831
18089	330764188
18936	329757133
22569	320236499
28255	321306022
28683	320853901
31804	311688502
7933	324216743

$k = 75$	
Seed	Opt
13642	365150433
14441	374041360
15320	370132837
19655	364224909
223	368534208
25415	369334665
26342	369300324
29001	376175177
31682	366127730
6115	377356725

$k = 80$	
Seed	Opt
1080	423144297
11442	422412343
13072	427116250
14351	418671807
15964	410831544
19703	430114984
23390	428526647
30813	414180192
5645	425872291
7281	426672072

$k = 85$	
Seed	Opt
10215	467825042
11179	477196491
16125	476320157
1717	465704743
18149	471497118
20858	477104803
25370	482438183
26323	473569889
30408	474399045
32733	481312064

$k = 90$	
Seed	Opt
11723	533220177
11946	529382174
12169	524132585
15935	546858680
20135	527458256
2293	553702134
26610	526188217
27266	529484077
27488	522234384
7832	526748720

$k = 95$	
Seed	Opt
10886	584204645
14201	602601022
19049	590767377
19350	590993482
22328	597801401
2869	577597448
32121	595326543
4824	582940077
4999	589115927
9806	606496065

$k = 100$	
Seed	Opt
10150	647462060
10546	656447418
10920	662711750
16990	665569618
22293	661488251
23610	667363738
25328	647582676
26671	655949369
2752	657715351
704	668132036

$k = 105$	
Seed	Opt
12131	705314014
14469	695811951
15247	727872937
19971	720330842
24300	721068736
27880	730942629
28964	720885443
29069	715974914
392	721280937
8024	721615970

Table A.9: Data on toroidal $(k \times k)$ grids with Gaussian distributed integral weights. 'Seed' lists the random seeds used for generating the instances.

$k = 110$	
Seed	Opt
1529	786076341
18142	796316551
27948	792672585
28395	817300054
28671	800360679
3596	782391463
5881	818299433
6067	814908693
727	794719145
882	786157215

$k = 115$	
Seed	Opt
13746	851492054
13903	883597024
16498	870966055
17921	865980373
18625	877859936
20000	863656579
25231	876770372
29545	882445579
7341	865413023
9454	858204759

$k = 120$	
Seed	Opt
10099	929775752
10653	940987796
1103	957288806
13763	952964879
21074	944261721
22230	940620699
25846	957891409
28174	949858046
293	953482659
32669	954207089

$k = 125$	
Seed	Opt
12182	1037328839
13065	1006709807
18946	1022713614
27086	1021566382
2775	1025226450
3001	1008264367
31446	1017882277
31905	1033180558
32173	1049685293
385	1012971346

$k = 130$	
Seed	Opt
11410	1117069724
13933	1107572557
16747	1105037481
20864	1101819808
28780	1113174714
29336	1120129406
32559	1114523437
478	1122924115
7819	1109309901
8096	1111734667

$k = 135$	
Seed	Opt
11824	1201815986
21560	1214623380
21999	1225925302
22415	1200555575
23518	1180843686
2555	1186016038
32098	1207605121
3419	1207536593
7720	1209588264
8013	1208970367

$k = 140$	
Seed	Opt
12683	1291907082
13278	1302748132
14073	1284999528
14599	1297722942
1693	1292972751
25499	1308500561
26255	1283518711
28779	1298772984
29164	1275833161
6552	1296919925

$k = 145$	
Seed	Opt
142	1388313544
15685	1386757987
25005	1393086100
25176	1390586096
32432	1369056460
333	1377066313
4141	1371876611
6133	1384196921
6342	1392459524
7961	1377978454

$k = 150$	
Seed	Opt
10454	1495662545
15681	1488132003
15866	1496016572
21662	1467898813
22332	1465634683
23695	1473809878
25116	1494238043
2701	1494610559
27114	1467453627
27690	1477356020

$k = 155$	
Seed	Opt
10714	1568656847
13010	1607258237
14703	1585294184
1907	1608298507
22800	1569569580
2715	1590096121
7110	1593678895
8419	1578562361
8459	1599849310
9522	1575205489

$k = 160$	
Seed	Opt
1190	1678959821
11993	1679440535
18065	1696606503
18207	1683659852
18400	1669539689
30336	1686943303
30669	1703245021
32286	1688962563
5331	1674970620
5860	1663114284

$k = 165$	
Seed	Opt
14695	1777307109
19582	1804372242
20233	1780198903
22910	1795283389
26169	1782142526
27124	1815654692
7043	1796130633
9082	1785623675
9127	1795794685
9	1787015996

$k = 170$	
Seed	Opt
13142	1918859526
14527	1913313421
17386	1903720020
17832	1897617261
20102	1902263711
25241	1907735069
27354	1910625897
2775	1893032209
32592	1913511760
4682	1893351395

$k = 175$	
Seed	Opt
14332	1993423320
16955	1998196998
17232	2007186053
19664	2008045971
23799	2010109644
31869	2030296750
32351	2033138819
4962	2007993837
5734	2004793199
9239	2011592040

$k = 180$	
Seed	Opt
11722	2088434640
11727	2133952195
11732	2120075491
15134	2127366027
2599	2128278450
2640	2120473523
28836	2139781710
31309	2135103581
31650	2120501740
5926	2134297213

$k = 185$	
Seed	Opt
12892	2249859504
13546	2269160653
13900	2227052097
25873	2258987453
26688	2256515990
31133	2244125602
5365	2276870074
6088	2234314746
8487	2237620682
8863	2239039934

Table A.10: Data on toroidal ($k \times k$) grids with Gaussian distributed integral weights (cont.).

$k = 5$		$k = 6$		$k = 7$		$k = 8$	
Seed	Opt	Seed	Opt	Seed	Opt	Seed	Opt
12146	112	15089	186	12691	302	10395	452
14840	108	18073	192	13147	310	11644	[450, 454]
16870	110	18549	192	15920	302	12169	456
19456	110	22061	190	18333	308	1237	[460, 462]
26312	110	2351	188	18420	308	15544	452
2721	110	24098	194	2081	302	16077	[456, 458]
31519	110	24312	190	26164	304	22	456
3375	110	31266	190	31795	306	2765	458
6254	110	9457	192	6390	302	27750	458
6842	112	9952	188	6652	306	8923	[454, 456]

Table A.11: Data on toroidal ($k \times k \times k$) grids with uniformly distributed ± 1 weights. 'Seed' lists the random seeds used for generating the instances.

$k = 5$		$k = 6$		$k = 7$		$k = 8$	
Seed	Opt	Seed	Opt	Seed	Opt	Seed	Opt
10441	9277464	10064	17713143	11280	25519376	10303	43592459
17970	9360308	12717	18017627	18835	30003302	10856	46344821
18923	12073082	19137	18063240	19609	25897495	14867	45205148
2188	10946102	25533	17315858	20848	28091114	16663	39981474
22776	9599268	2676	21077885	21148	30089554	19806	45373707
29184	8988032	27220	16919770	26425	25288073	31605	41832185
466	9829525	27526	16357410	26752	27387188	5962	42259324
5589	8909853	28027	18362589	27585	32567124	6720	45544519
8950	12791739	5368	19286131	29416	29275320	7186	39205148
9477	10390827	7798	19344366	32731	27507640	9278	38618779

$k = 9$		$k = 10$		$k = 11$	
Seed	Opt	Seed	Opt	Seed	Opt
1042	64108058	20294	85081904	10509	111586711
11330	62327221	21289	81325562	13445	[112234762, 112598912]
12787	63403107	22003	87880828	18545	[115135667, 115666548]
14098	58991509	23646	87831577	19131	115771689
19340	64382053	2498	86337174	20650	[111563590, 111639398]
22851	64756627	26472	84592459	24507	[112481294, 112602779]
5782	61752437	3168	83267779	28409	[113447934, 113761126]
6863	61533895	3205	86954873	31506	[111199947, 111906046]
70	56542382	32569	86902034	32307	109252542
9221	60596017	8840	85105005	6259	[110034060, 110237155]

Table A.12: Data on toroidal ($k \times k \times k$) grids with Gaussian distributed integral weights. 'Seed' lists the random seeds used for generating the instances.

Appendix B

Data on Computational Results

Below, we provide the collective data on our computational results for the different classes of test instances. The abbreviations CYC, CON, CLQ, and TC denote the separation scenarios that were introduced in Section 3.2. For those classes in which we could solve most or even all of the instances to optimality within the given time limit of ten hours, we provide the CPU times required by the different scenarios. The minimum CPU time is typeset bold for each instance. For the remaining classes, i. e., those in which none of the instances could be solved to optimality within ten hours regardless of the used scenario, we provide the relative gaps at the end of the computations. The minimum gap is typeset bold for each instance. Note that in case of the two-dimensional toroidal grids the scenarios CLQ and TC are equivalent to CON and are hence omitted.

Instance		CPU time [sec]			
k	Seed	CYC	CON	CLQ	TC
5:	12146	0	0	0	**0**
	14840	**2**	2	3	4
	16870	0	0	**0**	0
	19456	**4**	5	15	16
	26312	0	0	0	**0**
	2721	2	2	**1**	5
	31519	4	**3**	8	4
	3375	11	12	**6**	15
	6254	**5**	7	12	18
	6842	0	0	0	**0**
6:	15089	298	**161**	1071	493
	18073	1	1	**0**	0
	18549	**3**	4	5	4
	22061	**535**	774	4637	1366
	2351	100	**94**	390	146
	24098	**32**	46	122	74
	24312	**1**	2	1	1
	31266	65	**57**	688	114
	9457	**81**	93	701	120
	9952	157	**130**	767	444

Instance		CPU time [sec]			
k	Seed	CYC	CON	CLQ	TC
7:	12691	**2637**	3355	limit	3541
	13147	78	**61**	310	104
	15920	9884	10028	limit	**8000**
	18333	629	**316**	2190	583
	18420	**1732**	1862	6711	2785
	2081	902	**645**	3927	2290
	26164	**8894**	25508	limit	17296
	31795	639	131	639	**99**
	6390	1487	**1303**	5250	2011
	6652	460	369	1921	**354**
8:	10395	limit	limit	limit	limit
	11644	limit	limit	limit	limit
	12169	limit	limit	limit	limit
	1237	limit	limit	limit	limit
	15544	21267	16409	limit	**8345**
	16077	limit	limit	limit	limit
	22	**22914**	29770	limit	34471
	2765	**10089**	12835	limit	17875
	27750	**14891**	30625	limit	22836
	8923	limit	limit	limit	limit

Table B.1: CPU times for toroidal ($k \times k \times k$) grids with uniformly distributed ± 1 weights. 'Seed' lists the random seeds used for generating the instances.

Instance		CPU time [sec]		Instance		CPU time [sec]		Instance		CPU time [sec]	
k	Seed	CYC	CON	k	Seed	CYC	CON	k	Seed	CYC	CON
30:	11653	8	**0**	50:	12205	4595	**427**	70:	25063	**16710**	limit
	13441	91	**21**		14092	3145	**194**		26486	limit	limit
	14579	37	**0**		14211	395	**12**		28924	21222	**2541**
	16241	21	**0**		17695	339	**11**		2941	1348	**506**
	22000	13	**1**		18152	894	**46**		2964	limit	**2132**
	23307	25	**2**		24084	127	**18**		30920	2109	**300**
	30533	46	**1**		31133	2128	**264**		32012	1238	**1033**
	32642	18	**0**		3235	108	**20**		7014	3204	**82**
	4527	34	**2**		5102	403	**18**		9204	limit	limit
	7903	33	**2**		8545	189	**6**		9939	9407	**1776**
35:	14423	58	**3**	55:	11787	17925	**343**	75:	10738	4312	**2332**
	15415	1359	**48**		13203	866	**33**		11748	limit	limit
	15846	157	**15**		13295	1368	**106**		151	limit	limit
	20849	54	**2**		15123	523	**49**		20442	limit	limit
	23778	82	**7**		21910	910	**83**		241	limit	**5215**
	23798	71	**5**		23960	1105	**50**		27279	limit	limit
	27102	82	**3**		30290	4216	**407**		28720	**14352**	21860
	28373	183	**7**		3697	3073	**254**		28797	4989	**1319**
	28708	56	**2**		6263	701	**142**		31708	limit	limit
	457	24	**1**		6808	896	**300**		6726	limit	limit
40:	12504	296	**20**	60:	17049	7793	**3613**	80:	11489	limit	limit
	13024	25	**5**		19697	4526	**296**		18055	limit	limit
	17666	467	**56**		22181	2425	**51**		18637	19844	**925**
	21887	207	**43**		27127	**9765**	limit		22148	limit	limit
	24026	247	**14**		28147	532	**25**		22267	limit	limit
	2566	175	**5**		2987	2663	**354**		28635	limit	limit
	31352	334	**45**		4211	843	**22**		29439	limit	**1309**
	3336	241	**18**		5022	1117	**56**		3438	limit	limit
	8193	109	**9**		7774	1100	**54**		3932	limit	limit
	8614	288	**10**		8278	1514	**112**		5470	limit	limit
45:	12027	217	**13**	65:	10959	516	**33**	85:	10733	limit	limit
	12898	408	**11**		17716	limit	limit		11774	limit	limit
	14407	54	**7**		18062	**82**	117		13674	**2855**	4933
	16600	142	**47**		21384	**5062**	30571		14944	**13727**	limit
	16710	210	**21**		21907	**10709**	limit		15307	limit	limit
	19930	639	**256**		23164	4914	**182**		18788	limit	limit
	27477	3264	**116**		26400	**16494**	limit		22879	limit	limit
	4946	124	**3**		32274	7199	**681**		5367	limit	limit
	7500	1966	**90**		7291	347	**92**		7215	**14366**	limit
	965	68	**10**		8051	12029	**2613**		7602	3907	**2711**

Table B.2: CPU times for toroidal ($k \times k$) grids with uniformly distributed ± 1 weights. 'Seed' lists the random seeds used for generating the instances.

Instance		CPU time [sec]			
k	Seed	CYC	CON	CLQ	TC
5:	10441	0	**0**	0	0
	17970	0	**0**	0	0
	18923	0	**0**	0	0
	2188	0	**0**	**0**	**0**
	22776	0	0	**0**	0
	29184	0	0	**0**	0
	466	0	**0**	0	**0**
	5589	0	0	0	**0**
	8950	0	0	**0**	**0**
	9477	0	0	0	**0**
6:	10064	1	2	2	**1**
	12717	1	1	**1**	1
	19137	3	3	3	**3**
	25533	0	0	0	**0**
	2676	**18**	19	60	116
	27220	1	**0**	0	0
	27526	2	**2**	2	2
	28027	1	**1**	1	1
	5368	1	**0**	1	1
	7798	1	1	1	**1**
7:	11280	**5**	6	6	6
	18835	5	5	6	**5**
	19609	16	**12**	68	15
	20848	30	**21**	94	31
	21148	3	3	3	**3**
	26425	40	**22**	162	69
	26752	6	5	**4**	4
	27585	**2**	2	2	2
	29416	5	5	4	**4**
	32731	95	**71**	433	150
8:	10303	11	11	10	**9**
	10856	46	**36**	75	48
	14867	210	**122**	632	138
	16663	125	**50**	251	154
	19806	62	**47**	385	73

Instance		CPU time [sec]			
k	Seed	CYC	CON	CLQ	TC
8:	31605	**19**	19	22	19
	5962	72	**47**	106	92
	6720	**18**	21	24	20
	7186	52	58	97	**52**
	9278	1321	**458**	2930	1261
9:	1042	**40**	49	54	45
	11330	1808	**840**	4529	1490
	12787	limit	limit	limit	limit
	14098	5523	**3030**	16997	4004
	19340	1332	**663**	2760	1282
	22851	116	87	**86**	92
	5782	6930	**3739**	12638	4105
	6863	**20**	31	31	30
	70	1686	**1432**	8071	2052
	9221	153	**87**	289	119
10:	20294	4405	**1905**	14172	2418
	21289	4102	**1471**	8196	2562
	22003	11813	**9461**	25396	10244
	23646	limit	21839	limit	**20465**
	2498	1153	1100	2207	**929**
	26472	11056	**4371**	10173	7262
	3168	8083	**3690**	21456	4850
	3205	2586	**1204**	9136	2154
	32569	limit	limit	limit	**34661**
	8840	27466	**19281**	limit	25674
11:	10509	644	509	644	**458**
	13445	limit	limit	limit	limit
	18545	limit	limit	limit	limit
	19131	16161	**8815**	limit	10902
	20650	limit	limit	limit	limit
	24507	limit	limit	limit	limit
	28409	limit	limit	limit	limit
	31506	limit	limit	limit	limit
	32307	9578	**3980**	10121	4144
	6259	limit	limit	limit	limit

Table B.3: CPU times for toroidal ($k \times k \times k$) grids with Gaussian distributed integral weights. 'Seed' lists the random seeds used for generating the instances.

Instance		CPU time [sec]		Instance		CPU time [sec]		Instance		CPU time [sec]	
k	Seed	CYC	CON	k	Seed	CYC	CON	k	Seed	CYC	CON
30:	11495	2	**0**	50:	10929	67	**4**	70:	11214	158	**29**
	13235	4	**0**		11184	43	**5**		13627	208	**46**
	13605	2	**0**		16132	96	**19**		16057	313	**35**
	13937	2	**0**		16361	69	**10**		18089	204	**25**
	14964	2	**0**		1813	89	**19**		18936	410	**143**
	20209	4	**0**		25245	48	**6**		22569	258	**25**
	22312	3	**0**		32072	77	**8**		28255	204	**25**
	3934	4	**0**		3735	51	**5**		28683	162	**18**
	7382	4	**0**		7352	42	**2**		31804	303	**30**
	8289	10	**1**		7890	36	**14**		7933	166	**22**
35:	10018	11	**1**	55:	10166	106	**26**	75:	13642	567	**73**
	18452	19	**2**		12625	36	**7**		14441	390	**47**
	23988	8	**1**		18140	76	**20**		15320	226	**40**
	278	19	**1**		20590	119	**11**		19655	328	**43**
	30846	14	**4**		22691	119	**30**		223	276	**82**
	3294	7	**1**		25255	93	**14**		25415	250	**28**
	4430	3	**0**		26654	36	**4**		26342	255	**56**
	4948	11	**1**		29134	74	**17**		29001	477	**46**
	7040	9	**2**		8853	41	**5**		31682	395	**65**
	8883	4	**0**		8897	48	**7**		6115	317	**91**
40:	11761	16	**5**	60:	13540	179	**19**	80:	1080	392	**58**
	1312	17	**5**		15097	110	**26**		11442	321	**46**
	14029	19	**1**		15359	111	**10**		13072	311	**120**
	21074	14	**1**		18207	112	**15**		14351	369	**91**
	2480	9	**0**		18409	71	**10**		15964	451	**68**
	30667	18	**2**		27474	142	**31**		19703	250	**45**
	5273	13	**2**		4383	114	**25**		23390	238	**74**
	7115	26	**4**		5759	67	**7**		30813	396	**72**
	7974	9	**1**		6267	83	**12**		5645	407	**38**
	9512	17	**1**		6648	66	**7**		7281	456	**140**
45:	16083	18	**2**	65:	10361	114	**22**	85:	10215	474	**109**
	20652	21	**1**		14157	113	**13**		11179	464	**133**
	21007	38	**3**		1523	143	**12**		16125	660	**180**
	21053	27	**3**		17884	327	**54**		1717	636	**101**
	22054	53	**8**		21291	115	**46**		18149	815	**114**
	24987	15	**2**		25341	200	**40**		20858	473	**87**
	2848	18	**1**		3687	230	**32**		25370	484	**142**
	31108	66	**7**		7422	189	**69**		26323	271	**57**
	508	58	**8**		8705	68	**8**		30408	449	**98**
	8448	25	**3**		9236	249	**84**		32733	534	**131**

Table B.4: CPU times for toroidal ($k \times k$) grids with Gaussian distributed integral weights. 'Seed' lists the random seeds used for generating the instances.

Instance		CPU time [sec]	
k	Seed	CYC	CON
90:	11723	496	**126**
	11946	1013	**139**
	12169	652	**133**
	15935	885	**178**
	20135	428	**147**
	2293	1356	**326**
	26610	753	**58**
	27266	1238	**256**
	27488	770	**247**
	7832	1334	**217**
95:	10886	870	**338**
	14201	867	**270**
	19049	926	**390**
	19350	1088	**247**
	22328	773	**129**
	2869	994	**272**
	32121	701	**206**
	4824	805	**97**
	4999	969	**376**
	9806	802	**342**
100:	10150	787	**203**
	10546	1804	**756**
	10920	516	**284**
	16990	1299	**491**
	22293	862	**186**
	23610	904	**224**
	25328	930	**293**
	26671	1386	**453**
	2752	852	**223**
	704	1103	**226**
105:	12131	1148	**252**
	14469	1439	**592**
	15247	1868	**498**
	19971	2087	**737**
	24300	2559	**488**
	27880	2196	**1011**
	28964	1712	**265**
	29069	709	**326**
	392	1064	**415**
	8024	1012	**296**

Instance		CPU time [sec]	
k	Seed	CYC	CON
110:	1529	3340	**453**
	18142	770	**337**
	27948	773	**353**
	28395	1506	**683**
	28671	1665	**493**
	3596	1033	**591**
	5881	3943	**1315**
	6067	1345	**859**
	727	1613	**736**
	882	1269	**1011**
115:	13746	1186	**597**
	13903	3467	**1177**
	16498	2274	**546**
	17921	1960	**810**
	18625	982	**291**
	20000	2700	**1086**
	25231	4592	**923**
	29545	2312	**464**
	7341	1041	**357**
	9454	1458	**743**
120:	10099	1686	**935**
	10653	4184	**1800**
	1103	1717	**700**
	13763	2681	**1242**
	21074	1728	**1217**
	22230	2641	**737**
	25846	3094	**1947**
	28174	2649	**973**
	293	**2531**	2920
	32669	3647	**1303**
125:	12182	2997	**1177**
	13065	1186	**603**
	18946	2836	**1858**
	27086	3077	**1876**
	2775	2003	**759**
	3001	1154	**431**
	31446	2254	**1083**
	31905	2754	**1442**
	32173	2285	**1191**
	385	2250	**1432**

Instance		CPU time [sec]	
k	Seed	CYC	CON
130:	11410	4957	**4027**
	13933	2401	**2115**
	16747	1795	**1075**
	20864	1459	**682**
	28780	3103	**2747**
	29336	3103	**1074**
	32559	5245	**3187**
	478	3701	**1545**
	7819	4013	**1359**
	8096	3572	**1213**
135:	11824	4171	**2158**
	21560	1562	**538**
	21999	6468	**3233**
	22415	4441	**2464**
	23518	1775	**1035**
	2555	**2868**	6967
	32098	6814	**4320**
	3419	3028	**1573**
	7720	5158	**2333**
	8013	3443	**2455**
140:	12683	2544	**1458**
	13278	3598	**2030**
	14073	4710	**2310**
	14599	1972	**1342**
	1693	5146	**1680**
	25499	10670	**3329**
	26255	7105	**2370**
	28779	4141	**2032**
	29164	5459	**2853**
	6552	11814	**2672**
145:	142	6403	**3048**
	15685	4514	**2050**
	25005	8876	**3078**
	25176	5857	**3483**
	32432	8837	**4757**
	333	3342	**1905**
	4141	5502	**2578**
	6133	6368	**3051**
	6342	**3201**	3747
	7961	6768	**6288**

Table B.5: CPU times for toroidal ($k \times k$) grids with Gaussian distributed integral weights (cont.).

Instance		CPU time [sec]		Instance		CPU time [sec]	
k	Seed	CYC	CON	k	Seed	CYC	CON
150:	10454	**4931**	5295	170:	13142	11448	**6676**
	15681	5124	**4247**		14527	**4548**	6043
	15866	5210	**4629**		17386	12288	**6678**
	21662	**6210**	8581		17832	8619	**7036**
	22332	**3225**	9211		20102	10034	**7848**
	23695	9088	**7224**		25241	10079	**9412**
	25116	9604	**4469**		27354	**12817**	17937
	2701	4634	**3483**		2775	**10534**	12115
	27114	5418	**4354**		32592	14373	**12126**
	27690	4506	**2563**		4682	**14587**	15379
155:	10714	**3527**	6484	175:	14332	14706	**9590**
	13010	9056	**6200**		16955	10773	**10303**
	14703	10005	**8754**		17232	**9651**	12139
	1907	7294	**3188**		19664	7429	**4908**
	22800	9160	**5284**		23799	11448	**6780**
	2715	9234	**1402**		31869	**7818**	8414
	7110	6799	**4897**		32351	limit	**22750**
	8419	**6294**	15626		4962	12006	**8030**
	8459	8616	**5576**		5734	**12856**	12879
	9522	**7839**	8736		9239	10762	**8326**
160:	1190	6432	**4193**	180:	11722	**7343**	14066
	11993	5312	**1964**		11727	**12152**	12898
	18065	6720	**5685**		11732	**12520**	31134
	18207	8114	**4849**		15134	15861	**11434**
	18400	**14124**	20912		2599	18530	**14852**
	30336	14431	**11071**		2640	**24974**	35080
	30669	4573	**4033**		28836	10463	**9767**
	32286	5328	**3719**		31309	**15951**	limit
	5331	4927	**4877**		31650	**8491**	26801
	5860	6139	**3179**		5926	**21715**	31131
165:	14695	13855	**13710**	185:	12892	**14056**	20036
	19582	8784	**5683**		13546	29979	**26204**
	20233	**6805**	16319		13900	**19476**	24808
	22910	8128	**5759**		25873	**21303**	limit
	26169	9458	**8339**		26688	**10967**	14909
	27124	**6943**	9589		31133	**14626**	26184
	7043	7584	**6331**		5365	**26400**	28270
	9	**6183**	7318		6088	**20601**	limit
	9082	8625	**6421**		8487	14400	**10770**
	9127	**6400**	7068		8863	**10031**	19130

Table B.6: CPU times for toroidal ($k \times k$) grids with Gaussian distributed integral weights (cont.).

Instance	CPU time [sec]			
	CYC	CON	CLQ	TC
be120.3.1	5277	**3059**	11295	5532
be120.3.2	1827	**686**	2971	1677
be120.3.3	4505	**2543**	9176	3940
be120.3.4	2443	**1141**	4609	2153
be120.3.5	3389	**1712**	7350	3714
be120.3.6	2500	**1229**	5666	2618
be120.3.7	689	**95**	100	107
be120.3.8	1757	**792**	2082	1366
be120.3.9	limit	**26827**	limit	limit
be120.3.10	5094	**3317**	11906	5365
be250.1	9857	**4536**	10603	4989
be250.2	limit	limit	limit	limit
be250.3	30020	18721	limit	**18297**
be250.4	13377	**5855**	14525	8163
be250.5	limit	limit	limit	limit
be250.6	limit	limit	limit	limit
be250.7	10898	**4370**	16399	5574
be250.8	limit	limit	limit	limit
be250.9	limit	limit	limit	limit
be250.10	limit	limit	limit	limit
beasley50.1	0	**0**	**0**	**0**
beasley50.2	0	**0**	0	0
beasley50.3	0	0	**0**	0
beasley50.4	0	0	**0**	**0**
beasley50.5	0	**0**	**0**	0
beasley50.6	0	**0**	0	**0**
beasley50.7	0	**0**	0	0
beasley50.8	0	**0**	0	0
beasley50.9	0	**0**	**0**	**0**
beasley50.10	0	0	0	**0**
beasley100.1	5	**0**	**0**	0
beasley100.2	3	**0**	**0**	**0**
beasley100.3	3	**0**	**0**	**0**
beasley100.4	3	**0**	0	**0**
beasley100.5	3	0	0	**0**
beasley100.6	9	**0**	0	1
beasley100.7	4	0	**0**	**0**
beasley100.8	3	**0**	**0**	**0**
beasley100.9	2	**0**	0	0
beasley100.10	2	**0**	**0**	**0**
beasley250.1	18297	**10341**	limit	11526
beasley250.2	limit	limit	limit	limit
beasley250.3	7592	**2714**	7905	3041

Instance	CPU time [sec]			
	CYC	CON	CLQ	TC
beasley250.4	limit	limit	limit	limit
beasley250.5	6996	2331	2737	**2201**
beasley250.6	limit	limit	limit	limit
beasley250.7	limit	23769	limit	**23561**
beasley250.8	limit	limit	limit	limit
beasley250.9	limit	**29894**	limit	35494
beasley250.10	limit	limit	limit	limit
gka1a	0	**0**	**0**	**0**
gka2a	0	0	**0**	0
gka3a	0	0	**0**	**0**
gka4a	0	**0**	**0**	**0**
gka5a	0	0	0	**0**
gka6a	0	**0**	**0**	0
gka7a	0	**0**	**0**	**0**
gka8a	0	**0**	**0**	**0**
gka1b	0	**0**	0	532
gka2b	3	**1**	3	2410
gka3b	12	**4**	14	5610
gka4b	45	**12**	62	10449
gka5b	119	**29**	104	15217
gka6b	298	**74**	244	22709
gka7b	674	**223**	664	31338
gka8b	1500	**402**	1470	limit
gka9b	2971	**1202**	3169	limit
gka10b	13542	**9235**	17842	limit
gka1c	3	0	**0**	0
gka2c	9	0	0	**0**
gka3c	6	0	**0**	0
gka4c	9	**0**	0	0
gka5c	5	0	**0**	0
gka6c	1	**0**	0	0
gka7c	2	**0**	**0**	**0**
gka1d	3	0	**0**	**0**
gka2d	168	**67**	244	138
gka3d	361	**166**	511	188
gka4d	2980	**2094**	9590	4303
gka5d	limit	limit	limit	limit
gka6d	29047	**23959**	limit	limit
gka7d	limit	limit	limit	limit
gka8d	limit	limit	limit	limit
gka9d	limit	limit	limit	limit
gka10d	limit	limit	limit	limit

Table B.7: CPU times for quadratic 0/1 optimization problems.

Instance	Rel. gap [%] CYC	CON	CLQ	TC
be100.1	20.9	**17.6**	18.2	23.9
be100.2	31.0	**27.9**	28.8	34.8
be100.3	27.7	**23.6**	25.0	31.3
be100.4	25.5	**21.4**	22.9	27.8
be100.5	39.2	**34.1**	35.9	42.5
be100.6	30.9	**27.3**	28.5	33.4
be100.7	30.1	**26.1**	28.1	32.5
be100.8	30.4	**26.9**	29.0	34.8
be100.9	51.4	**46.3**	49.3	57.2
be100.10	44.0	**39.7**	41.3	47.9
be120.8.1	43.0	**39.2**	41.9	42.7
be120.8.2	38.8	**35.5**	38.0	38.8
be120.8.3	37.9	**34.7**	36.7	38.0
be120.8.4	31.6	**28.0**	29.4	32.0
be120.8.5	27.0	**23.8**	25.6	26.7
be120.8.6	40.7	**36.7**	39.1	40.4
be120.8.7	29.7	**27.0**	28.7	30.2
be120.8.8	39.4	**36.2**	39.0	40.9
be120.8.9	40.4	**36.7**	39.4	40.7
be120.8.10	35.5	**31.1**	33.5	35.9
be150.3.1	3.4	**2.6**	4.2	3.3
be150.3.2	4.9	**4.1**	5.9	5.1
be150.3.3	2.8	**1.9**	3.7	2.4
be150.3.4	1.6	**0.9**	2.3	1.5
be150.3.5	5.8	**5.4**	7.1	5.8
be150.3.6	7.9	**7.7**	9.9	8.7
be150.3.7	4.3	**4.0**	5.7	4.7
be150.3.8	6.7	**6.2**	8.3	7.0
be150.3.9	16.7	**16.6**	18.6	16.8
be150.3.10	5.4	**4.7**	6.7	5.6
be150.8.1	76.4	60.2	**56.9**	58.4
be150.8.2	67.2	64.3	**60.9**	62.4
be150.8.3	51.5	51.7	**50.1**	50.8
be150.8.4	**61.9**	63.6	62.8	64.0
be150.8.5	**52.1**	54.2	53.5	54.3
be150.8.6	53.2	53.3	52.7	**52.3**
be150.8.7	50.6	48.4	48.0	**48.0**
be150.8.8	52.0	52.9	52.7	**51.1**
be150.8.9	**63.9**	66.3	65.8	65.7
be150.8.10	**54.5**	**54.5**	54.5	54.7

Instance	Rel. gap [%] CYC	CON	CLQ	TC
be200.3.1	32.2	22.1	23.7	**21.8**
be200.3.2	29.1	20.0	21.6	**19.7**
be200.3.3	12.5	12.1	13.4	**11.6**
be200.3.4	13.6	13.2	14.5	**12.8**
be200.3.5	16.0	15.4	17.1	**15.3**
be200.3.6	18.8	18.4	19.9	**18.2**
be200.3.7	9.1	**8.4**	9.9	8.5
be200.3.8	16.2	15.4	17.2	**14.9**
be200.3.9	19.8	19.1	21.1	**18.8**
be200.3.10	22.6	21.8	24.0	**21.4**
be200.8.1	98.6	77.9	99.8	**75.2**
be200.8.2	**125.3**	149.1	147.6	146.1
be200.8.3	**115.8**	144.0	148.2	148.0
be200.8.4	**115.4**	142.6	144.1	146.6
be200.8.5	**120.7**	152.0	155.8	158.6
be200.8.6	**95.1**	117.3	119.7	116.8
be200.8.7	**100.1**	130.8	134.9	132.1
be200.8.8	**107.3**	134.8	142.8	135.0
be200.8.9	**109.9**	142.7	143.1	140.2
be200.8.10	**112.4**	143.9	144.0	150.3
beasley500.1	88.9	30.8	30.3	**30.3**
beasley500.2	80.1	22.4	21.7	**21.6**
beasley500.3	79.8	23.7	22.9	**22.7**
beasley500.4	73.3	25.0	24.3	**24.1**
beasley500.5	80.2	25.4	24.8	**24.5**
beasley500.6	86.3	27.2	27.0	**26.4**
beasley500.7	86.7	27.7	27.4	**26.6**
beasley500.8	82.4	29.0	28.3	**27.7**
beasley500.9	87.5	**28.6**	28.7	28.7
beasley500.10	74.2	**22.4**	22.9	23.5
gka1e	**0**	**0**	**0**	**0**
gka2e	6.8	**6.2**	7.4	6.2
gka3e	17.1	**17.0**	18.3	17.0
gka4e	28.1	18.2	19.5	**17.7**
gka5e	64.1	44.7	46.8	**42.8**
gka1f	81.0	27.3	**27.3**	27.4
gka2f	156.4	89.9	**89.0**	89.4
gka3f	**263.8**	348.8	351.2	350.3
gka4f	**349.1**	521.4	521.8	514.2
gka5f	**444.3**	672.7	673.9	662.3

Table B.8: Relative gaps for quadratic 0/1 optimization problems after ten hours CPU time.

Instance	CPU time [sec]			
	CYC	CON	CLQ	TC
g05_60.0	limit	—[a]	**6089**	13014
g05_60.1	limit	—[a]	**7330**	22177
g05_60.2	limit	—[a]	**17760**	limit
g05_60.3	limit	—[a]	**2864**	22492
g05_60.4	limit	—[a]	limit	limit
g05_60.5	limit	—[a]	**9494**	19342
g05_60.6	limit	—[a]	**15219**	limit
g05_60.7	limit	—[a]	**7633**	17601
g05_60.8	limit	—[a]	**18834**	limit
g05_60.9	limit	—[a]	**11419**	22269
pm1s_80.0	1	1	**0**	**0**
pm1s_80.1	3	**2**	6	3
pm1s_80.2	81	**68**	202	148
pm1s_80.3	6	**6**	14	10
pm1s_80.4	4	**3**	11	4
pm1s_80.5	7	**5**	18	7
pm1s_80.6	30	**30**	60	51
pm1s_80.7	10	**8**	28	15
pm1s_80.8	**11**	11	41	15
pm1s_80.9	5	**5**	14	6
pm1s_100.0	**359**	400	1119	589
pm1s_100.1	2331	**2285**	5772	3261
pm1s_100.2	343	**341**	971	569
pm1s_100.3	**1491**	2081	4596	4743
pm1s_100.4	784	**741**	2260	1356

Instance	CPU time [sec]			
	CYC	CON	CLQ	TC
pm1s_100.5	367	**357**	1906	642
pm1s_100.6	1480	**1019**	3411	1767
pm1s_100.7	**93**	109	343	139
pm1s_100.8	227	**225**	1065	404
pm1s_100.9	**85**	93	359	138
pw01_100.0	178	**144**	1047	183
pw01_100.1	**139**	184	1000	304
pw01_100.2	**182**	208	1177	372
pw01_100.3	356	**288**	1252	406
pw01_100.4	**35**	46	172	62
pw01_100.5	**62**	63	248	84
pw01_100.6	**268**	277	961	296
pw01_100.7	443	512	1336	**395**
pw01_100.8	24	**23**	99	36
pw01_100.9	**343**	382	2490	671
w01_100.0	**44**	46	203	65
w01_100.1	**30**	32	215	43
w01_100.2	**392**	570	1891	664
w01_100.3	**67**	85	327	114
w01_100.4	33	**21**	103	36
w01_100.5	68	**67**	284	118
w01_100.6	18	**16**	57	21
w01_100.7	**75**	75	549	177
w01_100.8	6	5	5	**4**
w01_100.9	**25**	31	99	27

[a] 'Out of memory' error.

Table B.9: CPU times for `rudy`-generated instances.

Instance	CPU time [sec]			
	CYC	CON	CLQ	TC
mannino_k48	8	**5**	5	5
mannino_k487a	269	**8**	71	99
mannino_k487b	limit	**68**	1262	1232
mannino_k487c	limit	limit	limit	limit

Table B.10: CPU times for Mannino instances.

Instance	Rel. gap [%]			
	CYC	CON	CLQ	TC
g05_80.0	13.3	—[a]	**1.9**	3.1
g05_80.1	11.9	—[a]	**0.9**	1.9
g05_80.2	12.8	—[a]	**1.6**	2.7
g05_80.3	14.2	—[a]	**2.5**	4.0
g05_80.4	13.1	—[a]	**1.8**	2.8
g05_80.5	13.8	—[a]	**2.4**	3.3
g05_80.6	13.5	—[a]	**1.9**	3.1
g05_80.7	13.3	—[a]	**1.9**	3.0
g05_80.8	13.8	—[a]	**2.3**	3.5
g05_80.9	14.2	—[a]	**2.5**	3.9
g05_100.0	15.4	11.6	**4.7**	5.7
g05_100.1	15.8	11.9	12.6	**6.3**
g05_100.2	15.4	11.4	**4.4**	5.4
g05_100.3	15.9	11.9	**5.1**	5.8
g05_100.4	14.7	10.7	**4.0**	5.0
g05_100.5	15.0	11.1	**4.2**	10.6
g05_100.6	15.2	11.2	**4.4**	5.4
g05_100.7	15.4	11.4	**4.5**	5.4
g05_100.8	15.3	11.3	**4.5**	5.5
g05_100.9	15.4	11.4	**5.0**	10.8
pm1d_80.0	112.3	**78.0**	97.4	103.1
pm1d_80.1	109.8	**79.6**	95.1	97.1
pm1d_80.2	95.1	**69.0**	78.9	89.1
pm1d_80.3	88.7	**62.9**	75.6	81.8
pm1d_80.4	104.4	**74.5**	91.6	96.8
pm1d_80.5	113.2	**83.5**	97.1	103.3
pm1d_80.6	128.3	**90.7**	115.6	118.0
pm1d_80.7	107.6	**77.5**	95.6	98.8
pm1d_80.8	90.4	**65.2**	78.5	83.6
pm1d_80.9	102.7	**73.3**	89.5	95.3
pm1d_100.0	136.2	**121.8**	127.1	130.0
pm1d_100.1	142.6	**126.5**	134.6	136.7
pm1d_100.2	119.3	**105.7**	107.7	112.6
pm1d_100.3	114.8	**101.8**	**101.8**	108.8
pm1d_100.4	125.6	**112.7**	117.6	119.3
pm1d_100.5	100.5	**90.5**	93.4	95.5
pm1d_100.6	120.4	**110.4**	113.1	114.2
pm1d_100.7	120.5	**107.2**	113.3	115.0
pm1d_100.8	116.1	**103.9**	109.1	111.2
pm1d_100.9	111.4	**99.5**	104.7	105.7

Instance	Rel. gap [%]			
	CYC	CON	CLQ	TC
pw05_100.0	9.7	8.5	**3.9**	5.0
pw05_100.1	9.4	8.3	**3.6**	4.8
pw05_100.2	9.2	8.0	**3.4**	4.6
pw05_100.3	9.1	7.9	**3.2**	4.4
pw05_100.4	9.6	8.5	**3.8**	4.8
pw05_100.5	9.1	7.7	8.9	**4.4**
pw05_100.6	9.2	8.3	**3.7**	4.7
pw05_100.7	8.8	7.7	**3.3**	4.4
pw05_100.8	8.8	7.9	**3.0**	4.2
pw05_100.9	9.6	8.6	**3.7**	4.8
pw09_100.0	19.0	18.0	17.3	**17.2**
pw09_100.1	18.9	17.7	**16.1**	17.6
pw09_100.2	19.0	18.1	**16.1**	17.9
pw09_100.3	18.8	17.8	**17.1**	17.6
pw09_100.4	18.9	17.9	**16.7**	17.7
pw09_100.5	19.2	18.1	**16.1**	17.6
pw09_100.6	19.3	18.1	**16.3**	17.8
pw09_100.7	19.4	18.2	**17.3**	17.8
pw09_100.8	18.9	17.9	**15.6**	17.4
pw09_100.9	18.8	17.8	17.4	**17.3**
w05_100.0	37.7	**22.7**	28.8	25.0
w05_100.1	39.4	**22.7**	30.6	23.8
w05_100.2	29.3	**16.7**	22.7	19.0
w05_100.3	39.3	**23.1**	30.9	25.1
w05_100.4	39.3	**23.3**	30.0	26.4
w05_100.5	40.0	**23.1**	31.2	26.2
w05_100.6	41.9	**23.0**	33.0	25.8
w05_100.7	27.2	**14.1**	20.3	16.1
w05_100.8	47.9	**27.7**	38.1	30.7
w05_100.9	29.2	**15.1**	21.1	17.8
w09_100.0	85.1	**69.5**	73.4	77.6
w09_100.1	88.9	**72.7**	77.7	81.3
w09_100.2	67.3	**54.6**	56.8	61.1
w09_100.3	94.6	**77.8**	81.5	85.1
w09_100.4	87.2	**71.0**	75.8	79.1
w09_100.5	68.4	**54.7**	58.2	60.8
w09_100.6	77.6	**62.5**	67.3	69.7
w09_100.7	78.6	**63.4**	68.3	71.3
w09_100.8	92.1	**74.4**	79.8	83.0
w09_100.9	87.9	**71.6**	76.4	80.4

[a] 'Out of memory' error.

Table B.11: Relative gaps for `rudy`-generated instances after ten hours CPU time.

List of Algorithms

List of Figures

List of Tables

References

[ABCC01] A. Applegate, R. Bixby, C. Chvátal, and W. Cook. TSP cuts which do not conform to the template paradigm. In M. Jünger and D. Naddef, editors, *Computational Combinatorial Optimization: Optimal or Provably Near-Optimal Solutions,* volume 2241 of *Lecture Notes in Computer Science*, pages 261–304. Springer-Verlag, 2001.

[Bar81] F. Barahona. Balancing signed toroidal graphs in polynomial time. Departamento de Matemáticas, Universidad de Chile, 1981.

[Bar82] F. Barahona. On the computational complexity of Ising spin glass models. *Journal of Physics A: Mathematical and General*, 15(10):3241–3253, 1982.

[Bar83] F. Barahona. The max cut problem in graphs not contractible to K_5. *Operations Research Letters*, 2:107–111, 1983.

[Bar93] F. Barahona. On cuts and matchings in planar graphs. *Mathematical Programming*, 60:53–68, 1993.

[BE07] A. Billionnet and S. Elloumi. Using a mixed integer quadratic programming solver for the unconstrained quadratic 0-1 problem. *Mathematical Programming*, 109(1, Series A):55–68, 2007.

[Bea98] J.E. Beasley. Heuristic algorithms for the unconstrained binary quadratic programming problem. Technical report, The Management School, Imperial College, London SW7 2AZ, England, 1998.

[BGJR88] F. Barahona, M. Grötschel, M. Jünger, and G. Reinelt. An application of combinatorial optimization to statistical physics and circuit layout design. *Operations Research*, 36(3):493–513, 1988.

[BGM85] F. Barahona, M. Grötschel, and A.R. Mahjoub. Facets of the bipartite subgraph polytope. *Mathematics of Operations Research*, 10(2):340–358, 1985.

[BH93] E. Boros and P.L. Hammer. Cut-polytopes, boolean quadric polytopes and nonnegative quadratic pseudo-boolean functions. *Mathematics of Operations Research*, 18:245–253, 1993.

[BLO08] C. Buchheim, F. Liers, and M. Oswald. Local cuts revisited. *Operations Research Letters*, 36(4):430–433, 2008.

[BM58] G.E.P. Box and M.E. Muller. A Note on the Generation of Random Normal Deviates. *Annals of Mathematical Statistics*, 29:610–611, 1958.

[BM86] F. Barahona and A.R. Mahjoub. On the cut polytope. *Mathematical Programming*, 36:157–173, 1986.

[Brø83] A. Brøndsted. *An Introduction to Convex Polytopes*, volume 90 of *Graduate Texts in Mathematics*. Springer-Verlag Berlin Heidelberg New York, 1983.

[Chr95] T. Christof. SMAPO: a library of linear descriptions of SMAll combinatorial POlytopes, 1995.
http://comopt.ifi.uni-heidelberg.de/software/SMAPO/index.html.

[CKC83] R.-W. Chen, Y. Kajitani, and S.-P. Chan. A graph-theoretic via minimization algorithm for two-layer printed circuit boards. *IEEE Transactions on Circuits and Systems*, 30(5):284–299, 1983.

[CLRS09] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2009.

[CR97] T. Christof and G. Reinelt. Efficient parallel facet enumeration for 0/1-polytopes. Technical report, University of Heidelberg, Germany, 1997.

[Dan51] G.B. Dantzig. Maximization of a linear function of variables subject to linear inequalities. In T.C. Koopmans, editor, *Activity analysis of production and allocation*, pages 339–347. John Wiley & Sons, New York, 1951.

[DDJ+95] C. De Simone, M. Diehl, M. Jünger, P. Mutzel, G. Reinelt, and G. Rinaldi. Exact ground states of Ising spin glasses: New experimental results with a branch and cut algorithm. *Journal of Statistical Physics*, 80:487–496, 1995.

[DDJ+96] C. De Simone, M. Diehl, M. Jünger, P. Mutzel, G. Reinelt, and G. Rinaldi. Exact ground states of two-dimensional $\pm J$ Ising spin glasses. *Journal of Statistical Physics*, 84:1363–1371, 1996.

[Die05] R. Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics*. Springer-Verlag Berlin Heidelberg New York, 2005.

[DL97] M.M. Deza and M. Laurent. *Geometry of Cuts and Metrics*, volume 15 of *Algorithms and Combinatorics*. Springer-Verlag, Berlin, 1997.

[DO72] Y.G. Dorfman and G.I. Orlova. Finding the maximal cut in a graph. *Engineering Cybernetics*, 10:502–506, 1972.

[DR94] C. De Simone and G. Rinaldi. A cutting plane algorithm for the max-cut problem. *Optimization Methods and Software*, 3:195–214, 1994.

[Dre86] A. Dress. Computing spin-glass Hamiltonians. Preprint, Universität Bielefeld, 1986.

[Dur64] R. Durstenfeld. Algorithm 235: Random permutation. *Communications of the ACM*, 7(7):420, 1964.

[EGJR01] M. Elf, C. Gutwenger, M. Jünger, and G. Rinaldi. Branch-and-Cut Algorithms for Combinatorial Optimization and Their Implementation in ABACUS. In M. Jünger and D. Naddef, editors, *Computational Combinatorial*

Optimization: Optimal or Provably Near-Optimal Solutions, volume 2241 of *Lecture Notes in Computer Science*, pages 157–222. Springer-Verlag, 2001.

[EJ73] J. Edmonds and E.L. Johnson. Matching, Euler tours, and the Chinese postman. *Mathematical Programming*, 5:88–124, 1973.

[FY38] R.A. Fisher and F. Yates. *Statistical Tables for Biological, Agricultural and Medical Research*. Oliver and Boyd, London, 1938.

[Ger85] B. Gerards. Testing the Odd Bicycle Wheel Inequalities for the Bipartite Subgraph Polytope. *Mathematics of Operations Research*, 10(2):359–360, 1985.

[GJ79] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, New York, 1979.

[GJR84] M. Grötschel, M. Jünger, and G. Reinelt. A Cutting Plane Algorithm for the Linear Ordering Problem. *Operations Research*, 32(6):1195–1220, 1984.

[GJR87] Martin Grötschel, Michael Jünger, and Gerhard Reinelt. Calculating Exact Ground States of Spin Glasses: A Polyhedral Approach. In J.L. van Hemmen and I. Morgenstern, editors, *Proceedings of the Heidelberg Colloquium on "Glassy Dynamics",* volume 275 of *Lecture Notes in Physics*, pages 325–353. Springer, 1987.

[GJS76] M.R. Garey, D.S. Johnson, and L. Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1:237–267, 1976.

[GKA98] F. Glover, G. Kochenberger, and B. Alidaee. Adaptive memory tabu search for binary quadratic programs. *Management Science*, 44(3):336–345, 1998.

[GL99] A. Galluccio and M. Loebl. On the Theory of Pfaffian Orientations. II. T-joins, k-cuts, and Duality of Enumeration. *Electronic Journal of Combinatorics*, 6, 1999.

[GLS93] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*, volume 2 of *Algorithms and Combinatorics*. Springer-Verlag Berlin Heidelberg, 1993.

[GN84] M. Grötschel and G.L. Nemhauser. A polynomial algorithm for the max-cut problem on graphs without long odd cycles. *Mathematical Programming*, 29:28–40, 1984.

[GP81] M. Grötschel and W.R. Pulleyblank. Weakly Bipartite Graphs and the Max-Cut Problem. *Operations Research Letters*, 1(1):23–27, 1981.

[Gri90] V.P. Grishukhin. All facets of the cut cone C_n for n = 7 are known. *European Journal of Combinatorics*, 11(2):115–117, 1990.

[GW95] M.X. Goemans and D.P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM*, 42(6):1115–1145, 1995.

[GY04] J.L. Gross and J. Yellen, editors. *Handbook of Graph Theory*. Discrete Mathematics and its Applications. CRC Press, 2004.

[Had75] F.O. Hadlock. Finding a maximum cut of a planar graph in polynomial time. *SIAM Journal on Computing*, 4:221–225, 1975.

[Ham65] P.L. Hammer. Some network flow problems solved with pseudo-boolean programming. *Operations Research*, 13:388–399, 1965.

[Han79] P. Hansen. Methods of nonlinear 0-1 programming. *Annals of Discrete Mathematics*, 5:53–70, 1979.

[Hel00] C. Helmberg. *Semidefinite Programming for Combinatorial Optimization*. Professorial dissertation, TU Berlin, 2000.

[HR95] C. Helmberg and F. Rendl. Solving quadratic (0,1)-problems by semidefinite programming and cutting planes. Technical report, Institut für Mathematik, Technische Universität Graz, Graz, Austria, 1995.

[HRVW96] C. Helmberg, F. Rendl, R.J. Vanderbei, and H. Wolkowicz. An interior-point method for semidefinite programming. *SIAM Journal on Optimization*, 6(2):342–361, 1996.

[Ilo02] Ilog. Cplex 8.1, 2002. By ILOG S.A., 9 Rue de Verdun, 94253 Gentilly Cedex, France, http://www.ilog.com/products/cplex.

[JT98] M. Jünger and S. Thienel. Introduction to ABACUS – a branch-and-cut system. *Operations Research Letters*, 22:83–95, 1998.

[JT00] M. Jünger and S. Thienel. The ABACUS System for Branch-and-Cut-and-Price Algorithms in Integer Programming and Combinatorial Optimization. *Software: Practice and Experience*, 30:1325–1352, 2000.

[Kar72] R.M. Karp. Reducibility among combinatorial problems. In R.E. Miller and J.W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[KL70] B.W. Kernighan and S. Lin. An Efficient Heuristic Procedure for Partitioning Graphs. *The Bell System Technical Journal*, 49(1):291–307, 1970.

[KM72] V. Klee and G.J. Minty. How Good is the Simplex Algorithm? In O. Sisha, editor, *Inequalities, Volume III*, pages 159–175. Academic Press, New York, 1972.

[KM06] K. Krishnan and J.E. Mitchell. A Semidefinite Programming Based Polyhedral Cut and Price Approach for the Maxcut Problem. *Computational Optimization and Applications*, 33(1):51–71, 2006.

[Knu97] D.E. Knuth. *Seminumerical Algorithms*, volume 2 of *The Art of Computer Programming*. Addison-Wesley, 1997.

[LP95] M. Laurent and S. Poljak. One-third-integrality in the max-cut problem. *Mathematical Programming*, 71:29–50, 1995.

[Mil76] P. Miliotis. Integer programming approaches to the traveling salesman problem. *Mathematical Programming*, 10:367–378, 1976.

[NW99] G.L. Nemhauser and L.A. Wolsey. *Integer and Combinatorial Optimization*. Interscience Series in Discrete Mathematics and Optimization. John Wiley & Sons, 1999.

[Pin84] R.Y. Pinter. Optimal layer assignment for interconnect. *Advances in VLSI and Computer Systems*, 1(2):123–137, 1984.

[PR74] J.C. Picard and H.D. Ratliff. Minimum cuts and related problems. *Networks*, 5(4):357–370, 1974.

[PR87] M.W. Padberg and G. Rinaldi. Optimization of a 532 City Symmetric Traveling Salesman Problem by Branch and Cut. *Operations Research Letters*, 6:1–7, 1987.

[PR90] P.M. Pardalos and G.P. Rodgers. Computational aspects of a branch and bound algorithm for quadratic zero-one programming. *Computing*, 45(2):131–144, 1990.

[PR91] M.W. Padberg and G. Rinaldi. A Branch-and-Cut Algorithm for the Resolution of Large-Scale Traveling Salesman Problems. *SIAM Review*, 33:60–100, 1991.

[PS00] G. Pataki and S.H. Schmieta. The DIMACS library of mixed semidefinite-quadratic-linear programs, 2000.
http://dimacs.rutgers.edu/Challenges/Seventh/Instances/.

[PT94] S. Poljak and Z. Tuza. The max-cut problem—A survey. Technical report, Institute of Mathematics, Academia Sinica, Nankang, Taipei, Taiwan, 1994.

[Rei94] G. Reinelt. *The Traveling Salesman: Computational Solutions for TSP Applications*, volume 840 of *Lecture Notes in Computer Science*. Springer-Verlag, 1994.

[Rei07] G. Reinelt. Effiziente Algorithmen 2. Lecture Notes, 2007.

[Rin98] G. Rinaldi. Rudy, 1998.
http://www-user.tu-chemnitz.de/~helmberg/rudy.tar.gz.

[Rin10] G. Rinaldi. Personal communication, 2010.

[RRW10] F. Rendl, G. Rinaldi, and A. Wiegele. Solving Max-Cut to Optimality by Intersecting Semidefinite and Polyhedral Relaxations. *Mathematical Programming*, 121(2):307–335, 2010.

[RZ00] T. Regge and R. Zecchina. Combinatorial and topological approach to the 3D Ising model. *Journal of Physics A: Mathematical and General*, 33(4):741–761, 2000.

[Sch03] A. Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*, volume 24 of *Algorithms and Combinatorics*. Springer-Verlag Berlin Heidelberg, 2003.

[ST04] D.A. Spielman and S.-H. Teng. Smoothed Analysis Of Algorithms: Why The Simplex Algorithm Usually Takes Polynomial Time. *Journal of the ACM*, 51(3):385–463, 2004.

[Thi95] S. Thienel. *ABACUS: A Branch-And-CUt System*. PhD thesis, University of Cologne, 1995.

[Wag37] K. Wagner. Über eine Erweiterung des Satzes von Kuratowski. *Deutsche Mathematik*, 2:280–285, 1937.

[Wie06] A. Wiegele. *Nonlinear optimization techniques applied to combinatorial optimization problems*. PhD thesis, Alpen-Adria University of Klagenfurt, 2006.

[Wie07] A. Wiegele. Biq Mac Library, 2007.
http://biqmac.uni-klu.ac.at/biqmaclib.html.

[Yan78] M. Yannakakis. Node- and edge-deletion NP-complete problems. In *Proceedings of the 10th Annual ACM Symposium on Theory of Computing*, pages 253–264. Association for Computing Machinery, New York, 1978.

[Zie06] G.M. Ziegler. *Lectures on Polytopes*, volume 152 of *Graduate Texts in Mathematics*. Springer-Verlag Berlin Heidelberg New York, 2006.

Symbols and Notations

The entries below are divided into the following groups: We start with basic mathematical operators and relations. The subsequent group covers sets, spaces, and vectors as well as associated fundamental operations. The next three groups are related to complexity theory, graph theory, and polyhedral theory, respectively, in this order. Finally, we have the entries that come from combinatorial optimization in general as well as from the theory of the max-cut problem and the shrink separation in particular.

$\deg(v)$	Degree of a node v.	4
$\delta(U)$	Set of edges with precisely one end in U.	4
$(U : W)$	Set of edges with one end in U and the other one in W.	4
$\mathrm{aff}(X)$	Affine hull of a set X.	6
$\mathrm{cone}(X)$	Conic hull of a set X.	6
$\mathrm{conv}(X)$	Convex hull of a set X.	6
$\dim(F)$	Dimension of a face F.	7
$(\boldsymbol{a}, \alpha)$	Inequality $\boldsymbol{a}^T\boldsymbol{x} \leq \alpha$.	7
$H(\boldsymbol{a}, \alpha)$	Hyperplane defined by the equation $\boldsymbol{a}^T\boldsymbol{x} = \alpha$.	6
$K(\boldsymbol{a}, \alpha)$	Closed halfspace defined by the inequality $\boldsymbol{a}^T\boldsymbol{x} \leq \alpha$.	6
CUT	Cut polytope.	30
MET	Semimetric polytope.	34
$s_B(\cdot)$	Switching mapping alongside a set B.	42
$\tilde{\boldsymbol{z}}$	Switched LP solution.	44
$\overline{\boldsymbol{z}}$, $\overline{G}$	Contracted LP solution and associated graph.	47
$\overline{\boldsymbol{z}}'$, $\overline{G}'$	Extended LP solution and associated graph.	53
ξ_l, ξ_u	Lower resp. upper limit of a feasible LP range.	53
$\boldsymbol{c}(F)$	Aggregate weight of a set F w.r.t. a weight vector $\boldsymbol{c}$.	11

Index