Partitioning into Isomorphic or Connected Subgraphs

Vom Fachbereich Mathematik
der Technischen Universität Darmstadt
zur Erlangung des Grades
Doctor rerum naturalium
(Dr. rer. nat.)
genehmigte Dissertation

Von

Hendrik Lüthen, M. Sc.

aus Berlin

Referent:	Prof. Dr. Marc E. Pfetsch
Korreferentin:	Prof. Dr. Frauke Liers
Tag der Einreichung:	2. Februar 2018
Tag der mündlichen Prüfung:	4. Mai 2018

Darmstadt 2018

Lüthen, Hendrik: Partitioning into Isomorphic or Connected Subgraphs
Darmstadt, Technische Universität Darmstadt
Jahr der Veröffentlichung der Dissertation auf TUprints: 2019
Tag der mündlichen Prüfung: 04.05.2018

Acknowledgments

This work would not exist without the help of many people. Hereby, I want to show my gratitude.

First of all, I want to thank my supervisor Marc Pfetsch. He gave me the opportunity to work on this topic and guided me through my research. I am grateful for his support and his attention to detail concerning, in particular, mathematical accuracy, formulation details and LaTeX.

Many thanks also go to the referee of my thesis, Frauke Liers. Her, Martin Otto and Stefan Ulbrich I want to thank for their willingness to act as members of the thesis committee.

Moreover, I am indebted to the DFG for most of my funding within the Collaborative Research Centre 666: “Integral Sheet Metal Design with Higher Order Bifurcations”.

Special thanks go to the optimization group at TU Darmstadt and the colleagues from the CRC 666. It was always a pleasure working in both groups. In particular, I want to thank Christopher Hojny for many fruitful discussions on our combined and my own research. His mind was always open. For proofreading I want to thank Johanna Biehl, Tristan Gally, Thea Göllner, Oliver Habeck, Christopher Hojny, Philip Kolvenbach, Frederic Matter and Andreas Schmitt.

Furthermore, I thank Kai Habermehl and Anja Kuttich for being such great office mates. And, of course, many thanks to my groups of card game players enjoying a quiet little game of Skat or Doppelkopf—at least, now I do not receive a “Strich” for forgetting you.

Finally, I want to thank my friends and family for their help and empathy. Foremost for their encouragement and moral support I thank my fiancée Juliane and my son Theodor.

Zusammenfassung

In dieser Arbeit geht es hauptsächlich um Partitionierungen von Graphen und den Zusammenhang in Graphen. Zuerst wird das Problem betrachtet, die Knotenmenge eines Graphen in eine vorgegebene Anzahl an Teilmengen zu partitionieren, sodass die auf diesen Knotenteilmengen induzierten Subgraphen jeweils isomorph zueinander sind. Es wird gezeigt, dass dieses Problem NP-vollständig ist. Außerdem werden verschiedene Graphenklassen betrachtet, für die diese Frage in polynomieller Zeit beantwortet werden kann. Es kann darüber hinaus gezeigt werden, dass, wenn zusätzlich gefordert wird, dass die einzelnen Subgraphen zusammenhängend sind, dieses Problem immer noch NP-vollständig ist.

Im nächsten Abschnitt wird dieser letzte Aspekt näher betrachtet. Dafür wird das Polytop definiert, welches von Inzidenzvektoren von Knotenmengen aufgespannt wird, die einen zusammenhängenden Subgraphen induzieren. Hierfür werden neue facettendefinierende Ungleichungen hergeleitet und deren allgemeine Struktur näher untersucht. Für Kreise und vollständig bipartite Graphen wird eine vollständige Beschreibung des Polytops angegeben sowie in dem Fall, dass ein Graph um genau einen Knoten und eine zusätzliche Kante erweitert wird.

Anschließend wird diese Betrachtung um eine zusätzliche Partitionierung ergänzt, das heißt, es wird der Fall betrachtet, dass ein Graph in eine vorgegebene Anzahl an Teilmengen partitioniert wird, sodass die Teilmengen jeweils zusammenhängende Subgraphen induzieren. Für das entsprechende Polytop werden die Dimension und facettendefinierende Ungleichungen hergeleitet. Diese theoretischen Betrachtungen werden ergänzt durch das sog. Connected Max-$\mathcal{K}$-Cut Problem, welches wie folgt definiert ist: Gegeben ein gewichteter Graph G und eine Zahl $\mathcal{K}$, partitioniere die Knoten von G so in $\mathcal{K}$ Teilmengen, dass die einzelnen Knotenmengen jeweils einen zusammenhängenden Graphen induzieren und der Schnitt maximal ist. Hierfür werden verschiedene Techniken vorgestellt, um den Lösungsprozess von SCIP zu beschleunigen. Numerische Resultate zeigen dann den Einfluss der einzelnen Techniken.

Aus den Problemformulierungen haben sich ganz allgemeine Fragestellungen über gemischt-ganzzahlige Formulierungen (engl. mixed-integer program MIP) ergeben. Die in der Literatur zu findenden Aussagen beschränken sich häufig auf den zulässigen Bereich und betrachten die Zielfunktion nicht. Hierfür wird eine mögliche Definition vorgestellt, welche erlaubt, MIP-Formulierungen für Probleme mit nichtlinearer Zielfunktion anzugeben. Es werden dabei Eigenschaften der Definition diskutiert sowie Voraussetzungen herausgearbeitet, die die Existenz oder Nichtexistenz bestimmter Formulierungen belegen.

Im letzten Abschnitt wird eine MIP-Formulierung für das Problem angegeben, einen Graphen so zu partitionieren, dass alle bis auf eine Teilmenge isomorph sind. Die Größe dieser nichtisomorphen Teilmenge soll dabei minimiert werden. Es wird darüber hinaus ein Verfahren angegeben, welches mittels geschickten Ausprobierens das Problem für die hier betrachteten Graphen in kurzer Zeit optimal löst. Anstatt einen Graphen in isomorphe Subgraphen zu partitionieren, wird zum Schluss das Problem betrachtet, einen Graphen in *ähnliche* Teile zu partitionieren. Hierfür wird ein Programmiergerüst vorgestellt, welches Probleme dieser Art lösen kann und die verschiedenen implementierten Arten der Ähnlichkeit werden vorgestellt.

Contents

Chapter 1

Introduction

To motivate the problems appearing in this thesis we start with a short introduction to a few topics considered in the Collaborative Research Centre 666: "Integral Sheet Metal Design with Higher Order Bifurcations" (CRC 666). A more detailed overview, especially about the engineering part, can be found for example in [47].

1.1 Manufacturing

Usually, branched sheet metal is produced in differential style, that is, by welding, gluing or similar procedures. These techniques have several disadvantages: the resulting pieces are heavier, have lower thermal conductivity, corrode easier and are more likely to break at for example the welding or gluing point. This is why the CRC 666 studies processes, which allows for an integral production of branched sheet metal, called linear flow splitting.

Linear flow splitting is a profile-rolling process applied to sheet metal at room temperature and its basic principle is depicted in the upper left corner of Figure 1.1. Flat sheet metal is directed by a tooling system consisting of two supporting rolls and two splitting rolls. The Supporting rolls ensure the motion of the sheet metal as well as a fixation. Splitting roles are used for creating branches. Their distance is smaller than the width of the unprocessed material, which leads to a material flow depicted in the upper right corner of Figure 1.1. The material flows into the space between supporting and splitting roles thereby creating a bifurcation, also called a flange in Figure 1.1.

Using many different tooling systems subsequently, each with a smaller distance between their respective splitting roles than the previous tooling system, leads to longer flanges and a decreased width of the sheet metal, which is depicted

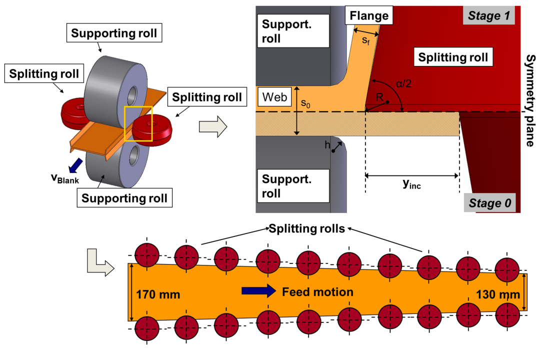


Figure 1.1: Linear flow splitting process, figure taken from [50]

in the lower part of Figure 1.1. In principle, the process can also be applied to branched sheet metal thereby creating bifurcations of a higher order.

Linear flow splitting can be combined with a roll forming process to produce for example the profile shown in Figure 1.2. Depicted is also the flower pattern to show the intermediate steps which are needed for creating the final profile.

If we first apply a roll forming process to bend the sheet metal and then use a process similar to linear flow splitting, we can create a bifurcation anywhere on the sheet metal not only on its borders. This procedure is called linear bend splitting and is depicted in Figure 1.3.

Combining these different techniques increases the number of producible profiles and thus the applicability of the process. For further details on various other techniques studied within the CRC 666 or applications of this process we refer to [47].

In this thesis we assume that we are given a profile, which should be produced using the mentioned procedures. In particular, we are not dealing with the optimization of a profile given a load case scenario, which can for example be found in [53, 55, 61]. The profile, which we want to produce may be given by a cross-section similar to the one found in Figure 1.4 (left). We translate this profile to a graph as depicted on the right-hand side of Figure 1.4. The graph is created by introducing a node for every intersection or end point. Therefore, the edges represent the material in the profile. This translation of a profile to a graph was already used in [52], where also a weight function was added to the edges, representing the thickness of the corresponding material.

An intersection in the profile can be the result of different production steps.

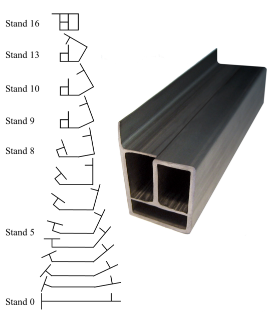


Figure 1.2: Picture of a profile produced on the integrated production line and its flower pattern, figure taken from [48]

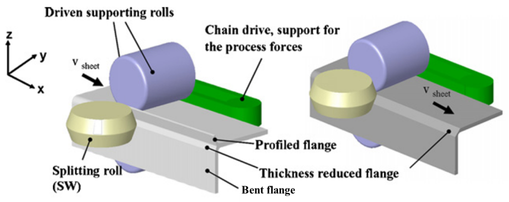


Figure 1.3: Linear bend splitting process, figure taken from [49]

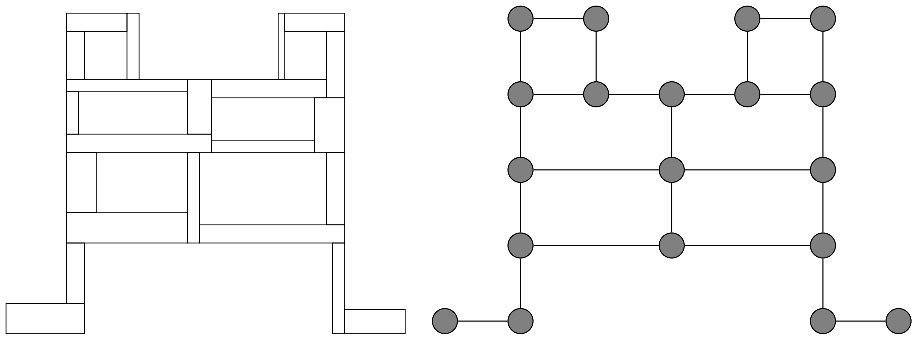


Figure 1.4: Sketch of the cross-section of an exemplary profile (left) and its derived graph (right), figures taken from [47]

Inspect for example an intersection of degree 3, meaning the meeting of three line segments or a node of degree 3 in the corresponding graph. There are ten different possibilities to produce this intersection: two joining operations, splitting material on either side or bending one side and joining the third. For a more detailed overview we refer to [52]. The different production steps also lead to different mechanical behavior of the resulting intersection. A mechanical analysis was applied to derive a rating of the different possibilities for creating intersections [52, 101].

These evaluations are used in [52] to calculate an optimal unrolling for a given profile, meaning an ordered list of steps for the production line, which results in a profile as close to the given one as possible while also choosing the best technologies for the intersections. Of course, the main restrictions are derived from the fact, that the profile should be producible. If these restrictions do not allow for the production of the profile using one piece of sheet metal, the algorithm automatically increases the number of allowed pieces as long as it is lower than a given upper bound. The whole optimization process is based on a Steiner tree formulation with various additional constraints. The Steiner tree is chosen in such a way that the tree representing the solution allows for reading off the calculated unrolling. More details can be found in [52].

The production capabilities of the CRC 666 consisted of one single production line, which implies that any profile containing more than one single piece has to be produced in at least five steps: arrange the production line for the production of the first piece, produce this piece, rearrange the production line for the production of the second piece, produce it and finally combine the two pieces. The rearranging of the production line, or retooling, is a very time consuming process for two reasons. First of all, the involved machinery is very heavy so that moving has to be done with cranes. Second, the fine-tuning takes a lot of time since there are many different parameters to consider. If, for example, there is a change in measurements for a tool used at the beginning of the production line, all tools succeeding this have to be changed as well.

1.2 Overview

The complicated retooling process gave rise to the idea to ask for a partition of the whole profile in order to ensure a minimization of the time needed for the retooling process. The whole process can of course be circumvented if all the different pieces are equal because then no retooling is needed at all. In Chapter 2 we therefore study the problem of partitioning a graph into a given number of subsets such that the induced graphs are isomorphic. We show that this problem is NP-complete and study graph classes for which this problem is solvable in polynomial time.

Because we want to specify the number of single pieces, we also have to ensure that the subgraphs are connected since every connected component corresponds to a single piece. For example, if the produced profile should consist of three pieces but each subgraph contains two connected components, the number of single pieces is six instead of three. This is why Chapter 3 is focused on graph connectivity. The first part is concerned with the connected subgraph polytope defined by node sets of a graph, which induce connected subgraphs. We study the polytope and show complete descriptions of this polytope for special graph classes. Thereafter, we investigate the polytope resulting from an extension to also include a partitioning of the nodes. Section 3.3 deals with the optimization problem Connected Max-$\mathcal{K}$-Cut, which is to partition the graph into connected components such that the induced cut is maximized, therefore extending on the theoretical results of the previous sections.

The theoretical inspections from Chapter 3 lead to questions about MIP formulations in general. In Chapter 4 we propose a definition of MIP formulation including an objective function since this topic is sparse in the literature. Furthermore, we discuss properties of the formulation and show results concerning the existence or non-existence of such formulations illustrated by examples.

In Chapter 5 we return to the retooling process. Since it is not always possible to find identical subgraphs in a given graph, we inspect the problem of partitioning a graph into isomorphic subgraphs such that there may exist one component, which is not isomorphic to the others. Optimization is utilized because we want this non-isomorphic part to be as small as possible. The idea is that in this case only one retooling step has to be executed and furthermore the additional piece is small such that it might be produced reasonably fast. Moreover, to minimize the time needed for the retooling process, it might not be the best idea to inspect *equal* pieces but *similar* pieces. In Section 5.2 we formally define what is meant by this and present a framework for solving problems in this context. Furthermore, we give a short overview of the functions we implemented for measuring the similarity.

Chapter 6 contains a conclusion and an outlook.

1.3 Notation

In this section we give a short overview of notation which is used throughout this thesis.

To shorten notation, we use $[n]$ for an integer n to denote the set $\{1,\dots,n\}$ and $[n]_0$ for $[n] \cup \{0\}$. The restriction of a vector a to a subset I of the indices is denoted by $a|_I$. The restriction of a set S to a subset induced by an index set I is also denoted by $S|_I$. For a set S, the set of all k-element subsets of S is denoted by $\binom{S}{k}$.

A partition of a set S is a family of non-empty subsets S_i, $i \in [n]$ of S with $S_i \cap S_j = \emptyset$ for all $i, j \in [n]$ with $i \neq j$ and $S_1 \cup \cdots \cup S_n = S$. Note that we assume partitions to be *ordered*, that is, we differentiate between the partitions (S_1, S_2) and (S_2, S_1). Furthermore, we use the term *subpartition* if the empty set is allowed in a partition.

The unit vector is denoted by e, i.e., e_i is 1 at index i and 0 otherwise. For shortening the formulas we use the notation

$$x(I) = \sum_{i \in I} x_i$$

for a vector or a function x and a subset of the indices I. If $|I| = 1$, we also write $x(i)$ instead of $x(\{i\})$ for $i \in I$. The incidence vector is denoted by χ, that is, for a subset of indices I it holds that $\chi(I) = \sum_{i \in I} e_i = \sum_{i \in I} \chi(I)_i$. Furthermore, conv denotes the convex hull.

We assume the reader to be familiar with basic graph theory. There are numerous textbooks available, for an introduction see for example [32].

Let $G = (V, E)$ be a graph. If $V' \subseteq V$, we write $G[V']$ for the subgraph induced by the nodes V', that is, $G[V'] = (V', E')$ with $E' := \{\{u, v\} \in V' \times V' \mid \{u, v\} \in E\}$. Analogously, we write $G[E']$ for the graph induced by the edges E', i.e., $G[E'] := (V', E')$ with $V' := \{v \in V \mid v \in e \text{ with } e \in E'\}$.

We denote the neighborhood of a node v by $\Gamma(v)$ and the degree of v by $\deg(v)$. Note that in our case $\Gamma(v)$ does not contain v itself. The complete graph on n nodes is denoted by K_n and we use the term empty graph for graphs without edges, i.e., an empty graph G is defined by $G = (V, \emptyset)$.

The term k-connectivity always refers to k-node-connectivity, that is, a graph $G = (V, E)$ is k-connected if $G[V \setminus V']$ is connected for every set $V' \subseteq V$ with $|V'| < k$, where $|\cdot|$ denotes both the cardinality and the absolute value.

One major problem which appears throughout the whole thesis is GRAPH ISOMORPHISM which is defined as follows.

Definition 1.1. Two graphs $G = (V, E)$ and $H = (V', E')$ are *isomorphic*, written as $G \cong H$, if there exists a bijection ϕ between the nodes of both graphs such that edges are preserved, meaning $\phi\colon V \to V'$ such that $\{u, v\} \in E$ if and only if $\{\phi(u), \phi(v)\} \in E'$.

Problem 1.2 (GRAPH ISOMORPHISM). *Given two graphs G and H. Does there exist a graph isomorphism between G and H?*

For recent research activity on the GRAPH ISOMORPHISM problem see for example [8]. The complexity of this problem is unknown, the current belief is that GRAPH ISOMORPHISM lies strictly between P and NP thereby defining the complexity class GI [102]. With this class GI-complete problems can be defined analogously to NP-complete problems. If P $\neq$ NP, it has been shown that so-called NP-intermediate problems have to exist [80].

Lastly, we define the treewidth of a graph.

Definition 1.3 (Tree decomposition). Let $G = (V, E)$ be a graph. A *tree decomposition* of G is a tree T with nodes $X_1, \ldots, X_n$, also called *bags*, where each X_i is a subset of V, if the following properties hold

- $\bigcup_{i \in [n]} X_i = V$,
- for all edges $\{v, w\} \in E$ there exists an $i \in [n]$ with $v, w \in X_i$, and
- for all $v \in V$ the subgraph of T induced by all X_i with $v \in X_i$ is a subtree of T, that is, in particular, connected.

The *width* of a tree decomposition T is the size of its largest bag minus 1, i.e., $\max_{i \in [n]} |X_i| - 1$.

Definition 1.4 (Treewidth). The *treewidth* of a graph G is the minimum width over all tree decompositions of G.

The treewidth is a graph parameter that often allows for faster algorithms if it is known that the treewidth of G is fixed. For a survey on treewidth see for example [16].

Chapter 2

Partitioning into Isomorphic Subgraphs

In this chapter we inspect the problem of partitioning the nodes of a graph into a given number of subsets such that the induced graphs on these node sets are isomorphic. We show in Section 2.1 that the decision problem is NP-complete by a reduction from a variant of the well-known 3-SAT problem. If the additional constraint that all subgraphs have to be connected is added, the problem is also NP-complete. For some particular graph classes though, we show in Section 2.2 that the problems are solvable in polynomial time. Furthermore, we give a formula for the number of different possibilities to partition a path graph in Section 2.3.

2.1 Complexity Result

We are interested in the following problem.

Problem 2.1 (Partition Isomorphism). *Let $G = (V, E)$ be a graph, $\mathcal{K} \in \mathbb{N}$. Does there exist a partition $V_1, \dots, V_{\mathcal{K}}$ of V such that all induced subgraphs are isomorphic, that is, $G[V_i] \cong G[V_j]$ for all $i, j \in [\mathcal{K}]$?*

Obviously, there can only exist a solution if $\mathcal{K}$ divides $|V|$, which we assume in the following. Note that isomorphism in particular implies that $V_i \neq \emptyset$ for all $i \in [\mathcal{K}]$.

As far as we know this problem has not been studied in the literature but there are variants which appear, for example in [10]. Here, $\mathcal{K} = 2$ and the aim is not to partition V but to find two disjoint subsets of nodes which induce isomorphic graphs with a given lower bound on the number of contained edges. There it is shown, that this problem is NP-complete. It becomes polynomially solvable if G and the isomorphic subgraphs are both restricted leading to the following problem.

Problem 2.2. *Given a k-connected graph G with treewidth k and $K \in \mathbb{N}$. Do there exist $V_1, V_2 \subseteq V$ with $V_1 \cap V_2 = \emptyset$ such that $G[V_1] \cong G[V_2]$ and*

- *$G[V_1]$ contains at least K edges and*
- *$G[V_1]$ is k-connected?*

In [19] it is shown that Problem 2.2 is solvable in polynomial time. However, the variant with $G[V_1]$ restricted to be at most $(k-1)$-connected for a k-connected graph G is again NP-complete [19].

In [110] the so-called k-isomorphism problem is inspected. Graphs $G = (V,E)$ and $G' = (V',E')$ are called *k-isomorphic* if there exist partitions $E = E_1 \cup \cdots \cup E_k$ and $E' = E'_1 \cup \cdots \cup E'_k$ of the edges such that $G[E_i] \cong G'[E'_i]$ for all $i \in [k]$. For $k = 1$ the problem is the same as Problem 1.2 (GRAPH ISOMORPHISM). For $k = 2$ it is shown in [110] that the k-isomorphism problem is NP-complete.

The article [75] inspects the problem of partitioning a graph into a prescribed set of isomorphic parts. For example, a perfect matching can be seen as a partitioning of a graph $G = (V,E)$ into $|V|/2$ many complete graphs on two nodes K_2. This concept is generalized to more complex parts and the complexity of the resulting problems is studied.

As Problem 2.1 can be seen as both a partition and an isomorphism problem, there exist different approaches to this problem. For the former perspective, a recent overview of the graph partitioning problem can be found in [20]. The latter perspective leads to similar well-known problems, namely the SUBGRAPH ISOMORPHISM and the MAXIMUM COMMON INDUCED SUBGRAPH problem.

Problem 2.3 (SUBGRAPH ISOMORPHISM). *Given two graphs G and H. Does there exist a subgraph G' of G that is isomorphic to H, i.e., $G' \cong H$?*

For a comparison of recent algorithms for solving this problem see for example [17]. The NP-completeness is shown easily, because if H is a clique or a cycle, SUBGRAPH ISOMORPHISM is equivalent to either the CLIQUE or the HAMILTONIAN CYCLE problem, also see [27] and [44, Problem GT48].

Problem 2.4 (MAXIMUM COMMON INDUCED SUBGRAPH). *Let $G_1 = (V_1,E_1)$ and $G_2 = (V_2,E_2)$ be two graphs. The task is to find a graph $G = (V,E)$ such that there exist subsets $V'_1 \subseteq V_1$ and $V'_2 \subseteq V_2$ with $G \cong G_1[V'_1]$ and $G \cong G_2[V'_2]$ with maximal cardinality of V.*

This problem is NP-complete since it includes Subgraph Isomorphism, see also [44, Problem GT49] where it is called Largest Common Subgraph. This problem is also discussed in Section 5.2.3. For applications and an algorithmic overview see for example [37].

Even though both Subgraph Isomorphism and Maximum Common Induced Subgraph seem to be obvious choices for proving NP-completeness of Problem 2.1, we were not able to find a suitable reduction. Instead, we used a variant of the well-known 3-SAT problem, that needs the introduction of additional notation.

Definition 2.5. A *Boolean variable* is a variable which only attains the two values TRUE or FALSE. A negation of a Boolean variable x is written as $\neg x$. A *Boolean formula* is a formula of Boolean variables concatenated by AND ($\wedge$) or OR ($\vee$) with brackets and negations allowed.

Problem 2.6 (SAT). *Let C be a Boolean formula of the Boolean variables $X := \{x_1, \dots, x_n\}$. The satisfiability problem* SAT *is the problem of deciding whether an assignment to the variables x_i exists such that the formula C is evaluated as* TRUE.

The famous Cook-Levin theorem shows that SAT is NP-complete [27, 84].

Definition 2.7. A Boolean variable or its negation in a Boolean formula is also called a *literal*. A *clause* is an expression formed by finitely many literals concatenated by a conjunction or disjunction (AND, OR, respectively). In our case, we are interested only in *disjunctive clauses*, i.e., clauses connected by OR, which is why we omit the term *disjunctive* in the following.

Problem 2.8 (3-SAT). *Let $C := c_1 \wedge c_2 \wedge \dots \wedge c_m$ be a Boolean formula of the clauses c_i with $|c_i| = 3$ for all $i \in [m]$ containing literals of the Boolean variables $X := \{x_1, \dots, x_n\}$. The problem of deciding whether an assignment to the variables x_i exists such that the formula C is evaluated as* TRUE *is called* 3-SAT.

3-SAT is also NP-complete as shown in [74]. Furthermore, the following restriction of 3-SAT is NP-complete [100].

Problem 2.9 (1-in-3-SAT). *The problem* 1-in-3-SAT *is the special case of* 3-SAT *in which the task is to find an assignment for the variables such that every clause contains exactly one* TRUE-*literal.*

This allows us to finally state the SAT variant used for the NP-completeness proof of Problem 2.1.

Problem 2.10 (2-in-4-SAT). *Let $C := c_1 \wedge c_2 \wedge \dots \wedge c_m$ be a Boolean formula of the clauses c_i with $|c_i| = 4$ for all $i \in [m]$ containing literals of the Boolean variables $X := \{x_1, \dots, x_n\}$. Is there a truth-assignment to the variables x_i such that every clause c_i is fulfilled with exactly two* TRUE-*values?*

Corollary 2.11 ([10]). 2-IN-4-SAT *is* NP-*complete.*

Proof. Reduction from 1-IN-3-SAT by adding a new TRUE-variable to every clause. □

Note that $\neg C$ defined from C by negating every literal in every clause is also a 2-IN-4-SAT instance. This fact is the central idea for the NP-completeness proof of Problem 2.1.

The following lemma shows that even a restricted form of 2-IN-4-SAT is NP-complete.

Lemma 2.12. *The following restricted version of* 2-IN-4-SAT *is still* NP-*complete:*

- *No clause contains a variable both in negated and non-negated form.*
- *No clause contains a variable twice.*

Proof. In the first case, a clause $(x \vee \neg x \vee a \vee b)$ can be replaced by the two clauses $(x \vee y \vee a \vee b)$ and $(x \vee x \vee y \vee y)$ after introducing another variable $y \notin X$. The second clause be can interpreted as $x = \neg y$.

In the second case, one can introduce three additional variables z_1, z_2 and z_3 and replace the clause $a \vee a \vee b \vee c$ by $z_1 \vee a \vee b \vee c$ and add the following six clauses:

$$\begin{aligned}
&\neg a \vee \neg z_1 \vee z_2 \vee z_3,\\
&\neg a \vee z_1 \vee \neg z_2 \vee z_3,\\
&\neg a \vee z_1 \vee z_2 \vee \neg z_3,\\
&a \vee \neg z_1 \vee \neg z_2 \vee z_3,\\
&a \vee \neg z_1 \vee z_2 \vee \neg z_3,\\
&a \vee z_1 \vee \neg z_2 \vee \neg z_3.
\end{aligned}$$

Basically, these formulas imply $a = z_1 = z_2 = z_3$.

Note that both these constructions are in polynomial time and the size of the resulting 2-IN-4-SAT instance is also polynomial. □

The first statement already appears in [10], where $(x \vee \neg x \vee a \vee b)$ is replaced by $(x \vee \neg y \vee a \vee b) \wedge (\neg x \vee y \vee a \vee b)$.

With this we can finally state and prove the main theorem of this chapter.

Theorem 2.13. *Problem 2.1 is* NP-*complete for* $\mathcal{K} = 2$.

Proof. The proof is done by a reduction from Problem 2.10 which uses multigraphs. Note that GRAPH ISOMORPHISM on multigraphs is GI-complete [111] and therefore using multigraphs does not increase the complexity.

The multigraph $G = (V,E)$ is defined from a given 2-in-4-SAT instance in the following way (nomenclature used from Problem 2.10):

$$V := \{c_1, \neg c_1, \ldots, c_m, \neg c_m\} \cup \{x_i^1, \neg x_i^1, \ldots, x_i^4, \neg x_i^4 \mid x_i^h \in c_i, i = 1, \ldots, m, h = 1,2,3,4\},$$

meaning there are two nodes for each clause, corresponding to the original clause and to its negation, respectively. There are also two nodes for every variable in every clause, meaning eight nodes per clause. Due to Lemma 2.12, we may assume that the variable nodes are pairwise different.

The edges E are defined by joining a clause c_i with $2i-1$ edges to all the contained literals and $2i$ edges to the negation of every contained literal. Whereas, for the complement clauses $\neg c_i$, the role is reversed: Clause i is connected with $2i-1$ edges to every negation of a literal and $2i$ edges to every contained literal. Additionally, every variable x_i^h is connected to its negation $\neg x_i^h$ by $2m+i$ edges. If any variable appears in at least two clauses, it is connected to every negation of it by $3m+i$ edges, see Figure 2.1 for an example, where a number displays the cardinality of edges. More formally,

$$\begin{aligned} E :=& \{(2i-1)\{c_i, a_j\}, (2i-1)\{\neg c_i, a_j\} \mid c_i = (a_1 \vee a_2 \vee a_3 \vee a_4) \in C, j = 1,2,3,4\} \\ &\cup \{2i\{c_i, \neg a_j\}, 2i\{\neg c_i, \neg a_j\} \mid c_i = (a_1 \vee a_2 \vee a_3 \vee a_4) \in C, j = 1,2,3,4\} \\ &\cup \{(2m+i)\{x_i^h, \neg x_i^h\} \mid x_i^h \in c_i, h = 1,2,3,4\} \\ &\cup \{(3m+i)\{x_i^{h_1}, \neg x_j^{h_2}\} \mid x_i^{h_1} \equiv x_j^{h_2}, i < j, h_1, h_2 \in \{1,2,3,4\}\}, \end{aligned}$$

where $i\{j,k\}$ indicates that there are i edges between j and k. Furthermore, we introduce two additional nodes even if the variable appears before, that is, we have four variable-nodes in G even if x_1 is an element of both c_1 and c_2. Additionally, if the nodes $x_i^{h_1}$ and $x_j^{h_2}$ both correspond to the same variable, we write $x_i^{h_1} \equiv x_j^{h_2}$. Note that there are $10m$ nodes and $\mathcal{O}(m^2)$ edges in G (since $n \le 4m$, which implies $n \in \mathcal{O}(m)$) such that G can be constructed in polynomial time.

We first deduce a truth-assignment from a given solution to Problem 2.1. Let there be a partition V_1, V_2 of V such that $G[V_1] \cong G[V_2]$ with isomorphism ϕ. The construction of G implies that nodes x_i^h and $\neg x_i^h$ cannot be in the same partition since there is no other node pair in G connected by exactly $2m+i$ edges. The same argument is used to show that if $x_i^{h_1} \equiv x_j^{h_2}$, then $x_i^{h_1}$ cannot be in the same partition as $\neg x_j^{h_2}$ because those nodes are connected by $3m+i$ edges.

Remember that $|c_i| = 4$ and because of Lemma 2.12 c_i is therefore connected to eight different nodes in G . If c_i is in V_1, it follows that c_i is connected to exactly four variable-nodes in V_1 and the remaining four neighbors of c_i are in V_2. The

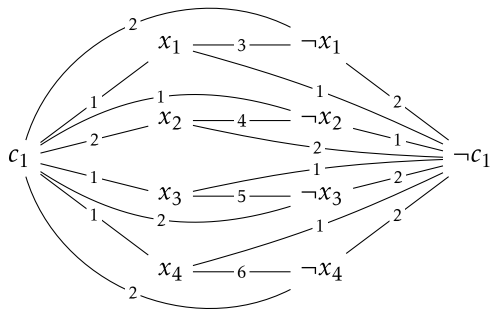


Figure 2.1: Graph created from the Boolean formula $c_1 := x_1 \vee \neg x_2 \vee x_3 \vee x_4$, where the numbers on the edges represent the number of multiple edges and the superscript h is removed since there is only one clause. Note that this way there do not exist edges between a variable and a negation of one appearing in a different clause.

number of edges to every node is unique, see Lemma 2.12, and the only other node in G with the same number of edges is $\neg c_i$. Therefore, $\phi(c_i) = \neg c_i$.

Let $c_i = (a \vee b \vee c \vee d)$ be a clause. The node c_i has to be connected to exactly two of the literals a, b, c or d in V_1 because otherwise it cannot be mapped to $\neg c_i$ by the isomorphism: If c_i is in the same partition as a, b, c and $\neg d$, then $\neg c_i$ is in the same partition as $\neg a, \neg b, \neg c$ and d. But then c_i has degree $3 \cdot (2i - 1) + 1 \cdot 2i$, whereas $\neg c_i$ has degree $1 \cdot (2i - 1) + 3 \cdot 2i$, which would be a contradiction to the isomorphism. The cases that c_i is connected to 0, 1 or 4 nodes of a, b, c and d are analogous.

That means for every clause $c_i = (a \vee b \vee c \vee d) \in V_1$ if we set both literals that are in V_1 to TRUE and the other two to FALSE, the clause c_i contains exactly two TRUE and two FALSE literals. Therefore, if an isomorphism ϕ exists, there also is an assignment for the variables such that the given 2-in-4-SAT instance can be evaluated as TRUE. The same result can be achieved by choosing all variables that are in V_2 resulting in the negation of those in V_1. Because it holds that for every variable x_i^h its negation $\neg x_i^h$ is not in the same set, meaning $x_i^h \in V_k$ implies $\neg x_i^h \in V_\ell$ for $k, \ell \in [2]$, $k \neq \ell$, this assignment is correct.

To show the reverse direction, that is, creating a solution to Problem 2.1 from a solution to a given 2-in-4-SAT instance C, assume there exists an assignment such that C can be evaluated as TRUE. Let $c_i = (a_{i1} \vee a_{i2} \vee a_{i3} \vee a_{i4})$ be a clause where w.l.o.g. a_{i1} and a_{i2} are the two TRUE variables. Note that Lemma 2.12

implies $a_{i1} \neq a_{i2}$. Define V_1 as follows

$$V_1 := \bigcup_i c_i \cup a_{i1} \cup a_{i2} \cup \neg a_{i3} \cup \neg a_{i4},$$

that is, V_1 contains all clause nodes and all the nodes which are TRUE along the two nodes which are FALSE. Notice that in particular for every variable $x \in V_1$, its negation $\neg x$ is not in V_1. Define V_2 as the complement, that is, $V_2 := V \setminus V_1$ as well as $\phi(c_i) := \neg c_i$. In the following we show that ϕ can be completed to be an isomorphism (also containing the nodes corresponding to variables).

The node $c_i = (a_{i1} \vee a_{i2} \vee a_{i3} \vee a_{i4}) \in V_1$ is connected to exactly two nodes corresponding to TRUE, w.l.o.g. a_{i1} and a_{i2}. In particular, this implies that a_{i3}, $a_{i4} \in V_2$. Defining the isomorphism additionally by $\phi(a_{i1}) := a_{i3}$ all $2i-1$ edges from c_i to a_{i1} in $G[V_1]$ correspond to the $2i-1$ edges between a_{i3} and $\neg c_i$ in $G[V_2]$, meaning for every edge (c_i, a_{i1}) in $G[V_1]$, the image $(\phi(c_i), \phi(a_{i1})) = (\neg c_i, a_{i3})$ is an edge in $G[V_2]$. Similar arguments hold for a_{i2}. Since $x_i \in V_1$ implies $\neg x_i \in V_2$, no edges exist between a variable and its negation in V_1. The same argument shows that the $2i$ edges $(c_i, \neg a_{i3})$ are also corresponding to the $2i$ edges $(\phi(c_i), \phi(\neg a_{i3})) = (\neg c_i, \neg a_{i1})$ in $G[V_2]$ as well as the case of $(c_i, \neg a_{i4})$.

Thus, we have shown that for all nodes $u, v \in V_1$ the number of edges $\{u, v\}$ is the same as the number of edges in the image of ϕ, i.e. $\{\phi(u), \phi(v)\}$. This in particular means that ϕ defines an isomorphism. □

As we have seen in the introduction, connectedness of the subgraphs is essential for the retooling problem. This is why we study the extension of Problem 2.1 by the additional constraint of connectedness.

Problem 2.14 (Connected Partition Isomorphism). *Let $G = (V, E)$ be a graph, $\mathcal{K} \in \mathbb{N}$. Does there exist a partition $V_1, \ldots, V_{\mathcal{K}}$ of V such that all induced subgraphs are connected and isomorphic, i.e., $G[V_i] \cong G[V_j]$ for all $i, j \in [\mathcal{K}]$?*

Corollary 2.15. *Problem 2.14 is* NP*-complete even for $\mathcal{K} = 2$.*

Proof. We can use the same proof as of Theorem 2.13 after adding the tautologies $x_i \vee \neg x_i \vee x_j \vee \neg x_j$ for all pairs $i \neq j$ and applying Lemma 2.12. The tautologies ensure that the resulting subgraphs are connected. □

Remark 2.16. For particular values of $\mathcal{K}$, Problems 2.1 and 2.14 are easy to solve, namely $\mathcal{K} = 1, n/2, n$. In the first case, there is only one set in the partition, which therefore is the whole graph and in the last case every set of the partition contains a single node. In the second case, every subgraph contains exactly two nodes. Then either the nodes are connected or not. If they are connected, Problem 2.1 can be answered positively if G contains a perfect matching. This question can be answered in polynomial time, for example with Edmond's blossom algorithm [36].

Furthermore, for simple graphs $G = (V, E)$ without loops, let the complement $\overline{G}$ of G be defined by $\overline{G} = (V, \overline{E})$ with $\overline{E} := \{\{u,v\} \in V \times V \mid u \neq v, \{u,v\} \notin E\}$. Solving Problem 2.1 in G with pairwise non-adjacent nodes is equivalent to finding a perfect matching in $\overline{G}$ and is therefore also solvable in polynomial time.

Remark 2.17. It can be shown that in general the graphs appearing in the proof of Theorem 2.13 are not planar. Let $B := (a_1 \vee a_2 \vee a_3 \vee a_4) \wedge (a_1 \vee a_2 \vee a_3 \vee a_4)$ be a Boolean formula. If we apply the construction of graph G from the proof of Theorem 2.13, select only one of the parallel edges and contract the edges between a_{i1}, a_{i2} and $\neg a_{i1}$, $\neg a_{i2}$ for $i = 1,2,3,4$, the resulting graph is isomorphic to the complete bipartite graph $K_{4,4}$ which in particular includes $K_{3,3}$. Thus, with Wagner's theorem [106] the graph G is not planar. Therefore, the question whether Problem 2.1 is NP-complete for planar graphs cannot be answered using the construction from the proof of Theorem 2.13.

2.2 Polynomial Time Solvable Graph Classes

Even though we have shown that Problems 2.1 and 2.14 are NP-complete, there are graphs classes where these problems are solvable in polynomial time for all values of $\mathcal{K}$. This is easy to see for complete or empty graphs. For the former, every subset of size $k := |V|/\mathcal{K}$ induces a complete graph on k nodes, which implies the isomorphism as well as the connectedness. The empty graph can of course in general only solve Problem 2.1, where connectedness is not a constraint. In this case, all induced graphs are also empty and hence isomorphic.

This section shows polynomial time solvability for trees, outerplanar and Turán graphs.

The following proposition inspects the case of G being a tree. It is shown that the restriction of subgraphs to the same size suffices for solving Problem 2.14 without the isomorphism constraint.

Proposition 2.18. *Let $G = (V, E)$ be a tree and let $\mathcal{K} \in \mathbb{N}$. Partitioning V into $\mathcal{K}$ subsets $V_1, \ldots, V_{\mathcal{K}}$ of the same size such that $G[V_i]$ is connected for every $i \in [\mathcal{K}]$ has a unique solution if it exists. Furthermore, this solution can be found in linear time.*

Proof. W.l.o.g. assume $\mathcal{K} \geq 2$. For the sake of contradiction assume there exist two different solutions $S_1, \ldots, S_{\mathcal{K}}$ and $T_1, \ldots, T_{\mathcal{K}}$. Let G_S be the graph resulting from contracting all edges of $G[S_i]$ for every $i \in [\mathcal{K}]$. Note that G_S is a tree on the nodes S_i and define G_T as the contraction of edges in $G[T_i]$ accordingly. Let S_ℓ be a leaf of G_S and let $e \in G_S$ be the edge connecting S_ℓ to a node from G_S. Note that e has a corresponding edge in G since it was not contracted in the construction of G_S.

Assume that there exists a $j \in [\mathcal{K}]$ with $e \in G[T_j]$. As a result of $|S_\ell| = |T_j|$, there exists a node $q \in S_\ell$ with $q \notin T_j$. Any path from q to a node in $G \setminus S_\ell$ contains the edge e since the path from one node to another is unique in a tree. This means there cannot exist an index k with $q \in T_k$, $|T_k| = |S_\ell|$ such that T_k is connected, which contradicts the assumption.

Therefore, there does not exist an $i \in [\mathcal{K}]$ with $e \in G[T_i]$. This implies that there has to exist a $j \in [\mathcal{K}]$ such that $S_\ell = T_j$ because of the size restriction. This shows that the subsets of nodes corresponding to leaves of G_S and G_T are equal. After removing the leaves from both trees, the same argument can be used again to show that the whole partitions are equal by iteratively removing leaves. This contradicts the assumption and therefore proves the statement.

If at least two nodes have to be in one partition, it is clear that every leaf has to be in the same partition as its neighbor. Contracting the corresponding edge and proceeding iteratively results either in the unique solution or in the result that no solution exists. This method thus finds the solution (if it exists) in linear time. □

Corollary 2.19. *Problem 2.14 can be solved in time $\mathcal{O}(n\mathcal{K})$ for trees.*

Proof. If the components have to be isomorphic, they in particular have to have the same size and hence Proposition 2.18 applies. Thus, Problem 2.14 can be solved by first finding a partition via the above procedure and then checking the resulting partitions for isomorphisms. Since Graph Isomorphism for trees can be answered in linear time [1] and has to be called $\mathcal{K}$ times, Problem 2.14 can be solved in time $\mathcal{O}(n\mathcal{K})$. □

Obviously, trees are graphs of treewidth 1. In fact, a graph with at least two nodes has treewidth 1 if and only if it is a tree. For $\mathcal{K} = 2$ we can show how to solve Problem 2.14 on a special graph class, that has treewidth 2, the so-called *outerplanar* graphs.

Definition 2.20 (Outerplanar graph). A graph G is *outerplanar* if G has a planar drawing such that all nodes belong to the outer face.

Proposition 2.21. *If G is an outerplanar graph and $\mathcal{K} = 2$, Problem 2.14 can be solved in time $\mathcal{O}(n^2)$.*

Proof. W.l.o.g. we may assume that all nodes lie on a fixed circle and all edges lie inside this circle and do not intersect. Let the nodes be ordered consecutively on the circle $v_0, v_1, \ldots, v_{n-1}$. A set of nodes W is called *consecutive* if $W = \{v_i, v_{i+1}, \ldots, v_{|W|+i-1}\}$, where the indices are considered modulo n. The set W is called *maximal consecutive* if there does not exist a consecutive set W' with $W' \supsetneq W$ and $|W| + 1 = |W'|$.

Let $V_1 \cup V_2$ be a partition of V such that both V_1 and V_2 consist of at least two different sets of maximal consecutive nodes. Because the edges of G lie only inside the circle and do not intersect, $G[V_1]$ and $G[V_2]$ cannot be both connected.

This implies that in order to solve Problem 2.14 at least one of the sets V_1 and V_2 has to be consecutive.

Assume w.l.o.g. that V_1 is consecutive. There exist $n/2$ possible solutions for V_1 because V_1 contains $n/2$ nodes implying the symmetry of V_1 and $V_2 = V \setminus V_1$. Since GI can be solved in linear time for planar graphs [59], Problem 2.14 can be solved in time $\mathcal{O}(n^2)$ by checking every possible solution. □

Note that since trees are in particular outerplanar, the above idea can also be applied to trees resulting in the same running time as in Proposition 2.18, however, only for $\mathcal{K} = 2$.

Since outerplanar graphs have treewidth at most 2, both results Corollary 2.19 and Proposition 2.21 are restricted to graphs with bounded treewidth (as are empty graphs). Besides the already mentioned complete graphs, the so-called Turán graphs do not share this restriction.

Definition 2.22 (Turán graph [105]). The *Turán graph* $T(n,r)$ is a complete multipartite graph on n nodes. The r subsets are sized as equal as possible, that is, T has $n \bmod r$ subsets of size $\lceil n/r \rceil$ and $r-(n \bmod r)$ subsets of size $\lfloor n/r \rfloor$. Two nodes are connected if and only if they belong to different subsets.

To show that Problems 2.1 and 2.14 are solvable in polynomial time for Turán graphs, we construct one possible solution. In order to prove the correctness of this solution, we first need the following technical lemma.

Lemma 2.23. *Let a_i be a sequence of length n with ordered values, that is, $a_i \geq a_{i+1}$ for all $i \in [n-1]$, such that consecutive entries differ in at most one position by exactly one, i.e., $a_j = a_{j+1} + 1$ for at most one $j \in [n-1]$. Let b_i be a sequence with the same properties. Furthermore, let the element-wise difference be $c_i := a_i - b_i, i \in [n]$. The entries c_i differ by at most one, that is, $|c_i - c_j| \leq 1$ for all $i,j \in [n]$.*

Proof. Let $\overline{a} = \max\{a_i\}$, $\overline{b} = \max\{b_i\}$, $\underline{a} = \min\{a_i\}$, and $\underline{b} = \min\{b_i\}$. W.l.o.g. we can assume $\overline{a} \neq \underline{a}$ and $\overline{b} \neq \underline{b}$ because the statement is trivially fulfilled otherwise. The only possibility for generating a difference of at least two is if two indices $k, \ell \in [n]$ exist such that $c_k = \overline{a} - \underline{b}$ and $c_\ell = \underline{a} - \overline{b}$. But this is not possible due to the order of the sequences. □

Theorem 2.24. *It is possible to partition the nodes of $T(n,r)$ into $\mathcal{K}$ subsets such that the induced subgraphs are isomorphic. The induced subgraphs are always connected, which implies the solvability of Problems 2.1 and 2.14.*

Proof. Let $n = \mathcal{K}u$, i.e., there are u nodes in every subgraph. We assume an ordering of the subsets of $T(n,r)$ such that the subsets are ordered by size, that is, the first p subsets are of size $m+1$ and the last q subsets are of size m, where $p(m+1) + qm = n$ (see Definition 2.22). Any partition of the nodes can now be described as a vector v in $\mathbb{N}^{p+q}$ with the meaning of taking v_i nodes of subset i. This suffices because it does not matter, which nodes are chosen from a subset since all nodes of a subset have exactly the same neighborhood.

Two induced subgraphs are isomorphic if they correspond to permutations of the same vector because of the symmetry of $T(n,r)$. Thus, it suffices to find $\mathcal{K}$ vectors v^j with $\sum_{i=1}^{p+q} v_i^j = u$ for all $j \in [\mathcal{K}]$ that are permutations of each other and add up to the vector $s := (m+1,\dots,m+1,m,\dots,m) \in \mathbb{N}^{p+q}$.

Let v^1 be the vector defined by

$$\left(\left\lceil \frac{u}{p+q} \right\rceil, \dots, \left\lceil \frac{u}{p+q} \right\rceil, \left\lfloor \frac{u}{p+q} \right\rfloor, \dots, \left\lfloor \frac{u}{p+q} \right\rfloor\right),$$

where ceiling- and floor-functions are used such that the entries add up to u. Note that this does not necessarily imply that the first p entries are rounded up and the last q are rounded down.

By using Lemma 2.23 it is clear that $s - v^1$ contains only entries that differ by at most one. Adding up the entries of $s - v^1$ results in a number that is divisible by u since $p(m+1)+qm = n = u\mathcal{K}$ and $\sum_{i=1}^{p+q} v_i^1 = u$. Since the entries of v^1 differ by at most one, it is possible to find v^2 as a permutation of the entries of v^1 such that $s - v^1 - v^2$ contains only entries that differ by at most one. To see this, $s - v^1$ may be ordered and s can be replaced by $s - v^1$ to use Lemma 2.23 again. As before $s - v^1 - v^2$ is still divisible by u. Let v^i, $i = 3,\dots,\mathcal{K}$ be defined accordingly. Then $s - \sum_{j=1}^{\mathcal{K}} v^j = 0$ because the properties of divisibility and of maximal difference of one are maintained within the process.

When choosing at least two different subsets, the induced graphs are connected because of the definition of Turán graphs. This means that induced subgraphs are only disconnected if at least two nodes are chosen from the same subset and none of the others. Since by the definition of v^j this does not happen, all solutions are connected. For $\mathcal{K} = n$ the partitions contain exactly one node and are therefore also connected. □

The above partition derived from vectors v^j can be found in linear time and thus Problems 2.1 and 2.14 can be answered in constant time.

Note that the result from [19], which shows the polynomial time solvability of Problem 2.2, does not only rely on bounded treewidth but also on a restriction of the graphs that have to be isomorphic. In our case it is not known if Problems 2.1 and 2.14 can be solved in polynomial time if the treewidth is bounded. However, as shown above an unbounded treewidth does not automatically lead to NP-complete problems.

It is clear that the partition given in the proof of Theorem 2.24 is not the only possible solution. For example $T(6,4)$ can be partitioned into two $T(3,3)$ or into two $T(3,2)$ (the proof uses the first case). It is an interesting task to find the number of non-isomorphic partitions of $T(n,r)$ into $\mathcal{K}$ parts. In the next section we examine this problem on the much simpler path graph.

2.3 Combinatorics on the Path Graph

The previous section deals with a decision problem and thus we are only interested in whether a solution exits or not. Even tough we have shown NP-completeness, the problem is easier when restricted to special graph classes. If for example G is the complete graph on n nodes, Problem 2.1 can be solved directly. Because every induced subgraph is obviously connected, Problem 2.14 does not differ from Problem 2.1 for complete graphs.

Since the decision problem is easy to solve, one question that follows naturally is how many different solutions there are. For complete graphs all subsets of size $n/\mathcal{K}$ induce isomorphic graphs. This implies that the number of non-isomorphic solutions is 1. Obviously, the above also holds for the empty graph if connectedness is not an additional constraint.

In the following we inspect the case where $G = (V,E)$ is a path on n nodes, that is, if $V = \{1,\dots,n\}$, the edges are $E = \{\{i,i+1\} \mid i \in \{1,\dots,n-1\}\}$. The formal statement of the problem is then the following.

Problem 2.25. *How many different solutions are there for Problem 2.1 for given $\mathcal{K} \in \mathbb{N}$ if G is a path on n nodes?*

This problem is answered by an explicit formula, which first needs one additional notation.

Definition 2.26. Given an integer r, let p denote a partition of r into a sum of positive integers, for example $5 = 4+1$ or $5 = 2+2+1$. Or more formally, p is a sequence $p_1,\dots,p_k$ of positive integers such that $\sum p_i = r$. We want to write a partition in a form where every integer used in the partition is given once and the number of occurrences in the partition is written in superscript, that is, $5 = (4^1, 1^1)$ and $5 = (2^2, 1^1)$, respectively for the examples above. As usual and for our purpose the order does not matter. This description is similar to the *frequency representation*, see, e.g., [6].

If a given partition $p = (a_1^{m_1}, a_2^{m_2}, \dots, a_q^{m_q})$ is written as a sum, the number of different possibilities to arrange the numbers a_i in this sum is

$$\binom{m_1 + m_2 + \cdots + m_q}{m_1}\binom{m_2 + \cdots + m_q}{m_2}\cdots\binom{m_q}{m_q} = \frac{(m_1 + m_2 + \cdots + m_q)!}{m_1!m_2!\dots m_q!}. \tag{2.1}$$

Before the formula can be stated, one last theorem is needed.

Theorem 2.27 ([38]). *Given k classes of elements with s_i identical objects in the i-th class, the number of sequences of length $s_1 + \cdots + s_k$ which consist of these elements and in which no two objects from the same class are adjacent is given by*

$$M(s_1, \ldots, s_k) = \sum_{i=1}^{k-1} \sum_{j_i=1}^{s_i} (-1)^{s_1+\cdots+s_{k-1}-j_1-\cdots-j_{k-1}} \times \prod_{t=1}^{k-1} \left[\frac{1}{j_t!} \binom{s_t - 1}{j_t - 1} \right] \binom{j_1 + \cdots + j_{k-1} + 1}{s_k} (j_1 + \cdots + j_{k-1})!.$$

With this we can finally state the result for answering Problem 2.25.

Lemma 2.28. *Let $A(n, \mathcal{K})$ denote the number of possibilities stated in Problem 2.25 ignoring possible interchanges of classes. Then,*

$$A(n, \mathcal{K}) := \sum_{p \text{ partition of } \frac{n}{\mathcal{K}}} \left(\frac{(m_1 + m_2 + \cdots + m_q)!}{m_1! m_2! \ldots m_q!} \right)^{\mathcal{K}} M(s_1, s_2, \ldots, s_{\mathcal{K}}), \qquad (2.2)$$

where $p = (a_1^{m_1}, a_2^{m_2}, \ldots, a_q^{m_q})$ and $s_1 = s_2 = \cdots = s_{\mathcal{K}} = \sum_{i=1}^{q} m_i$. If $n/\mathcal{K} \notin \mathbb{N}$, this sum is empty, and in particular, $A(n, \mathcal{K}) = 0$.

Proof. Every subgraph of the path G is a path or the union of paths. The subgraph of G induced by a subset of the nodes can therefore be described by the length of every connected component. This description also suffices for deciding isomorphism. Hence, to describe the induced subgraphs, we can also use partitions of integers, and two subgraphs are isomorphic if and only if they can be represented as the same partition. Thus, there exist solutions for every integer partition of $n/\mathcal{K}$, which explains the sum.

Let $p = (a_1^{m_1}, a_2^{m_2}, \ldots, a_q^{m_q})$ be a partition. Since in every solution to Problem 2.1 the $\mathcal{K}$ partitions have to be of equal size, the number of solutions to arrange one partition has to be raised to the $\mathcal{K}$-th power to account for all different arrangements of all these partitions. As the number of these arrangements is stated in (2.1), this results in the coefficient of $M(s_1, \ldots, s_{\mathcal{K}})$ in (2.2).

This only explains the number of possibilities to arrange the elements in one partition ignoring the arrangements of elements of different partitions. This is remedied by the factor $M(s_1, \ldots, s_{\mathcal{K}})$ from Theorem 2.27. Note that two different elements from the same partition cannot be adjacent since that would change their size. If for example two subgraphs of the same partition are adjacent, one of length 2 and the other of length 3, they do not form two separate subgraphs, but one of length 5. $\square$

Remark 2.29. Equation (2.2) does not count the number of different isomorphisms, but only the number of different partitions. Every connected component consisting of one node can be mapped via one isomorphism to another component with one node. For every component consisting of two or more nodes, there are exactly two possible isomorphisms because the subgraphs are all paths. Equation (2.2) can be modified accordingly.

Remark 2.30. Note that the solutions are symmetric as the partition classes can be changed without changing the solution. Equation (2.2) can be changed easily to remedy this fact by dividing M by $\mathcal{K}!$ since there are $\mathcal{K}!$ possibilities for exchanging the partition classes.

Remark 2.31. There is no explicit formula for the number of integer partitions but many different approximations or schemes for calculating the number explicitly are known, see for example [63, Sequence A000041].

Chapter 3

Connected Subgraphs and Partitioning into Connected Subgraphs

The difference between Problem 2.1 (Partition Isomorphism) and Problem 2.14 (Connected Partition Isomorphism) from the previous chapter is the additional constraint of graph connectivity, which is the main focus of this chapter. First, we inspect the polytope that is defined by subsets of nodes that induce a connected subgraph in Section 3.1. After a recollection of known results from the literature, we introduce new facet-defining inequalities in Section 3.1.1. In the subsequent sections we show complete descriptions of the connected subgraph polytope for cycles, complete bipartite graphs and for the case that exactly one node and one edge are added to a graph in Sections 3.1.2 to 3.1.4, respectively.

Section 3.2 deals with the problem of partitioning a graph into at most $\mathcal{K}$ components such that induced subgraphs are connected for a given integer $\mathcal{K}$. In Section 3.2.1 we analyze the polytope that is defined by subpartitions, that is, only a subset of the nodes is partitioned. In the succeeding section, we enforce every node to belong to one partition and also examine the corresponding polytope.

The last part, Section 3.3, introduces the Connected Max-$\mathcal{K}$-Cut problem, which is defined by maximizing the weight of edges between the $\mathcal{K}$ different partitions, which each induce connected subgraphs. We present our implementation for solving the problem based on the framework SCIP and show numerical results.

3.1 The Connected Subgraph Polytope

In this section, we analyze the connected subgraph polytope associated with a given graph G, which is defined as follows.

Definition 3.1. The connected subgraph polytope for a given graph $G = (V, E)$ is defined as

$$\mathcal{P}(G) := \operatorname{conv}(\{\chi(V') \in \{0,1\}^V \mid V' \subseteq V,\, G[V'] \text{ is connected}\}).$$

Example 3.2. *The connected subgraph polytope of the complete graph on n nodes K_n is the n-dimensional hypercube. This can easily be seen as every subset of nodes induces a connected subgraph.*

Example 3.3. *The connected subgraph polytope of the empty graph on n nodes is the n-dimensional simplex because the only non-zero points in $\mathcal{P}(G)$ are the unit vectors.*

The connected subgraph polytope has been studied amongst others in [3, 4, 108]. The authors of [78, Theorem 3.6 on p. 168] give a full description of $\mathcal{P}(G)$ if G is a tree (see also [108, Theorem 4]). To state this description, we need one more definition.

Definition 3.4 ([78, p. 168]). Given an undirected graph $G = (V, E)$, a vector $d \in \mathbb{R}^V$ is said to be an *indegree vector* if for some orientation $D = (V, A)$ of G, the indegree of each node v is d_v. For each indegree vector d of G, there is a corresponding *indegree inequality*:

$$\sum_{v \in V} (1 - d_v) x_v \leq 1. \tag{3.1}$$

The validity of indegree-inequalities follows from [108, Lemma 11].

Theorem 3.5 ([78, Theorem 3.6 on p. 168]). *If $G = (V, E)$ is a tree, the following holds:*

$$\mathcal{P}(G) = \{x \in \mathbb{R}^V \mid x_v \geq 0,\, x \text{ satisfies all indegree inequalities}\}.$$

Furthermore, each indegree inequality induces a facet.

In [3, 4] the inspected polytope is studied for directed graphs, which may be rooted. There, the focus lies on different MIP formulations for the Maximum Weight Connected Subgraph problem, which can be stated as follows.

Problem 3.6 (Maximum Weight Connected Subgraph). *Given an undirected graph $G = (V, E)$ and a weight function $\omega\colon V \to \mathbb{R}$ on the nodes. Find a subset $V' \subseteq V$ with maximal weight such that $G[V']$ is connected.*

Problem 3.6 is NP-hard even if the weights are restricted to $+1, -1$ and G is planar with degree at most 3 [66]. In [34] it is shown, that for undirected graphs, Problem 3.6 is equivalent to the Price-Collecting Steiner Tree problem:

Problem 3.7 (Price-Collecting Steiner Tree). *Given a graph $G = (V, E)$ with edge and node weights, $\omega_E \colon E \to \mathbb{R}$ and $\omega_V \colon V \to \mathbb{R}$, respectively. The node weights are non-negative, and the edge weights are non-positive. Find a subtree $T = (V_T, E_T)$ such that the sum of $\omega_E(E_T)$ and $\omega_V(V_T)$ is maximized.*

This problem has been studied, especially computationally, for example in [40, 85].

In [15] a related polytope has been studied, where the edges induce a connected graph:

$$\mathcal{P}_E(G) := \operatorname{conv}(\{\chi(E') \in \{0,1\}^E \mid E' \subseteq E,\ G[E'] \text{ is connected}\}). \tag{3.2}$$

The connection between the polytopes $\mathcal{P}(G)$ and $\mathcal{P}_E(G)$ is given through the *line graph*.

Definition 3.8. Given a graph $G = (V, E)$, the *line graph* $L(G) = (E, E')$ of G has node set E and two nodes in $L(G)$ are adjacent if the corresponding edges in G have the same endpoint, that is, they are adjacent.

Thus, the results of [15] only hold for $\mathcal{P}(G)$ if G is the line graph of a graph. It is known, that line graphs can be characterized by forbidden subgraphs [13]. This implies, that the results from [15] do not necessarily hold for these forbidden subgraphs or supergraphs containing them. If the results from [15] apply in the following, it is explicitly mentioned.

A concept that is used frequently in the context of connectedness, is the following.

Definition 3.9 (Separator). Let $G = (V, E)$ be an undirected graph. For two distinct nodes u and v in V with $\{u, v\} \notin E$, a subset $N \subseteq V \setminus \{u, v\}$ is called an *u-v-node separator* if and only if there is no path from u to v in $G[V \setminus N]$. A separator N is *minimal* if $N \setminus \{w\}$ is not an u-v-node separator for every $w \in N$. Note that we leave out the term *node* if it is clear from context. Let $\mathcal{N}(u, v)$ be the set of all minimal u-v-separators.

Let $G = (V, E)$ be an undirected graph and define the following inequalities:

$$x_v \geq 0, \qquad v \in V, \tag{3.3}$$

$$x_v \leq 1, \qquad v \in V, \tag{3.4}$$

$$x(N) \geq x_u + x_v - 1, \qquad \{u, v\} \in \tbinom{V}{2},\ \{u, v\} \notin E,\ N \in \mathcal{N}(u, v). \tag{3.5}$$

With this, it is obvious that the connected subgraph polytope $\mathcal{P}(G)$ can also be defined by:

$$\mathcal{P}(G) = \operatorname{conv}(\{x \in \mathbb{Z}^V \mid x \text{ fulfills (3.3) to (3.5)}\}),$$

see also [22]. Note that is sufficient to use the *minimal* separators, as these inequalities dominate the inequalities which use normal separators.

Connectivity can also be modeled in different ways, e.g., by using extended formulation based on flow formulations, for more details see Section 3.3. In [108] one such possibility is presented using a multicommodity flow.

3.1.1 Facets

In this section, we introduce facet-defining inequalities for $\mathcal{P}(G)$ and some of their general structure. Lemmata 3.10 to 3.12 and 3.14 are known from [108]. Nevertheless, we include them here for completeness and we give alternative proofs for Lemmata 3.12 and 3.14. Furthermore, the ideas are used in the proof of Lemma 3.16, which is a generalization of Lemma 3.14.

Basic Results

Lemma 3.10. *The polytope $\mathcal{P}(G)$ is full-dimensional, e.g.,* $\dim \mathcal{P}(G) = |V|$.

Proof. Every solution containing only one node is in $\mathcal{P}(G)$: $e_v \in \mathcal{P}(G)$ for all $v \in V$. Also, the empty solution is in $\mathcal{P}(G)$: $\chi(\emptyset) \in \mathcal{P}(G)$. □

Lemma 3.11. *Inequality* (3.3) *is facet-defining for $\mathcal{P}(G)$.*

Proof. Consider $v \in V$. Every vector e_w with $w \in V \setminus \{v\}$ fulfills $e_v^\top e_w = 0$ and lies in $\mathcal{P}(G)$ as does the empty solution $\chi(\emptyset)$. □

Lemma 3.12. *Inequality* (3.4) *is facet-defining for $\mathcal{P}(G)$ if and only if G is connected.*

Proof. The case with $|V| = 1$ is obvious, therefore we only prove the case with $|V| > 1$. We start with the direction "$\Leftarrow$" and consider $v \in V$. Since G is connected, there exists a spanning tree T containing v. The incidence vector $\chi(T)$ is in $\mathcal{P}(G)$ and $\chi(T)_v = 1$.

As a tree on more than one node always contains at least two leafs, we can find a leaf w in T with $w \neq v$. Let T_w be the tree that is created by removing w and its incident edge from T. Its incidence vector is also in $\mathcal{P}(G)$ and it is 1 at position v: $\chi(T_w) \in \mathcal{P}(G)$ and $\chi(T_w)_v = 1$. The process can be repeated until there is only the node v left. These $|V| - 1$ vectors $\chi(T_w)$ and $\chi(T)$ are affinely independent, which proves that (3.4) is facet-defining for $\mathcal{P}(G)$ if G is connected.

On the other hand, if G is not connected and w is a node that is in a different connected component than v, the inequality $x_w + x_v \leq 1$ is feasible for $\mathcal{P}(G)$.

Together with $-x_w \leq 0$ this inequality adds up to $x_v \leq 1$, which thus implies that (3.4) cannot be facet-defining. □

To prove that minimal separator inequalities are also facet-defining, the following property of minimal separators is needed. This lemma first appeared in [45] as an exercise but since we did not find any proof in the literature, we include it here for completeness.

Lemma 3.13. *The set $N \subseteq V$ is a minimal u-v-separator of a graph $G = (V,E)$ if and only if every path from u to v contains a member of N and for every $w \in N$ there exists a path P from u to v with $P \cap N = \{w\}$.*

Proof. We start with the direction "$\Leftarrow$". If there is a path P from u to v with $P \cap N = \emptyset$, then N cannot be a separator. Let $w \in N$ be a member of the separator. If for every path P from u to v with $w \in P$ it holds that $|P \cap N| \geq 2$, then $N \setminus \{w\}$ also is a u-v-separator contradicting the minimality of N.

For the other direction assume that there exists a path P_w from u to v with $P_w \cap N = \{w\}$ for every $w \in N$. If N is not minimal, there exists an element $w \in N$ such that $N \setminus \{w\}$ is a separator. But then the path P_w connects u and v in $G[V \setminus (N \setminus \{w\})]$ which is a contradiction to the assumption that $N \setminus \{w\}$ is a separator. □

To simplify the notation we introduce the following two definitions. If N is a separator in $G = (V,E)$, the set $C_v \subseteq V$ denotes the connected component containing v in $V \setminus N$. Note that we leave out the separator N, meaning $C_v = C_v(N)$. A set S is *tight* for an inequality $\alpha^\top x \leq \beta$ if $\chi(S)$ fulfills that inequality with equality, that is, $\alpha^\top \chi(S) = \beta$.

Lemma 3.14. *Inequality* (3.5) *is facet-defining for the polytope $\mathcal{P}(G)$ for a connected graph $G = (V,E)$ if and only if N is a minimal u-v-separator.*

Proof. We first show that (3.5) is facet-defining for $\mathcal{P}(G)$ by constructing $|V|$ affinely independent tight solutions for (3.5).

Let $u,v \in V$ be two different nodes with $\{u,v\} \notin E$ and $N \in \mathcal{N}(u,v)$ be a minimal u-v-separator. There exists a spanning tree T in C_u since C_u is connected. As in the proof of Lemma 3.12 we can find spanning trees T_i which contain u. The $|C_u|$ vectors $\chi(T)$ and $\chi(T_i)$ are affinely independent. The connected component C_v can be treated analogously.

For every node $w \in N$ there exists a path P_w from u to v containing only nodes from C_u, C_v and w since N is a minimal separator and hence Lemma 3.13 applies. All the vectors $\chi(P_w)$ are affinely independent for different $w \in N$.

Every spanning tree T of C_b, $b \in V \setminus (C_u \cup C_v \cup N)$ is connected to a node w in N. Let c be a node in T that is adjacent to w. The vector $\chi(P_w) + \chi(T)$ is in

$\mathcal{P}(G)$. As above, we can find trees T_i that contain the connecting node c such that $\chi(P_w) + \chi(T_i)$ lies in $\mathcal{P}(G)$. All these solutions are affinely independent.

Together, this amounts to $|V|$ affinely independent feasible solutions which are tight for (3.5), thus (3.5) is facet-defining.

On the other hand, if N is not a separator, there exists a path P from u to v in $G[V \setminus N]$ since G is connected. Thus, Inequality (3.5) is not valid since $\chi(P)$ is feasible, $P \cap N = \emptyset$ and $x_u + x_v - 1 = 1 > 0 = x(N)$ if we insert $\chi(P)$ into (3.5). Furthermore, if N is not minimal, there exists a node $w \in N$ such that $N' := N \setminus \{w\}$ is a separator. Since $x(N') \geq x_u + x_v - 1$ is valid and together with $x_w \geq 0$ adds up to (3.5), it cannot be facet-defining. $\square$

Further Results

As a next step, we want to generalize Inequality (3.5). If (3.5) is written in the form $\alpha^\top x \leq \beta$, it holds that α contains exactly two positive entries. In the following generalization α may contain a larger number of positive entries. In order to describe this generalization, we first have to introduce additional notation.

Definition 3.15. Let $G = (V, E)$ be a graph and U be an independent set of nodes in G. Define N_U as an U-separator if the graph $G' = G[V \setminus N_U]$ is disconnected and u and v are in different connected components for every pair $u, v \in U, u \neq v$. Note that if $|U| = 2$, this definition is equal to the u-v-separator defined in Definition 3.9.

For every independent set U define the following inequality:

$$x(U) - 1 \leq (|U| - 1)x(N_U). \tag{3.6}$$

Note that in case of $|U| = 1$, Inequality (3.6) is equivalent to (3.4) and for $|U| = 2$ it is equal to (3.5).

Lemma 3.16. *If G is connected, Inequality* (3.6) *is facet-defining for $\mathcal{P}(G)$ if and only if N_U is a minimal U-separator for all $u, v \in U$, $u \neq v$.*

Proof. We proceed as in the proof of Lemma 3.14 by constructing $|V|$ affinely independent tight solutions to (3.6).

Because of the comments above, we can assume w.l.o.g. that $|U| > 2$. Let $G' = (V', E') = G[V \setminus N_U]$ be the subgraph of G induced by the nodes $V \setminus N_U$. As N_U is a separator, V' can be partitioned into connected components C_i, that is, $V' = C_1 \cup \cdots \cup C_k$. For every C_i there exists a spanning tree T_i. Note that $\chi(T_i)$ is tight for (3.6) if $T_i \cap U \neq \emptyset$. Let $t \in T_i \cap U$. As in the proof of Lemma 3.12 we can remove leaves from T_i which are not t such that all these trees are tight for (3.6).

All of these solutions are affinely independent. This procedure can be repeated for every connected component which contains an element of U.

Let $C \subseteq V'$ be the set of nodes consisting of all the nodes from all connected components of G' which contain an element of U, in particular, $C \cap U = U$ and $(V' \setminus C) \cap U = \emptyset$. For every node $w \in N_U$ there exists a spanning tree T_w in $G[C \cup \{w\}]$ because N_U is by assumption a minimal u-v-separator for all $u, v \in U, u \neq v$. Since in particular $T_w \cap N_U = \{w\}$ and $T_w \cap U = U$, the solutions $\chi(T_w)$ are tight for (3.6). Furthermore, these are affinely independent for all $w \in N_U$.

If there exist a connected component C_j in G' which does not contain an element of U, one can proceed as in the proof of Lemma 3.14: Select a spanning tree T of C_j and choose the vector $\chi(T) + \chi(T_w)$ for the appropriate $w \in N_U$ and the T_w defined as above (that is, $G[T_w \cup T]$ is connected). Then the same argument as in the proof of Lemma 3.14 allows for removing leaves from T leading to $|C_j|$ affinely independent solutions.

Together, this amounts to $|V|$ affinely independent solutions which are tight for (3.6), that is, (3.6) is facet-defining for $\mathcal{P}(G)$.

If N_U is not a separator for a pair of nodes $u \neq v \in U$, the same arguments from the proof of Lemma 3.14 can be applied to show that (3.6) cannot be facet-defining. To show the minimality assume for the sake of contradiction that there exist $u \neq v \in U$ such that N_U is not a minimal u-v-separator. Then there exists $w \in N_U$ such that $N_U \setminus \{w\}$ is still a u-v-separator. We want to show that the inequality

$$x(U) - 1 \leq (|U| - 1)x(N_U) - x_w \tag{3.7}$$

is valid for $\mathcal{P}(G)$ because together with the valid inequality $0 \leq x_w$ Inequality (3.7) adds up to (3.6) showing that (3.6) cannot be facet-defining.

To see this, we differentiate for $U' \subseteq U$ between the three cases $|U'| < |U|$, $U' = U$ with $w \in S$ and $U' = U$ with $w \notin S$. In the first case we have seen that N_U is a separator for the nodes from U' and $S \cap N_U \neq \emptyset$. This implies the following

$$\chi(S \cap U) - 1 = \chi(U') - 1 \leq |U| - 1 - 1 \leq (|U| - 1)\chi(S \cap N_U) - x_w,$$

which holds since $|U'| < |U|$ (first inequality) and $S \cap N_U \neq \emptyset$ (last inequality).

In the second case, it follows that $S \cap (N_U \setminus \{w\}) \neq \emptyset$, which implies $|S \cap N_U| \geq 2$. Since $N_U \setminus \{w\}$ is a separator and $|U| \geq 2$, it follows that

$$\chi(U) - 1 \leq 2(|U| - 1) - 1 \leq (|U| - 1)\chi(S \cap N_U) - x_w.$$

In the last case, it holds that $U' = U$ and $w \notin S$. This implies $|N_U \setminus \{w\} \cap S| \geq 1$, which also means $|N_U| \geq 2$. It then follows

$$\chi(U) - 1 \leq 2(|U| - 1) - 1 \leq (|U| - 1)\chi(N_U) - x_w,$$

which concludes the proof. □

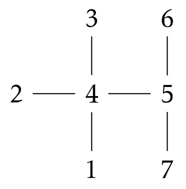


Figure 3.1: Graph used as an example to show that iteratively applying the idea from the proof of Lemma 3.16 is not possible in general

Remark 3.17. The last part of the previous proof leads to an interesting idea. Let U be an independent set and N_U be a U-separator with the corresponding inequality

$$x(U) - 1 \leq (|U| - 1)x(N_U). \tag{3.8}$$

If there exists a pair $u, v \in U$ such that there is an element $w \in N_U$ with the property that $N_U \setminus \{w\}$ is an u-v-separator, decrease the coefficient of x_w in (3.8) by 1. The question arises if repeatedly applying this idea leads to facet-defining inequalities. Unfortunately, it is not obvious on how to repeat the process or when to end it. For example, let $G = (V, E)$ be the graph defined by $V := \{1, \dots, 7\}$ and $E := \{\{1,4\}, \{2,4\}, \{3,4\}, \{4,5\}, \{5,6\}, \{5,7\}\}$ (see Figure 3.1). Let $U := \{1,2,3,6,7\}$ and $N_U := \{4,5\}$. In this case Inequality (3.8) is the following

$$x_1 + x_2 + x_3 + x_6 + x_7 - 1 \leq 4(x_4 + x_5),$$

which is not facet-defining, see Theorem 3.5. Applying the idea multiple times can lead to

$$x_1 + x_2 + x_3 + x_6 + x_7 - 1 \leq 2(x_4 + x_5),$$

which is facet-defining: direct the edges from the lower number to the larger number, instead of $\{5,7\}$ which is directed as $(7,5)$ and apply Theorem 3.5. After further applications of the idea, the resulting inequality is

$$x_1 + x_2 + x_3 + x_6 + x_7 - 1 \leq x_4 + x_5,$$

which is not valid since $\{1,2,3,4\}$ is a feasible solution cut off by the above inequality.

As a final statement in this section we prove a result about the general structure of facet-defining inequalities. Let

$$\sum_{i \in V} \pi_i x_i \leq \pi_0 \tag{3.9}$$

be a non-trivial facet-defining inequality. We can order the elements π_i by their sign:

$$\sum_{i \in U} \pi_i x_i - \sum_{i \in N_U} \pi_i x_i \leq \pi_0 \tag{3.10}$$

where $\pi_i > 0$ for all $i \in U \cup N_U$ and $\pi_i = 0$ for all $i \in V \setminus \{U \cup N_U\}$. Obviously, U and N_U can be defined by this inequality, but we will shortly show that U is independent and N_U defines a separator. In [108, Lemma 2] it is shown that $\pi_0 = 0$ if and only if the inequality is a scalar multiple of a non-negativity bound (3.3) (in Lemma 3.43 we show a generalization of this result). Moreover, $\pi_0 < 0$ is not possible since $\chi(\emptyset) \in \mathcal{P}(G)$. Therefore, we can assume $\pi_0 > 0$. Furthermore, in [108, Lemma 6] it is shown that no pair of adjacent nodes can have positive coefficients in a facet-defining inequality, meaning that in particular U is an independent set in G.

We want to show that N_U is a separator for U by extending [108, Lemma 5], which states that for two adjacent nodes a certain inequality holds.

Lemma 3.18. *Given a facet-defining inequality* (3.10)*, the nodes $u \in U$ are not connected in $G[V \setminus N_U]$, i.e., N_U is a separator for U.*

Proof. Given two nodes $u, v \in U$, we prove that if there exists a path P in $G[V \setminus N_U]$ from u to v, the following inequality holds:

$$(\pi_u + \pi_v)x_u + 0x_v + \sum_{w \in V \setminus \{u,v\}} \pi_w x_w \leq \pi_0. \tag{3.11}$$

The proof of the lemma then is similar to [108, Lemma 6]: If (3.11) holds, then, by exchanging u and v, so does

$$0x_u + (\pi_u + \pi_v)x_v + \sum_{w \in V \setminus \{u,v\}} \pi_w x_w \leq \pi_0. \tag{3.12}$$

Multiplying (3.11) by $\pi_u/(\pi_u + \pi_v)$ and (3.12) by $1 - \pi_u/(\pi_u + \pi_v)$ and summation of both resulting inequalities leads to Inequality (3.9), showing that (3.9) cannot be facet-defining. Note that this is clear if $\pi_u + \pi_v = 0$.

We assume that (3.11) is not valid. Then there exists $W \subseteq V$ such that $G[W]$ is connected but $(\pi_u + \pi_v)\chi(W)_u + 0\chi(W)_v + \sum_{w \in V \setminus \{u,v\}} \pi_w \chi(W)_w > \pi_0$. Consider two cases:

- Case $|\{u,v\} \cap W| = 0$ or 2:

$$\sum_{w \in V} \pi_w \chi(W)_w = (\pi_u + \pi_v)\chi(W)_u + 0\chi(W)_v + \sum_{w \in V \setminus \{u,v\}} \pi_w \chi(W)_w > \pi_0.$$

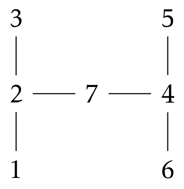


Figure 3.2: Graph used as an example to show that in a facet-defining inequality the separator for independent sets containing more than two elements does not have to be minimal

- Case $|\{u,v\} \cap W| = 1$. Then $W' = W \cup P$ is connected, and

$$\begin{aligned}\sum_{w\in V} \pi_w \chi(W')_w &= (\pi_u + \pi_v)\chi(W')_u + 0\chi(W')_v + \sum_{w\in V\setminus\{u,v\}} \pi_w \chi(W')_w \\ &\geq (\pi_u + \pi_v)\chi(W)_u + 0\chi(W)_v + \sum_{w\in V\setminus\{u,v\}} \pi_w \chi(W)_w \\ &> \pi_0.\end{aligned}$$

  The first inequality holds because $\pi_u, \pi_v > 0$ and $\pi_p \geq 0$ for $p \in P$ since $P \cap N_U = \emptyset$.

Thus, in all cases, the inequality $\sum_{w\in V} \pi_w x_w \leq \pi_0$ is not valid. □

In [108, Lemma 9] it is shown, that for $|U| = 2$, the resulting inequality has to be of type (3.5), meaning that N_U is a minimal u-v-separator. For $|U| > 2$, the separator N_U does not have to be minimal. For example consider the following graph $G = (V,E)$ with $V := \{1,\dots,7\}$ and $E := \{\{1,2\},\{2,3\},\{2,7\},\{4,7\},\{4,5\},\{4,6\}\}$ (see Figure 3.2). One facet-defining inequality is given by

$$x_1 + x_3 + x_5 + x_6 - (x_2 + x_4 + x_7) \leq 1,$$

which can be verified by Theorem 3.5. The independent set is, thus, given by $U = \{1,3,5,6\}$ and the separator by $N_U = \{2,4,7\}$. Note, however, that $N_U = \{2,4\}$ would suffice as a separator. Thus, direct generalizations of Lemma 3.14 are not possible and further restrictions are needed as for example the pairwise-minimality, see Lemma 3.16.

3.1.2 Complete Description of Cycles

Theorem 3.20 shows a complete description of $\mathcal{P}(G)$ in the case that G is a cycle. Cycles already have been treated in [15] and because the line graph of a cycle also is a cycle of the same length, the result from [15] is equivalent to Theorem 3.20. The result presented here was found independently from [15] and uses a different technique.

The following lemma, which holds also for general graphs G and not only for cycles, is used frequently throughout this thesis.

Lemma 3.19 ([78, p. 169]). *Let $\alpha^\top x \leq 1$ be a facet-defining inequality of $\mathcal{P}(G)$. For every pair of indices u and v there exists a tight solution containing exactly one of them.*

Proof. Assume that $x_u = x_v$ holds for all vertices on the hyperplane $\alpha^\top x = 1$. This means, it must be the hyperplane $x_u - x_v = 0$, which is impossible. □

Let $C_n = (\{0,\dots,n-1\},\{\{0,1\},\dots,\{n-2,n-1\},\{n-1,0\}\})$ be a cycle on n nodes. In this section all indices are considered modulo n.

Theorem 3.20. *The following inequalities are facet-defining for $\mathcal{P}(C_n)$:*

$$\sum_{\substack{j\in[q]\\ j\ odd}} x_{i_j} - \sum_{\substack{j\in[q]\\ j\ even}} x_{i_j} \leq 1, \tag{3.13}$$

$$\sum_{\substack{j\in[q]\\ j\ even}} x_{i_j} - \sum_{\substack{j\in[q]\\ j\ odd}} x_{i_j} \leq 1, \tag{3.14}$$

where $q \geq 4$ and $(i_1,\dots,i_q)$ is an even-length subsequence of $(1,\dots,n)$. *Moreover,* (3.3), (3.4), (3.13) *and* (3.14) *fully describe $\mathcal{P}(C_n)$.*

Before we prove the theorem, we need some more notation.

Definition 3.21 (Unimodal path). Given a facet-defining inequality $\alpha^\top x \leq \beta$ of $\mathcal{P}(C_n)$, we say that a path P from u to v, $u \neq v$ is *unimodal* if $\alpha_u > 0$, $\alpha_v > 0$ and $\alpha_i \leq 0$ for all $i \in P \setminus \{u,v\}$.

Note that from Lemma 3.18 it follows that for a unimodal path P there exists at least one node $w \in P$ with $\alpha_w < 0$.

For proving the theorem, we need a number of lemmata. For these, we assume that $\alpha^\top x \leq 1$ is a facet-defining inequality for $\mathcal{P}(C_n)$ with at least two positive entries in α.

Lemma 3.22. *The positive and negative entries in α alternate, that means, they are ordered as in* (3.13) *or* (3.14).

Proof. Because of Lemma 3.18 there is no path between two nodes with positive entries in α without using a node with a negative entry.

Assume there exists a path P between two distinct nodes u and v with $\alpha_u < 0$ and $\alpha_v < 0$. Furthermore, we assume P to be a shortest path between u and v, which means that $\alpha_w \geq 0$ for all $w \in P \setminus \{u,v\}$. If there exists a node $w \in P$ between u and v with $\alpha_w > 0$, there is nothing to prove. Thus, w.l.o.g. $\alpha_i = 0$ for all $i \in P \setminus \{u,v\}$. There must exist a tight solution S that contains exactly one element of $\{u,v\}$ because of Lemma 3.19. Note that S is a path and assume w.l.o.g. $u \in S$ and $v \notin S$. We can remove all end nodes j with $\alpha_j = 0$ from S without changing the fact that it is tight and connected. Call the remaining set S'. Then u has to be an end node of S'. The set $S' \setminus \{u\}$ is still connected and thus feasible but it violates $\alpha^\top x \leq 1$. □

Note that from Lemma 3.22 it follows that for any unimodal path P there exists exactly one node $w \in P$ with $\alpha_w < 0$.

Lemma 3.23. *The tight solutions S of $\alpha^\top x \leq 1$ are paths in C_n such that $S = P_1 \cup P_2 \cup P_3$ with $\alpha_i = 0 = \alpha_j$ for all $i \in P_1$ and $j \in P_3$ and $\alpha_u > 0$, $\alpha_v > 0$ for the end nodes u and v of P_2. In particular, both P_1 and P_3 are possibly empty. Furthermore, $S \neq C_n$.*

Proof. Note that the only connected sets in C_n are either the full cycle or paths. Consider S to be the full cycle. Then, by Lemma 3.22, there has to exist at least one node w with $\alpha_w < 0$. Removing the node from S results in a connected set which violates $\alpha^\top x \leq 1$.

Because P_1 and P_3 do not change the tightness of S, we can assume w.l.o.g. that $P_1 = P_3 = \emptyset$. Assume P_2 ends at a node v with $\alpha_v < 0$. Removing v from P_2 results in a connected set which violates $\alpha^\top x \leq 1$. □

Lemma 3.23 allows to assume that for every end node v of a tight solution S, it holds $\alpha_v > 0$ since otherwise $S \setminus \{v\}$ is still tight and feasible. This assumption is used in the following.

Lemma 3.24. *Let P be a unimodal path from u to v. If $\alpha_u = \alpha_v = 1$, then for every node $w \in P$ with $\alpha_w < 0$ it holds that $\alpha_w \leq -1$.*

Proof. Assume otherwise that $\alpha_w > -1$. Recall that u, v and w are the only nodes in P with a non-zero entry in α because of Lemma 3.22. Then the path P violates the inequality: $\alpha^\top \chi(P) > 1$. This is a contradiction because P is a feasible solution. □

Lemma 3.25. *If $\alpha_i = 1$ for all positive entries in α, then $\alpha_j = -1$ for all negative entries in α.*

Proof. From Lemma 3.23 we know that all tight solutions S of $\alpha^\top x \leq 1$ are concatenations of unimodal paths and Lemma 3.24 implies that all negative entries in α have to be less or equal to -1. The number of nodes with a negative entry in α has to be exactly one less than the number of nodes with a positive entry in α because of Lemma 3.22. Together with the tightness of S, this implies that all negative entries have to be -1. □

Lemma 3.26. *Let P be a unimodal path from u to v such that $w \in P$ is the node with $\alpha_w < 0$. If $\alpha_u < 1$ and $\alpha_v < 1$, then $|\alpha_w| \le \min\{\alpha_u, \alpha_v\}$.*

Proof. Let $P' \neq P$ be a unimodal path with v as an end node. Because of Lemma 3.22 there has to exist a node $q \in P'$ with $\alpha_q < 0$. Note that $q \neq w$. Every tight solution that contains q has also to contain v because of Lemma 3.23. Together with Lemma 3.19 this implies that there exists a tight solution S that contains v but not q. Because α_v is less than 1 and S is tight, it follows that $u \in S$. This also implies that $w \in S$ because of Lemma 3.23. W.l.o.g., we can assume that v is an end node of S. The set $S' := (S \setminus P) \cup \{u\}$ is connected and therefore $\alpha^\top \chi(S') \le 1$ has to hold. This leads to

$$\alpha^\top \chi(S') = \alpha^\top \chi(S) - \alpha_w - \alpha_v \le 1 = \alpha^\top \chi(S),$$

which implies $-\alpha_w \le \alpha_v$ or $|\alpha_w| \le \alpha_v$. Interchanging u and v leads to $|\alpha_w| \le \alpha_u$. □

Lemma 3.27. *If $\alpha_i \in \{0, -\gamma, \gamma\}$ for all i, then $\gamma = 1$.*

Proof. Let S be a tight solution to $\alpha^\top x \le 1$. Then

$$\alpha^\top x^S = \sum_{i:\alpha_i>0} \alpha_i \chi(S) + \sum_{j:\alpha_j<0} \alpha_j \chi(S) = 1.$$

From Lemma 3.23 we know that every tight solution is a concatenation of unimodal paths (with a possible addition of nodes with a zero entry in α). Lemma 3.22 implies that every unimodal path contains exactly one node with a negative entry in α. Therefore, the first sum contains exactly one more summand than the second sum. Thus,

$$1 = \sum_{i:\alpha_i>0} \alpha_i \chi(S) + \sum_{j:\alpha_j<0} \alpha_j \chi(S) = \sum_{i:\alpha_i>0} \gamma - \sum_{j:\alpha_j<0} \gamma = \gamma. \qquad \square$$

Lemma 3.28. *Let P be a path from u to v with $\alpha_u > 0$ and $\alpha_v > 0$. If $p \notin P$ with $\alpha_p > 0$ is in a unimodal path $\tilde{P}$ starting at an end node of P, the path P can be extended to a feasible solution $P' \supseteq P$ such that $p \in P'$. Furthermore, if $w \in \tilde{P}$ is the node with $\alpha_w < 0$ and P is a tight solution, then it holds that $|\alpha_w| \ge \alpha_p$.*

Proof. Define $P' := P \cup \tilde{P}$. If P is a tight solution, $\alpha^\top \chi(P) = 1$ holds. Because of the feasibility of P' we have that $\alpha^\top \chi(P') \leq 1$. Therefore,

$$\alpha^\top \chi(P) = 1 \geq \alpha^\top \chi(P') = \alpha^\top \chi(P) + \alpha_w + \alpha_p,$$

which implies $-\alpha_w \geq \alpha_p$ or $|\alpha_w| \geq \alpha_p$. □

Lemma 3.29. *There has to exist at least one index i with $\alpha_i = 1$.*

Proof. Assume that $\alpha_i < 1$ for all i. Let P be a unimodal path from u to v with w the node with $\alpha_w < 0$. There has to exist a tight solution S that does not contain v, otherwise $\alpha^\top x \leq 1$ would be contained in the trivial inequality $x_v \leq 1$.

A visualization of this proof can be seen in Figure 3.3.

If $u \notin S$, let z be the end node of S such that there is a path $\hat{P}$ from z to u with $\hat{P} \cap P = \{u\}$ and $\hat{P} \cap S = \{z\}$. In particular, $\alpha_z > 0$. Let $\tilde{P} \subset \hat{P}$ be a unimodal path from z to some node $p \in \hat{P}$ and $w' \in \tilde{P}$ the node with $\alpha_{w'} < 0$. Applying Lemma 3.28 leads to the feasible solution S' and $|\alpha'_w| \geq \alpha_p$. From Lemma 3.26 it follows that $|\alpha_{w'}| \leq \alpha_p$ which then implies $|\alpha_{w'}| = \alpha_p$. This also means that S' is a tight solution.

This process can be iterated to show that there exists a tight solution which contains u but not v. By applying Lemma 3.28 once more, we can show that $|\alpha_w| = \alpha_v$. Interchanging u and v leads to $\alpha_v = |\alpha_w| = \alpha_u$. Applying this procedure to all end nodes u and v of all unimodal paths leads to the fact that the absolute values of all non-zero entries of α have to be equal, meaning $\alpha_i \in \{0, \gamma, -\gamma\}$. Using Lemma 3.27 leads to a contradiction. □

Lemma 3.30. *Let P be a unimodal path from u to v and denote by $w \in P$ the node with $\alpha_w < 0$. If $\alpha_u = 1$ and $\alpha_v < 1$, then $|\alpha_w| = \alpha_v$.*

Proof. Because $\{u\}$ is tight and P is a feasible solution, $\alpha^\top \chi(P) \leq 1$ implies $|\alpha_w| \geq \alpha_v$:

$$\alpha^\top \chi(\{u\}) = 1 \geq \alpha^\top \chi(P) = \alpha^\top \chi(\{u\}) + \alpha_w + \alpha_v,$$

which implies $-\alpha_w \geq \alpha_v$ or $|\alpha_w| \geq \alpha_v$.

Let $P' \neq P$ be a unimodal path starting in v and let $w' \in P'$ be the node with $\alpha_{w'} < 0$. Every tight solution that contains w' also has to contain v because of Lemma 3.23. Together with Lemma 3.19 this implies that there exists a tight solution S that contains v but not w'. As before, we can assume w.l.o.g. that v is an end node of S. Note that S also has to contain w because $\alpha_v < 1$ and u because of Lemma 3.23. The set $S' := (S \setminus P) \cup \{u\}$ is connected and feasible. Therefore, $\alpha^\top \chi(S') \leq 1$ leads to

$$\alpha^\top \chi(S) = 1 \geq \alpha^\top \chi(S') = \alpha^\top \chi(S) - \alpha_w - \alpha_v,$$

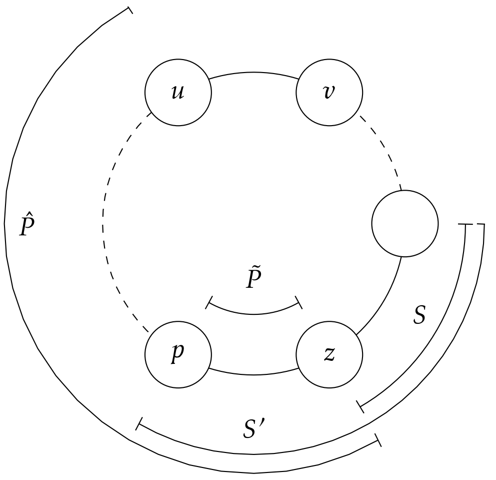


Figure 3.3: Visualization used for the proof of Lemma 3.29

which implies $\alpha_v \geq -\alpha_w$ or $|\alpha_w| \leq \alpha_v$. Together with the above this shows $|\alpha_w| = \alpha_v$. □

Lemma 3.31. *Let P be a unimodal path from u to v with $w \in P$ the node with $\alpha_w < 0$. If $0 < \alpha_u < 1$ and $0 < \alpha_v < 1$, then $\alpha_v = \alpha_u = |\alpha_w|$.*

Proof. Because of Lemma 3.29 there has to exist a node z with $\alpha_z = +1$. Let P' be the path from u to z with $v \notin P'$. W.l.o.g. assume that $\alpha_i < 1$ for all $i \in P' \setminus \{z\}$. The set $S := \{z\}$ is a tight solution. Let $\tilde{P} \subseteq P'$ be the unimodal path starting at z. Because of Lemma 3.30 we know that $S' := S \cup \tilde{P}$ is also a tight solution. As seen in the proof of Lemma 3.29 we can use Lemma 3.28 repeatedly to show that S' can be extended to $\tilde{S}$ until $\tilde{S} = P'$. With the same arguments as in the proof of Lemma 3.29 we see that $\tilde{S} = P'$ also has to be tight (see Lemma 3.28).

The set $\hat{S} := P \cup \tilde{S}$ is feasible and hence $\alpha^\top \chi(\hat{S}) \leq 1$ has to hold. Therefore,

$$\alpha^\top \chi(\tilde{S}) = 1 \geq \alpha^\top \chi(\hat{S}) = \alpha^\top \chi(\tilde{S}) + \alpha_w + \alpha_v,$$

which implies $-\alpha_w \geq \alpha_v$ or $|\alpha_w| \geq \alpha_v$. Because we can w.l.o.g. assume that $\alpha_u \leq \alpha_v$, Lemma 3.26 implies $\alpha_u = \alpha_v = |\alpha_w|$. □

Lemma 3.32. *Let P be a unimodal path from u to v and let $w \in P$ be the node with $\alpha_w < 0$. If $\alpha_u = \alpha_v = 1$, then $\alpha_w = -1$.*

Proof. Lemma 3.24 shows that $\alpha_w \leq -1$. Assume that $\alpha_w < -1$. There has to exist a tight solution S which contains w, otherwise $\alpha^\top x \leq 1$ would be contained in the trivial facet $x_w = 0$. From Lemma 3.23 we know that $u, v \in S$.

Because $\alpha^\top \chi(P) < 1$ holds, $P \subsetneq S$. Furthermore, there exists a unimodal path P' from a node $s \in S$ to another node $t \in S$ with the property that $|\alpha_{w'}| < \alpha_s$ or $|\alpha_{w'}| < \alpha_t$ for the unique node $w' \in P'$ with $\alpha_{w'} < 0$ in order to fulfill the tightness of S. Lemmata 3.30 and 3.31 show that this cannot happen. □

Additionally, we need one more theorem from the literature.

Theorem 3.33 ([108, Theorem 5]). *The indegree inequality corresponding to an orientation $D = (V, A)$ of G induces a facet of $\mathcal{P}(G)$ if and only if for every $u, v \in V$ there is at most one directed $u - v$ walk in D.*

With this, we can finally prove Theorem 3.20.

Proof of Theorem 3.20. First, we prove that (3.13) is facet-defining by stating an orientation D such that the corresponding indegree inequality is facet-defining. Let (3.13) be given by $\alpha^\top x \leq 1$. Let P be a path from u to w such that $\alpha_u = 1$, $\alpha_w = -1$ and $\alpha_i = 0$ for all $i \in P \setminus \{u, w\}$. The direction D is given by letting all edges in P be directed from u in the direction of w. Using this definition for every possible P results in a fully directed graph $D = (V, A)$. Inequality (3.13) implies that this is a feasible orientation. For every node u with $\alpha_u = 1$ also $d_u = 0$ holds, for every node w with $\alpha_w = -1$ it holds that $d_w = 2$ and $\alpha_v = 0$ implies $d_v = 1$ for every node v. Therefore, the corresponding indegree inequality is the same as (3.13). It is easy to see that for every pair of nodes s and t there exists at most one path from s to t. This means, that we can apply Theorem 3.33 to show that (3.13) is facet-defining.

For (3.14) the proof is exactly the same.

Let $\alpha^\top x \leq \beta$ define a facet F of $\mathcal{P}(C_n)$. First note that $\beta \geq 0$ because the empty set is connected. Also, if and only if $\beta = 0$, the inequality is a multiple of (3.3) [108, Lemma 2]. Therefore, we can assume that $\beta = 1$. If α contains exactly one positive entry α_i, then α_i has to be 1 because $\mathcal{P}(C_n)$ is full-dimensional (Lemma 3.10). Moreover, there cannot exist an index j with $\alpha_j < 0$ because this would imply $x_j = 0$. Then $x_j = 0$ and $x_i = 1$ has to hold for each point in F which implies that F cannot be a facet. Hence, $\alpha_j = 0$ for all $j \neq i$ and the inequality is the same as Inequality (3.4).

Therefore, let $\alpha^\top x \leq 1$ be a facet-defining inequality and let α contain at least two positive entries. From Lemma 3.22 we know that the positive and negative entries in α alternate.

It remains to show that the entries in α are 0, +1 or −1. Obviously, the positive entries can not be greater than 1 because every single node is connected.

Lemma 3.25 concludes the proof if the positive entries in α are +1 and Lemma 3.29 shows that not all positive entries in α are strictly less than 1. Because of Lemmata 3.30 to 3.32 the only remaining possibility is to have two different positive values: +1 and $0 < \gamma < 1$.

By Lemma 3.31 we can assume that $\alpha_i \in \{0, \gamma, -\gamma, +1, -1\}$ for all i. If there exists only one node w with $\alpha_w = 1$, then there does not exist a tight solution with $x_w = 0$ because $\gamma < 1$ and the positive and negative entries in α alternate. This is a contradiction because the equality $x_w = 1$ is not feasible for $\mathcal{P}(C_n)$. Therefore, there exist at least two nodes with an entry of $+1$ in α. Select two such nodes u and v with connecting path P such that P contains at least one node j with $\alpha_j = \gamma$. If there does not exist such a path, we have nothing more to prove. W.l.o.g. we can assume $\alpha_i \neq 1$ for all $i \in P \setminus \{u, v\}$. This implies that all nodes in $P \setminus \{u, v\}$ with a positive entry in α have value γ. From Lemmata 3.30 and 3.31 we know that $\alpha_i \neq -1$ for all $i \in P$. This implies that all nodes in P that have a negative entry in α have value $-\gamma$. Since there exists at least one node $j \in P$ with $\alpha_j = \gamma$, it holds that $\alpha^\top \chi(P) > 1$. This contradiction shows that $\alpha_i \in \{0, \pm 1\}$ which concludes the proof. □

3.1.3 Complete Description of Complete Bipartite Graphs

For n, $m \in \mathbb{N} \setminus \{0\}$ let $K_{n,m} = (A \cup B, \{\{a, b\} \mid a \in A, b \in B\})$ with $A \cap B = \emptyset$, $|A| = n$ and $|B| = m$ be a complete bipartite graph.

Theorem 3.34. *The following inequalities are facet-defining for $\mathcal{P}(K_{n,m})$:*

$$\sum_{a \in A'} x_a - (|A'| - 1) \sum_{b \in B} x_b \leq 1, \qquad A' \subseteq A,\ A' \neq \emptyset, \tag{3.15}$$

$$\sum_{b \in B'} x_b - (|B'| - 1) \sum_{a \in A} x_a \leq 1, \qquad B' \subseteq B,\ B' \neq \emptyset. \tag{3.16}$$

Moreover, (3.3), (3.15), *and* (3.16) *fully describe $\mathcal{P}(K_{n,m})$.*

Note that (3.15) and (3.16) are indegree inequalities if $A' = A$ or $B' = B$ and are otherwise of the type shown in (3.6). Before we can prove the theorem, we need two lemmata.

Lemma 3.35. *Let $\alpha^\top x \leq 1$ be a facet-defining inequality for $\mathcal{P}(K_{n,m})$. If $A^+ := \{i \in A \mid \alpha_i > 0\} \neq \emptyset$, then the tight solutions S are one of the following types:*

1. *$|S \cap B| = 0$. Then also $|S \cap A^+| = 1$ and $|S| = 1$.*

2. *$|S \cap B| = 1$. Then also $S \cap A^+ = A^+$.*

Furthermore, it holds that $\alpha_a \geq 0$ for all $a \in A$ and $\alpha_b \leq 0$ for all $b \in B$. Also, if $|A^+| > 1$, then $\alpha_b = \alpha_{b'} < 0$ for all b, $b' \in B$, otherwise, $\alpha^\top x \leq 1$ is the trivial inequality $x_a \leq 1$.

Proof. If S is feasible and $|S \cap B| > 1$, then also $|S \cap A| > 1$ has to hold: Let b, $b' \in B \cap S$, $b \neq b'$. If α_b and $\alpha_{b'}$ are both negative, the set $S \setminus \{b'\}$ is also feasible but violates $\alpha^\top x \leq 1$. That means there exists at most one element $b \in B$ with $\alpha_b < 0$ in S. If a tight solution S' does not contain any node from B, the only connected sets are the single node solutions $S' = \{i'\}$ for $i' \in A^+$ and in particular $\alpha_{i'} = 1$.

For every tight solution S with $b \in S$ it has to hold that $A^+ \subseteq S$. Otherwise, there would exist some $a' \in A^+$ which is not in S. Then, we can define $S \cup \{a'\}$ to get a feasible solution that violates $\alpha^\top x \leq 1$.

Assume that there exists an element $b \in B$ with $\alpha_b > 0$. Since every tight solution S contains at least one element from A or B, the set S has to contain all elements from $A \cup B$ that have a positive entry in α. This can be seen as above, because extending a tight solution that does not contain all elements from $A \cup B$ results in a violation of the inequality. Because there has to exist a tight solution, that only contains either $a \in A^+$ or b, there also has to be an element $q \in A \cup B$ with $\alpha_q < 0$ in S. Removing q from S is still feasible and violates the inequality. Therefore, there cannot exist an element $b \in B$ with $\alpha_b > 0$.

Assume that there exists an $\tilde{a} \in A$ with $\alpha_{\tilde{a}} < 0$. Then in order to have a tight solution S with $\tilde{a} \in S$, one element from B has to be in S and also $A^+ \subseteq S$ has to hold as shown above. The set $S \setminus \{\tilde{a}\}$ violates the inequality, showing that this is not possible.

If there exists a $b \in B$ with $\alpha_b < 0$, every tight solution S containing b has to contain all elements from A^+. If there also exists an element $b' \in B$ with $\alpha_{b'} > \alpha_b$, then $S \setminus \{b\} \cup \{b'\}$ is feasible and violates $\alpha^\top x \leq 1$. Of course, if $\alpha_{b'} < 0$, the roles of b and b' can be reversed to show that $\alpha_b = \alpha_{b'}$. This implies that either $\alpha_b = 0$ for all $b \in B$ or $\alpha_b = \alpha_{b'} < 0$ for all b, $b' \in B$.

If in the case that $\alpha_b = 0$ for all $b \in B$, it also holds that there exists an element $a' \neq a$, $a, a' \in A^+$, then for every tight solution S, it has to hold $a, a' \in S$ which is a contradiction to the dimension, see Lemma 3.19. Therefore, if $|A^+| > 1$, then $\alpha_b < 0$ for all $b \in B$. □

Lemma 3.36. *Let $\alpha^\top x \leq 1$ be a facet-defining inequality for $\mathcal{P}(K_{n,m})$. Define $V' = \operatorname{supp}(\alpha)$ as the support of α and $G' = G[V']$. Let f be the projection from $\mathbb{Z}^V$ to $\mathbb{Z}^{V'}$. If the facet induced by $\alpha^\top x \leq 1$ is projected to $\mathcal{P}(G')$, this projection is a facet of $\mathcal{P}(G')$, i.e., $f(\alpha)^\top x' \leq 1$ is a facet-defining inequality for $\mathcal{P}(G')$.*

Proof. If α contains exactly one positive entry, Lemmata 3.12 and 3.35 show the statement. So w.l.o.g. let α contain at least two positive entries with at least one $a \in A$ such that $\alpha_a > 0$.

Lemma 3.35 shows that $B \subseteq V$ is a subset of V'. This also means that G' is a complete bipartite graph $K_{|A'|,|B|}$ with $A' := \{i \in A \mid \alpha_i > 0\}$.

If $\chi(S)$ is a tight solution, then it holds that

$$1 = \alpha^\top x = \sum_{i\in V} \alpha_i x_i = \sum_{i\in V'} \alpha_i x_i + \sum_{i\in V\setminus V'} \alpha_i x_i. \tag{3.17}$$

Because the second sum is zero according to the definition of V', the set $f(S)$ is a tight solution for $\mathcal{P}(G')$.

Lemma 3.35 states the structure of every tight solution and both types are connected in G'. Let $W := \operatorname{span}\{f(S) \mid S \text{ tight solution to } \alpha^\top x \leq 1 \text{ in } \mathcal{P}(G)\}$ be the projection of the span of all tight solutions. Because $\dim W = \dim V'$, it follows that $f(\alpha)^\top x' \leq 1$ is a facet-defining inequality for $\mathcal{P}(G')$. □

With these we can finally prove Theorem 3.34.

Proof of Theorem 3.34. To prove that (3.15) is facet-defining, it suffices to construct $|A|+|B|$ affinely independent tight solutions because of Lemma 3.10. Given a subset $A' \subseteq A$ and an inequality of the type (3.15), we see that e_i for $i \in A'$ is a tight solution. Also $\sum_{i'\in A'} e_{i'} + e_j$ is a tight solution for every $j \in B$. Finally, $\sum_{i'\in A'} e_{i'} + e_b + e_a$ is a tight solution for some $b \in B$ and for every $a \in A \setminus A'$. That all these solutions are affinely independent is easy to see because every solution introduces one new variable. The proof that (3.16) is facet-defining follows by interchanging A and B.

To show that this gives a full description, let $\alpha^\top x \leq \beta$ be a facet-defining inequality. As in the proof of Theorem 3.20 it suffices to assume that $\beta = 1$. If there exists only one positive entry in α, Lemma 3.35 shows that the inequality is trivial. So assume that α contains at least two positive entries. This in particular means that $|A'| > 1$ holds for $A' := \{i \in A \mid \alpha_i > 0\}$.

Given $a, a' \in A'$, Lemma 3.19 guarantees that there has to exist a tight solution S such that w.l.o.g. $a \in S$ and $a' \notin S$. From Lemma 3.35 we can see that this means that $S = \{a\}$. Therefore, $\alpha_a = 1$. Repeating the argument for all pairs $\{i, i'\}, i, i' \in A'$ and $i \neq i'$ shows that $\alpha_{i'} = 1$ for all $i' \in A'$ with only one possible exception $i^\star \in A'$.

We have shown that there are $|A'|-1$ tight solutions which contain exactly one node from A' and there are $|B|$ tight solutions containing exactly one node from B. But there are also $2^{|A|-|A'|} - 1$ possible tight solutions for the nodes from $A \setminus A'$ because every subset of these nodes can be added to every solution containing at least one node from B. We also have shown that these are the only possibilities for tight solutions in Lemma 3.35. Hence, in the case of $A' = A$ there is one solution missing for the inequality to be facet-defining which can only be remedied by setting $\alpha_{i^\star} = 1$. Thus, the proof is concluded in the case of $A' = A$.

Let $V' = \operatorname{supp}(\alpha)$ be the support of α and let f be the projection from V to V' and $G' = G[V']$. If there is some facet-defining inequality with a node $i^\star$ with $0 < \alpha_{i^\star} < 1$, Lemma 3.36 shows that the $f(\alpha)^\top x' \leq 1$ should also be a facet of

$\mathcal{P}(G')$. Since we know a complete description of $\mathcal{P}(G')$ by the first part of the proof and there are no nodes with an entry of 0 in $f(\alpha)$, there cannot exist such an $i^\star$. $\square$

3.1.4 Adding One Node and One Edge

Let $G = (V,E)$ be a graph and let $G' = (V',E')$ be a graph that is obtained by adding one additional node q to V and one edge connecting q to a node $w \in V$, meaning $V' := V \cup \{q\}$ and $E' := E \cup \{q,w\}$. This implies in particular, $\dim \mathcal{P}(G) = \dim \mathcal{P}(G') - 1$.

Lemma 3.37. *If a full description of $\mathcal{P}(G)$ is known, a full description of $\mathcal{P}(G')$ can be derived as follows: For every inequality $\alpha^\top x \le \beta$ in the full description of $\mathcal{P}(G)$, add the two inequalities*

1. $(\alpha')^\top x \le \beta$ *and*
2. $(\alpha')^\top x + \beta x_q - \beta x_w \le \beta$

to $\mathcal{P}(G')$, where $\alpha' \in \mathbb{R}^{V'}$ with $\alpha'|_V = \alpha$ and $\alpha'_q = 0$. Furthermore, the non-negativity constraint $x_q \ge 0$ has to be added. The resulting description is also complete.

Proof. First, we show that the stated inequalities are facet-defining. For the non-negativity constraint, see Lemma 3.11. Note that all solutions feasible in G are also feasible in G'. Thus, in case of the first inequality, there exist a set X containing $|V|$ tight affinely independent points because $\alpha^\top x \le \beta$ is facet-defining for $\mathcal{P}(G)$. From these, we want to generate a set X' of size $|V'| = |V| + 1$ points S' feasible for $\mathcal{P}(G')$, which are tight for $(\alpha')^\top x \le \beta$.

For every $S \in X$, we first define $S' \in X'$ by $S'|_V = S$ and $\chi(S)_q = 0$ resulting in $|X'| = |X|$. Furthermore, we can assume w.l.o.g. that there has to exist at least one tight solution $S \in X$ with $\chi(S)_w = 1$ because otherwise $\chi(S)_w = 0$ for all tight solutions S, which implies that the inequality is trivial. Define an additional solution $S'' \in X'$ for this S by $S''|_V = S$ and $\chi(S'')_q = 1$. Since $\chi(S)_w = 1$, the solution $\chi(S'')$ is feasible and also tight for $\mathcal{P}(G')$. Moreover, $\chi(S'')$ is affinely independent from the above $|X'| - 1$ points, meaning that X' contains $|V'| = |V| + 1$ affinely independent tight solutions.

For the second type of inequality the argument is analogous: There exist $|V|$ tight affinely independent solutions because $\alpha^\top x \le \beta$ is facet-defining for $\mathcal{P}(G)$. For each of these solutions S we set $\chi(S')_q = 1$ if $\chi(S)_w = 1$ and $\chi(S')_q = 0$ otherwise. It is clear, that the solutions for $\mathcal{P}(G')$ are also tight for $(\alpha')^\top x + \beta x_q - \beta x_w \le \beta$ and affinely independent. Moreover, the solution e_q is also tight and affinely independent from the other solutions, meaning $|V'|$ affinely independent tight solutions in total.

Second, to show completeness of the description let $(\alpha')^\top x \le \beta$ be a facet-defining inequality for $\mathcal{P}(G')$. W.l.o.g. we can assume $\beta > 0$ because otherwise, the inequality is a scalar multiple of a non-negativity constraint, [108, Lemma 2].

It has to be shown that $(\alpha')^\top x \le \beta$ is of one of the remaining two types defined above. We distinguish the two cases $\alpha'_q = 0$ and $\alpha'_q \neq 0$. In the first case, any tight solution $S' \in X'$ for $\mathcal{P}(G')$ gives rise to a tight solution $S \in X$ for $\mathcal{P}(G)$ if we define $S := S'|_V$. Thus, for a set X' of affinely independent tight solutions S', the span of X has exactly one dimension less then X' since this process is a projection. Consequently, the inequality $(\alpha')^\top x \le \beta$ is of the first type.

For the second case with $\alpha'_q \neq 0$ we first show that $\alpha'_q = \beta$. Because e_q is feasible, we have $\alpha'_q \le \beta$. If $\alpha'_q < 0$, then removing q from a tight solution containing q results in increasing the left-hand side to a value larger than β, which is a contradiction. Therefore, $0 < \alpha'_q \le \beta$. Because of affine independence, there has to exist a tight solution S with $\chi(S)_q \neq \chi(S)_w$. If $\chi(S)_w = 1$ and $\chi(S)_q = 0$ in S, then setting $\chi(S)_q = 1$ results in a feasible solution with left-hand side larger than β, which is a contradiction. Thus, there has to exist a feasible solution S with $\chi(S)_q = 1$ and $\chi(S)_w = 0$. Because of the construction of G', the only feasible solution is $\{q\}$, meaning that e_q has to be a tight solution, which implies $\alpha'_q = \beta$.

If we write $(\alpha')^\top x \le \beta$ as $(\alpha'')^\top x + \beta x_q - \beta x_w \le \beta$ with $\alpha = \alpha''|_V$, the inequality $\alpha^\top x \le \beta$ is facet-defining for $\mathcal{P}(G)$: As shown above, the only tight solution S with $\chi(S)_w \neq \chi(S)_q$ is e_q, thus $\chi(S)_w = \chi(S)_q$ holds for every other tight solution. Therefore, all solutions unequal to e_q are also tight for $\alpha^\top x \le \beta$, which means that we can derive $|V|$ affinely independent feasible tight solutions for $\mathcal{P}(G)$, showing that $\alpha^\top x \le \beta$ is facet-defining for $\mathcal{P}(G)$ which proves the statement. □

Note that the two inequalities above are indegree inequalities given in Definition 3.4 in the case of $\beta = 1$. Directing the new edge into q results in q having indegree 1, meaning a coefficient of 0 and thus the first type of inequality from Lemma 3.37. Directing the edge from q to w results in the second type. Since trees can be completely described by adding exactly one node and one edge, Lemma 3.37 applied repeatedly leads to the same result as Theorem 3.5.

3.2 The Connected Subpartition and Partition Polytopes

In this section we generalize the polytope $\mathcal{P}(G)$ by also allowing different connected components. This directly corresponds to the topic of Chapter 2, where both Problems 2.1 and 2.14 contained partitioning problems. First, in Section 3.2.1, we allow a node to not belong to any partition, thereby directly generalizing $\mathcal{P}(G)$. Second, in Section 3.2.2, we study the case of full partitions, meaning that every node has to be in exactly one partition.

3.2.1 The Connected Subpartition Polytope

Because in this section only a subset of the nodes of an undirected graph $G = (V, E)$ needs to be partitioned, we use the term subpartition. The polytope which is studied in this section can be defined as follows.

Definition 3.38. Let $G = (V, E)$ be a graph and $\mathcal{K} \in \mathbb{N}$ be an integer. Furthermore, let $V_0 \cup V_1 \cup \dots \cup V_{\mathcal{K}} = V$ be a *subpartition* of V, that is, $V_0 \cup V_1 \cup \dots \cup V_{\mathcal{K}}$ is a partition, where $V_i = \emptyset$ is allowed for $i \in [\mathcal{K}]_0$. The connected subpartition polytope $\mathcal{P}^{\leq}_{\mathcal{K}}(G)$ is defined as

$$\begin{aligned}\mathcal{P}^{\leq}_{\mathcal{K}}(G) := \operatorname{conv}\Bigg(\Bigg\{\chi\Bigg(\bigcup_{i\in[\mathcal{K}]}(V_i \times \{i\})\Bigg) \in \{0,1\}^{V\times[\mathcal{K}]} \;\Bigg|\; & \\ V_0 \cup V_1 \cup \dots \cup V_{\mathcal{K}} = V \text{ is a subpartition of } V, & \\ G[V_i] \text{ is connected for every } i \in [\mathcal{K}]\Bigg\}\Bigg). &\end{aligned}$$

Note that in particular there are no constraints on V_0. Moreover, this definition implies that $\mathcal{P}^{\leq}_{1}(G)$ and $\mathcal{P}(G)$ are isomorphic, see Definition 3.1.

Similar to the previous section, we can also define $\mathcal{P}^{\leq}_{\mathcal{K}}(G)$ by using separator inequalities with additional packing constraints:

$$x^i_v \geq 0, \qquad v \in V, i \in [\mathcal{K}], \tag{3.18}$$

$$\sum_{i\in[\mathcal{K}]} x^i_v \leq 1, \qquad v \in V, \tag{3.19}$$

$$\sum_{v\in N} x^i_v \geq x^i_u + x^i_w - 1, \qquad \{u,w\} \in \binom{V}{2}, \{u,w\} \notin E, N \in \mathcal{N}(u,w), i \in [\mathcal{K}], \tag{3.20}$$

where we use the superscript to denote the partition index. We also use this notation for the unit vectors, that is, e^i_v is 1 for node v and partition i and 0 otherwise. Note that in slight abuse of notation the term *vector* is used instead of matrix but the corresponding spaces are isomorphic. Inequality (3.19) ensures that every node is in at most one partition and (3.20) is used for connectedness as in the previous section. The connected subpartition polytope $\mathcal{P}^{\leq}_{\mathcal{K}}(G)$ can then also be defined as follows (compare Section 3.1):

$$\mathcal{P}^{\leq}_{\mathcal{K}}(G) = \operatorname{conv}(\{x \in \mathbb{Z}^{V\times[\mathcal{K}]} \mid x \text{ fulfills (3.18) to (3.20)}\}).$$

Note that, in contrast to Problem 2.14, we do not only omit the isomorphism but also allow partitions consisting of less than $\mathcal{K}$ components. To enforce that the $\mathcal{K}$ partitions are each non-empty, define the polytope $\tilde{\mathcal{P}}^{\leq}_{\mathcal{K}}(G)$ by adding the following constraint to $\mathcal{P}^{\leq}_{\mathcal{K}}(G)$

$$\sum_{v \in V} x^i_v \geq 1, \qquad i \in [\mathcal{K}]. \tag{3.21}$$

Studying the polytope $\tilde{\mathcal{P}}^{\leq}_{\mathcal{K}}(G)$, however, is more complicated than the subpartition polytope $\mathcal{P}^{\leq}_{\mathcal{K}}(G)$. This can be seen by the following example with $\mathcal{K} = 2$. Let $G_1 = (\{1,2,3,4\}, \{\{1,2\},\{2,3\},\{3,4\}\})$ be a path on four nodes and $G_2 = K_4$ be a complete graph on four nodes. There exist 6 integer points in $\tilde{\mathcal{P}}^{\leq}_{2}(G_1)$ resulting in a dimension of 3, whereas the 14 integer points in $\tilde{\mathcal{P}}^{\leq}_{2}(G_2)$ lead to a dimension of 4. Note that although both G_1 and G_2 are connected, the dimensions of $\tilde{\mathcal{P}}^{\leq}_{\mathcal{K}}(G_1)$ and $\tilde{\mathcal{P}}^{\leq}_{\mathcal{K}}(G_2)$ are not the same. This implies that in general the dimension of $\tilde{\mathcal{P}}^{\leq}_{\mathcal{K}}(G)$ depends on the structure of G more heavily than the other polytopes inspected here. A detailed study is out of scope of this thesis and instead we focus on $\mathcal{P}^{\leq}_{\mathcal{K}}(G)$ in this section.

Lemma 3.39. *The polytope $\mathcal{P}^{\leq}_{\mathcal{K}}(G)$ is full dimensional.*

Proof. Every unit vector e^i_v is contained in $\mathcal{P}^{\leq}_{\mathcal{K}}(G)$ as well as the zero vector. Affine independence is obvious. □

Lemma 3.40. *Inequality* (3.18) *is facet-defining for $\mathcal{P}^{\leq}_{\mathcal{K}}(G)$.*

Proof. Every unit vector e^i_w with $w \neq v$ or $i \neq k$ is contained in $\mathcal{P}^{\leq}_{\mathcal{K}}(G)$ and tight for (3.18). The same holds for the zero vector. Affine independence is obvious. □

To simplify the notation for this and the next section, we introduce a shifted modulo calculation.

Definition 3.41. The k-modulo mod_k for $k \in \mathbb{N}$ is defined via

$$a \,\mathrm{mod}_k\, k := (a - 1 \mod k) + 1.$$

This notation ensures that the result of a calculation mod_k is in $[k]$ instead of $[k-1]_0$, which is the case when using the normal modulo calculation.

The $\mathcal{K}$-modulo is implicitly used in the following proofs and also in the next section.

Lemma 3.42. *Inequality* (3.19) *is facet-defining for $\mathcal{P}^{\leq}_{\mathcal{K}}(G)$ if $\mathcal{K} > 1$.*

Proof. For all $i \in [\mathcal{K}]$ define the solutions e_v^i and $e_v^i + e_w^{i+1}$ for $w \neq v$. These are $|V|\mathcal{K}$ many tight and feasible solutions to (3.19), which are obviously affinely independent. □

Note that if $\mathcal{K} = 1$, Inequality (3.19) is facet-defining if and only if G is connected, see Lemma 3.12.

As a next step, we want to show that (3.20) is facet-defining. This is done by first generalizing [108, Lemma 2] to a broader class of polytopes. Although we assume that this result is already known, we could not find any reference.

Lemma 3.43. *Let $\mathcal{P}$ be an n-dimensional polytope in $\mathbb{R}_+^n$ with $0 \in \mathcal{P}$. Furthermore, let there exist numbers $q_i > 0$ such that $q_i e_i \in \mathcal{P}$ for all $i \in [n]$. If*

$$\alpha^\top x \leq \beta \tag{3.22}$$

defines a facet of $\mathcal{P}$, it holds that $\beta \geq 0$ and $\beta = 0$ if and only if Inequality (3.22) *is a scalar multiple of a trivial inequality $x_i \geq 0$.*

Proof. Because $x = 0$ is feasible, $\beta \geq 0$.

Let $\beta = 0$. Since $q_i e_i \in \mathcal{P}$, it follows that $\alpha_i \leq 0$ for all $i \in [n]$. Since $x \in \mathbb{R}_+^n$, the inequality $\alpha_i x_i \leq 0$ is valid. This implies that $\alpha^\top x \leq \beta$ can be written as the sum of multiples of non-negativity constraints $-x_i \leq 0 \Leftrightarrow x_i \geq 0$ and is in particular not facet-defining.

If there are two entries k, ℓ with $\alpha_k < 0$ and $\alpha_\ell < 0$, then $\alpha_\ell x_\ell \leq 0$ is valid and also

$$\sum_{\substack{i=1\\ i \neq \ell}}^{n} \alpha_i x_i \leq 0$$

is valid because $\alpha_i \leq 0$. As the sum of these two is (3.22), it cannot be facet-defining. Hence, there is at most one coefficient k with $\alpha_k < 0$. This means that (3.22) is a scalar multiple of the trivial inequality $-x_i \leq 0 \Leftrightarrow x_i \geq 0$.

To show the other direction, $x_i \geq 0$ obviously is of the form $\alpha^\top x \leq \beta$ with $\beta = 0$. □

Obviously, Lemma 3.43 can be applied to $\mathcal{P}_{\mathcal{K}}^{\leq}(G)$. To show that (3.20) is also facet-defining, the following lemma shows a more general result. Basically, we can obtain facets for $\mathcal{P}_{\mathcal{K}}^{\leq}(G)$ by lifting from $\mathcal{P}(G)$ with a specific $i \in [\mathcal{K}]$. The idea is similar to the one found in [64].

Lemma 3.44. *If*

$$\alpha^\top x = \sum_{v \in V} \alpha_v x_v \leq \beta \tag{3.23}$$

defines a non-trivial facet of $\mathcal{P}(G)$, then the inequality

$$\sum_{v \in V} \alpha_v x_v^i \le \beta \tag{3.24}$$

defines a facet of $\mathcal{P}_{\mathcal{K}}^{\le}(G)$ for all $i \in [\mathcal{K}]$.

Proof. Lemma 3.43 implies $\beta > 0$ since (3.23) has to define a non-trivial facet. Because (3.23) defines a facet, there exists a set X of size $|V|$ of affinely independent feasible points that are tight for (3.23). For a point $x \in X$ and $i \in [\mathcal{K}]$ define $\tilde{x} \in \mathcal{P}_{\mathcal{K}}^{\le}(G)$ by $\tilde{x}_v^i = x_v$ and $\tilde{x}_v^\ell = 0$ for all $\ell \in [\mathcal{K}]$, $\ell \ne i$. Then $\tilde{x}$ is tight for (3.24) and because x is feasible for $\mathcal{P}(G)$, the point $\tilde{x}$ is feasible for $\mathcal{P}_{\mathcal{K}}^{\le}(G)$.

For each $v \in V$ there exists an $x \in X$ with $x_v = 0$: Otherwise, $x_v = 1$ for all $x \in X$ and (3.23) would be contained in the trivial inequality $x_v \le 1$, which would be a contradiction to the fact that (3.23) is non-trivial. For this $v \in V$ we define $\mathcal{K} - 1$ points $\tilde{x} + e_v^\ell$ for all $\ell \ne i$, which are feasible for $\mathcal{P}_{\mathcal{K}}^{\le}(G)$ and tight for (3.24). Since this holds for all $v \in V$, these are $|V|(\mathcal{K} - 1)$ points. The feasibility follows from the fact that sets containing a single node are connected by definition, compare also to Section 3.1.1.

Together with the $|X| = |V|$ points from above, these are $|V|\mathcal{K}$ points, which are easily seen to be affinely independent. □

Corollary 3.45. *Inequality (3.20) is facet-defining for $\mathcal{P}_{\mathcal{K}}^{\le}(G)$.*

Proof. Inequality (3.5) defines a non-trivial facet of $\mathcal{P}(G)$, see Lemma 3.14. Applying Lemma 3.44 shows that (3.20) is facet-defining for $\mathcal{P}_{\mathcal{K}}^{\le}(G)$. □

Similar to this corollary, Lemma 3.44 can also be used to show other inequalities to be facet-defining, for example derived from (3.6) or (3.1).

Corollary 3.46. *If G is connected, the inequality*

$$\sum_{u \in V} x_u^i - 1 \le (|U| - 1) \sum_{s \in N_U} x_s^i, \qquad i \in [\mathcal{K}]$$

is facet-defining for $\mathcal{P}_{\mathcal{K}}^{\le}(G)$ if and only if N_U is a minimal U-separator for all $u, v \in U$, $u \ne v$.

Proof. Use Lemma 3.16 and Lemma 3.44. □

Corollary 3.47. *The inequality*

$$\sum_{v \in V} (1 - d_v) x_v^i \le 1, \qquad i \in [\mathcal{K}]$$

for an indegree vector d of an orientation $D = (V, A)$ of G induces a facet of $\mathcal{P}_{\mathcal{K}}^{\le}(G)$ if and only if for every $u, v \in V$ there is at most one directed $u - v$ walk in D.

Proof. Apply Theorem 3.33 and Lemma 3.44. □

3.2.2 The Connected Partition Polytope

In this section we are interested in the connected partition polytope defined as follows.

Definition 3.48. Given a graph $G = (V, E)$ and an integer $\mathcal{K} \in \mathbb{N}$, the connected partition polytope $\mathcal{P}^{=}_{\mathcal{K}}(G)$ is defined as

$$\mathcal{P}^{=}_{\mathcal{K}}(G) := \operatorname{conv}\left(\left\{\chi\left(\bigcup_{i\in[\mathcal{K}]}(V_i \times \{i\})\right) \in \{0,1\}^{V\times[\mathcal{K}]} \;\middle|\; \begin{array}{l} V_1 \cup \cdots \cup V_{\mathcal{K}} = V \text{ is a subpartition of } V, \\ G[V_i] \text{ is connected for every } i \in [\mathcal{K}] \end{array}\right\}\right).$$

In particular, there is no V_0 in contrast to $\mathcal{P}^{\leq}_{\mathcal{K}}(G)$.

The difference between $\mathcal{P}^{\leq}_{\mathcal{K}}(G)$ and $\mathcal{P}^{=}_{\mathcal{K}}(G)$ is that in the first polytope a node does not have to belong to a partition, or in other words, only a subset of the nodes is partitioned instead of the whole set of nodes. This implies that $\mathcal{P}^{=}_{\mathcal{K}}(G)$ is a face of $\mathcal{P}^{\leq}_{\mathcal{K}}(G)$ and can be defined by the IP formulation derived from the following inequalities:

$$x_v^i \geq 0, \qquad v \in V, i \in [\mathcal{K}], \tag{3.25}$$

$$\sum_{i\in[\mathcal{K}]} x_v^i = 1, \qquad v \in V, \tag{3.26}$$

$$\sum_{v\in N} x_v^i \geq x_u^i + x_w^i - 1, \qquad \{u,w\} \in \binom{V}{2}, \{u,w\} \notin E, N \in \mathcal{N}(u,w), i \in [\mathcal{K}], \tag{3.27}$$

where the only difference to $\mathcal{P}^{\leq}_{\mathcal{K}}(G)$ is the equality in (3.26), see Section 3.2.1.

There exist similar partition problems involving connectivity in the literature. For example, the problem whether a given sequence $n_1, \ldots, n_{\mathcal{K}}$ with $\sum n_i = |V|$ is realizable for a graph $G = (V, E)$ is studied. A sequence is *realizable*, if there exists a partition $V_1, \ldots, V_{\mathcal{K}}$ of V such that $G[V_i]$ is connected and $|V_i| = n_i$ for all $i \in [\mathcal{K}]$. The article [14] gives a recent overview of the complexity for this and variations of this problem.

In [67] the so-called s-partition problem is studied. This is defined as follows: For $s \in \mathbb{N}$, an *s-partition* of $G = (V, E)$ is a partition of E into $E_1 \cup \cdots \cup E_{\mathcal{K}}$, where $|E_i| = s$ for $i \in [\mathcal{K} - 1]$ and $1 \leq |E_{\mathcal{K}}| \leq s$ such that $G[E_i]$ is connected for each $i \in [\mathcal{K}]$.

A more general concept can be found in [35], which studies the problem of partitioning the nodes or edges of a graph into equal sized parts such that these induce a particular type of graph. In particular, connected graphs are one of these types. The paper is mainly concerned with the computational complexity of these problems.

Returning to $\mathcal{P}^{=}_{\mathcal{K}}(G)$, let c be the number of connected components of G. If $\mathcal{K} = c$, the dimension of $\mathcal{P}^{=}_{\mathcal{K}}(G)$ is $(\mathcal{K}-1)^2$ since it is isomorphic to the Birkhoff polytope, which is the convex hull of all doubly stochastic $\mathcal{K} \times \mathcal{K}$ matrices (see for example [113]). A square matrix $A = (a_{ij})$ is *doubly stochastic* if its entries are non-negative real numbers such that both the row and column sum is 1, that is,

$$\sum_i a_{ij} = 1 = \sum_j a_{ij}.$$

If the entries are also integers, i.e., in $\{0,1\}$, matrix A is called a *permutation matrix*. These two polytopes are isomorphic since all nodes of a connected component have to be in the same part and we can therefore assume w.l.o.g. that G is an empty graph with $|V| = c$. As each node of G has to be in a different part to correspond to a point in $\mathcal{P}^{=}_{\mathcal{K}}(G)$, the solutions directly correspond to permutation matrices. Together with the Birkhoff-von Neumann theorem (see for example [86]), which states that the vertices of the Birkhoff polytope are permutation matrices, the isomorphism between the Birkhoff polytope and $\mathcal{P}^{=}_{\mathcal{K}}(G)$ follows.

If $\mathcal{K} < c$, the polytope $\mathcal{P}^{=}_{\mathcal{K}}(G)$ is empty because there does not exist a single feasible point. Therefore, we assume $c < \mathcal{K}$ in the following.

Note that every linear independent equality reduces the dimension by one and since there are exactly $|V|$ equalities in the description of $\mathcal{P}^{=}_{\mathcal{K}}(G)$, its maximal dimension is $|V|\mathcal{K} - |V| = |V|(\mathcal{K}-1)$. In the following, we show that this is indeed the dimension of $\mathcal{P}^{=}_{\mathcal{K}}(G)$.

Remark 3.49. The $\mathcal{K}$-partition polytope inspected in [24] is defined by additionally using edge-variables and it is mentioned that its dimension is $|V|(\mathcal{K}-1) + |E|$ referring to [25] for a proof.

For the next proofs, we introduce the concept of a *cyclic shift*, which is used repeatedly in the following. This shift allows us to define $\mathcal{K}-2$ additional feasible points of $\mathcal{P}^{=}_{\mathcal{K}}(G)$ from one given feasible point and one part index. Remember that the $\mathcal{K}$-modulo from Definition 3.41 is used if it is applicable.

Definition 3.50. Let S be a subset of the nodes V and $i \in [\mathcal{K}]$ be an index. Let $x \in \mathcal{P}^{=}_{\mathcal{K}}(G)$ be a feasible point such that exactly the nodes from S are in part i, i.e., $x^i_v = 1$ for all $v \in S$ as well as $x^i_w = 0$ for all $w \in V \setminus S$. One *cyclic shift* of x is defined by the point $\hat{x}$ as a result of moving every node which is not in S to the next part and every node in S is fixed to part i. Formally, define the shift $\hat{x}$ of x as

$$\hat{x}^j_v := \begin{cases} x^j_v, & \text{if } v \in S, \\ x^{j-1}_v, & \text{if } v \in V \setminus S, \end{cases}$$

where the part index $j-1$ is the maximal number less than j, that is not i and is calculated $\mathrm{mod}_{\mathcal{K}}$. For example, if $\mathcal{K}=4$, $i=1$ and $j=2$, then $j-1=4$ or if $i=\mathcal{K}$ and $j=1$, then $j-1=\mathcal{K}-1$. Note that in particular, $\hat{x}$ is a point in $\mathcal{P}^{=}_{\mathcal{K}}(G)$.

Note that we can apply a cyclic shift also to the point $\hat{x}$, meaning that we apply two cyclic shifts to the point x. In particular, a cyclic shift can be applied $\mathcal{K}-2$ times to a feasible point in $\mathcal{P}^{=}_{\mathcal{K}}(G)$ to generate a set of $\mathcal{K}-1$ affinely independent points that are in $\mathcal{P}^{=}_{\mathcal{K}}(G)$. This idea is crucial for the following proofs.

To determine the dimension of $\mathcal{P}^{=}_{\mathcal{K}}(G)$, we first inspect the special case that G is connected in Lemma 3.51 and proceed with an empty graph in Lemma 3.52. The ideas of both proofs are then combined to determine the dimension of $\mathcal{P}^{=}_{\mathcal{K}}(G)$ for general graphs in Lemma 3.53.

Lemma 3.51. *The dimension of $\mathcal{P}^{=}_{\mathcal{K}}(G)$ for connected graphs $G=(V,E)$ is $|V|(\mathcal{K}-1)$ if $\mathcal{K}>1$.*

Proof. W.l.o.g. we can assume that G is a spanning tree with root node r since more edges only increase the number of feasible points and therefore the dimension cannot be decreased by adding edges. Because G is connected, all $\mathcal{K}$ points $\chi(V_i)$ for $i\in[\mathcal{K}]$, meaning all nodes V are in part i, are feasible. W.l.o.g. we can also assume $|V|>1$.

For every of the $|V|-1$ edges $e\in E$, the graph $G[E\setminus\{e\}]$ has exactly two connected components. Let these components be called C_r and C_s, respectively. W.l.o.g. we assume for the root node r that $r\in C_r$. Define the point $x\in\mathcal{P}^{=}_{\mathcal{K}}(G)$ by the following partition of G: part 1 contains all nodes from C_r and part 2 all nodes from C_s, that is, $V\setminus C_r$. Define $\mathcal{K}-2$ additional feasible solutions by cyclically shifting x while fixing the nodes from C_r to part 1, meaning that only the nodes from C_s are shifted. Iterating this process for every edge results in $(\mathcal{K}-1)(|V|-1)$ feasible points. Together with the $\mathcal{K}$ points from above this yields $|V|(\mathcal{K}-1)+1$ feasible points of $\mathcal{P}^{=}_{\mathcal{K}}(G)$. Let this set be called S.

We show affine independence of the points in S by iteratively removing elements from S, which are the unique points having a 1 in some specific entry. We can remove the $\mathcal{K}-1$ points, where the root node r is in another part than the first since r is in part $i\neq 1$ for exactly one point in S. Let the resulting subset be $S_1\subsetneq S$.

For every neighbor $v\in\Gamma(r)$, the node v is only in parts $2,\dots,\mathcal{K}$ if in the above process the edge $\{v,r\}$ is removed from G. That allows for the removal of the corresponding points from S_1 leading to $S_2\subsetneq S_1$. For every neighbor $w\in\Gamma(v)$ with $w\neq r$, there exists only one point in S_2 with w being in part i for $i\in\{2,\dots,\mathcal{K}\}$, namely, when the edge $\{w,v\}$ is removed. Thus, the argument above can be iterated to remove all points from S until only the point remains, where every node is in part 1. Thus, we have shown that S is affinely independent, which concludes the proof. □

Lemma 3.52. *The dimension of $\mathcal{P}_{\mathcal{K}}^{=}(G)$ for empty graphs $G = (V, E)$ is $|V|(\mathcal{K} - 1)$ if $|V| = c < \mathcal{K}$.*

Proof. Let the nodes be numbered from 1 to n. For each node $v \in [n]$, we create $\mathcal{K} - 1$ different points that are in $\mathcal{P}_{\mathcal{K}}^{=}(G)$. Basically, one node is fixed to part 1 and the others are sorted in the remaining parts, where a cyclic shift of these leads to $\mathcal{K} - 1$ different solutions. More formally, the parts are defined by $V_1 := \{v\}$ and the remaining nodes $(1, \dots, v-1, v+1, \dots, n)$ are in that order the only elements of the parts $V_{2+j}, V_{3+j}, \dots, V_{n+j}$ for $j \in [\mathcal{K} - 2]_0$, since we assume $|V| < \mathcal{K}$. The index $\ell + j$ of the parts is calculated by

$$\ell + j = \begin{cases} \ell + j, & \text{if } \ell + j < \mathcal{K} + 1, \\ (\ell + j) \bmod (\mathcal{K} + 1) + 2, & \text{if } \ell + j \geq \mathcal{K} + 1, \end{cases}$$

with $\ell \in \{2, \dots, n\}$. Together these are $|V|(\mathcal{K} - 1)$ points in $\mathcal{P}_{\mathcal{K}}^{=}(G)$. Since $\mathcal{K} > |V|$, there also exists a point in $\mathcal{P}_{\mathcal{K}}^{=}(G)$ where part 1 is empty, for example the point corresponding to $V_2 := \{1\}, \dots, V_{n+1} := \{n\}$. Together these add up to $|V|(\mathcal{K} - 1) + 1$ points in $\mathcal{P}_{\mathcal{K}}^{=}(G)$. Thus, it only remains to show affine independence.

Assume there exists a linear combination of the corresponding vectors that adds up to the zero vector with the property that all coefficients also add up to 0. To show affine independence, we have to prove that all coefficients have to be 0. Let the coefficients be named $a_1^1, \dots, a_1^{\mathcal{K}-1}, a_2^1, \dots, a_n^{\mathcal{K}-1}$ and q corresponding to the above created points in the same order.

Inspect the subset of points corresponding to solutions with a node in part 1. Since the corresponding subset of the linear combination has to be 0 and q is the only coefficient not in this particular subset, the coefficient q is 0.

It follows from the construction that if in one of the vectors x from above we have $x_v^i = 1$, then also $x_{v+1}^{i+1} = 1$ holds for $v < |V|$ and $1 < i < \mathcal{K}$. From the linear combination of the zero vector follows that the summation of all vectors with $x_v^i = 1$ has to be 0 and so does the summation of all vectors with $x_{v+1}^{i+1} = 1$. Note that the difference between the corresponding sets of vectors only lies in those two vectors, where either v or $v + 1$ is assigned to part 1, that is, in the first set exists exactly one vector with $x_{v+1}^1 = 1$ and in the second one with $\tilde{x}_v^1 = 1$. This implies equality of the corresponding coefficients. This argument can be iterated to show that all coefficients have to be equal since for every v exist $i \in [\mathcal{K}]$ and $w \in V \setminus \{v\}$ such that $x_w^i = 1$ and $x_v^1 = 1$. This implies that all coefficients have to be 0 since their sum is 0. □

Combining both lemmata leads to the dimension of $\mathcal{P}_{\mathcal{K}}^{=}(G)$ for all graphs after we define the following construction. Let T be a tree with root node r. Assume T to be directed "away from the root", i.e., if the nodes are numbered after the appearance in a BFS tree started at r, the arcs are directed from nodes with a lower number to nodes with a larger number. The subtree T' *rooted at* v is then defined as the subtree containing node v and all nodes reachable from v in the directed tree T.

Lemma 3.53. *The dimension of $\mathcal{P}^{=}_{\mathcal{K}}(G)$ for a graph $G = (V,E)$ is $|V|(\mathcal{K}-1)$ if $c < \mathcal{K}$.*

Proof. Again, we construct $|V|(\mathcal{K}-1)+1$ affinely independent feasible points by combining the ideas from the proofs of Lemmata 3.51 and 3.52. Let the connected components C_i of G be numbered from 1 to c. For every C_i let T_i be a spanning tree with root r_i.

Let $i \in [c]$ and $v_i \in T_i$. All nodes from C_i defined by the subtree T_{v_i} rooted at v_i constitute part 1. All remaining nodes from C_i constitute part $i+1$. The remaining parts are then defined by the other connected components C_j, $j \neq i$ ordered by their respective label. Or, in other words, part 1 contains the nodes from T_{v_i}, part $i+1$ contains the nodes from C_i that are not in T_{v_i} and part $1+j$ contains the nodes from C_j. The parts $2,\dots,c+1$ are then cyclically shifted similar to the proof of Lemma 3.51 fixing part 1. Applying this procedure for every node v results in $\mathcal{K}-1$ feasible points in $\mathcal{P}^{=}_{\mathcal{K}}(G)$ generated from v, that is, $|V|(\mathcal{K}-1)$ in total.

One additional vector q is created by defining the parts $V_2,\dots,V_{c+1}$ by the connected components $1,\dots,c$, meaning that the nodes of every connected component belong to the same part and part 1 is empty. Together, this results in $|V|(\mathcal{K}-1)+1$ feasible points in $\mathcal{P}^{=}_{\mathcal{K}}(G)$, which we call S.

As in the proof of Lemma 3.52 we again show the affine independence by proving that the coefficients in a linear combination of the zero vector have to be 0. Since the sum over all vectors corresponding to solutions with nodes in part 1 have to be 0, the coefficient corresponding to q has to be 0 because q is the only point with $\chi(q)^1_v = 0$ for all $v \in V$ and the linear combination of all vectors adds up to 0.

Let r_1 and r_2 be two roots of two connected components C_{r_1} and C_{r_2} with $C_{r_2} = C_{r_1}+1$. Let $i \in [\mathcal{K}]$, $1 < i < \mathcal{K}$. Let $S_{r_1} \subseteq S$ be the subset of vectors $\tilde{x}$ with $\tilde{x}^i_{r_1} = 1$ and $S_{r_2} \subseteq S$ be the subset of vectors $\hat{x}$ with $\hat{x}^{i+1}_{r_2} = 1$. The subset of the above linear combination containing the vectors in S_{r_1} has to be zero as well as the subset consisting of the vectors in S_{r_2}. This in particular means that the corresponding sums are equal. The construction implies that for almost all vectors it holds that, whenever $x \in S_{r_1}$ then also $x \in S_{r_2}$, with the only exception of the one vector $y \in S_{r_1}$ with $y^1_{r_2} = 1$ (compare to the proof of Lemma 3.52). The same argument implies that there exists one vector $z \in S_{r_2}$ with $z^i_{r_1} = 0$, namely the one where $z^1_{r_1} = 1$. Combining both arguments implies that the coefficients corresponding to y and z have to be equal. This argument can be iterated and modified for all root nodes also for $C_{r_2} = C_{r_1}+j$, $j > 1$ where only the part index has to be adapted accordingly.

To extend this idea to the children of root nodes, let v be a child of some root node r. Furthermore, let S_v be the subset of vectors for which v is in part 1 and let S_r be the subset of vectors for which node r is in part 1. It holds that $S_r \subseteq S_v$ since whenever node r is in part 1, so is v. Since the sum over all coefficients corresponding to vectors from S_v or S_r has to be 0, this also holds for the coefficients corresponding to the vectors from $S_v \setminus S_r$. Then the argument from above can be applied to show the equality of the corresponding coefficients. Obviously, this argument can be applied iteratively to show that all coefficients have to be equal. This concludes the proof since the coefficients add up to 0, in which case equality implies that all have to be 0. $\square$

Next, we want to show that Inequality (3.25) is facet-defining. Again, we proceed as in the proof of the dimension, that is, we first show in Lemma 3.54 that this holds true in the special case that G is connected. We proceed by showing the special case that G is an empty graph in Lemma 3.55. Both ideas are then combined to finally show this statement for general graphs G in Lemma 3.56.

Lemma 3.54. *Inequality* (3.25) *is facet-defining for $\mathcal{P}^{=}_{\mathcal{K}}(G)$ for connected graphs $G = (V, E)$ if $1 = c < \mathcal{K}$.*

Proof. We prove the statement by constructing $|V|(\mathcal{K}-1)$ affinely independent feasible tight solutions to (3.25). Let T be a spanning tree of G with root node v.

For every node $w \in V$, $w \neq v$ define part i by all the nodes of the subtree of T rooted at w. The remaining nodes constitute part j, $j = \min\{\ell \in [\mathcal{K}], \ell \neq i\}$. By cyclically shifting these remaining nodes and fixing part i, we construct $\mathcal{K}-1$ feasible tight solutions to (3.25). Repeating this process for every $w \neq v$ therefore results in $(|V|-1)(\mathcal{K}-1)$ feasible tight solutions because v is the root node of T.

Additional $\mathcal{K}-1$ solutions are generated by defining the parts $V_j := V$ for all $j \in [\mathcal{K}]$, $j \neq i$. Hence, the number of feasible tight solutions generated by these ideas is $|V|(\mathcal{K}-1)$.

To show the affine independence we assume a linear combination of the zero vector as before. Let S_1 be the set of vectors with v in part $j \neq i$ and let $q \in V$ be a child of v in T. Define S_2 as the set of vectors corresponding to solutions with q in part j. Then $S_2 \subseteq S_1$ and $|S_1 \setminus S_2| = 1$ with the solution corresponding to q being assigned to part i. Since as before the summation over the corresponding coefficients has to be 0, this implies that the coefficient corresponding to q is 0. This process can be iterated for all other children of v and its grandchildren, showing that all coefficients corresponding to vectors with some node assigned to part i are zero. The remaining $\mathcal{K}-1$ vectors, in which no node is assigned to part i are easily seen to be affinely independent since every node is exactly once in every part except part i. $\square$

Lemma 3.55. *Inequality* (3.25) *is facet-defining for* $\mathcal{P}^{=}_{\mathcal{K}}(G)$ *for empty graphs* $G = (V,E)$ *if* $|V| = c < \mathcal{K}$.

Proof. We prove the statement by constructing $|V|(\mathcal{K}-1)$ affinely independent feasible tight solutions to (3.25). For every node $w \in V$, $w \neq v$, fix w to part i and cyclically shift the remaining nodes as explained in the proof of Lemma 3.52. This results in $(|V|-1)(\mathcal{K}-1)$ feasible tight solutions. The missing $\mathcal{K}-1$ vectors are created by defining the parts V_j, $j \in [\mathcal{K}], j \neq i$ by the nodes $1,\dots,|V|$, leaving part i empty and cyclically shifting these solutions while fixing part i as in the proofs above.

Assume a labeling of the nodes from 1 to $|V|$ and assume that v has number 1. Let S_v be the set of vectors x with $x_v^j = 1$ for $j \neq i$. Furthermore, let $w = v+1$ and S_w be the set of vectors x with $x_w^{j+1} = 1$. It holds that $S_w \subseteq S_v$ and $|S_v \setminus S_w| = 1$. Let $q \in S_v \setminus S_w$. Note that for q it holds that $\chi(q)^i_w = 1$, meaning in q the node w is assigned to part i. Since the sum over all vectors from S_v and S_w has to be 0, respectively, the coefficient for q has to be 0. The same argument can also be applied to other nodes from V, with the corresponding correct part number. By iterating the above argument for all parts $j \neq i$ and all nodes $w \neq v$ this shows that all coefficients corresponding to vectors in which a node is assigned to part i have to be 0. That the remaining coefficients have to be 0 as well can be shown as in the proof of Lemma 3.54. □

Lemma 3.56. *Inequality* (3.25) *is facet-defining for* $\mathcal{P}^{=}_{\mathcal{K}}(G)$ *for every graph* $G = (V,E)$ *if* $c < \mathcal{K}$.

Proof. We prove the statement by constructing $|V|(\mathcal{K}-1)$ affinely independent feasible tight solutions to (3.25) using a combination of the ideas from the proofs of Lemmata 3.54 and 3.55. Let the connected components C_ℓ be numbered $1,\dots,c$, with v being in the first component. Let T_ℓ be a spanning tree of C_ℓ with root node r_ℓ and let $r_1 := v$.

For every connected component C_j, $j \in [c]$, $j \neq 1$ create the vectors as in the proof of Lemma 3.53, that is, for every node $w \in C_j$ fix all nodes contained in the corresponding subtree rooted at w to part i and cyclically shift the remaining nodes. For component C_1 we do the same, except for node v. Therefore, this process results in $(|V|-1)(\mathcal{K}-1)$ feasible tight solutions. Additional solutions are created by partitioning the nodes such that all nodes belonging to a connected component are in the same part and the components are sorted with respect to their labeling skipping part i (compare to the proof of Lemma 3.55). These solutions are again cyclically shifted fixing the empty part i to yield the missing $\mathcal{K}-1$ feasible tight solutions.

To prove the affine independence, we explain the idea for the root node r of component 2, which then can easily be adapted to the other root nodes of

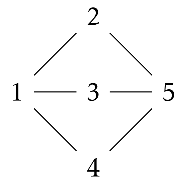


Figure 3.4: Graph used as a counterexample to show that Inequality (3.27) does in general not define a facet for $\mathcal{P}^{=}_{\mathcal{K}}(G)$

the remaining components, then to their children and with that for all nodes of other connected components than C_1. The idea is the same as in the proof of Lemma 3.55. Whenever v is in part $j \neq i$, $j \neq i-1$, node r is in part $j+1$ with the sole exception of the case that r is in part i. The corresponding coefficient therefore has to be 0. Note that if $j = i-1$, then instead of $j+1$, the part $i+1$ has to be used.

For nodes from C_1, we use the same argument as in the proof of Lemma 3.54 to show that the coefficients of vectors corresponding to solutions in which nodes from C_1 are assigned to part i have to be 0.

As in both proofs of Lemmata 3.54 and 3.54 it can be shown that the coefficients corresponding to vectors which represent the solutions with no node being assigned to part i have to be 0. This concludes the proof. □

Obviously, Equality (3.26) cannot define a facet. Moreover, for Inequality (3.27) we cannot use the technique for the subpartition polytope $\mathcal{P}^{\leq}_{\mathcal{K}}(G)$ from Section 3.2.1 because if $\mathcal{K} = 1$ and all nodes have to belong to a partition, there is only one solution if the graph is connected and none otherwise. This means, there is no obvious choice for a polytope from which the inequalities could be lifted. Furthermore, Inequality (3.27) does in general not define a facet. To see this, let $G = (\{1,\ldots,5\},\{\{1,2\},\{1,3\},\{1,4\},\{2,5\},\{3,5\},\{4,5\}\})$ (see Figure 3.4) and $\mathcal{K} = 2$. Note that $\{2,3,4\}$ is a minimal 1-5-separator and inspect the inequality $x^1_2 + x^1_3 + x^1_4 \geq x^1_1 + x^1_5 - 1$. The only two feasible tight solutions for this inequality are derived from the parts $V_1 = \{1\}, V_2 = \{2,\ldots,5\}$ and $V_1 = \{5\}, V_2 = \{1,\ldots,4\}$ and therefore there are not enough affinely independent solutions to allow for the inequality to be facet-defining.

3.3 Solving Connected Max-K-Cut

This section deals with the definition and solution method for the Connected Max-$\mathcal{K}$-Cut problem. We present a MIP formulation of Connected Max-$\mathcal{K}$-Cut and solve it with the framework SCIP [87]. After introducing the problem in Section 3.3.1 we show how to enforce connectedness of the solutions in Section 3.3.2. We extend the solution process by using our own propagation routine, which is described in Section 3.3.3. Furthermore, we add additional cuts and explain their separation routines in Section 3.3.4. The additional heuristics that we implemented are introduced in Section 3.3.5, followed by branching rules in Section 3.3.6. Since the solutions contain symmetries, we explain how we handle them for solving CONNECTED MAX-$\mathcal{K}$-CUT in Section 3.3.7. Finally, in Section 3.3.8 we present our numerical results. This section is based on the joint work with Hojny, Joormann and Schmidt [56].

3.3.1 Problem Definition

To state the CONNECTED MAX-$\mathcal{K}$-CUT problem, we first need one additional definition.

Definition 3.57. Let $G = (V,E)$ be a graph and $\mathcal{K} \in \mathbb{N}$. For a partition $V_1,\dots,V_{\mathcal{K}}$ we define a *cut* δ as $\delta(V_1,\dots,V_{\mathcal{K}}) := \{\{v,w\} \in E \mid v \in V_i, w \in V_j, i \neq j\}$, that is, the edges between different partitions. In particular, for $V' \subseteq V$ we define $\delta(V') := \delta(V', V \setminus V')$. If G is weighted with weight function $\omega\colon E \to \mathbb{R}$, we define the weight of cut δ as the sum over the weight of all edges in the cut, meaning $\omega(\delta(V_1,\dots,V_{\mathcal{K}})) := \sum_{\{v,w\}\in\delta(V_1,\dots,V_{\mathcal{K}})} \omega(\{v,w\})$.

With this we can define the main problem of this section.

Problem 3.58 (CONNECTED MAX-$\mathcal{K}$-CUT). *Let $G = (V,E)$ be a weighted graph with weight function $\omega\colon E \to \mathbb{R}$ and let $\mathcal{K} \in \mathbb{N}$. Find a partition $V_1,\dots,V_{\mathcal{K}}$ with $V_i \neq \emptyset$ and $G[V_i]$ connected for every $i \in [\mathcal{K}]$ such that the weight over all edges in the induced cut $\delta(V_1,\dots,V_{\mathcal{K}})$ is maximized.*

CONNECTED MAX-$\mathcal{K}$-CUT is NP-complete for planar graphs and $\mathcal{K} = 2$ [54]. If we do not require the connectedness of each partition, the problem is called MAX-$\mathcal{K}$-CUT and in the special case of $\mathcal{K} = 2$ its name is MAX-CUT. The literature available for both these problems is huge and there also exist many variations. For extensive surveys on MAX-CUT see [30, 31, 95]. Polyhedral results can be found for example in [23, 24]. A recent article on solving the MAX-$\mathcal{K}$-CUT using semidefinite programming can be found in [7].

In terms of complexity, it suffices to inspect Problem 3.58 for connected graphs since otherwise there exists a polynomial time reduction as the following lemma shows. Note, that Lemma 3.59 is a result from the author of this thesis and does not appear in [56].

Lemma 3.59. *Let $G = (V,E)$ be a disconnected graph. By solving a polynomial number of instances of Problem 3.58 for connected graphs, Problem 3.58 can be solved for G.*

Proof. Let c be the number of connected components of G. Any connected component can contain at most $\mathcal{K}-c+1$ partitions in order to solve Problem 3.58. For each connected component C_ℓ of G, solve Problem 3.58 with $\mathcal{K} = 1,\dots,\mathcal{K}-c+1$ and call these solutions s_ℓ^i for $\ell \in [c]$ and $i \in [\mathcal{K}-c+1]$. We create a weighted directed acyclic graph $D = (W, A)$ with $\omega\colon A \to \mathbb{R}$ in which a special longest path describes a solution to Problem 3.58 for G. Note that longest paths in acyclic directed graphs can be determined in polynomial time, see, e.g., [103].

The nodes W are defined by all solutions s_ℓ^i and two additional nodes s and t. The arcs A are defined as

$$A := \{(s, s_1^i) \mid i \in [\mathcal{K}-c+1]\} \cup \bigcup_{\ell\in[c-1]} \{(s_\ell^i, s_{\ell+1}^j) \mid i,j \in [\mathcal{K}-c+1]\} \cup \{(s_c^i, t) \mid i \in [\mathcal{K}-c+1]\},$$

that is, s has arcs to every solution of the first component, whereas every solution of component ℓ has an arc to every solution of component $\ell+1$ and every solution of the last component has an arc to t. We define the weight of every arc a as the value of the solution s_ℓ^i corresponding to the head of a and all arcs ending in t are weighted with 0.

Moreover, we have to introduce a budget-function $b\colon A \to \mathbb{N}$ on the arcs. If (d, s_ℓ^i) is an arc, we define its budget to be i, that is, the budget for an arc is the number of partitions used in component ℓ in the solution s_ℓ^i. On every arc ending in t we define the budget to be 0.

Every longest path P from s to t corresponds to a maximal cut in G. In particular, every longest path P which satisfies the budget constraint $b(P) = \mathcal{K}$ is an optimal solution to Problem 3.58 for G. Since the budget constraint is integer-valued, finding P is possible in polynomial time, see, e.g., [89]. Additionally, this can be seen by creating $\mathcal{K}$ copies of D, where choosing an arc with budget q corresponds to going from copy j to copy $j+q$ but further details are omitted. □

Because of Lemma 3.59 we assume G to be connected in the following.

Problem 3.58 can be modeled as a mixed-integer program as follows. First, we define the binary variable x_v^i for every node $v \in V$ and $i \in [\mathcal{K}]$. These should be 1 if node v is in partition i and 0 otherwise. The variable y_{vw} for $v, w \in V$ models whether nodes v and w belong to different parts, i.e., $y_{vw} = 1$ means that v and w are in different parts, whereas $y_{vw} = 0$ implies that v and w belong to the same part. The objective function can then be written as

$$\max \sum_{\{v,w\}\in E} \omega(\{v,w\}) y_{vw}. \tag{3.28}$$

The partition constraint, which models that every node is in exactly one partition, can be stated as

$$\sum_{i\in[\mathcal{K}]} x_v^i = 1, \qquad v \in V, \tag{3.29}$$

see also (3.26). If there exist $i \in [\mathcal{K}]$ and an edge $\{v,w\} \in E$ such that the values x_v^i and x_w^i differ, the variable y_{vw} has to be forced to 1 for modeling that the corresponding edges connects two nodes from different parts. This can be achieved by the following constraints

$$x_v^i - x_w^i \le y_{vw}, \qquad \{v,w\} \in E, i \in [\mathcal{K}], \tag{3.30}$$

$$x_w^i - x_v^i \le y_{vw}, \qquad \{v,w\} \in E, i \in [\mathcal{K}]. \tag{3.31}$$

Enforcing y_{vw} to 0 if x_v^i and x_w^i are both equal to 1 for an edge $\{v,w\} \in E$ and one $i \in [\mathcal{K}]$ to model the reverse direction, leads to the following constraint

$$x_v^i + x_w^i + y_{vw} \le 2, \qquad \{v,w\} \in E, i \in [\mathcal{K}]. \tag{3.32}$$

As we want to enforce to use all $\mathcal{K}$ parts, we use (3.21). Note that depending on the edge weights, these constraints are not always necessary. For example, if all edge weights are positive, any optimal solution uses the maximum of $\mathcal{K}$ parts implying the validity (3.21).

We use two different approaches to model the connectedness. The first is by using flow variables, whereas the second uses separators. In the first case we need additional variables but only a polynomial number of constraints. In the second case we do not need additional variables but the number of constraints may be exponential. Here we show both possibilities and compare them in the numerical results.

We start with the description of using flow variables. The idea is that every node has to have an outflow of 1, except $\mathcal{K}$ distinguished nodes, which serve as representative nodes modeling an "artificial sink". Flow is only allowed on edges, that are not in the cut. This implies that every node has to be connected to a representative node in order for the outflow to vanish using only edges that are not in the cut. This implies connectivity of the parts.

The binary variables ζ_v^i model if node v is the representative of part i, i.e., $\zeta_v^i = 1$ if v is the representative for part i and 0 otherwise. Since we want to model exactly one representative for every part, we use the constraint

$$\sum_{v \in V} \zeta_v^i = 1, \qquad i \in [\mathcal{K}]. \tag{3.33}$$

Since a node can only be a representative of part i, if it is a member of part i, we use the constraint

$$\zeta_v^i \le x_v^i, \qquad i \in [\mathcal{K}], v \in V. \tag{3.34}$$

For modeling the flow, we direct each edge in both possible directions by introducing the directed graph $D = (V, A)$, where $A := \{(v,w),(w,v) \mid \{v,w\} \in E\}$. We introduce the variable $f_{vw} \in \mathbb{R}_+$ on all arcs A from D directly corresponding to the edges in G. To allow only flow on edges that do not lie in the cut, we add the following constraint

$$f_{vw} + f_{wv} \leq M(1 - y_{vw}), \qquad \{v, w\} \in E, \tag{3.35}$$

where M is a sufficiently large constant, for example $M = |V| - \mathcal{K} + 1$. To model that every node, which is not a representative, has to have an outflow of at least 1, we use

$$\sum_{(v,w)\in A} f_{vw} - \sum_{(w,v)\in A} f_{wv} \geq 1 - M \sum_{i\in[\mathcal{K}]} \zeta_v^i, \qquad v \in V \tag{3.36}$$

with the same M as in (3.35). There also exist other possibilities for modeling connectedness using flow variables, examples using multi-commodity flows can be found, e.g., in [33, 46].

To model connectedness using only the x-variables, we use the separator inequalities introduced in Section 3.1

$$\sum_{u\in N} x_u^i \geq x_v^i + x_w^i - 1, \qquad v, w \in V, i \in [\mathcal{K}], N \text{ some } v\text{-}w\text{-seperator}. \tag{3.37}$$

These inequalities are dominated by using minimal separators, that is,

$$\sum_{u\in N} x_u^i \geq x_v^i + x_w^i - 1, \qquad v, w \in V, i \in [\mathcal{K}], N \in \mathcal{N}(v, w), \tag{3.38}$$

also see Section 3.1.

Thus, the polytope over which we want to optimize (3.28) is given by

$$\mathcal{S}_{\mathcal{K}}(G) := \operatorname{conv}(\{(x,y) \in \{0,1\}^{V\times[\mathcal{K}]} \times \{0,1\}^{V\times V} \mid \\ (x,y) \text{ fulfills (3.29) to (3.32)}, (3.38), (3.21)\}).$$

Since in the branch-and-bound procedure we often generate solutions in the polytope, where the integrality constraints are relaxed, we introduce the relaxed polytope, in which the vertices are not necessarily integer valued:

$$\mathcal{S}_{\mathcal{K}}^r(G) := \{(x,y) \in [0,1]^{V\times[\mathcal{K}]} \times [0,1]^{V\times V} \mid \\ (x,y) \text{ fulfills (3.29) to (3.32)}, (3.38), (3.21)\}.$$

The polytope resulting from using the flow formulation from above is defined as

$$\mathcal{S}_{\mathcal{K}}^f(G) := \operatorname{conv}(\{(x,y,f,\zeta) \in \{0,1\}^{V\times[\mathcal{K}]} \times \{0,1\}^{V\times V} \times \mathbb{R}_+^{E\times[2]} \times \{0,1\}^{V\times[\mathcal{K}]} \mid \\ (x,y,f,\zeta) \text{ fulfills (3.29) to (3.36)}, (3.21)\}).$$

It can be shown that the projection of $\mathcal{S}_{\mathcal{K}}^f$ to the space of the x- and y-variables is $\mathcal{S}_{\mathcal{K}}$ by using the max-flow min-cut theorem [39, 41] but the details are omitted.

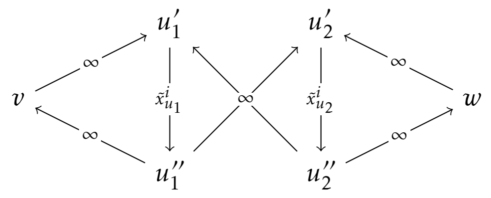


Figure 3.5: Directed graph D derived from the undirected graph $G = (\{v,u_1,u_2,w\},\{\{v,u_1\},\{u_1,u_2\},\{u_2,w\}\})$

Remark 3.60. Note that the y-variables are only needed to formulate the objective function and for the feasibility problem it suffices to inspect the polytope $\mathcal{S}^p_{\mathcal{K}}(G)$ resulting from a projection of $\mathcal{S}_{\mathcal{K}}(G)$ to the space of the x-variables (see also Section 4.2). Furthermore, Inequality (3.21) implies that, in general, $\mathcal{S}^p_{\mathcal{K}}(G) \neq \mathcal{P}^{=}_{\mathcal{K}}(G)$ (compare Sections 3.1 and 3.2). Within the solution process, we often find elements in $\mathcal{S}^r_{\mathcal{K}}(G)$ and since often the main interest are the x-variables, we introduce $\mathcal{S}^{r,x}_{\mathcal{K}}(G)$ as the projection of $\mathcal{S}^r_{\mathcal{K}}(G)$ to the space of the x-variables.

3.3.2 Separation of the Connectedness Constraint

The constraint (3.38) can be separated in polynomial time by using maximum flow calculations [40]. We include the description here for completeness. Let $\tilde{x}$ be a fractional solution for the x-variables. For each pair of distinct nodes v and w with $\{v,w\} \notin E$, $\tilde{x}^i_v + \tilde{x}^i_w - 1 > 0$ for $i \in [\mathcal{K}]$ construct a digraph $D = (W,A)$ as follows. For each node we introduce two copies u' and u'' forming the node set $W := \{u',u'' \mid u \in V \setminus \{v,w\}\} \cup \{v,w\}$, whereas the arcs are given by

$$\begin{aligned} A := \{&(v''_1,v'_2),(v''_2,v'_1) \mid \{v_1,v_2\} \in E, v_1,v_2 \notin \{v,w\}\} \\ &\cup \{(u'',v),(v,u') \mid \{u,v\} \in E\} \cup \{(u'',w),(w,u') \mid \{u,w\} \in E\} \\ &\cup \{(u',u'') \mid u \in V \setminus \{v,w\}\}, \end{aligned}$$

compare Figure 3.5 for a visualization. The capacity of arcs (u',u'') with $u \in V$ are set to be $\tilde{x}^i_u$ and all other capacities are set to ∞. The minimum cut between v and w (which can for example be found by a maximum flow calculation) only uses those arcs with a finite capacity. Since these arcs are the ones between the two copies of nodes of G, the minimum cut directly translates to the set of nodes which can be used for the violated connectivity constraint.

In [40] it is mentioned that computational experiments show that applying the above procedure is very time-consuming because for every pair there may be a maximum flow calculation. The authors proposed a linear time method for finding infeasible integer points, which is also applied here.

To explain the method, the following property of minimal separators, which first appears in [45] as an exercise, is crucial.

Lemma 3.61. *Let $G = (V,E)$ be a graph and N be a v-w-separator. Let C_v and C_w be the connected components of $G[V \setminus N]$ containing v and w, respectively. Then N is a minimal v-w-separator if and only if every node in N is adjacent to at least one node in C_v and at least one node in C_w.*

A proof relying on Lemma 3.13 can be found in [76].

Let $\tilde{x}$ be an integer solution and denote by $G^i_{\tilde{x}} = (V, E_{\tilde{x}})$ the graph induced by $\tilde{x}$, meaning $E_{\tilde{x}} = \{\{v,w\} \in E \mid \tilde{x}^i_v = \tilde{x}^i_w = 1\}$ for some $i \in [\mathcal{K}]$. If $\tilde{x}$ is infeasible, there exist two connected components C_v and C_w with $v \in C_v$ and $w \in C_w$ in $G^i_{\tilde{x}}$. Let $\Gamma(C_v)$ denote the neighboring nodes of the connected component C_v in G, that is, $\Gamma(C_v) := \{u \in V \setminus C_v \mid \exists \{u,k\} \in E, u \in C_v\}$. Obviously, $\Gamma(C_v)$ is a v-w-separator but not necessarily minimal w.r.t. cardinality. To derive a minimal separator, we use Algorithm 1. The general idea to find a minimal v-w-separator is to start with $\Gamma(C_v)$ and remove all nodes p, which are not adjacent to C_w, when Lemma 3.61 implies the minimality.

Algorithm 1 : Finding a minimal separator

```
  Input : Graph $G = (V,E)$, $i \in [\mathcal{K}]$, infeasible solution $\tilde{x}$, $C_v$, $C_w$ two
          connected components of $G^i_{\tilde{x}}$ with $v \in C_v$, $w \in C_w$
  Output : Minimal $v$-$w$-separator $N$
1 Find neighbors $\Gamma(C_v)$ of $C_v$ in $G$;
2 Delete all edges in $C_v$ and $\Gamma(C_v)$ from $G$;
3 Find the set $R \subseteq V$ of nodes reachable from $w$ in the modified graph $G$;
4 return $N = R \cap \Gamma(C_v)$
```

Note that Line 3 in Algorithm 1 can be performed in linear time by using a BFS in G with the additional constraint that nodes from $\Gamma(C_v)$ are never added to the queue. A proof of correctness can be found in [40].

Our experiments have shown that the determination of a minimal separator may not be the fastest way for solving Connected Max-$\mathcal{K}$-Cut. Instead, using $\Gamma(C_v)$ is faster most of the time. This was also observed in [5].

For calculating the neighborhoods $\Gamma(C_v)$ of connected components we use Algorithm 2. It is based on the disjoint-set data structure (first described in [42], see for example [28]) using the library boost [18]. Note that we do not need to calculate the neighborhood of nodes that do not belong to any part.

Given a fractional solution $\tilde{x} \in \mathcal{S}^{r,x}_{\mathcal{K}}$, the idea is to iterate once over the graph $G = (V,E)$ and find the connected components induced by nodes in the same part while additionally creating the neighborhood for those components. We say that node v belongs to part i if there exists $i \in [\mathcal{K}]$ such that $\tilde{x}^i_v > 0.5$ (see Line 4).

Let $F_i \subseteq V$ be the subset of nodes which belong to part i and let $G_i := G[F_i]$ be the subgraph induced by F_i. Algorithm 2 finds the connected components of all G_i and if v is a representative of connected component C_v, then Algorithm 2 also finds the neighborhood of C_v in G, denoted by $\Gamma(v)$.

For each node which belongs to a part we create a set using the disjoint-set data structure in Line 5. Then we iterate over all edges $\{v,w\}$ of G and distinguish the two cases whether v and w belong to the same part or not. If they do, we use the disjoint-set data structure to select one representative and create the union of the neighborhoods of v and w (see Lines 9 and 10).

Otherwise, if v and w do not belong to the same part, we differentiate further whether v and w belong to a part or not. If v belongs to a part, we add w to its neighborhood in Line 13 or we add v to the neighborhood of w if w belongs to a part in Line 16.

Algorithm 2 : Finding connected components and their neighborhoods

```
   Input : Graph $G = (V,E)$, fractional solution $\tilde{x}$
   Output : Connected components of $G_i$ for $i \in [\mathcal{K}]$ given by
            representatives $v$ and their neighborhoods $\Gamma(v)$ in $G$
 1 foreach node $v \in V$ do
 2     $\Gamma(v) := \emptyset$;
 3 end
 4 foreach node $v$ with $\tilde{x}_v^i > 0.5$ for an $i \in [\mathcal{K}]$ do
 5     MakeSet($v$);
 6 end
 7 foreach edge $\{v,w\} \in E$ do
 8     if $v$ and $w$ belong to the same part then
 9         Union($v,w$) and w.l.o.g. let $v$ be its representative;
10         $\Gamma(v) := \Gamma(v) \cup \Gamma(w)$;
11     else
12         if if $v$ belongs to a part then
13             $\Gamma(v) := \Gamma(v) \cup \{w\}$;
14         end
15         if if $w$ belongs to a part then
16             $\Gamma(w) := \Gamma(w) \cup \{v\}$;
17         end
18     end
19 end
```

The running time of Algorithm 2 is as follows. To decide for each node, whether it belongs to a part or not, we need time $\mathcal{O}(\mathcal{K}|V|)$. If theses parts are saved, we only need to iterate over the edges of G once in time $\mathcal{O}(|E|)$. The time complexity of the disjoint-set data structure is $\mathcal{O}(k \cdot \alpha(k,\ell))$, where k is the number of disjoint-set operations, ℓ is the number of elements and $\alpha(k,\ell)$ is the inverse Ackermann function (see [18, 28]). In our case, $k = |V| + |E|$ and $\ell = |V|$. Together this results in $\mathcal{O}(\mathcal{K}|V| + |E| + (|V| + |E|)\alpha(|V| + |E|, |V|))$. Since we can assume $\alpha(k,\ell) \leq 4$ for any practical purpose, the running time of Algorithm 2 is linear.

In theory, it is enough to find one minimal separator and add the corresponding inequality to the problem to cut off an infeasible point. But it might be a good idea to add more separator inequalities if the computational overload can be handled. Even for two connected components C_v and C_w, there are many different possible separator inequalities. Since in general the neighborhoods are not equal, the separator can either be $\Gamma(C_v)$ or $\Gamma(C_w)$. Furthermore, instead of v and w other nodes from C_v or C_w can be chosen for defining a separator inequality (3.37). If there exist three different connected components, the separator inequalities could also be created for every pair of connected components or for example only for ordered pairs.

In our case, Algorithm 2 needs to be called only once for calculating all connected components and their respective neighborhoods. Thus, the overload of creating additional separator inequalities is small. Our experiments have shown that it is best to add all violated separator inequalities for all connected components of all G_i. Furthermore, our experiments have shown that it is best to add the two described separator inequalities for every pair of connected components.

3.3.3 Propagation

Within the branch-and-bound procedure, we may be at a node in the branch-and-bound tree where some variables are fixed to a value and for others there may exist upper and/or lower bounds. This is called a *partial solution* in the following. A partial solution may be extended to a feasible solution for CONNECTED MAX-$\mathcal{K}$-CUT or it may be infeasible. The propagation algorithms works on a partial integer solution and can fix some entries of the solution to 0 and others to 1. The basic idea is that nodes cannot belong to part i if they cannot be reached from nodes that already belong to part i. Furthermore, if two nodes from part i can only be connected by using node $u \in V$, then node u has to be in part i as well. To explain the last part we need the following definition.

Definition 3.62 (Articulation node). Let $G = (V, E)$ be a graph. A node $v \in V$ is called an *articulation node* of G if $G[V \setminus \{v\}]$ has more connected components than G.

The propagation algorithm as shown in Algorithm 3 works as follows. First, for each part the nodes that are already fixed to this part are saved as sets F_i (Line 2). Then for every part the auxiliary graph G' is created by removing all nodes (and incident edges) that are fixed to another part (Lines 5 and 6). Afterwards, for a node q of F_i a BFS tree starting in q is calculated (Line 8). All nodes v in G that are not already fixed to a part that cannot be reached from q cannot be in part i because any path would have to use nodes that are already fixed to other parts (Line 10). Thus, $x_v^i = 0$ can be fixed. If node $u \notin F_i$ is an articulation node of G' such that $G'[\tilde{V} \setminus \{u\}]$ has at least two connected components which both contain nodes from F_i (Line 14), the node u has to belong to part i because otherwise the solution cannot be connected (Line 15). Note that after a fixing to i the values for the other parts can be set to 0, that is, $\tilde{x}_u^k = 0$ for all $k \neq i \in [\mathcal{K}]$, because of the partition equality (3.29).

The running time of Algorithm 3 is $\mathcal{O}(\mathcal{K}|V||E|)$. This can be shown as follows. The time is dominated by the for-loop for each part (Line 4). The auxiliary graph G' and the BFS tree can both be calculated in linear time $\mathcal{O}(|E|+|V|)$. The same holds for Line 10. All articulation nodes in Line 12 can be found in linear time $\mathcal{O}(|E|+|V|)$ by one BFS call and additional linear time checking [60]. The for-loop in Line 13 needs time $\mathcal{O}(|V|)$, whereas Line 14 can be implemented by BFS in linear time $\mathcal{O}(|E|+|V|)$.

Remark 3.63. As a last step inside the main for-loop the creation of separator inequalities (3.37) can be added. This can be achieved by first contracting all edges of G' if both endpoints are not yet fixed to some part, meaning all nodes $u \in V \setminus \bigcup_{j\in[\mathcal{K}]} F_j$. Let $G^\star$ be the resulting graph. In $G^\star$ the articulation nodes can be calculated (in linear time [60]) and one can proceed similar to Line 13: If an articulation node which is not fixed to part i separates at least two nodes v, w from part i, all the nodes corresponding to u before the contraction step define a separator for v and w and the inequality (3.37) can be added to our model. In theory this creates additional inequalities in linear time. In practice, however, the propagation usually is not the phase, in which additional inequalities are added and hence, this was not implemented. Furthermore, it is not clear, whether the contraction leads to an increased running-time and if the found separators are useful. For example the separator that is found may contain almost all variables, whereas the corresponding minimal separator contains only a few vertices.

3.3.4 Cuts

This section describes six additional valid cuts for $\mathcal{S}_{\mathcal{K}}(G)$. The first three are derived from the connectivity constraint, whereas the last three are known from the Max-$\mathcal{K}$-Cut literature. For a recent overview of cuts for the Max-$\mathcal{K}$-Cut problem see, e.g., [7].

Algorithm 3 : Propagation

```
   Input : Graph $G = (V,E)$, solution $\tilde{x}$, number of parts $\mathcal{K}$
 1 foreach $i \in [\mathcal{K}]$ do
 2     $F_i := \{v \in V \mid \tilde{x}_v^i$ fixed to 1$\}$;
 3 end
 4 foreach $i \in [\mathcal{K}]$ do
 5     $\tilde{V} := V \setminus \bigcup_{j \neq i \in [\mathcal{K}]} F_j$;
 6     $G' := G[\tilde{V}]$;
 7     $q \in F_i$;
 8     $T :=$ BFS tree of $G'$ starting in $q$;
 9     foreach node $u \in V$ which is not in $\bigcup_{j \in [\mathcal{K}]} F_j$ and not in $T$ do
10         $\tilde{x}_u^i := 0$;
11     end
12     calculate all articulation nodes $U_1$ of $G'$;
13     foreach $u \in U_1$ and $u \notin F_i$ do
14         if $u$ separates two nodes in $F_i$ then
15             $\tilde{x}_u^i := 1$;
16         end
17     end
18 end
```

Bounded Edge Cuts

For bounding the objective value we use the following constraint

$$\sum_{\{v,w\} \in E} y_{vw} \leq |E| - |V| + \mathcal{K}.$$

Validity can be seen as follows

$$\sum_{\{v,w\} \in E} y_{vw} \leq |E| - \sum_{i \in [\mathcal{K}]} (|V_i| - 1) = |E| + \mathcal{K} - \sum_{i \in [\mathcal{K}]} |V_i| = |E| + \mathcal{K} - |V|.$$

The first estimate is due to the fact that every partition with $|V_i|$ nodes has to contain at least $|V_i| - 1$ edges to be connected and the last equality is due to the fact that the V_i are a partition of V.

Articulation Cuts

Since any articulation node is in particular a separator, the *articulation cuts* are a special case of separator inequalities. Let v be an articulation node and let u and

w be two nodes from different connected components of $G[V \setminus \{v\}]$. Since v is a minimal separator for u and w, the following inequality is valid

$$x_u^i + x_w^i - x_v^i \leq 1, \qquad i \in [\mathcal{K}], \tag{3.39}$$

compare to (3.37). We experimented with adding all articulation cuts to our problem at the beginning of the algorithm. However, for many graphs from our test set, there were simply too many. Therefore, we tested a special articulation cut, which is explained in the next paragraph.

Leaf Cuts

In the special case, in which u is a leaf and u is adjacent to v, Inequality (3.39) is valid for every $w \in V \setminus \{u,v\}$ since we assume G to be connected. This is called a *leaf cut* in the following. As with the articulation cuts above, we also tested to add all possible leaf cuts at the beginning of the algorithm, but for many graphs from our test set, there were still too many possible cuts. This is why we tested separating these leaf cuts directly by initially calculating all leaves and then checking for every leaf and every partition if one of the $|V|-2$ separator inequalities is violated. This gives a $\mathcal{O}(\mathcal{K}|V|^2)$ separation routine. Of course, these inequalities are also separated by the calculation of minimal separators given in Algorithm 1. But the idea was to simplify the calculation and derive valid cuts in shorter time. See Section 3.3.8 for the numerical results.

Cycle Cuts

For the case $\mathcal{K} = 2$, we are in the special case of Max-Cut, for which many additional cuts are known. Because any cycle crosses a cut an even number of times, the so called odd cycle cuts are valid:

$$\sum_{\{v,w\}\in F} y_{vw} - \sum_{\{v,w\}\in C\setminus F} y_{vw} \leq |F|-1, \qquad F \subseteq C, |F| \text{ odd},$$

where C is a cycle in G [11]. These inequalities can be separated in polynomial time by a shortest path calculation in a auxiliary graph G', see [11]. Let $G' = (V', E')$ be derived from $G = (V,E)$ by having two nodes v' and v'' for each $v \in V$. Every edge $\{v,w\} \in E$ gives rise to two edges $\{v',w'\}$ and $\{v'',w''\}$ in E' with weights y_{vw} and two edges $\{v',w''\}$ and $\{v'',w'\}$ with weights $1-y_{vw}$. For every node $v \in V$ we calculate a shortest path from v' to v''. The minimum length of all these paths gives rise to the required cycle. The running time of this separation procedure is $\mathcal{O}(|V|(|V|\log|V|+|E|))$ by using Dijkstra's algorithm (see for example [28]) from the boost library [18], which needs $\mathcal{O}(|V|\log|V|+|E|)$ many calculations.

Triangle Cuts

For $\mathcal{K} = 2$ no cut can contain exactly one edge from any triangle. Therefore, the following triangle cuts are valid if the nodes u, v, w form a triangle, see [24]:

$$\begin{aligned} y_{vw} - y_{wu} - y_{uv} &\leq 0, \\ -y_{vw} + y_{wu} - y_{uv} &\leq 0, \\ -y_{vw} - y_{wu} + y_{uv} &\leq 0. \end{aligned}$$

We separate these cuts by first calculating all triangles in G and iterating over all of them to check if one of them is violated.

Clique Cuts

In [24] the so-called clique cuts were introduced:

$$\sum_{\substack{v,w \in V' \\ v \neq w}} y_{vw} \leq \binom{\mathcal{K}+1}{2} - 1, \qquad V' \subseteq V, |V'| = \mathcal{K}+1, G[V'] \text{ clique.}$$

The feasibility can be seen as for any subset of $\mathcal{K}+1$ nodes, at least two nodes have to be in the same part. This implies for a clique V' on $\mathcal{K}+1$ nodes that at least one of the possible $\binom{\mathcal{K}+1}{2}$ edges cannot be in the cut. We separate these cuts for $\mathcal{K} = 2$ by calculating all cliques on 3 nodes in the beginning and iterating over all of them to check if one of them is violated.

3.3.5 Primal Heuristics

In this section we describe the three heuristics we implemented. Note that it suffices to treat only the x-variables as mentioned in Remark 3.60. The first heuristic is a starting heuristic, which finds a feasible solution leading to hopefully good bounds. The second one creates an integer solution $x \in \mathcal{S}^p_{\mathcal{K}}$ from a relaxed solution $\hat{x} \in \mathcal{S}^{r,x}_{\mathcal{K}}$. Finally, the third heuristic tries to improve a given feasible integer solution $\tilde{x} \in \mathcal{S}^p_{\mathcal{K}}$.

Tree Heuristic

The main idea is to use a minimum spanning tree to ensure connectedness of the parts. Each edge in the spanning tree defines a cut in G into exactly two parts such that both resulting components are connected. This idea is applied repeatedly on edges maximizing the weight of the cut (and thereby the objective function). As the edges in the calculated tree tend to be inside parts and not in the cut, we start with a minimum spanning tree with respect to w. In more detail the heuristic can be seen in Algorithm 4.

The algorithm proceeds by assigning the root of the tree to the first part and the leaves to other parts. If the number of leaves is too small (Line 6), meaning that there are more different parts then leaves, the procedure generates parts containing exactly one node. These single node parts are chosen such that they are leaves in the minimum spanning tree with a maximal weight of the induced cut. After a leaf is assigned to a part, it is removed from the tree and the leaves are recalculated. This is carried out until the number of leaves is equal to the number of partitions left.

The other parts are assigned to contain the single leaf after sorting them decreasingly with respect to the weight of their induced cut. If there are leaves left, which are not assigned to a part, these leaves with their complete paths to the root node are assigned to the first part which contains the root node (Line 18).

The main loop starts in Line 21. For every part we inspect the path from the leaf to the root node until one node is reached, which already belongs to a part (at least the root node belongs to part 1). For every node along this path, the weight of the corresponding induced cut is calculated and the part in then chosen as the subpath, which maximizes the weight of the induced cut. As a last step, all nodes that do not belong to a part are put in the first part (Line 33).

As we use the boost library [18], the minimum spanning tree in Line 2 is be found in time $\mathcal{O}(|E|\log|V|)$ by using Kruskal's algorithm [79]. The weights in Line 4 can be found in time $\mathcal{O}(|V||E|)$. The loop in Line 8 can be implemented in time $\mathcal{O}(\mathcal{K}|V|)$. The time used for sorting in Line 16 is $\mathcal{O}(|V|\log|V|)$. The loop in Line 18 runs in $\mathcal{O}(\mathcal{K}|V|)$. The time needed for the main loop in Line 21 is $\mathcal{O}(\mathcal{K}|V|^2)$ because of the calculation of the weight in Line 27. Thus, the overall time needed for the tree heuristic is $\mathcal{O}(\mathcal{K}|V|^2 + |V||E|)$.

Algorithm 4 : Tree Heuristic

```
   Input : Graph $G = (V, E)$, number of parts $\mathcal{K}$
   Output : Integer solution $\tilde{x}$
1  $\tilde{x} := 0$;
2  compute a minimum spanning tree $T$ of $G$ w.r.t. $\omega$ with root node $r$;
3  let $L$ be the set of leaves in $T$;
4  calculate $\omega(\delta(\{v\}))$ for every $v \in V$;
5  $k := \mathcal{K}$;
6  if $|L| < \mathcal{K} - 1$ then
7      $j := \mathcal{K}$;
8      while $j > |L|$ do
9          let $\ell$ be the largest leaf in $L$ with respect to $\omega(\delta(\{\ell\}))$;
10         $V_j := \{\ell\}$;
11         remove $\ell$ and its incident edge from $T$; recompute leaves $L$ of $T$;
12         set $\tilde{x}_\ell^j := 1$ and $j := j - 1$;
13     end
14     $k := |L|$;
15 end
16 set $V_1 := \{r\}$ and sort $\ell_i \in L$ decreasing w.r.t. $\omega(\delta(\{\ell_i\}))$;
17 for $i = 1, \dots, k$ do  $V_{i+1} := \{\ell_i\}$ ;
18 for $i = k + 1, \dots, |L|$ do
19     let $p$ be the path from $r$ to $\ell_i$ in $T$ and set $V_1 := V_1 \cup V(p)$;
20 end
21 for $i = 2, \dots, k$ do
22     $\omega_1 := \omega(\delta(V_i))$, $S_1 := V_i$;
23     let $p$ be the path from $V_i$ to $r$ ($|V_i| = 1$) in $T$ where $p_j$ denotes element $j$;
24     $j := 2$ ($p_1 = V_i$);
25     while $p_j \neq \{r\}$ and $p_j$ does not already belong to any part do
26         $S_j := S_{j-1} \cup \{p_j\}$;
27         $\omega_j := \omega(\delta(S_j))$;
28         $j := j + 1$;
29     end
30     set $j' := \arg\max\{\omega_j\}$ and $V_i := S_{j'}$;
31     foreach node $v$ in $V_i$ do  $\tilde{x}_v^i := 1$ ;
32 end
33 foreach $v \in V_1$ or $v$ not in any $V_i, i > 1$ do $\tilde{x}_v^1 := 1$;
34 return $\tilde{x}$
```

Rounding Heuristic

This heuristic originated from [46]. The main idea is to round a relaxed solution $\hat{x}$ from $\mathcal{S}_{\mathcal{K}}^{r,x}(G)$ to be integer valued in accordance to the entries in $\hat{x}$. The pseudocode can be seen in Algorithm 5.

In the following we assume that the relaxed solution $\hat{x}$ is given in the form of a vector. Because this vector is sorted (Line 2), we use the functions node(x) and part(x) to denote the node and the part of the specified entry respectively. For example if $\hat{x}_u^i$ is one entry of $\hat{x}$ which is in position ℓ after sorting, then node$(\ell) = u$ and part$(\ell) = i$. The value ℓ can be seen as the probability that node node(ℓ) belongs to part part(ℓ).

We start by assigning the nodes with the highest probability to the corresponding part (Line 6) until every part contains exactly one node. The set M serves as the set of already assigned nodes and serves both as a stopping criterion (Line 14) and for easier checks of assignment (Line 16). Then we iterate over the sorted vector and assign nodes to parts if they do not violate the connectedness constraint (Line 15). This process is repeated until all nodes are assigned to a part (Line 14). This while-loop is needed because otherwise nodes may exist that are not assigned to some part after the first iteration over the sorted vector $\hat{x}$.

Note that the check for connectedness in Line 17 can be realized by checking if node(ℓ) is in the neighborhood of $V_{\text{part}(\ell)}$. The time complexity is thus $\mathcal{O}(|E|)$. As sorting in Line 2 can be implemented in $\mathcal{O}(\mathcal{K}|V|\log(\mathcal{K}|V|)) = \mathcal{O}(\mathcal{K}|V|\log(|V|))$, the complexity is dominated by the while-loop in Line 14. Because of connectedness, the overall complexity is therefore in $\mathcal{O}(\mathcal{K}|V|^2|E|)$.

Algorithm 5 : Rounding heuristic

```
Input : Graph $G = (V, E)$, number of parts $\mathcal{K}$, relaxation solution $\hat{x}$
Output : Integer solution $\tilde{x}$
$\tilde{x} := 0$;
sort $\hat{x}$ in descending order;
$V_i := \emptyset$ for all $i \in [\mathcal{K}]$;
$\ell := 1$;
$M := \emptyset$;
while there exists $i \in [\mathcal{K}]$ with $V_i = \emptyset$ do
    if $V_{\text{part}(\ell)} = \emptyset$ then
        $V_{\text{part}(\ell)} := \{\text{node}(\ell)\}$;
        $M := M \cup \{\text{node}(\ell)\}$;
        $\tilde{x}^{\text{part}(\ell)}_{\text{node}(\ell)} := 1$;
    end
    $\ell := \ell + 1$;
end
while $M \neq V$ do
    for $\ell = 1, \ldots, [\mathcal{K}|V|]$ do
        if $\{\text{node}(\ell)\} \notin M$ then
            if $G[V_{\text{part}(\ell)} \cup \{\text{node}(\ell)\}]$ is connected then
                $V_{\text{part}(\ell)} := V_{\text{part}(\ell)} \cup \{\text{node}(\ell)\}$;
                $M := M \cup \{\text{node}(\ell)\}$;
                $\tilde{x}^{\text{part}(\ell)}_{\text{node}(\ell)} := 1$;
            end
        end
    end
end
return $\tilde{x}$
```

Improving Heuristic

The main idea of this heuristic is to locally improve a given integer solution by moving a node from one part to another. This change of part is only performed if both involved parts are connected. For example if node v is in part V_i and should be moved to part j, both parts $V_i \setminus \{v\}$ and $V_j \cup \{v\}$ have to be connected. Node v is only moved if the objective function value increases. The details of this heuristic can be found in Algorithm 6.

Algorithm 6 : Improving Heuristic

```
   Input : Graph $G = (V, E)$, number of parts $\mathcal{K}$, feasible integer solution $x'$
   Output : Integer solution $\tilde{x}$ with at least the same objective value
1  $\tilde{x} := x'$;
2  foreach $v \in V$ do
3      let $i \in [\mathcal{K}]$ be the part of $v$, that is, $\tilde{x}_v^i = 1$;
4      if $\exists j \in [\mathcal{K}], j \neq i$ such that both $G[V_i \setminus \{v\}]$ and $G[V_j \cup \{v\}]$ are connected,
        and $\emptyset \neq V_i \setminus \{v\}$ then
5          if objective value increases by moving $v$ from $V_i$ to $V_j$ then
6              $\tilde{x}_v^i := 0$;
7              $\tilde{x}_v^j := 1$;
8              return $\tilde{x}$
9          end
10     end
11 end
12 return $\tilde{x}$
```

This heuristic runs in $\mathcal{O}(|V|\mathcal{K}|E|^2)$ since we have to check for each node first if $V_i \setminus \{v\}$ is connected in time $\mathcal{O}(|E|)$. Then we have to check each other of the $\mathcal{K} - 1$ parts j if $V_j \cup \{v\}$ is connected also in time $\mathcal{O}(|E|)$.

3.3.6 Branching Rules

Let x be a fractional solution and let F_i denote the set of nodes that are fixed to be in part i, i.e., $F_i = \{v \in V \mid x_v^i = 1\}$ as introduced in Section 3.3.3. Furthermore, let F_i^0 denote the set of nodes v which are fixed to a value of 0 for part i, that is, for every $v \in F_i^0$ it holds that $x_v^i = 0$. Both may occur within the solution process for example as a result of the propagation algorithm. Let $G_i := G[F_i]$ be the graph induced by the nodes fixed to part i and $G_i^0 := G[V \setminus \bigcup_{j \neq i \in [\mathcal{K}]} F_j]$ be the graph induced on the nodes fixed to part i as well as those which are not yet fixed to any part.

Articulation Branching

Let v be an articulation node of $G = (V,E)$. By definition, $G[V \setminus \{v\}]$ contains connected components $C_1,\dots,C_k$ with $k > 1$. If node v is defined to be in part i and both $u \in C_\ell$ and $w \in C_q$ belong to the same part, they have to be in part i. This also implies that parts other than i can contain nodes from at most one connected component C_ℓ. Therefore, the number of possibilities for parting the nodes drastically reduces once an articulation node is fixed to a part. The idea of articulation branching therefore is to first branch on articulation nodes to a part if possible.

All articulation nodes are calculated in the beginning (in linear time [60]) and are stored. The branching rule also stores the last processed node and proceeds with the next articulation node in order to allow for different fixings. For each articulation node v it is checked if v is already fixed to some part, meaning $v \in F_j$ for some $j \in [\mathcal{K}]$, in which case it is ignored. If v is not fixed to any part yet, the children of the current branch are then generated by fixing the articulation node v to every possible part j if $v \notin F_j^0$ holds.

Note that the time needed for this branching rule is dominated by the calculation of the branching nodes, which is only needed once. Otherwise, $\mathcal{O}(|V| + \mathcal{K})$ calculations are needed.

Infeasibility Branching

The idea of the infeasibility branching rule is to create infeasible children. Since infeasibility of integer solutions can be checked by fast algorithms (see, e.g., Section 3.3.3), hopefully infeasible decisions are detected early to rule out many branching decisions that would otherwise occur later within the solution process.

To this end, let v be an unassigned node $v \notin F_i$ for all $i \in [\mathcal{K}]$ with the least number of unassigned neighbors. If there is more than one such node v, we let v be the node with the least number of possible part assignments, because there may exists parts $i \in [\mathcal{K}]$ such that $v \in F_i^0$. The idea is to create as few new branches as possible since the hope is that they are infeasible. If there is still more than one possible node v, we choose v to have the maximal degree with the idea of increasing the likelihood of infeasible branches. The children for the branch-and-bound tree are then generated by fixing v to every possible part.

Note that for every node v, we need time $\mathcal{O}(\mathcal{K})$ to find the number of unassigned parts. If these numbers are calculated once, the running time of the branching rule is $\mathcal{O}(|V|\Delta)$, where Δ denotes the maximal degree of a node in G. This results in an overall running time of $\mathcal{O}(|V|(\Delta + \mathcal{K}))$.

Objective Branching

To incorporate the objective function into the branching decision, the idea of objective branching is to find node v with the most assigned neighbors. Then fixing v to one part may lead to many edges in the induced cut, thereby increasing the objective function.

We decide for each node v if it is fixed or branched. Then, we choose v to be the node with the most assigned neighbors and create the children for the branch-and-bound tree by fixing v to every possible part. The running time of this procedure is $\mathcal{O}(|V|(\Delta + \mathcal{K}))$.

Note that this procedure may also create infeasible solutions, since fixing a node to a part only a small amount of its neighbors are fixed to, is likely to be infeasible. Then the arguments from the infeasibility branching rule can be applied.

Path Branching

The idea of this branching rule is to enforce connectedness by finding paths between different connected components. In contrast to the previous rules, we do not fix a node to a part but rather create two child nodes by adding one additional constraint.

First, we need to find a disconnected part. Therefore, let $i \in [\mathcal{K}]$ such that the graph G_i induced by the nodes fixed to part i is disconnected. If no such i exists, this branching rule is terminated and another one is called. If the partial solution can be extended to a feasible solution, then there exists a path $P = (V_P, E_P)$ connecting two different connected components of G_i such that P contains only nodes from G_i^0, which is the graph induced by nodes fixed to part i as well as the nodes not yet fixed to any part. One child is generated by fixing all nodes in V_P to part i adding the constraint

$$\sum_{v \in V_P} x_v^i \geq |V_P|.$$

The other child is created by adding the constraint that at least one of the nodes in P is not fixed to part i

$$\sum_{v \in V_P'} x_v^i \leq \left|V_P'\right| - 1,$$

where $V_P' := \{v \in V_P \mid v \notin F_i\}$.

Note that both, the connected components of G_i and the path P, can be found by (modified) BFS algorithms. This implies that the running time of the path branching rule in $\mathcal{O}(\mathcal{K}|E|)$.

3.3.7 Symmetry

For any solution of Connected Max-$\mathcal{K}$-Cut, exchanging part labels creates a different solution with the same objective value. More formally, if σ is a permutation of $[\mathcal{K}]$, for any solution x the point $\bar{x}$ defined by $\bar{x}^i_v = x^{\sigma(i)}_v$ is also a solution derived by a so-called *partitioning symmetry*. Furthermore, if there exists a graph automorphism $\tilde{\sigma}\colon V \to V$ of $G = (V, E)$, then $\tilde{x}$ defined by $\tilde{x}^i_v = x^i_{\tilde{\sigma}(v)}$ is also feasible with the same objective value as x. This symmetry is called a *graph symmetry* in the following.

As symmetric solutions often lead to an increased running-time of a branch-and-bound algorithm due to the fact that similar branches have to be calculated, there exist many procedures to handle such symmetries. An overview can be found for example in [88]. A computational study comparing different techniques for mixed-integer programs is [93]. Next, we want to describe how we handle the above two types of symmetries by applying procedures from the literature.

Partitioning Symmetries

Every point x in $\mathcal{S}^p_{\mathcal{K}}$ in particular is in $\{0,1\}^{V\times[\mathcal{K}]}$, meaning that it can be written as a $|V| \times \mathcal{K}$ matrix. The permutation σ acting on the part labels can thus be seen as acting on the columns of x. For handling these column symmetries, [72] introduced the concept of *orbitopes*. Since every permutation of the columns generates a symmetric solution, only one of these permutations needs to be inspected in the branch-and-bound procedure. Thus, the main idea of orbitopes is to chose one solution, where the columns are sorted such that they are lexicographically maximal in their symmetry class. The so-called *full orbitope* $O_{m,n}$ is defined as the convex hull of all binary $m \times n$ matrices, whose columns are sorted lexicographically non-increasing. This implies that inequalities derived from the orbitope can be used for our mixed-integer formulation to decrease the number of possible solutions.

Since due to the partitioning constraint (3.29), every solution x contains exactly one 1-entry in every row, we can even restrict $O_{m,n}$ to the *partitioning orbitope*. The partitioning orbitope is defined as the convex hull of vertices of $O_{m,n}$ with exactly one 1-entry in every row. In [72] the authors state a complete description of the partitioning orbitope and prove that it can be separated in time $\mathcal{O}(|V|\mathcal{K})$. Furthermore, in [71] it is shown that propagation can be achieved also in time $\mathcal{O}(|V|\mathcal{K})$. To handle partitioning symmetries for the Connected Max-$\mathcal{K}$-Cut problem we use these propagation and separation routines.

Graph Symmetries

The same idea of lexicographically maximal elements from above can be applied in the case of graph automorphisms. The general idea can be formulated by the so-called symresacks. For a permutation s of the symmetric group on n elements, the *symresack* P_s is defined as

$$P_s := \operatorname{conv}(\{x \in \{0,1\}^n \mid \sum_{i=1}^{n} (2^{n-s(i)} - 2^{n-i}) x_i \leq 0\}), \tag{3.40}$$

see [58]. It can be shown that P_s contains exactly the binary vectors which are lexicographically not smaller than their permutation with respect to s. Thus, inequalities derived from symresacks can also be applied in the case that s is a graph automorphism. Note that a graph automorphism only acts on the rows of solution matrices x_v^i and n therefore is not equal to $|V|\mathcal{K}$. For brevity of presentation the details are omitted. Due to the large coefficients in (3.40), adding the corresponding inequalities may lead to numerical problems.

In [58] it is shown that symresacks can be described by an IP formulation with left-hand side coefficients in $\{0, \pm 1\}$, which can be separated in time $\mathcal{O}(n\alpha(n,n))$, where α is the inverse Ackermann function. This in particular means, that the corresponding inequalities do not introduce the above mentioned numerical instabilities. Moreover, [58] formulates a linear algorithm for solving the propagation over P_s. Both, separation and propagation, are implemented for solving the Connected Max-$\mathcal{K}$-Cut problem.

To find graph automorphisms we use the program nauty [91].

3.3.8 Numerical Results

Each of the presented techniques was implemented as an extension to SCIP and we compare in this section their influence on the solution process. We tested various different settings on three different test sets.

The experiments on all test sets are using SCIP 4.0.1 [87] as branch-and-bound framework and CPLEX 12.7.1 [62] as LP-solver. The tests were run on a Linux cluster with Intel Xeon E5 3.5 GHz quad core processors and 32 GB memory; the code was run using one thread and running a single process at a time. The time limit was set to 3600 s per instance.

Our test instances consist of three different sets: Color02, I080 and Random. The first contains a subset of the instances found in [26] which constitute a benchmark for graph coloring problems. We removed the largest instances since there was no hope to solve them within our time frame. The second set is a part of the SteinLib library, called I080 and can be found in [104]. These are a part of a benchmark set for solving Steiner tree problems. The last set of instances were

Table 3.1: The three test sets and their properties: minimal, maximal and average number of nodes and edges, as well as average value of density and number of articulation nodes

	Nodes			Edges			Density	Art. Nodes
Name	min.	max.	avg.	min.	max.	avg.	avg.	avg.
Color02	11	2368	167.5	20	110871	4441.27	0.2255	0.235
I080	80	80	80.0	120	3160	884.40	0.2799	3.920
Random	50	100	84.0	107	1016	486.57	0.1402	0.647

randomly generated with the idea of creating sparse connected graphs. Sparse instances are of particular interest to us since connectivity is in this case a very restrictive constraint and the idea is that our techniques are especially beneficial then. An overview of the three test sets with details of some properties can be found in Table 3.1, where the density of a graph $G = (V, E)$ is calculated by

$$\frac{2|E|}{|V|(|V|-1)}.$$

Preliminary test have shown that out of the four presented branching rules only the articulation branching may lead to a faster solution process, which is why we do not include results of the others. Furthermore, we only include the clique and bounded edge cuts in the following analysis since articulation, leaf and cycle cuts did in most cases not result in an increased performance of the algorithm. We tested both the flow formulation and the separator-based formulation on the three test sets shown in Table 3.1. The results using different parameter settings can be found in Table 3.2. In all cases we used both symmetry breaking techniques explained in Section 3.3.7 since this decreased the running time in most cases. For a comparison of the different symmetry handling techniques and their influence see Appendix A.

The names of the tested settings in Table 3.2 are explained as follows. The letter "f" indicates that the flow formulation is used and we optimize over the polyhedron $\mathcal{S}^f_{\mathcal{K}}$, whereas "nf" indicates the separator-based formulation given by $\mathcal{S}_{\mathcal{K}}$. In case of the flow-formulation "cut 0" means that no additional cuts are used, in contrast to "cut 1", where clique cuts are applied and separator inequalities are separated. In case of the separator formulation, "cut 1" indicates the additional use of clique cuts over the setting "cut 0". Furthermore, we always include the bounded edge cuts independent of the setting. In both formulations "branch 0" indicates that none of our branching rules is applied, whereas "branch 1" implies the use of the articulation node branching rule. Lastly, "heur 0" means that none of the presented heuristics are applied, while "heur 1" indicates the use of all three heuristics: tree heuristic, rounding heuristic and improving heuristic.

The columns of the table contain the name of the setting as explained above, the number of applied separator cuts, the time needed for finding these separator cuts, the number of domain reductions resulting from the propagation routine, the time needed for the propagation, the number of optimally solved instances and the time needed for the solution. With the exception of the number of optimally solved instances, the numbers are all calculated using the shifted geometric mean, which is defined as

$$(\Pi(a_i + s))^{1/n} - s$$

for the n values $a_1, \dots, a_n$ and the shift s. We use a shift of 10 for the times values as well as the number of domain reductions and a shift of 1000 for the number of separator cuts.

First, it can be seen that the time needed for the propagation routine is negligible. While separating the separator inequalities is more time-consuming, it uses only a small percentage of the total time. Second, using cuts is favorable in all cases. The influence of the other techniques is not as notably. For the Color02 test set heuristics are in general increasing the running time with the sole exception of the flow formulation with additional cuts. For the other test sets heuristics do not have a clear influence on the overall performance. In some setting, it is preferable to use heuristics, in others it is the opposite and in some case the influence of the heuristics is negligible.

Since the articulation branching rule depends on the number of articulation nodes in the graph, this setting has virtually no influence on the Color02 test set because there are only two instances, which contain an articulation node. In case of the I080 test set, the branching rule has more influence since the average number of articulation nodes is much higher even though the graphs are on average denser compared to the other two test sets (see Table 3.1). Still, there is no clear indication on when it is best to use our special branching rule. Even though the density for the Random test set is smaller than the average density of the other test sets, the number of articulation nodes is less than the number of articulation nodes in the I080 test set. That is one possible explanation on why the branching rule may not have a big influence on the overall running time. Only if additional cuts are applied in the separator-based formulation is it clearly beneficial to use articulation branching.

A more detailed analysis is planned to appear in [56].

Table 3.2: Comparison of different parameter settings for solving the Connected Max-$\mathcal{K}$-Cut problem

Setting	#SepaCuts	SepaTime	#DomRed	PropTime	#Opt	Time
`Color02`:						
f cut 0 branch 0 heur 0	0.0	0.0	0.0	0.00	7	2130.45
f cut 0 branch 0 heur 1	0.0	0.0	0.0	0.00	7	1939.66
f cut 0 branch 1 heur 0	0.0	0.0	0.0	0.00	7	2130.90
f cut 0 branch 1 heur 1	0.0	0.0	0.0	0.00	7	1940.32
f cut 1 branch 0 heur 0	10131.4	11.8	22.1	1.64	13	1267.08
f cut 1 branch 0 heur 1	8092.9	10.5	17.2	0.37	15	1094.92
f cut 1 branch 1 heur 0	9881.4	11.7	22.5	1.65	13	1243.33
f cut 1 branch 1 heur 1	8124.5	10.5	17.0	0.37	15	1100.98
nf cut 0 branch 0 heur 0	21296.3	23.0	21.1	1.58	19	820.60
nf cut 0 branch 0 heur 1	22459.7	21.9	21.9	0.52	20	845.86
nf cut 0 branch 1 heur 0	21191.8	22.9	21.4	1.57	19	812.90
nf cut 0 branch 1 heur 1	22751.3	21.7	22.1	0.53	20	837.61
nf cut 1 branch 0 heur 0	20100.2	15.9	22.0	1.47	20	751.17
nf cut 1 branch 0 heur 1	21771.0	16.8	21.0	0.65	20	800.75
nf cut 1 branch 1 heur 0	19802.8	15.8	21.2	1.46	20	741.76
nf cut 1 branch 1 heur 1	21417.3	16.9	20.2	0.64	20	799.23
`I080`:						
f cut 0 branch 0 heur 0	0.0	0.0	0.0	0.00	24	1402.00
f cut 0 branch 0 heur 1	0.0	0.0	0.0	0.00	23	1510.28
f cut 0 branch 1 heur 0	0.0	0.0	0.0	0.00	25	1466.49
f cut 0 branch 1 heur 1	0.0	0.0	0.0	0.00	26	1379.09
f cut 1 branch 0 heur 0	7363.0	4.9	50.5	0.21	89	122.51
f cut 1 branch 0 heur 1	5505.8	3.7	43.8	0.13	95	99.82
f cut 1 branch 1 heur 0	5492.4	3.9	35.7	0.19	92	100.99
f cut 1 branch 1 heur 1	5468.1	3.9	39.5	0.15	92	99.10
nf cut 0 branch 0 heur 0	20577.4	11.5	57.4	0.35	75	290.66
nf cut 0 branch 0 heur 1	19025.7	12.6	55.0	0.37	71	293.39
nf cut 0 branch 1 heur 0	15592.9	9.8	51.7	0.39	71	231.35
nf cut 0 branch 1 heur 1	15968.8	9.6	51.5	0.28	75	238.62
nf cut 1 branch 0 heur 0	17292.1	7.0	66.7	0.64	92	88.46
nf cut 1 branch 0 heur 1	12994.2	4.5	61.7	0.34	92	66.64
nf cut 1 branch 1 heur 0	10429.4	4.1	47.5	0.41	93	61.35
nf cut 1 branch 1 heur 1	9960.4	3.7	48.2	0.30	93	56.27
`Random`:						
f cut 0 branch 0 heur 0	0.0	0.0	0.0	0.00	13	3325.69
f cut 0 branch 0 heur 1	0.0	0.0	0.0	0.00	9	3363.59
f cut 0 branch 1 heur 0	0.0	0.0	0.0	0.00	13	3342.07
f cut 0 branch 1 heur 1	0.0	0.0	0.0	0.00	9	3441.01
f cut 1 branch 0 heur 0	2976.5	3.3	4.9	0.04	134	312.48
f cut 1 branch 0 heur 1	3001.5	3.4	5.4	0.01	132	319.86
f cut 1 branch 1 heur 0	3223.3	3.7	5.1	0.04	132	328.40
f cut 1 branch 1 heur 1	2859.2	3.2	5.0	0.01	134	305.63
nf cut 0 branch 0 heur 0	11451.7	21.9	7.1	0.06	62	1389.09
nf cut 0 branch 0 heur 1	12499.0	22.4	8.0	0.07	61	1424.69
nf cut 0 branch 1 heur 0	9447.7	19.6	5.5	0.05	63	1222.95
nf cut 0 branch 1 heur 1	10305.8	20.3	5.8	0.02	65	1246.45
nf cut 1 branch 0 heur 0	5165.3	3.9	6.8	0.06	141	157.51
nf cut 1 branch 0 heur 1	4912.0	4.1	7.2	0.07	139	154.00
nf cut 1 branch 1 heur 0	4344.8	2.9	5.2	0.05	144	138.05
nf cut 1 branch 1 heur 1	4669.6	3.4	6.1	0.03	143	145.83

Chapter 4

MIP Formulations

There exist integer programming formulations for problems where a subset of the variables is only needed to formulate the objective function. To answer the feasibility problem, it thus suffices to examine a lower-dimensional polytope. The question then arises whether it is possible to find a formulation over the lower-dimensional polytope such that there exists a linear objective function which corresponds to the original objective. We examine the question whether an optimization problem can be solved by solving a mixed-integer program and find answers for specific cases. Furthermore, we propose a definition of mixed-integer programming formulations including objective functions and discuss some of their properties. Section 4.1 serves as an introduction, whereas Section 4.2 provides the main part. This chapter is based on a joint work with Hojny and Pfetsch [57].

4.1 Introduction

A subproblem of Problem 2.1 is to partition the node set of a given graph $G = (V, E)$. This problem is known as the Graph Partitioning problem where the objective function is to maximize the weight of edges between different partitions.

Problem 4.1. *Given a graph $G = (V, E)$ with weight function $\omega\colon E \to \mathbb{R}$ and an integer $\mathcal{K} \in \mathbb{N}$, the* Graph Partitioning *problem is to partition the nodes V of G into $\mathcal{K}$ subsets such that the sum of the weights of edges between different partitions is maximized.*

Note that this is similar to Problem 3.58, where additional connectedness is the task. On the other hand, this can be formulated analogously as minimizing the edge weights that completely lie in one partition.

One way of formulating this problem as a mixed-integer program (MIP) is the following (see, e.g., [24]):

$$
\begin{aligned}
\max \quad & \sum_{\{v,w\}\in E} \omega(\{v,w\}) y_{vw} && && \text{(4.1a)}\\
\text{s. t.} \quad & \sum_{i\in[\mathcal{K}]} x_v^i = 1, && v \in V, && \text{(4.1b)}\\
& x_w^i - x_v^i \le y_{vw}, && \{v,w\}\in E,\ i\in[\mathcal{K}], && \text{(4.1c)}\\
& x_v^i - x_w^i \le y_{vw}, && \{v,w\}\in E,\ i\in[\mathcal{K}], && \text{(4.1d)}\\
& x_w^i + x_v^i + y_{vw} \le 2, && \{v,w\}\in E,\ i\in[\mathcal{K}], && \text{(4.1e)}\\
& x_v^i \in \{0,1\}, && v\in V,\ i\in[\mathcal{K}], && \text{(4.1f)}\\
& y_{vw} \in \{0,1\}, && \{v,w\}\in E. && \text{(4.1g)}
\end{aligned}
$$

In this case, the binary variable x_v^i encodes whether node v is in partition i and the binary variable y_{vw} models whether the edge $\{v,w\}$ connects two nodes from different partitions (compare also to the formulation of Connected Max-$\mathcal{K}$-Cut in Section 3.3.1).

Equation (4.1b) is the partitioning constraint already introduced in (3.29). Inequalities (4.1c) and (4.1d) model that, if the two nodes v and w are in different partitions, the variable y_{vw} has to be 1. Inequality (4.1e) models that, if the two nodes v and w are in the same partition, the variable y_{vw} cannot be 1. Finally, (4.1f) and (4.1g) ensure the binarity of the variables. Other formulations can for example be found in [2], but the details are not of interest here.

Given the x-variables, it is easy to compute the y-variables. But if the roles are reversed, calculating the x-variables is NP-hard.

Lemma 4.2. *Given a graph $G = (V,E)$ and a binary vector $y \in \{0,1\}^E$. It is* NP-*hard to find the minimum number $\mathcal{K} \in \mathbb{N}$ such that y corresponds to the cut induced by a partitioning of V into $\mathcal{K}$ components.*

Proof. Let $y_{vw} = 1$ for all $\{v,w\} \in E$. Since this implies that each edge has to be in the induced cut, the endpoints of every edge have to be in different components of the partition. Thus, solving the problem of finding the minimum number of partitions needed to ensure that all edges are in the cut, is equivalent to solving the Chromatic Number problem: find the minimum number of colors needed to color the nodes of a graph such that no edge has two endpoints of the same color. Chromatic Number is NP-complete (see, e.g., [44, Problem GT4]). □

The following formulation models the Graph Partitioning problem by only using the x-variables. Note that in this case the objective function is quadratic.

$$\begin{aligned}
\max \quad & \sum_{\{v,w\}\in E} \omega(\{v,w\}) \left(1 - \sum_{i\in[\mathcal{K}]} x_v^i x_w^i \right) & & \\
\text{s.t.} \quad & \sum_{i\in[\mathcal{K}]} x_v^i = 1, & & v \in V, \\
& x_v^i \in \{0,1\}, & & v \in V,\ i \in [\mathcal{K}].
\end{aligned} \tag{4.2}$$

We refer to this problem as the Quadratic Graph Partitioning problem. The question that arises is how to reformulate this problem to use a linear objective function. One possibility is linearization, which introduces additional variables and results in Formulation (4.1). But is there any way to model the problem on the x-variables alone with a linear objective function? This serves as a guiding question for the next section. Compare also to Remark 3.60, where this question is mentioned concerning the polytope $\mathcal{S}_{\mathcal{K}}(G)$.

4.2 Problem Formulation

In order to generalize the question of representing an optimization problem as a MIP, let $X \subseteq \mathbb{Z}^p \times \mathbb{R}^q$ be non-empty and *hole-free*, that is, $X = \operatorname{conv}(X) \cap (\mathbb{Z}^p \times \mathbb{R}^q)$. Note that hole-free is a necessary condition for finding a mixed-integer formulation for X in the original space. Furthermore, let $f : X \to \mathbb{R}$ be an objective function with a finite maximal value. The question from the previous section then generalizes to the following problem.

Problem 4.3. *Given an optimization problem* Π *of the following form*

$$\max\{f(x) \mid x \in X\}, \tag{4.3}$$

can Π *be solved by a MIP?*

To formalize Problem 4.3, some careful definitions are needed. First, it has to be clarified what a MIP is. One common definition is the following.

Definition 4.4. A *mixed-integer program (MIP)* is an optimization problem

$$\max\{g(x) \mid x \in Q \cap (\mathbb{Z}^p \times \mathbb{R}^q)\},$$

specified by a polyhedron $Q \subseteq \mathbb{R}^p \times \mathbb{R}^q$ and an affine function $g \colon \mathbb{R}^p \times \mathbb{R}^q \to \mathbb{R}$. We use the tuple (Q, g) to denote a MIP. Furthermore, a MIP is called *feasible* if $Q \cap (\mathbb{Z}^p \times \mathbb{R}^q) \neq \emptyset$.

Since the set X in Problem 4.3 is not specified other than being hole-free and Definition 4.4 contains a polyhedron Q, we have to extend the above definition to also include X.

Definition 4.5. A *mixed-integer formulation* of a set $X \subseteq \mathbb{Z}^p \times \mathbb{R}^q$ is a pair (Q, π) with a polyhedron $Q = \{(y,z) \in \mathbb{R}^{p'} \times \mathbb{R}^{q'} \mid Ay + Bz \leq b\}$ and an affine function $\pi \colon \mathbb{R}^{p'} \times \mathbb{R}^{q'} \to \mathbb{R}^p \times \mathbb{R}^q$ such that

$$X = \pi\Big(Q \cap (\mathbb{Z}^{p'} \times \mathbb{R}^{q'})\Big),$$

where $A \in \mathbb{R}^{m \times p'}$, $B \in \mathbb{R}^{m \times q'}$, and $b \in \mathbb{R}^m$. If π is the identity, the formulation is *in the original space*, otherwise *in an extended space* (or an *extended formulation* for short). Furthermore, if $q' = 0$, we call the the formulation an *integer formulation*.

Since Definition 4.5 does not include the objective function, we propose the following definition.

Definition 4.6. A *mixed-integer programming formulation (MIP formulation)* for an instance I of Π is a triple (Q, g, τ) with a polyhedron $Q \subseteq \mathbb{R}^{p'} \times \mathbb{R}^{q'}$, an affine function $g \colon \mathbb{R}^{p'} \times \mathbb{R}^{q'} \to \mathbb{R}$ and a function $\tau \colon \mathbb{R}^{p'} \times \mathbb{R}^{q'} \to \mathbb{R}^p \times \mathbb{R}^q$ such that

1. (Q, g) is a feasible MIP with finite maximum value,
2. $\tau(\arg\max\{g(y) \mid y \in Q \cap (\mathbb{Z}^{p'} \times \mathbb{R}^{q'})\}) \subseteq \arg\max\{f(x) \mid x \in X\}$.

Assume we are given an optimization problem Π as in (4.3). If there exists a MIP formulation of an instance I, we can solve I by solving the maximization problem of g over the mixed-integer set $Q \cap (\mathbb{Z}^{p'} \times \mathbb{R}^{q'})$ obtaining a solution y. By applying the transformation τ on y, we obtain a solution $\tau(y) \in X$, which by Definition 4.6 Part 2 is an optimal solution of I. Therefore, if there exists a MIP formulation of (4.3), the optimization problem Π can be solved by a MIP.

This allows us to refine Problem 4.3.

Problem 4.7. *Given an optimization problem* Π *of the form*

$$\max\{f(x) \mid x \in X\}, \tag{4.4}$$

does there exist a MIP formulation for Π*?*

The question whether a set X can be described as a mixed-integer set, that is, whether this set can be represented as the projection of a polytope, has been fully characterized in [65]. An equivalent algebraic description can also be found in [12]. This means that Problem 4.7 reduces to the question whether there exists a linear objective function for finding an optimum for f.

Observe that we cannot expect to find a MIP formulation (Q, g, τ) in the original space for every problem Π if τ is the identity. For example, assume that f has a unique integral maximizer $x^\star$ in the relative interior of $\mathrm{conv}(X)$. If (Q, g, id) was a MIP formulation of Π in the original space with $Q \cap X = X$, there would exist an optimal point on the boundary of $\mathrm{conv}(X)$, a contradiction to Property 2 of Definition 4.6. If we have $X \subseteq \{0,1\}^n$, however, this problem cannot occur since binary sets do not have relative interior integer points.

Another problem of Definition 4.6 is that such a formulation always exists, but might not be useful. It is clear that Q, g, and τ can be chosen easily once an optimal value of (4.3) is known. For example, let an optimum of (4.3) be attained at $x^\star$. In this case, setting g to be constant and τ mapping everything to $x^\star$, Q can be any non-empty polyhedron, in particular it can be chosen as a single point. Since solving (4.3) to obtain $x^\star$ might be NP-hard, we cannot expect that such a MIP formulation can be computed efficiently and the formulation will in general depend on the given data, i.e., the set X. Thus, the definition of a MIP formulation has to be refined.

Definition 4.8. Let I be an instance of an optimization problem with objective f. A MIP formulation (Q, g, τ) of I is called *efficient* if

1. Q, g, and τ can be generated in polynomial time in $\langle I \rangle$,
2. $g(y)$ can be evaluated in polynomial time in $\langle I \rangle$ for all $y \in Q$,
3. $\tau(y)$ can be evaluated in polynomial time in $\langle I \rangle$ for all $y \in Q \cap (\mathbb{Z}^{p'} \times \mathbb{R}^{q'})$,

where $\langle \cdot \rangle$ denotes the encoding length (see for example [51]).

Formulation (4.1) is an example of an efficient MIP formulation. Note that this formulation is given in an extended space if we consider the original space to be given by the x-variables alone. Furthermore, note that in Definition 4.8 it is necessary to require g to be computable efficiently, since otherwise, once an optimal solution is known, g can easily be chosen to be only optimal in that particular solution.

In general, the following theorem shows that an efficient integer programming formulation always exists if f is linear, $X \subseteq \{0,1\}^n$, and the membership problem is in NP since the proof is constructive.

Theorem 4.9 ([73]). *Consider a 0/1-problem that defines $X^I \subseteq \{0,1\}^{n(I)}$ for each instance I, and let the membership problem for X^I be in* NP. *Then there exists a polynomial p such that for each instance I, there is a system $Ax + By \leq b$ of at most $p(n(I))$ linear inequalities and $k(I) \leq p(n(I))$ auxiliary variables with*

$$X^I = \{x \in \{0,1\}^{n(I)} \mid \exists\, y \in \mathbb{Z}^{k(I)} \text{ with } Ax + By \leq b\}.$$

If the membership problem is in P, *the integrality condition on y can be dropped.*

Formulation (4.1) and Theorem 4.9 only hold for extended spaces, but what about the original space? Under some assumptions we can show that there does not exist an efficient formulation.

Lemma 4.10. *Let I be an instance of an* NP*-hard problem Π, and let (Q, g, τ) be a MIP formulation of I (either in the original or an extended space) such that*

$$\max\{g^\top x \mid x \in Q \cap (\mathbb{Z}^{p'} \times \mathbb{R}^{q'})\}$$

can be solved in polynomial time in $\langle I \rangle$. As long as $\mathsf{P} \neq \mathsf{NP}$*, the MIP formulation (Q, g, τ) is not efficient.*

Proof. The assumptions guarantee that the formulation can be derived in polynomial time and that the computation of an optimal solution of I can be achieved in polynomial time as well. □

The previous lemma can directly be used to show that, for the Graph Partitioning problem, no efficient formulation exists in the original x-space if $\mathsf{P} \neq \mathsf{NP}$. To this end, for $G = (V, E)$ and $\mathcal{K} \in \mathbb{N}$, let the set of feasible partitionings be defined as

$$X_V^{\mathcal{K}} := \{x \in \{0,1\}^{V \times [\mathcal{K}]} \mid \sum_{i \in [\mathcal{K}]} x_v^i = 1, v \in V\}.$$

Observe that optimization of a linear objective function over $X_V^{\mathcal{K}}$ is possible in linear time. To see this, inspect the optimization problem $\max\{g^\top x \mid x \in X_V^{\mathcal{K}}\}$. For each node v the entry i can be chosen such that g_v^i, $i \in [\mathcal{K}]$ is maximal. If this is carried out for each node v, the maximum value is attained because of the partitioning constraint. Thus, we only have to inspect the objective function once.

Corollary 4.11. *There does not exist an efficient MIP formulation for the* Graph Partitioning *problem over $X_V^{\mathcal{K}}$.*

Proof. Since linear optimization over $X_V^{\mathcal{K}}$ is possible in linear time, the statement follows from Lemma 4.10 because the Graph Partitioning problem is NP-hard, see Garey and Johnson [44, Problem ND14]. □

The idea of using that optimization over an integer set is possible in polynomial time, whereas the corresponding optimization problem is NP-hard, to show that no efficient formulation exists, can also be applied to other examples. We include here the Traveling Salesman Problem or TSP for short, the Multiprocessor Scheduling and the Maximum Biclique problem.

The TSP on a weighted undirected graph $G = (V, E)$ is to find a weight minimal *Hamiltonian cycle* in G, that is, a cycle containing each node. One possible formulation is to give every node $v \in V$ a number from 1 to $|V|$, in which case every permutation describes a possible Hamiltonian cycle in G. In particular, this implies that the solutions are vertices of the permutahedron, which is the polytope spanned by all permutations (see, e.g., [113]). Note that the well-known Miller-Tucker-Zemlin formulation [92] extends on the permutahedron by also using edge-variables.

Given a set of tasks V, a number of processors $\mathcal{K}$ and an integer length for each task, the Multiprocessor Scheduling problem is to find a schedule for the tasks with the earliest possible deadline such that at most $\mathcal{K}$ tasks are performed at the same time. For an edge or node weighted bipartite graph $G = (V,E)$, the Maximum Biclique problem on G is to find an induced complete bipartite subgraph of G whose (node or edge) weight is maximal.

Similar to Corollary 4.11 we can apply Lemma 4.10 to show that efficient formulations cannot exist for these problems. To this end, let

$$X^G := \{x \in \{0,1\}^V \mid x = \chi(V'),\ V' \subseteq V \text{ is the node set of biclique in } G\}.$$

Corollary 4.12. *As long as* P ≠ NP *there do not exist efficient formulations for the* TSP *over the vertices of the permutahedron, the* Multiprocessor Scheduling *problem over* $X_V^{\mathcal{K}}$ *as well as the edge version of the* Maximum Biclique *problem over* X^G.

Proof. Since the permutahedron can be derived from a linear projection of the Birkhoff polytope (see, e.g., [70]) over which one can optimize in polynomial time, the statement for the TSP follows (for NP-hardness see, e.g., [44, Problem ND22]). Moreover, solutions of the multiprocessor scheduling problem can be modeled as vectors in $X_V^{\mathcal{K}}$ over which linear optimization is possible in linear time, but Multiprocessor Scheduling is NP-hard, see [44, Problem SS8]. Finally, the node version of the maximum biclique problem is solvable in polynomial time, see [109, Theorem 4]. Thus, a linear objective over X^G can be maximized in polynomial time. This proves the assertion, because the edge version of the biclique problem is NP-hard, see [29]. In all cases, applying Lemma 4.10 leads to the stated result. □

Remark 4.13. From a practical point of view, there are additional requirements that seem useful for the general notion of a MIP formulation.

1. If a MIP formulation is known and f is changed, it would be preferable that Q and τ can be chosen as before and only g has to be changed, that is, Q and τ are independent of f.

2. Q should be tractable, meaning polynomial time separable.

3. Part 2 in Definition 4.6 could be changed to equality, i.e., all optimal solutions can be generated.

The first part can be guaranteed in the case that f and τ are affine, since if $f(x) = f^\top x$ and $\tau(y) = \tau y$, for $f \in \mathbb{R}^{p+q}$, $\tau \in \mathbb{R}^{(p'+q')\times(p+q)}$, we obtain:

$$\max_{x\in X} f(x) = \max_{y\in Y} f(\tau(y)) = \max_{y\in Y} f^\top \tau y = \max_{y\in Y} (\tau^\top f)^\top y =: \max_{y\in Y} g(y),$$

where $Y := Q \cap (\mathbb{Z}^p \times \mathbb{R}^q)$. The first equality holds because of Definition 4.6 Condition 2 and the others because of linearity.

To expand on Part 3 of Remark 4.13, we return to the GRAPH PARTITIONING problem. As we have seen, there indeed exists an efficient formulation in extended space given by (4.1) that furthermore fulfills Condition 2 of Definition 4.6 with equality. One natural question is whether this can also be achieved in the original space. Because of the quadratic function, this intuitively should not be the case. As shown in Lemma 4.15 this intuition is correct.

Note that the following lemmata are independent of P vs. NP and state the non-existence of an objective function independent of the time of its calculation in difference to Lemma 4.10. For these lemmata we use the following simple observation and define $P_{\mathcal{K}}(G) := \operatorname{conv}(X_V^{\mathcal{K}})$.

Observation 4.14. *Given a polytope $\mathcal{P}$ and a linear objective function ω. If two points $x, y \in \mathcal{P}$ are both optimal with respect to ω, they have to lie in a common face of $\mathcal{P}$.*

Lemma 4.15. *There does not exist a linear objective function $\omega\colon \mathbb{R}^{V\times[\mathcal{K}]} \to \mathbb{R}$ for the polytope $P_{\mathcal{K}}(G)$ such that the maximizers of ω over $P_{\mathcal{K}}(G)$ are exactly the maximizers of the* QUADRATIC GRAPH PARTITIONING *problem if $\mathcal{K} \geq 2$ and not all solutions of the quadratic problem are optimal.*

Proof. For the sake of contradiction, assume there is a non-trivial face F of $P_{\mathcal{K}}(G)$ that contains exactly the maximizers of the QUADRATIC GRAPH PARTITIONING problem. Let $a^\top x \leq \beta$ be an inequality that induces F. Since exactly the points in F maximize $a^\top x \leq \beta$, we have $a^\top x = \beta$ for all $x \in F$, and $a^\top x < \beta$ for all $x \in P_{\mathcal{K}}(G) \setminus F$.

Observe that $x \in \{0,1\}^{V\times[\mathcal{K}]}$ is a vertex of $P_{\mathcal{K}}(G)$ if and only if for every $v \in V$ there is exactly one $i(v) \in [\mathcal{K}]$ with $x_v^{i(v)} = 1$. Hence, a vertex is contained in F if and only if $i(v) \in \mathcal{A}^v := \arg\max\{a_v^i \mid i \in [\mathcal{K}]\}$ for every $v \in V$.

Let x be a vertex of $P_{\mathcal{K}}(G)$ that is contained in F. If there exists $\tilde{v} \in V$ such that $\mathcal{A}^{\tilde{v}} \neq [\mathcal{K}]$, there exist an index $j \in \mathcal{A}^{\tilde{v}}$ and an index $\tilde{\jmath} \in [\mathcal{K}] \setminus \mathcal{A}^{\tilde{v}}$. By exchanging the entries of x in columns j and $\tilde{\jmath}$, we obtain another vertex $\tilde{x}$ of $P_{\mathcal{K}}(G)$ that is optimal for the QUADRATIC GRAPH PARTITIONING problem, because changing the labels of assigned partitions does not affect the objective value. But $\tilde{x}$ cannot be contained in F, because for every $v \in V \setminus \{\tilde{v}\}$

$$\sum_{i\in[\mathcal{K}]} a_v^i \tilde{x}_v^i \leq \sum_{i\in[\mathcal{K}]} a_v^i x_v^i \qquad \text{and} \qquad \sum_{i\in[\mathcal{K}]} a_{\tilde{v}}^i \tilde{x}_{\tilde{v}}^i = a_{\tilde{v}}^{\tilde{\jmath}} < a_{\tilde{v}}^j = \sum_{i\in[\mathcal{K}]} a_{\tilde{v}}^i x_{\tilde{v}}^i.$$

Consequently, $a^\top \tilde{x} < a^\top x$. For this reason, $\mathcal{A}^v = [\mathcal{K}]$ for all $v \in V$.

But if $\mathcal{A}^v = [\mathcal{K}]$ for every $v \in V$, then every vertex of $P_{\mathcal{K}}(G)$ maximizes $a^\top x \leq \beta$ due to the partitioning constraint (4.2). This contradicts the assumption that not all vertices of $P_{\mathcal{K}}(G)$ are optimal for the quadratic problem. $\square$

Note that this lemma does not depend on the Graph Partitioning problem, but only on assigning nodes to partitions with the requirement that interchanging two partitions does not change the objective value. Thus, in the original space of the x-variables there does not exist a linear objective function that allows for finding all optimal solutions.

Observation 4.14 can also be used for the TSP by relying on the following property of the permutahedron.

Remember that the permutahedron $\Pi_{n-1} \subseteq \mathbb{R}^n$ is defined as the convex hull of all permutations of the coordinates in the vector $(1, 2, \dots, n) \in \mathbb{R}^n$. In particular, the permutahedron is an $(n-1)$-dimensional polytope embedded in an n-dimensional space (see [113]). All the k-dimensional faces of Π_{n-1} are in one-to-one correspondence to (ordered) partitions of $[n]$ into $n-k$ non-empty parts (see [113]). For example, in the case of $n = 4$, one facet F (which has dimension $k = n - 2 = 2$) can be described by the partition $(\{1,3\},\{4,2\})$. The four vertices of F derived from this partition are: $(1,3,4,2)$, $(1,3,2,4)$, $(3,1,4,2)$ and $(3,1,2,4)$, that is, all points generated from all permutations of the coordinates in each set of the partition while maintaining the order of the sets.

This description of faces of Π_{n-1} allows the following observation: there does not exist a non-trivial face of Π_{n-1} which contains both a point and any non-trivial cyclic shift of that point. For example, a cyclic shift applied to the vertex $(1,3,2,4)$ results in the vertex $(4,1,3,2)$. Since the faces can be described by *ordered* partitions, both these points cannot be contained in a common face F if $F \neq \Pi_3$. This observation allows for showing the non-existence of a formulation of the TSP over the permutahedron (compare to Corollary 4.12).

Lemma 4.16. *Provided that not every solution of a* TSP *is optimal, there does not exist a linear objective function ω over the permutahedron such that the maximizers of ω correspond exactly to the optimal solutions of the* TSP.

Proof. Note that for any optimal solution, every cyclic shift of a vertex of the permutahedron yields exactly the same solution of the TSP. Therefore, any face which contains an optimal solution over the permutahedron also has to contain every cyclic shift. The arguments above show that this cannot happen. $\square$

Chapter 5

Different Approaches for Handling the Retooling Process

In this chapter we return to our starting problem: Given a profile, how should it be produced in order to minimize the needed retooling steps (see Section 1.2)? Since we cannot hope to produce all profiles by combining just one piece, we start in Section 5.1 with the problem of partitioning a graph such that all except one of the subgraphs are isomorphic. The size of that part, which does not have to be isomorphic to the others, is minimized. The idea is to create the whole profile nearly completely out of one piece and the missing piece should be as small as possible to ensure a fast production. First, we show a MIP formulation for this problem in Section 5.1.1 and then present in Section 5.1.2 an algorithm that quickly solves this problem for our tested profiles.

In the second part of this chapter, we present a different approach. Instead of looking for identical parts, we search for *similar* parts of a profile. We first formalize the problem of partitioning a graph subject to a given similarity measure in Section 5.2.1 and present a framework for solving this problem in Section 5.2.2. The measures that we implemented can be found in Section 5.2.3. The first measure we propose is called Degree Similarity, for which we also show a MIP formulation. The other measures are Graph Isomorphism, Graph Edit Distance and Maximum Common Induced Subgraph, which are already known from the literature.

5.1 Minimizing the Non-Isomorphic Remaining Part

In this section we are interested in the following problem.

Problem 5.1 (MIN NON-ISOMORPHIC PART). *Given a graph $G = (V, E)$ and an integer $\mathcal{K} \in \mathbb{N}$, the problem is to partition the nodes V of G into $\mathcal{K} + 1$ subsets $V_0, \dots, V_{\mathcal{K}}$ such that the following conditions hold:*

1. *$G[V_i] \cong G[V_j]$ for all $i, j \in [\mathcal{K}]$,*
2. *$G[V_i]$ is connected for all $i \in [\mathcal{K}]_0$,*
3. *$|V_0|$ is as small as possible.*

In Section 5.1.1, we show how to model Problem 5.1 as a mixed-integer program, whereas in Section 5.1.2, it is solved by a brute-force algorithm.

5.1.1 MIP Formulation

In the IP formulation of the CONNECTED MAX-$\mathcal{K}$-CUT problem in Section 3.3 we are mainly interested in edges between different partitions because these are the edges crucial for the objective function, whereas the edges with both endpoints in the same partition are only used for modeling the connectivity. In this section, however, the situation is different. Since we want to model isomorphisms of induced graphs, the edges with both endpoints in the same partition are the main focus, whereas edges between different partitions are of no interest at all. This is why in this section the variables y_{vw} model whether the endpoints are in the same partition, in other words, $y_{vw} = 1$ implies that v and w are in the same partition in contrast to its meaning in Chapters 3 and 4.

To formulate Problem 5.1 as a MIP, we use a binary variable x_v^i to denote if node $v \in V$ is in partition $i \in [\mathcal{K}]_0$. The partitioning constraint can then be written as

$$\sum_{i \in [\mathcal{K}]_0} x_v^i = 1, \qquad v \in V,$$

which is the same as (3.29) from Section 3.3. Furthermore, the binary variable y_{vw} models for every pair of nodes v, $w \in V$ with $v \neq w$ if v and w belong to the same partition. This can be realized by

$$y_{vw} = y_{wv}, \qquad v, w \in V, v \neq w,$$

and the coupling conditions

$$x_v^i + x_w^i - y_{vw} \leq 1, \qquad v, w \in V, v \neq w, i \in [\mathcal{K}]_0, \tag{5.1}$$

$$x_v^i - x_w^i + y_{vw} \leq 1, \qquad v, w \in V, v \neq w, i \in [\mathcal{K}]_0. \tag{5.2}$$

Inequality (5.1) models that if v and w are in the same partition, y_{vw} has to be 1, whereas Inequality (5.2) models that if v and w are not in the same partition, y_{vw} has to be 0 since there is an i such that $x_v^i - x_w^i$ is equal to 1. Note that we define y_{vw} for all pairs of nodes with $v \neq w$ even if there does not exist an edge $\{v, w\}$ because the variable is also used for the isomorphism as explained next.

To model an isomorphism, we use the binary variable z_{vw} to encode whether node $v \in V$ is mapped to $w \in V$ by the isomorphism. Since an isomorphism is a symmetric function, the following constraint has to hold

$$z_{vw} = z_{wv}, \qquad v, w \in V.$$

Since the partition V_0 does not need to be isomorphic to any other partition V_i, $i \in [\mathcal{K}]$, the variable z_{vw} is set to 0 if v belongs to partition V_0, that is,

$$z_{vw} \leq 1 - x_v^0, \qquad v, w \in V. \tag{5.3}$$

Every partition V_i, $i \in [\mathcal{K}]$, other than V_0 has to be isomorphic to exactly $\mathcal{K} - 1$ other partitions. This means that the following equality has to hold

$$\sum_{w \in V} z_{vw} = \mathcal{K} - 1 - x_v^0(\mathcal{K} - 1), \qquad v \in V,$$

which implies (5.3). This also means that we know an upper bound for the total number of isomorphic mappings: Every partition contains a maximum number of $\left\lfloor \frac{|V|}{\mathcal{K}} \right\rfloor$ nodes and each of these nodes is mapped to a maximum of $\mathcal{K} - 1$ nodes. Since the maximum number of isomorphic mappings is attained if V_0 is empty, the above argumentation has to hold for all of the $\mathcal{K}$ partitions. This results in the inequality

$$\sum_{v \in V} \sum_{w \in V} z_{vw} \leq \left\lfloor \frac{|V|}{\mathcal{K}} \right\rfloor \mathcal{K}(\mathcal{K} - 1).$$

In our case, a node v cannot be mapped to another node w if they both belong to the same partition. This can be formalized as

$$x_v^i + x_w^i + z_{vw} \leq 2, \qquad i \in [\mathcal{K}], v, w \in V.$$

Note that we do not have to add this restriction if a node belongs to partition 0, because, in this case, the value of z_{vw} is set to zero by (5.3).

If two nodes u, v are in the same partition, the isomorphism cannot map a third node w to both u and v, since the isomorphism has to map one node to exactly one node of every other partition excluding the zero-partition. This implies the inequality

$$z_{wu} + z_{wv} \leq 2 - y_{uv}, \qquad u, v, w \in V.$$

Note that none of the stated inequalities ensures that z really corresponds to isomorphisms as we have not yet mentioned the property defining an isomorphism: If and only if $\{v,w\}$ is an edge of G, then $\{\phi(v),\phi(w)\}$ has to be an edge, too. We formalize this by excluding the case that the endpoints of an edge are mapped to two points which are not connected by an edge. This results in the inequality

$$z_{vs} + z_{vt} + z_{ws} + z_{wt} \leq 9 - 4y_{vw} - 4y_{st}, \qquad \{v,w\} \in E, \{s,t\} \notin E.$$

If both y_{vw} and y_{st} are 1, the nodes v, w as well as s, t are in the same partition. Since there is an edge between v and w but not between s and t, at most one of z_{vs}, z_{vt}, z_{ws}, z_{wt} can be 1. If at least one of y_{vw} or y_{st} is 0, the inequality becomes redundant.

Obviously, this model contains many symmetries (compare to Section 3.3.7). For example any permutation of the partitions $1,\ldots,\mathcal{K}$ results in a symmetric solution. To remove at least some of these symmetries, we can add the inequality

$$\sum_{\substack{i \in [\mathcal{K}]_0 \\ i \leq v}} x_v^i \geq 1, \qquad v \in V, \tag{5.4}$$

where we assume the nodes to be numbered from 1 to $|V|$. The inequality implies that the first node has to be either in partition 0 or 1, the second has to be in partition 0, 1 or 2 and so forth.

The only remaining part is to model the connectivity, where we again use flow variables as in Section 3.3. Note that in contrast to the "artificial sink" in Section 3.3, we use an "artificial source" as a representative node in this section, but the idea is the same. Every node has to receive an inflow of 1, except $\mathcal{K}+1$ distinguished nodes, which serve as representatives. Flow is only allowed on edges with both endpoints in the same partition. This implies that every representative has to be connected to nodes of the same partition in order for them to receive a flow value. This implies the connectivity of all partitions.

The binary variables ζ_v^i model whether node v is the representative in partition i. Since there can be at most one representative for every partition,

$$\sum_{v \in V} \zeta_v^i \leq 1, \qquad i \in [\mathcal{K}]_0$$

has to hold. Furthermore, a node can only be a representative of partition i if it is a member of partition i, which implies

$$\zeta_v^i \leq x_v^i, \qquad i \in [\mathcal{K}]_0, v \in V.$$

For modeling the flow, we direct each edge in both possible directions by introducing the directed graph $D = (V,A)$, where $A := \{(v,w),(w,v) \mid \{v,w\} \in E\}$.

We introduce the variable $f_{vw} \in \mathbb{R}_+$ for all arcs from D directly corresponding to the edges in G. To model that every node receives a flow value of at least 1 unless it is the representative, we use

$$\sum_{(w,v)\in A} f_{wv} - \sum_{(v,w)\in A} f_{vw} \geq 1 - |V| \sum_{i\in[\mathcal{K}]_0} \zeta_v^i, \qquad v \in V.$$

Note that the constant $|V|$ cannot be reduced because, in the worst case of $V_0 = V$, the representative needs to have an outflow of size $|V|-1$ in contrast to the model shown in Section 3.3.

Finally, there can only be a flow over existing edges, meaning that their endpoints are in the same partition and the corresponding y-variable is 1, i.e.,

$$f_{vw} + f_{wv} \leq |V| y_{vw}, \qquad \{v,w\} \in E.$$

Since we want to maximize the size of the partitions V_i, $i \in [\mathcal{K}]$, thereby minimizing the nodes in V_0 and because the variable z_{vw} can only be 1 for nodes not in V_0, we can state the objective function as

$$\max \sum_{v\in V} \sum_{w\in V} z_{vw}.$$

Of course, connectivity can also be modeled by using separator inequalities as shown in Section 3.3. Our experiments, however, have shown that there exists a faster method for solving Problem 5.1 as explained in the next section.

5.1.2 Brute-Force Algorithm

Apart from the MIP formulation presented in the previous section, we also implemented a direct solution approach for Problem 5.1 (Min Non-Isomorphic Part), which is explained in this section. The general idea is to test every possible solution while exploiting the structure of the underlying graph. Without a reduction of the possible subsets, the number of possible partitions is huge. To see this, let $G = (V,E)$ be a graph on $|V| = n$ nodes. The maximum number of nodes in a partition i, $i \neq 0$ is $\lfloor n/\mathcal{K} \rfloor$. To simplify the notation, we assume that n is divisible by $\mathcal{K}$. Furthermore, we assume that there exists a solution to Problem 5.1 with $|V_0| = 0$. The number of possible partitions of G into $\mathcal{K}$ subgraphs is then given by

$$\frac{1}{\mathcal{K}!} \cdot \binom{n}{n/\mathcal{K}} \binom{n - n/\mathcal{K}}{n/\mathcal{K}} \cdots \binom{n - (\mathcal{K}-1)n/\mathcal{K}}{n/\mathcal{K}}, \tag{5.5}$$

where the division by $\mathcal{K}!$ is due to the partition symmetry since every permutation of partition labels results in a symmetric solution (compare Section 3.3.7). In order for any of these subsets to be a solution to Problem 5.1, connectivity and isomorphism have to be tested as well. Needless to say, this simple approach is not practical.

Our approach exploits connectivity. The main idea is to start with small subsets of nodes which induce isomorphic subgraphs and extend those subgraphs step-by-step with additional nodes. Each node is only added to a subgraph if connectivity is maintained. The outline of the algorithm can be found in Algorithm 7 on page 108.

We start the algorithm in Line 1 with every possible subset $T = \{v_1, \ldots, v_{\mathcal{K}}\}$ which contains exactly $\mathcal{K}$ nodes and define each of the partitions $V_i := \{v_i\}$, $i \in [\mathcal{K}]$, to contain exactly one node. For this partition it obviously holds that $G[V_i] \cong G[V_j]$ for all $i, j \in [\mathcal{K}]$, and $G[V_i]$ is also connected for every $i \in [\mathcal{K}]$. With this possible solution the Expand procedure is called with partition index 1 in Line 2.

In order for a partition to be a solution to Problem 5.1, it has to be checked whether $G[V_0]$ is connected, which is done in Line 12. This solution is saved if it is the best solution found so far and returned if it is optimal. The optimality for a feasible solution follows directly if each partition contains the maximum number of $\lfloor n/\mathcal{K} \rfloor$ nodes (see Line 15).

After the Expand procedure enlarges V_1 by every possible node that ensures connectedness (Line 18), the procedure is called again with the next partition index and the enlarged V_1. If the Expand procedure is called with an index $i \in [\mathcal{K}]$, $i \neq 1$, every possibility to enlarge V_i while ensuring the existence of an isomorphism is tested. For each of these possibilities, the Expand procedure is called with the next partition index in Line 26. If $i = \mathcal{K} + 1$, the procedure is called again with partition index 1 in Line 24.

Lemma 5.2. *Algorithm 7 is correct, meaning all conditions of Problem 5.1 are fulfilled and the obtained solution minimizes the size of V_0.*

Proof. The above description shows that the conditions on the connectedness and isomorphism are fulfilled since, otherwise, no solution is returned and the properties are maintained within every step. Thus, it is only left to show that the minimum value for $|V_0|$ is attained.

We can assume w.l.o.g. $\mathcal{K} = 2$ because larger values for $\mathcal{K}$ do not change the argument. Let $Q^\star$ be an optimal solution for Problem 5.1 and $\phi\colon V_1 \to V_2$ be the isomorphism between V_1 and V_2.

Select an arbitrary node $v_1 \in V_1$, and label the nodes by their appearance in a BFS tree rooted in v_1, i.e., $v_1, v_2, \ldots$. Note that this tree is in V_1 and this ordering of nodes also appears in Algorithm 7 as a possible solution for the set V_1. W.l.o.g. let $w_i = \phi(v_i)$ for $w_i \in V_2$ and $v_i \in V_1$, and let the ordering of w_i be induced by the ordering of v_i.

The isomorphism between the graphs $G[V_1]$ and $G[V_2]$ implies that the subgraphs $G[\{v_1,\dots,v_i\}]$ and $G[\{w_1,\dots,w_i\}]$ are isomorphic for every $i \in [|V_1|]$, because the isomorphism of $G[V_1]$ and $G[V_2]$ implies isomorphism of subsets induced by ϕ. This also means the ordering of V_2 is inspected by the algorithm in Line 26.

This implies that the solution $Q^\star$ is inspected in Algorithm 7 if there does not exist an optimal solution that is inspected first. In this case, however, the objective value is the same and a solution equivalent to $Q^\star$ is returned. □

To show the improvement achieved by Algorithm 7 over the direct approach stated in (5.5), we analyze its running time next.

Lemma 5.3. *The running time of Algorithm 7 is $\mathcal{O}(n!\,n^{\mathcal{K}})$.*

Proof. First note that Algorithm 7 starts in Line 1 with a loop over every subset of V that contains $\mathcal{K}$ elements. Since there are $\binom{n}{\mathcal{K}}$ such subsets Line 1 is called $\mathcal{O}(n^{\mathcal{K}})$ times.

As a next step, we analyze the Expand procedure. In the first call, it holds that $|V_0| = n - \mathcal{K}$. Since in every subsequent call of Expand one node is removed from V_0, the maximum depth of calls to Expand is $n - \mathcal{K}$, which is in the order $\mathcal{O}(n)$. Lines 21 and 29 are called at most $\mathcal{O}(n)$ times. Furthermore, testing for an isomorphism in Line 26 is possible in $\mathcal{O}(n)$ steps because it only needs to be checked if adding a new node w leads to a feasible isomorphism. To see this in more detail, let v be the node that was added to V_1. Then verifying if V_1 and $V_i \cup \{w\}$ are isomorphic is done by testing whether every edge containing v in $G[V_1]$ also is an edge in $G[V_i \cup \{w\}]$ containing w.

To analyze the recursive structure of the Expand procedure, let $M(n)$ be its running time. The maximal depth is n, and, for every call of Expand, the procedure calls itself with the value $|V_0|$ decreased by one. Moreover, in every call an additional work of $\mathcal{O}(n)$ has to be done as explained above. Therefore, the recursive formula for the running time is $M(n) = nM(n-1) + n$ with $M(0) = 1$. Note that we disregard the big O notation in this formula for simplicity reasons. Repeatedly applying the formula to itself leads to the following claim

$$M(n-j) = n(n-1)\cdots(n-j)M(n-j-1) + \sum_{k=0}^{j}\prod_{\ell=0}^{k}(n-\ell). \tag{5.6}$$

We prove (5.6) by induction, where the base case $j = 0$ is obvious. The inductive step $j \longrightarrow j+1$ is shown by the following calculation

$$\begin{aligned}
M(n-j) &= n(n-1)\cdots(n-j)M(n-j-1) + \sum_{k=0}^{j}\prod_{\ell=0}^{k}(n-\ell)\\
&= n(n-1)\cdots(n-j)\,[(n-j-1)(M(n-j-2)+(n-j-1)] + \sum_{k=0}^{j}\prod_{\ell=0}^{k}(n-\ell)\\
&= n(n-1)\cdots(n-j-1)M(n-j-2) + n(n-1)\cdots(n-j-1) + \sum_{k=0}^{j}\prod_{\ell=0}^{k}(n-\ell)\\
&= n(n-1)\cdots(n-j-1)M(n-j-2) + \prod_{\ell=0}^{j+1}(n-\ell) + \sum_{k=0}^{j}\prod_{\ell=0}^{k}(n-\ell)\\
&= n(n-1)\cdots(n-j-1)M(n-j-2) + \sum_{k=0}^{j+1}\prod_{\ell=0}^{k}(n-\ell)\\
&= M(n-(j+1)).
\end{aligned}$$

With $M(0)=1$ and $j=n$ this leads to

$$\begin{aligned}
M(n) &= n(n-1)\cdots(n-j)M(n-j-1) + \sum_{k=0}^{j}\prod_{\ell=0}^{k}(n-\ell)\\
&= n!\cdot M(0) + \sum_{k=0}^{n}\prod_{\ell=0}^{k}(n-\ell)\\
&= n! + \mathcal{O}(n!).
\end{aligned}$$

Thus, the running time of the Expand procedure is $\mathcal{O}(n!)$. Together with the previously mentioned running time of the main loop, the resulting running time for Algorithm 7 is $\mathcal{O}(n!\,n^{\mathcal{K}})$. □

Note that, for sparse graphs, the running time of Algorithm 7 may be much smaller. We estimated in the proof of Lemma 5.3 that the number of calls for Lines 18 and 26 is $\mathcal{O}(n)$. But if we analyze it more closely, we can see that the number of calls is in the order of the maximal degree of a node, which can be much smaller than $\mathcal{O}(n)$. Our practical examples, see Section 1.1, for instance, are always planar. By using Euler's formula, it can easily be shown that the average degree of a node in a planar graph is strictly less than 6, see, e.g., [32, Chapter 4]. This implies that Lines 18 and 26 are called a constant number of times on average for planar graphs.

If we therefore assume a constant degree of c, the recursive formula for the running time of Expand is $M(n) = cM(n-1) + c$ since Expand calls itself c times and testing for isomorphism also needs time c. Proceeding similar to the proof of Lemma 5.3 leads to

$$M(n-j) = c^{j+1}M(n-j-1) + \sum_{k=1}^{j+1} c^j,$$

which can be proven by induction as above. Solving with $M(0) = 1$ results in

$$M(n) = c^n \cdot 1 + \sum_{k=1}^{n} c^k,$$

which is in $\mathcal{O}(c^n)$. Together with $\mathcal{O}(n^{\mathcal{K}})$ for the number of $\mathcal{K}$-element subsets, the total running time of Algorithm 7 is $\mathcal{O}(c^n n^{\mathcal{K}})$.

Algorithm 7 : Brute-force algorithm for solving Problem 5.1

```
Input : Graph $G = (V, E)$, $\mathcal{K} \in \mathbb{N}$
Output : Solution to Problem 5.1, that is, a partition $V_0, \ldots, V_\mathcal{K}$ of the
         nodes, such that $G[V_i] \cong G[V_j]$ for $i, j \in [\mathcal{K}]$ and $G[V_i]$ is
         connected for $i \in [\mathcal{K}]_0$ and $|V_0|$ is as small as possible
foreach $\mathcal{K}$-subset $T := \{v_1, \ldots, v_\mathcal{K}\}$ of $V, V_i := \{v_i\}, i \in [\mathcal{K}]$, $V_0 := V \setminus T$ do
    Expand(1, $V_0, \ldots, V_\mathcal{K}$);
end
if a solution was found then
    return best solution $V_0, \ldots, V_\mathcal{K}$ found;
else
    if $G$ is connected then return $V_0 = V$;
    else return $\emptyset$;
end
Procedure Expand(index of partition $i$, intermediate solution $V_0, \ldots, V_\mathcal{K}$)
    if $i = 1$ then
        if $|V_0|$ is smaller than best solution found so far and $G[V_0]$ is connected
         then
            save solution $V_0, \ldots, V_\mathcal{K}$;
            if $|V_1| = \lfloor n/\mathcal{K} \rfloor$ then
                return optimal solution $V_0, \ldots, V_\mathcal{K}$;
            end
        end
        foreach node $v \in V_0$ that is connected to $V_1$ do
            $V_1 := V_1 \cup \{v\}$;
            $V_0 := V_0 \setminus \{v\}$;
            Expand(2, $V_0, \ldots, V_\mathcal{K}$);
        end
    else if $i = \mathcal{K} + 1$ then
        Expand(1, $V_0, \ldots, V_\mathcal{K}$);
    else
        foreach node $w \in V_0$ such that $G[V_1] \cong G[V_i \cup \{w\}]$ do
            $V_i := V_i \cup \{w\}$;
            $V_0 := V_0 \setminus \{w\}$;
            Expand($i + 1$, $V_0, \ldots, V_\mathcal{K}$)
        end
    end
end
```

5.2 Graph Similarity Measures

Instead of partitioning a graph into isomorphic subgraphs as done in Section 2.1 or the previous section, one can also consider the problem of partitioning a graph into a given number of similar components, which are not necessarily equal. We first define the corresponding problem in Section 5.2.1, and show a general framework for solving these types of problems in Section 5.2.2. In Section 5.2.3 we give a short overview of the four similarity measures we implemented: Degree Similarity, Graph Isomorphism, Graph Edit Distance and Maximum Common Induced Subgraph. Note that the first is tailored for our specific retooling problem introduced in Section 1.2, whereas the remaining measures are known from the literature.

5.2.1 Problem Definition

We will first define how to formally measure the difference between two graphs.

Definition 5.4. Let $\mathcal{G}$ be the space of all graphs. A *similarity measure* between two graphs is a function $\Psi\colon \mathcal{G} \times \mathcal{G} \to \mathbb{R}^+$ with $\Psi(G,G) = 0$ for all graphs G. A similarity measure Ψ is *symmetric* if $\Psi(G_1,G_2) = \Psi(G_2,G_1)$ holds for all graphs G_1 and G_2.

With this we can define the problem that we focus on in this section.

Problem 5.5. *Given a graph $G = (V,E)$, a similarity measure Ψ and an integer $\mathcal{K} \in \mathbb{N}$, the task is to partition V into $\mathcal{K}$ subsets $V = V_1 \cup V_2 \cup \cdots \cup V_{\mathcal{K}}$ such that the sum of all pairwise similarity measures is minimized, that is,*

$$\min \sum_{i,j \in [\mathcal{K}]} \Psi(G[V_i], G[V_j]). \tag{5.7}$$

This allows us for example to reformulate Problem 2.1 in terms of a similarity measure. To do so, we first need to define a measure derived from Graph Isomorphism.

Definition 5.6. Let G_1 and G_2 be two graphs and define the similarity measure Ψ_I by

$$\Psi_I(G_1,G_2) := \begin{cases} 0, & \text{if } G_1 \cong G_2, \\ 1, & \text{if } G_1 \ncong G_2. \end{cases}$$

Thus, Problem 2.1 (Partition Isomorphism) can be formulated as the problem of deciding whether the optimal value to Problem 5.5 is 0 for the measure Ψ_I.

Similar to Problem 2.14 we can also extend Problem 5.5 by additionally requiring the subgraphs to be connected.

Problem 5.7. *Given a graph $G = (V,E)$, a similarity measure Ψ and an integer $\mathcal{K} \in \mathbb{N}$, the task is to partition V into $\mathcal{K}$ subsets $V = V_1 \cup V_2 \cup \cdots \cup V_{\mathcal{K}}$ such that all subgraphs $G[V_i]$ are connected for every $i \in [\mathcal{K}]$ and the sum of all pairwise similarity measures is minimized, that is,*

$$\min \sum_{i,j\in[\mathcal{K}]} \Psi(G[V_i], G[V_j]). \tag{5.8}$$

Thus, solving Problem 5.7 with Ψ_I as a similarity measure also answers Problem 2.14.

If a similarity measure is symmetric, we do not have to calculate all terms of (5.7) or (5.8) since they can be rewritten as

$$\sum_{i,j\in[\mathcal{K}]} \Psi(G[V_i], G[V_j]) = 2\left(\sum_{\substack{i<j\\ i,j\in[\mathcal{K}]}} \Psi(G[V_i], G[V_j]) \right) + \sum_{i\in[\mathcal{K}]} \Psi(G[V_i], G[V_i]).$$

Since $\Psi(G,G) = 0$ holds for all measures, the objective functions (5.7) or (5.8) of Problems 5.5 and 5.7, respectively, can be further simplified to

$$\min \sum_{\substack{i<j\\ i,j\in[\mathcal{K}]}} \Psi(G[V_i], G[V_j]). \tag{5.9}$$

Note that the assumption of a symmetric measure is fulfilled for the similarity measures appearing in this section.

5.2.2 General Solving Framework

One possibility to solve Problems 5.5 and 5.7 is to create MIP formulation for every similarity measure that we are interested in. The three measures appearing in this section allow for such a formulation as is shown in Section 5.2.3. As the construction of such a formulation might be difficult or even impossible, we present an exact framework for solving Problems 5.5 and 5.7 that allows for easier integration of different similarity measures.

When using a different measure one only has to specify the difference between two graphs and the framework allows for calculating the optimal partition such that the given measure is minimized. This means that we can use the fastest algorithm for calculating the measure by relying on methods already implemented, which may be faster than a MIP formulation. The framework is modular, meaning that different parts can easily be interchanged or new ones can be added. We also allow for the integration of lower bounds that may reduce the number of times that a measure has to be calculated explicitly, since, for example, solving Graph Edit Distance or Maximum Common Induced Subgraph are itself NP-hard (see Section 5.2.3). In the following, we give a more detailed description of this framework mentioning also the downsides of the chosen procedure.

The framework is written in the programming language Java, and there exist two classes for implementing the two main operations: partitioning and similarity. The first class is used for creating the partitions on which the similarity measures are calculated. The general idea is to call the next partition given by a partitioner subclass and calculate its similarity measure given by a subclass of the partitioner superclass. This procedure is repeated until either an optimum is found or the partitioner ends with the result that all partitions have already been returned. In particular, the general idea is to use brute-force for testing every possible partition of the graph and calculating the similarity measure. As mentioned in Section 5.1.2, this approach might be faster for sparse graphs, when for example connectivity is a very restrictive constraint.

Partitioner

For symmetric similarity measures, many of the possible partitions can be excluded since we know that the objective function is independent of the ordering (compare to (5.9)). First of all, we can therefore assume that the first node always is in the first partition. Furthermore, we can restrict the order of different partitions (compare to (5.4)). To explain the idea, let $G = (V,E)$ be a graph. We represent a possible partition by an integer vector of length $|V|$ containing values in $[\mathcal{K}]$, thus denoting the partition index for every node. Let $p = (p_1,\dots,p_{|V|})$ be such a partition vector. To reduce the above-mentioned symmetries, we can restrict the vector p to fulfill the constraint

$$p_{\ell+1} \leq \max\{p_j \mid j \in [\ell]\} + 1$$

for every $\ell \in [|V|-1]$. Note that this restriction is stronger than the one described by (5.4).

Furthermore, the number of possible partitions can be reduced by additional constraints. For example, in order to solve Problem 5.7, the subgraphs have to be connected. In this case, we can restrict the partition vector to those which induce connected subgraphs. We implemented testing for connectivity by using a linear time `depth-first search` algorithm, which is generally much faster than calculating a similarity value. Note that this idea can be extended to include other constraints if variations of Problem 5.7 are to be solved.

Depending on the similarity measure, there may be further partitions which can be excluded. For example, in the case of Graph Isomorphism, we can restrict the partitions to those corresponding to subgraphs which contain the same number of nodes since this is a necessary constraint for graphs to be isomorphic.

In our implementation, this is achieved by calculating permutations of a starting array that is lexicographically sorted. For example, on a graph with four nodes and two different partitions, we start with the array $[1,1,2,2]$. We calculate all permutations of this array by creating the next partition as the next array in the lexicographic ordering. This can be seen in Algorithm 8, which is repeatedly called until it returns -1. Algorithm 8 is taken from [77].

Algorithm 8 : Generate the next permutation in lexicographic order

```
Input : Array of numbers $a$ of length $n$
Output : Array of numbers $b$ such that $b$ is lexicographically larger than $a$
         and is the lexicographically smallest array with this property or
         $-1$ if $a$ is lexicographically maximal
1 $\ell := \max\{i \in [n] \mid a[i] < a[i+1]\}$;
2 if no such $\ell$ exists then
3     return $-1$
4 end
5 $m := \max\{i \in [n] \mid a[i] > a[\ell], i > \ell\}$;
6 swap values $a[\ell]$ and $a[m]$;
7 reverse the subarray $[a[\ell+1],\dots,a[n]]$;
8 return $a$
```

Note that the idea of using specialized partitioners can be adapted for other similarity measures as well.

Calculating the Similarity Value

The superclass for calculating the similarity measure basically contains two functions. The first one returns a similarity value for two given graphs, and the second one returns a lower bound. This last method may be useful when the calculation of the similarity value in itself is complicated, since returning a bound may lead to fewer calculations of the correct similarity value. For example, in the case of Graph Isomorphism, the function Ψ_I as defined in Definition 5.6 can return a value of 1 (implying that the given graphs are not isomorphic) in any of the following cases:

- the number of nodes in both given graphs is not equal, or
- the number of edges in both given graphs is not equal, or
- the degree-vectors are not the same.

Of course, there exist many other necessary conditions for two graphs to be isomorphic, but those mentioned above can be verified very easily. Note that these prerequisites can be examined in linear time, whereas the best known algorithm for Graph Isomorphism requires time $\mathcal{O}(2^{\mathcal{O}(\sqrt{n \log n})})$ [9]. If the returned lower bound is already larger than the best known solution to the problem, we know that calculating the correct similarity value is not needed.

5.2.3 Implemented Measures

In this section we give a short overview of the four similarity measures we implemented: Graph Isomorphism, Degree Similarity, Graph Edit Distance and Maximum Common Induced Subgraph. For Degree Similarity also a MIP formulation is given since the other measures are already known from literature.

Degree Similarity

As mentioned in Section 1.2, the graphs model profiles and the degree of a node therefore models the complexity of the corresponding junction. This leads to the idea of defining a measure based on the degrees of nodes in the graphs. One possibility is to generate the sorted degree sequence for each graph, that is, a sequence with the degree of every node sorted in decreasing order. By comparing these two sequences, a similarity measure for graphs is created from a similarity measure for two sequences. This idea of defining a graph measure from a measure between sequences has also been used before, see for example Graph Edit Distance, which is explained next.

What we do in this section is calculating the number of nodes with a specified degree and use the sum of absolute values of differences of these values to define a similarity measure. More formally, let z_d^1 denote the number of nodes with degree d in G_1, and let z_d^2 be defined accordingly for G_2. If Δ denotes the maximum degree in both G_1 and G_2, that is, $\Delta := \max\{\deg(v) \mid v \in V_1 \cup V_2\}$, Degree Similarity is defined by the measure

$$\Psi_D(G_1, G_2) := \sum_{d=0}^{\Delta} |z_d^1 - z_d^2|. \tag{5.10}$$

Note that, in particular, this definition implies that Ψ_D is symmetric. Also note that $\Psi_D(G_1, G_2) = 0$ does not automatically imply $G_1 \cong G_2$ as is the case with the other three measures defined in this section.

Problem 5.7 with similarity measure Ψ_D can also be modeled by an IP. In order to do so, we use the binary variables x_v^i to model whether node v is in partition i and the binary variable y_{vw} to model if nodes v and w are in the same partition. Since this is equivalent to the model shown in Section 5.1.1, we do not repeat the details here.

The binary variable z_{vd} models if node v has degree d, which is enforced by the two constraints

$$\sum_{d\in[\Delta]_0} z_{vd} = 1, \qquad v \in V, \tag{5.11}$$

$$\sum_{d\in[\Delta]_0} d \cdot z_{vd} = \sum_{w\in V} y_{vw}, \qquad v \in V. \tag{5.12}$$

Equality (5.11) implies that, for each node v exactly one, z_{vd} is 1 for exactly one d, and (5.12) models that, for this d, the degree of v is equal to the number of edges from v to another node w in the same partition as v.

Since we need the degree sequence of every partition, we introduce the binary variable z^i_{vd} to denote if node v has degree d and is in partition i. This can be modeled by the inequalities

$$z_{vd} + x^i_v - z^i_{vd} \leq 1, \qquad v \in V, d \in [\Delta]_0, i \in [\mathcal{K}], \tag{5.13}$$

$$z^i_{vd} \leq z_{vd}, \qquad v \in V, d \in [\Delta]_0, i \in [\mathcal{K}], \tag{5.14}$$

$$z^i_{vd} \leq x^i_v, \qquad v \in V, d \in [\Delta]_0, i \in [\mathcal{K}], \tag{5.15}$$

Inequality (5.13) models that, if node v is in partition i and has degree d, the variable z^i_{vd} has to be 1. Inequality (5.14) models that, if z^i_{vd} is 1, node v has to have degree d. Lastly, Inequality (5.15) models that if z^i_{vd} is 1, node v has to be in partition i.

The last variable we introduce counts the difference of nodes of the same degree in different partitions and therefore is integer-valued. For given partitions i and j, the variable z^{ij}_d denotes the absolute value of the difference of nodes in i and j with degree d by the inequalities

$$\sum_{v\in V}(z^i_{vd} - z^j_{vd}) \leq z^{ij}_d, \qquad i,j \in [\mathcal{K}], d \in [\Delta]_0, \tag{5.16}$$

$$\sum_{v\in V}(z^j_{vd} - z^i_{vd}) \leq z^{ij}_d, \qquad i,j \in [\mathcal{K}], d \in [\Delta]_0. \tag{5.17}$$

To model the absolute value, the inequalities (5.16) and (5.17) only differ in the sign of the left-hand-side.

If instead of solving Problem 5.5, one wants to solve Problem 5.7, additional connectivity-enforcing constraints have to be added. We refer to Section 3.3.1 for more details.

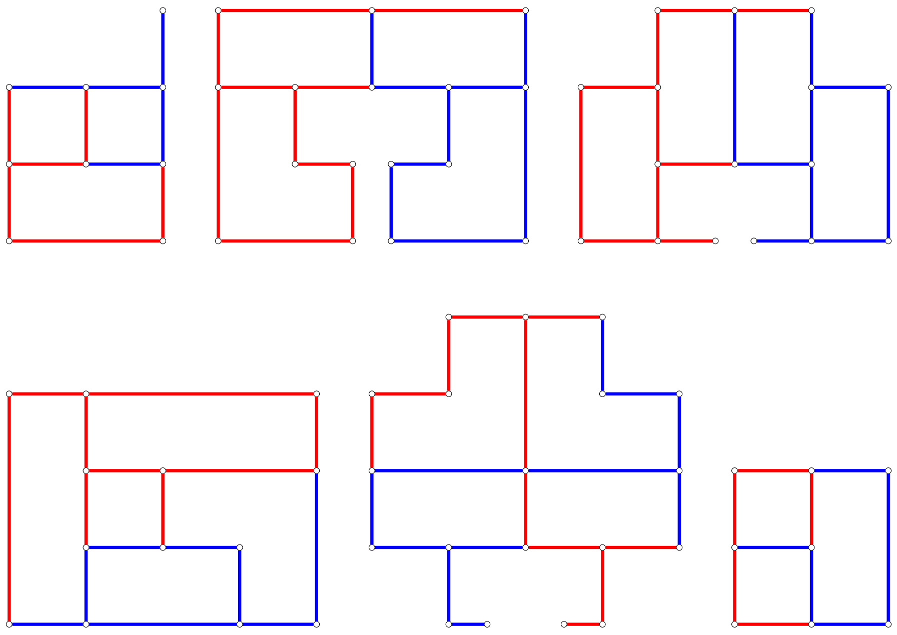
Note that $G_1 \cong G_2$ implies $\Psi_D(G_1, G_2) = 0$, which in particular implies $\Psi_D(G, G) = 0$. Together with the above-mentioned fact that Degree Similarity is symmetric, this implies that we can use the simplifications shown in (5.9) to model the objective function as

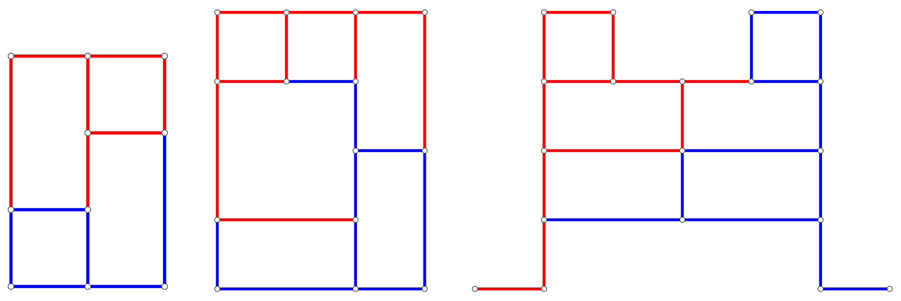
$$\min \sum_{d \in [\Delta]_0} \sum_{i<j} z_d^{ij}.$$

Results on Degree Similarity

To show the solutions we use the line graphs of nine different profiles, which already served as examples in [52, 55]. For these examples we always choose $\mathcal{K} = 2$ with red and blue representing the respective subgraphs.

Note that points with four edges adjacent need additional effort for practical use since it cannot be manufactured this way (see the fifth example below).



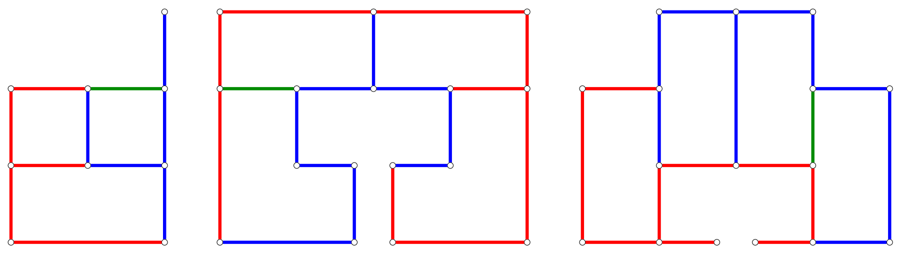


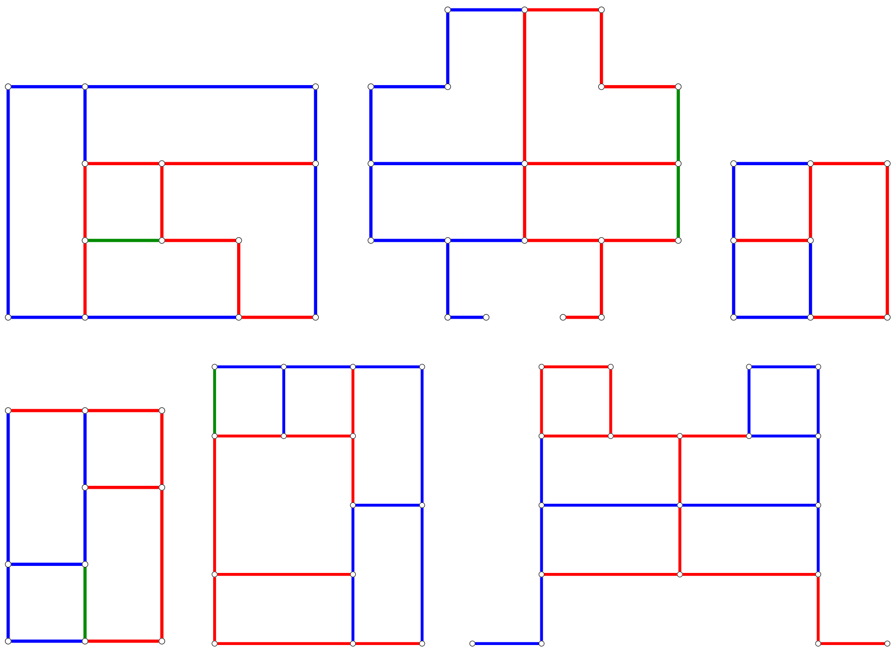
Graph Isomorphism

GRAPH ISOMORPHISM is implemented twice: first as a brute-force method checking for every permutation of nodes if the adjacency-matrices are equal and second by using `bliss` [68]. We do not include more details here since they can be found in Section 5.2.2, where GRAPH ISOMORPHISM serves as an example.

Results on Graph Isomorphism

Similar to the solutions on DEGREE SIMILARITY (see page 115) we include the results on GRAPH ISOMORPHISM here. This results are solutions to Problem 5.1 (MIN NON-ISOMORPHIC PART) by using Algorithm 7. The remaining part is denoted by green edges. Note that the similarity measure given in Definition 5.6 derived from GRAPH ISOMORPHISM is implied by these solutions since the absence of green edges implies isomorphic subgraphs and if there exists a green edge, no isomorphic partitioning is possible. Additionally, see that the solutions also may not be used in practice as the last example shows that metal intersection has to be forbidden.





Graph Edit Distance

Graph Edit Distance is based on distances for strings [82, 107]. For graphs, the first formalization can be found in [99]. There is a vast amount of available literature, which cannot be mentioned in detail here, see for example the book [97]. A recent survey article of the history of Graph Edit Distance and different approaches for its calculation is [43].

The basic idea is as follows: Given two graphs G_1 and G_2, how can G_1 be transformed to result in G_2? This transformation can use a specified set of operations, where each is assigned a cost. Of all possible sequences of operations, the one with the minimal cost value is defined to be the Graph Edit Distance. Here we refrain from defining the general setting, which allows, for example, labeled graphs, additional transformations and general cost functions and restrict attention to the cases that are relevant in this context.

Definition 5.8. We consider the following operations:

A. Relabel a node.

B. Add a node.

C. Remove a node and all its adjacent edges.

D. Insert an edge between two existing nodes.

E. Remove an edge.

The cost of Operation A is 0, Operations B, D and E are defined to cost 1, whereas the cost of Operation C is 1 plus the number of removed adjacent edges.

A finite sequence of these operations p is called an *edit path* and the cost $c(p)$ of p is defined as the sum of the costs of all operations in the path. An edit path is called *complete* if its set of operations transforms graph G_1 into G_2. The set of complete edit paths from G_1 to G_2 is denoted by $\Omega(G_1, G_2,)$.

Definition 5.9. The GRAPH EDIT DISTANCE between two graphs G_1 and G_2 is defined as

$$\min_{p \in \Omega(G_1, G_2)} c(p). \tag{5.18}$$

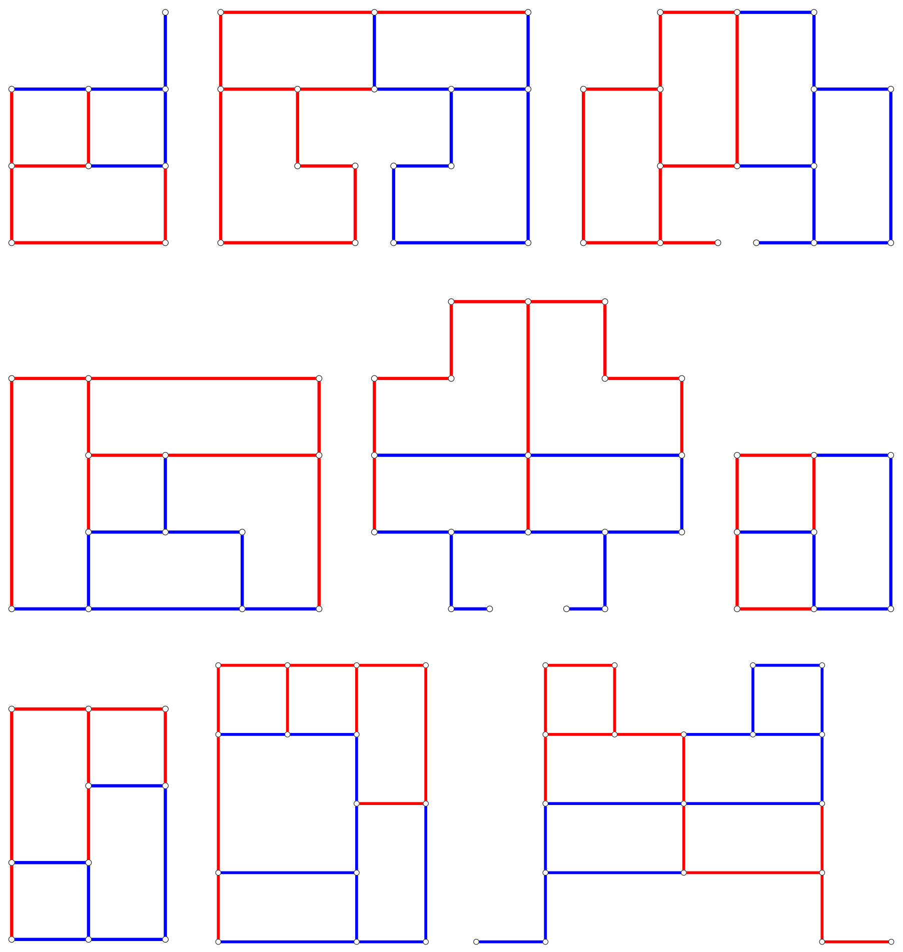
Let Ψ_E denote the similarity measure defined by (5.18).

The definition of the costs for the different operations imply that in our case the measure is symmetric. Moreover, it holds that Ψ_E is a generalization of graph isomorphism, meaning that $\Psi_E(G_1, G_2) = 0$ if and only if $G_1 \cong G_2$. This also means that if we solve Problem 5.5 or Problem 5.7 with Ψ_E as a similarity measure, we can also answer Problem 2.1 or Problem 2.14.

In general, calculating the GRAPH EDIT DISTANCE is NP-complete, see [112] but there are various approaches for calculating Ψ_E. The first IP formulation can be found in [69] and a recent formulation and a computational study can be found in [81]. Note that the LP relaxation of the IP formulation may also serve as a lower bound to be implemented in the framework presented in Section 5.2.2. The time needed for the calculation is $\mathcal{O}(n^7)$ by using an interior point algorithm (which runs in time $\mathcal{O}(n^{3.5})$ and is used on n^2 variables for the edges), see [69]. In our case, however, we rely on the graph matching toolkit [98] for the calculation of the GRAPH EDIT DISTANCE.

Results on Graph Edit Distance

See here the results on the profiles mentioned on page 115. Note that intersection is still allowed here, as can be seen in the fifth and ninth example, compare to GRAPH ISOMORPHISM in the previous section.



Maximum Common Induced Subgraph

The Maximum Common Induced Subgraph problem (see Problem 2.4) can be used in different ways for creating a similarity measure. The one that was used in our experiments is the following.

Definition 5.10. Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two graphs and let $G = (V, E)$ be their Maximum Common Induced Subgraph. The similarity measure Ψ_S

between G_1 and G_2 is defined by

$$\Psi_S(G_1, G_2) := \begin{cases} 0, & \text{if } |V| = |V_1| = |V_2|, \\ \frac{1}{|V|}, & \text{otherwise.} \end{cases} \tag{5.19}$$

Note that this is a direct generalization of GRAPH ISOMORPHISM (Definition 5.6) since $\Psi_D(G_1, G_2) = 0$ holds if and only if $G_1 \cong G_2$ holds, that is, $\Psi_I(G_1, G_2) = 0$. This also has the same implications as in the case of the GRAPH EDIT DISTANCE: Solving Problem 5.5 or Problem 5.7 with Ψ_D as a similarity measure also solves Problem 2.1 or Problem 2.14.

Note that MAXIMUM COMMON INDUCED SUBGRAPH can also be modeled as an IP, see [94]. Furthermore, there exists a cost function for the GRAPH EDIT DISTANCE such that solving the MAXIMUM COMMON INDUCED SUBGRAPH problem is equivalent to calculating the GRAPH EDIT DISTANCE with this particular cost function [21].

For the implementation in our framework presented in Section 5.2.2, we calculated the MAXIMUM COMMON INDUCED SUBGRAPH by the computation of a maximum clique in the so-called compatibility graph [83] defined as follows.

Definition 5.11. Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two graphs. The *compatibility graph* $G_1 \nabla G_2 = (V, E)$ is defined by the nodes $V := V_1 \times V_2$, and there exists an edge $\{(u_i, v_j), (u_k, v_\ell)\} \in E$ if and only if $u_i \neq u_k$ and $v_j \neq v_\ell$ as well as

- $\{u_i, u_k\} \in E_1$ and $\{v_j, v_\ell\} \in E_2$ or
- $\{u_i, u_k\} \notin E_1$ and $\{v_j, v_\ell\} \notin E_2$.

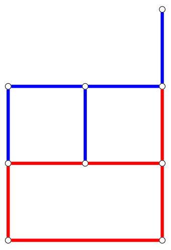
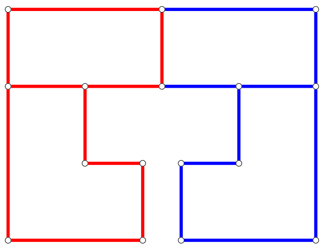
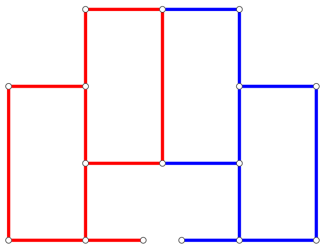
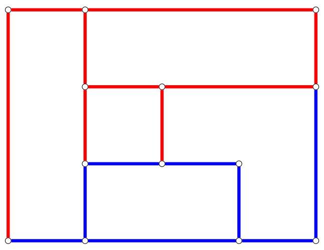
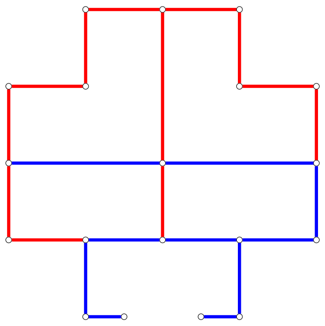
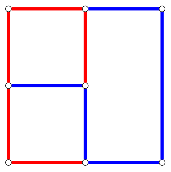
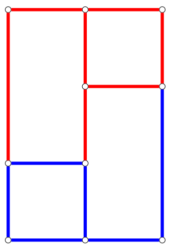
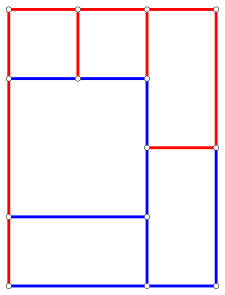
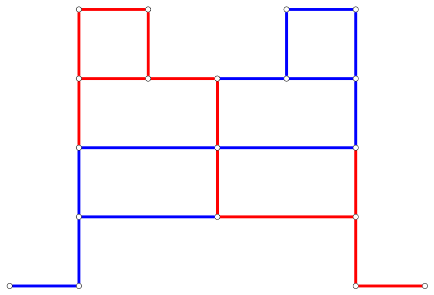
The compatibility graph is also known as *association graph* or *weak modular product*.

Let there exist an edge in $G_1 \nabla G_2$ between the nodes (u_i, v_j) and (u_k, v_ℓ). The definition of $G_1 \nabla G_2$ implies that $\phi(u_i) = v_j$ and $\phi(u_k) = v_\ell$ is a valid mapping for an isomorphism ϕ since either an edge is mapped to an edge or a pair of non-adjacent nodes is mapped to a pair of nodes that also is non-adjacent. Thus, a maximum clique in $G_1 \nabla G_2$ maximizes the pairwise *compatible* nodes, that is, nodes for which edges or *non-edges* are preserved by the corresponding mapping. A proof that a maximum clique in the compatibility graph corresponds to a MAXIMUM COMMON INDUCED SUBGRAPH can for example be found in [83].

To find a maximum clique in a graph, we use the algorithm presented in [96]. Note that various other methods exist for this problem, which, depending on the graphs, may be faster than using the compatibility graph. One example is Constrained Programming, see for example [90] for a comparison between these two methods.

Results on Maximum Common Induced Subgraph

Finally, the examples used before, see page 115. Note that the fifth and ninth example show that the idea might not be used in practice without additional constraints (compare to Graph Edit Distance or Graph Isomorphism).

Chapter 6

Conclusion and Outlook

We dealt with different topics in this thesis and in each of them are many interesting directions for further research. First of all, Section 2.1 proves NP-completeness of Problem 2.1 (Partition Isomorphism). Moreover, we showed that the problem is solvable in polynomial time for outerplanar graphs in the case of $\mathcal{K} = 2$. Remark 2.17 furthermore mentions that the NP-completeness proof cannot be applied to the case of planar graphs. But especially this graph class is of interest since the graphs derived from profiles are in particular planar. Unfortunately, we were not able to show either NP-completeness or the existence of a polynomial algorithm despite all efforts. Other graph classes might also be of interest.

The combinatorial question inspected in Section 2.3 relies on the very simple path graph. Extending this result to more complex graph classes is also a possible direction for future research.

The connected subgraph polytope inspected in Section 3.1 can be fully described in the cases of G being a tree or a cycle. This begs the question of a full description for cactus graphs, which are defined by the property that each edge is contained in at most one cycle. Basically, cactus graphs are a combination of trees and cycles. Furthermore, there are open questions raised in Section 3.1.1 concerning the general structure of facet-defining inequalities for all graphs.

For both the connected subpartition and partition polytopes studied in Sections 3.2.1 and 3.2.2, respectively, the facet-defining inequalities are either trivial or derived from the connected subgraph polytope. Are there others? A similar question regards the Connected Max-$\mathcal{K}$-Cut problem. The cuts from Section 3.3.4 either hold because of connectivity or because of the cut-property. But are there cuts which combine these two properties? Furthermore, in Section 3.2.1 we mention problems which arise when the inequality (3.21), implying that the

$\mathcal{K}$ partitions are each non-empty, is added. The corresponding polytope has not been studied here.

Chapter 4 proposes a definition of a MIP formulation concerning the objective function. There are open problems already mentioned in Remark 4.13, where additional properties of MIP formulations are presented. Under which assumptions can these additional properties be guaranteed? In particular, what is a guarantee for fulfilling Condition 2 of Definition 4.6 with equality?

The minimization of the time needed for the retooling process is revisited in Chapter 5, where we present other solution methods and a general solution framework. One possibility for further research is to develop a similarity measure, which directly corresponds to the production process. For example, in our model, two graphs are isomorphic independent from the measurements of their corresponding profiles. As it is straight-forward to also include weight functions on the edges or nodes, it is much more complicated to also include other aspects into the similarity measure. For instance, in the introduction, it is mentioned that changing a tool in the beginning of the production line leads to a change of all succeeding tools. This time aspect is not taken into account in any of the presented measures.

Appendix A

Influence of Symmetry Handling for Connected Max-K-Cut

This section contains more detailed tables also showing the influence of symmetry handling techniques when solving Problem 3.58 (Connected Max-$\mathcal{K}$-Cut). A detailed description can be found in Section 3.3.8. The additional information is given by the "sym" setting. The numbers indicate, which symmetry handling technique is used (see Section 3.3.7). The text "sym 0" means that no technique is used, whereas "sym 1" indicates the handling of partitioning symmetries and "sym 2" represents the handling of both, partitioning and graph symmetry. The tables can be found in Tables A.1 to A.3.

The randomly generated instances do not contain many graph symmetries, which implies that their handling is not beneficial. Otherwise, even when it is faster not to handle symmetries, the additional time for symmetry handling is minor. The test set I080 always profited from using the setting "sym 2". In the Color02 test set there only exist a few settings using the flow formulation with no additional cuts, where "sym 1" was superior to "sym 2". That is why in Section 3.3.8 we chose to always handle both types of symmetry.

Table A.1: Comparison of different parameter settings for solving the CONNECTED MAX-$\mathcal{K}$-CUT problem for the test set `Color02`

Setting	#SepaCuts	SepaTime	#DomRed	PropTime	#Opt	Time
f cut 0 branch 0 sym 0 heur 0	0.0	0.0	0.0	0.00	7	2090.24
f cut 0 branch 0 sym 0 heur 1	0.0	0.0	0.0	0.00	6	2232.15
f cut 0 branch 0 sym 1 heur 0	0.0	0.0	0.0	0.00	7	1998.94
f cut 0 branch 0 sym 1 heur 1	0.0	0.0	0.0	0.00	6	2119.19
f cut 0 branch 0 sym 2 heur 0	0.0	0.0	0.0	0.00	7	2130.45
f cut 0 branch 0 sym 2 heur 1	0.0	0.0	0.0	0.00	7	1939.66
f cut 0 branch 1 sym 0 heur 0	0.0	0.0	0.0	0.00	7	2091.52
f cut 0 branch 1 sym 0 heur 1	0.0	0.0	0.0	0.00	6	2232.42
f cut 0 branch 1 sym 1 heur 0	0.0	0.0	0.0	0.00	7	1999.50
f cut 0 branch 1 sym 1 heur 1	0.0	0.0	0.0	0.00	6	2119.33
f cut 0 branch 1 sym 2 heur 0	0.0	0.0	0.0	0.00	7	2130.90
f cut 0 branch 1 sym 2 heur 1	0.0	0.0	0.0	0.00	7	1940.32
f cut 1 branch 0 sym 0 heur 0	11 856.8	13.0	23.2	1.53	12	1498.28
f cut 1 branch 0 sym 0 heur 1	10 561.5	11.4	20.2	0.46	14	1367.26
f cut 1 branch 0 sym 1 heur 0	9769.4	11.2	18.4	1.46	13	1283.92
f cut 1 branch 0 sym 1 heur 1	9143.1	11.2	18.0	0.35	14	1242.55
f cut 1 branch 0 sym 2 heur 0	10 131.4	11.8	22.1	1.64	13	1267.08
f cut 1 branch 0 sym 2 heur 1	8092.9	10.5	17.2	0.37	15	1094.92
f cut 1 branch 1 sym 0 heur 0	11 404.6	12.7	21.4	1.50	12	1449.95
f cut 1 branch 1 sym 0 heur 1	10 639.7	11.5	21.3	0.45	14	1372.39
f cut 1 branch 1 sym 1 heur 0	9711.4	11.1	19.3	1.49	13	1277.79
f cut 1 branch 1 sym 1 heur 1	8898.9	11.1	16.8	0.34	14	1208.92
f cut 1 branch 1 sym 2 heur 0	9881.4	11.7	22.5	1.65	13	1243.33
f cut 1 branch 1 sym 2 heur 1	8124.5	10.5	17.0	0.37	15	1100.98
nf cut 0 branch 0 sym 0 heur 0	28 032.0	28.0	25.7	1.64	19	1080.17
nf cut 0 branch 0 sym 0 heur 1	29 897.3	27.4	25.1	0.80	17	1135.86
nf cut 0 branch 0 sym 1 heur 0	20 584.2	21.4	21.2	1.40	20	892.07
nf cut 0 branch 0 sym 1 heur 1	24 530.8	23.2	19.1	0.38	20	911.60
nf cut 0 branch 0 sym 2 heur 0	21 296.3	23.0	21.1	1.58	19	820.60
nf cut 0 branch 0 sym 2 heur 1	22 459.7	21.9	21.9	0.52	20	845.86
nf cut 0 branch 1 sym 0 heur 0	27 830.3	27.8	25.4	1.61	19	1074.70
nf cut 0 branch 1 sym 0 heur 1	30 355.3	27.3	25.0	0.80	17	1133.55
nf cut 0 branch 1 sym 1 heur 0	20 434.6	21.2	20.9	1.41	20	885.81
nf cut 0 branch 1 sym 1 heur 1	24 550.8	23.1	19.6	0.37	20	906.54
nf cut 0 branch 1 sym 2 heur 0	21 191.8	22.9	21.4	1.57	19	812.90
nf cut 0 branch 1 sym 2 heur 1	22 751.3	21.7	22.1	0.53	20	837.61
nf cut 1 branch 0 sym 0 heur 0	26 727.0	18.4	22.9	1.65	20	958.53
nf cut 1 branch 0 sym 0 heur 1	29 682.2	19.4	27.9	0.86	19	1024.24
nf cut 1 branch 0 sym 1 heur 0	20 575.2	15.5	19.9	1.32	20	759.64
nf cut 1 branch 0 sym 1 heur 1	22 281.2	16.8	19.6	0.36	20	807.49
nf cut 1 branch 0 sym 2 heur 0	20 100.2	15.9	22.0	1.47	20	751.17
nf cut 1 branch 0 sym 2 heur 1	21 771.0	16.8	21.0	0.65	20	800.75
nf cut 1 branch 1 sym 0 heur 0	27 160.2	18.5	22.9	1.65	20	952.20
nf cut 1 branch 1 sym 0 heur 1	29 990.9	19.5	26.3	0.83	19	1023.73
nf cut 1 branch 1 sym 1 heur 0	20 611.9	15.4	19.6	1.31	20	756.60
nf cut 1 branch 1 sym 1 heur 1	21 683.9	16.6	20.2	0.36	20	794.67
nf cut 1 branch 1 sym 2 heur 0	19 802.8	15.8	21.2	1.46	20	741.76
nf cut 1 branch 1 sym 2 heur 1	21 417.3	16.9	20.2	0.64	20	799.23

Table A.2: Comparison of different parameter settings for solving the Connected Max-$\mathcal{K}$-Cut problem for the test set I080

Setting	#SepaCuts	SepaTime	#DomRed	PropTime	#Opt	Time
f cut 0 branch 0 sym 0 heur 0	0.0	0.0	0.0	0.00	5	3099.69
f cut 0 branch 0 sym 0 heur 1	0.0	0.0	0.0	0.00	10	2639.37
f cut 0 branch 0 sym 1 heur 0	0.0	0.0	0.0	0.00	5	3104.14
f cut 0 branch 0 sym 1 heur 1	0.0	0.0	0.0	0.00	5	3342.92
f cut 0 branch 0 sym 2 heur 0	0.0	0.0	0.0	0.00	24	1402.00
f cut 0 branch 0 sym 2 heur 1	0.0	0.0	0.0	0.00	23	1510.28
f cut 0 branch 1 sym 0 heur 0	0.0	0.0	0.0	0.00	3	3349.15
f cut 0 branch 1 sym 0 heur 1	0.0	0.0	0.0	0.00	4	3074.86
f cut 0 branch 1 sym 1 heur 0	0.0	0.0	0.0	0.00	8	3167.08
f cut 0 branch 1 sym 1 heur 1	0.0	0.0	0.0	0.00	4	3383.18
f cut 0 branch 1 sym 2 heur 0	0.0	0.0	0.0	0.00	25	1466.49
f cut 0 branch 1 sym 2 heur 1	0.0	0.0	0.0	0.00	26	1379.09
f cut 1 branch 0 sym 0 heur 0	6398.0	6.5	46.3	0.18	73	331.17
f cut 1 branch 0 sym 0 heur 1	5533.3	6.9	38.6	0.11	75	308.20
f cut 1 branch 0 sym 1 heur 0	7514.8	7.2	48.1	0.21	67	315.69
f cut 1 branch 0 sym 1 heur 1	4102.3	4.3	36.1	0.09	76	204.46
f cut 1 branch 0 sym 2 heur 0	7363.0	4.9	50.5	0.21	89	122.51
f cut 1 branch 0 sym 2 heur 1	5505.8	3.7	43.8	0.13	95	99.82
f cut 1 branch 1 sym 0 heur 0	5806.4	5.9	38.6	0.17	75	298.70
f cut 1 branch 1 sym 0 heur 1	5924.9	7.0	38.4	0.12	73	304.66
f cut 1 branch 1 sym 1 heur 0	5639.7	5.3	39.3	0.19	73	247.41
f cut 1 branch 1 sym 1 heur 1	5449.5	5.7	39.9	0.11	73	242.71
f cut 1 branch 1 sym 2 heur 0	5492.4	3.9	35.7	0.19	92	100.99
f cut 1 branch 1 sym 2 heur 1	5468.1	3.9	39.5	0.15	92	99.10
nf cut 0 branch 0 sym 0 heur 0	23 669.3	17.2	64.3	0.55	48	834.40
nf cut 0 branch 0 sym 0 heur 1	21 950.7	15.5	62.3	0.44	52	846.77
nf cut 0 branch 0 sym 1 heur 0	18 727.8	15.5	56.6	0.39	54	710.55
nf cut 0 branch 0 sym 1 heur 1	20 497.5	17.5	59.6	0.47	49	791.81
nf cut 0 branch 0 sym 2 heur 0	20 577.4	11.5	57.4	0.35	75	290.66
nf cut 0 branch 0 sym 2 heur 1	19 025.7	12.6	55.0	0.37	71	293.39
nf cut 0 branch 1 sym 0 heur 0	17 478.3	15.0	53.5	0.35	53	674.75
nf cut 0 branch 1 sym 0 heur 1	13 414.1	11.2	46.3	0.29	54	581.97
nf cut 0 branch 1 sym 1 heur 0	15 595.1	13.3	51.2	0.34	52	588.56
nf cut 0 branch 1 sym 1 heur 1	13 560.3	13.1	42.8	0.27	54	589.95
nf cut 0 branch 1 sym 2 heur 0	15 592.9	9.8	51.7	0.39	71	231.35
nf cut 0 branch 1 sym 2 heur 1	15 968.8	9.6	51.5	0.28	75	238.62
nf cut 1 branch 0 sym 0 heur 0	11 462.5	6.7	52.3	0.43	75	183.83
nf cut 1 branch 0 sym 0 heur 1	8217.4	7.1	42.1	0.30	73	156.19
nf cut 1 branch 0 sym 1 heur 0	12 434.2	9.2	54.4	0.48	72	175.30
nf cut 1 branch 0 sym 1 heur 1	11 346.2	8.0	56.2	0.39	71	160.05
nf cut 1 branch 0 sym 2 heur 0	17 292.1	7.0	66.7	0.64	92	88.46
nf cut 1 branch 0 sym 2 heur 1	12 994.2	4.5	61.7	0.34	92	66.64
nf cut 1 branch 1 sym 0 heur 0	11 115.8	7.0	46.5	0.33	74	186.05
nf cut 1 branch 1 sym 0 heur 1	12 023.8	8.9	47.1	0.30	74	209.06
nf cut 1 branch 1 sym 1 heur 0	9635.4	7.5	44.1	0.33	74	153.82
nf cut 1 branch 1 sym 1 heur 1	8868.7	8.0	42.0	0.30	74	152.56
nf cut 1 branch 1 sym 2 heur 0	10 429.4	4.1	47.5	0.41	93	61.35
nf cut 1 branch 1 sym 2 heur 1	9960.4	3.7	48.2	0.30	93	56.27

Table A.3: Comparison of different parameter settings for solving the Connected Max-$\mathcal{K}$-Cut problem for the test set Random

Setting	#SepaCuts	SepaTime	#DomRed	PropTime	#Opt	Time
f cut 0 branch 0 sym 0 heur 0	0.0	0.0	0.0	0.00	14	3270.96
f cut 0 branch 0 sym 0 heur 1	0.0	0.0	0.0	0.00	15	3354.46
f cut 0 branch 0 sym 1 heur 0	0.0	0.0	0.0	0.00	13	3325.47
f cut 0 branch 0 sym 1 heur 1	0.0	0.0	0.0	0.00	9	3364.03
f cut 0 branch 0 sym 2 heur 0	0.0	0.0	0.0	0.00	13	3325.69
f cut 0 branch 0 sym 2 heur 1	0.0	0.0	0.0	0.00	9	3363.59
f cut 0 branch 1 sym 0 heur 0	0.0	0.0	0.0	0.00	14	3305.25
f cut 0 branch 1 sym 0 heur 1	0.0	0.0	0.0	0.00	13	3447.63
f cut 0 branch 1 sym 1 heur 0	0.0	0.0	0.0	0.00	13	3340.76
f cut 0 branch 1 sym 1 heur 1	0.0	0.0	0.0	0.00	9	3441.05
f cut 0 branch 1 sym 2 heur 0	0.0	0.0	0.0	0.00	13	3342.07
f cut 0 branch 1 sym 2 heur 1	0.0	0.0	0.0	0.00	9	3441.01
f cut 1 branch 0 sym 0 heur 0	3753.3	4.7	5.5	0.04	122	439.52
f cut 1 branch 0 sym 0 heur 1	3506.5	4.6	5.2	0.01	126	417.57
f cut 1 branch 0 sym 1 heur 0	2896.2	3.2	4.8	0.04	134	305.79
f cut 1 branch 0 sym 1 heur 1	2875.7	3.4	5.1	0.01	132	312.33
f cut 1 branch 0 sym 2 heur 0	2976.5	3.3	4.9	0.04	134	312.48
f cut 1 branch 0 sym 2 heur 1	3001.5	3.4	5.4	0.01	132	319.86
f cut 1 branch 1 sym 0 heur 0	2950.8	3.8	4.0	0.04	126	378.72
f cut 1 branch 1 sym 0 heur 1	2895.2	3.7	4.1	0.01	128	372.99
f cut 1 branch 1 sym 1 heur 0	3375.8	3.9	5.4	0.04	131	338.37
f cut 1 branch 1 sym 1 heur 1	2879.0	3.2	4.9	0.01	134	306.66
f cut 1 branch 1 sym 2 heur 0	3223.3	3.7	5.1	0.04	132	328.40
f cut 1 branch 1 sym 2 heur 1	2859.2	3.2	5.0	0.01	134	305.63
nf cut 0 branch 0 sym 0 heur 0	13 308.1	25.0	7.6	0.09	51	1692.06
nf cut 0 branch 0 sym 0 heur 1	10 740.5	20.6	6.7	0.04	58	1468.02
nf cut 0 branch 0 sym 1 heur 0	11 485.0	22.4	7.1	0.06	61	1401.37
nf cut 0 branch 0 sym 1 heur 1	12 400.7	22.1	7.9	0.06	62	1409.36
nf cut 0 branch 0 sym 2 heur 0	11 451.7	21.9	7.1	0.06	62	1389.09
nf cut 0 branch 0 sym 2 heur 1	12 499.0	22.4	8.0	0.07	61	1424.69
nf cut 0 branch 1 sym 0 heur 0	11 177.1	21.7	6.6	0.06	59	1492.26
nf cut 0 branch 1 sym 0 heur 1	10 817.0	20.7	6.3	0.05	57	1457.31
nf cut 0 branch 1 sym 1 heur 0	9572.9	19.7	5.7	0.05	62	1234.78
nf cut 0 branch 1 sym 1 heur 1	10 291.5	20.3	5.9	0.03	64	1247.61
nf cut 0 branch 1 sym 2 heur 0	9447.7	19.6	5.5	0.05	63	1222.95
nf cut 0 branch 1 sym 2 heur 1	10 305.8	20.3	5.8	0.02	65	1246.45
nf cut 1 branch 0 sym 0 heur 0	4921.1	4.0	5.5	0.05	135	195.93
nf cut 1 branch 0 sym 0 heur 1	4716.8	4.0	5.9	0.03	135	191.52
nf cut 1 branch 0 sym 1 heur 0	5102.7	3.7	6.8	0.06	142	155.46
nf cut 1 branch 0 sym 1 heur 1	5145.1	4.3	7.6	0.08	139	159.77
nf cut 1 branch 0 sym 2 heur 0	5165.3	3.9	6.8	0.06	141	157.51
nf cut 1 branch 0 sym 2 heur 1	4912.0	4.1	7.2	0.07	139	154.00
nf cut 1 branch 1 sym 0 heur 0	5067.9	3.8	5.9	0.06	134	198.69
nf cut 1 branch 1 sym 0 heur 1	4664.0	3.9	5.0	0.02	134	193.32
nf cut 1 branch 1 sym 1 heur 0	4367.8	2.9	5.0	0.05	144	138.57
nf cut 1 branch 1 sym 1 heur 1	4532.8	3.2	6.0	0.03	144	140.99
nf cut 1 branch 1 sym 2 heur 0	4344.8	2.9	5.2	0.05	144	138.05
nf cut 1 branch 1 sym 2 heur 1	4669.6	3.4	6.1	0.03	143	145.83

Bibliography

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The design and analysis of computer algorithms*. 1974 (cit. on p. 25).

[2] Z. Ales and A. Knippel. "An extended edge-representative formulation for the K-partitioning problem". In: *Electronic Notes in Discrete Mathematics* 52.Supplement C (2016). INOC 2015 - 7th International Network Optimization Conference, pp. 333–342. DOI: 10.1016/j.endm.2016.03.044 (cit. on p. 90).

[3] E. Álvarez-Miranda, I. Ljubić, and P. Mutzel. "The Maximum Weight Connected Subgraph Problem". In: *Facets of Combinatorial Optimization: Festschrift for Martin Grötschel*. Ed. by M. Jünger and G. Reinelt. Springer Berlin Heidelberg, 2013, pp. 245–270. DOI: 10.1007/978-3-642-38189-8_11 (cit. on p. 32).

[4] E. Álvarez-Miranda, I. Ljubić, and P. Mutzel. "The Rooted Maximum Node-Weight Connected Subgraph Problem". In: *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems: 10th International Conference, CPAIOR 2013, Yorktown Heights, NY, USA, May 18-22, 2013. Proceedings*. Ed. by C. Gomes and M. Sellmann. Springer Berlin Heidelberg, 2013, pp. 300–315. DOI: 10.1007/978-3-642-38171-3_20 (cit. on p. 32).

[5] E. Álvarez-Miranda and M. Sinnl. "A Relax-and-Cut framework for large-scale maximum weight connected subgraph problems". In: *Computers & Operations Research* 87 (2017), pp. 63–82. DOI: 10.1016/j.cor.2017.05.015 (cit. on p. 69).

[6] G. E. Andrews. *The Theory of Partitions*. Cambridge Mathematical Library. Cambridge University Press, 1998 (cit. on p. 28).

[7] M. F. Anjos, B. Ghaddar, L. Hupp, F. Liers, and A. Wiegele. "Solving k-Way Graph Partitioning Problems to Optimality: The Impact of Semidefinite Relaxations and the Bundle Method". In: *Facets of Combinatorial Opti-*

mization: Festschrift for Martin Grötschel. Ed. by M. Jünger and G. Reinelt. Springer Berlin Heidelberg, 2013, pp. 355–386. doi: 10.1007/978-3-642-38189-8_15 (cit. on pp. 64, 72).

[8] L. Babai, A. Dawar, P. Schweitzer, and J. Torán. "The Graph Isomorphism Problem (Dagstuhl Seminar 15511)". In: *Dagstuhl Reports* 5.12 (2016). Ed. by L. Babai, A. Dawar, P. Schweitzer, and J. Torán, pp. 1–17. doi: 10.4230/DagRep.5.12.1 (cit. on p. 14).

[9] L. Babai and E. M. Luks. "Canonical Labeling of Graphs". In: *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*. STOC '83. ACM, 1983, pp. 171–183. doi: 10.1145/800061.808746 (cit. on p. 113).

[10] S. Bachl. "Isomorphe Subgraphen und deren Anwendung beim Zeichnen von Graphen". ger. PhD thesis. Universität Passau, 2001 (cit. on pp. 17, 20).

[11] F. Barahona and A. R. Mahjoub. "On the cut polytope". In: *Mathematical Programming* 36.2 (1986), pp. 157–173. doi: 10.1007/BF02592023 (cit. on p. 74).

[12] A. Basu, K. Martin, C. T. Ryan, and G. Wang. "Mixed-Integer Linear Representability, Disjunctions, and Variable Elimination". In: *Integer Programming and Combinatorial Optimization: 19th International Conference, IPCO 2017, Waterloo, ON, Canada, June 26-28, 2017, Proceedings*. Ed. by F. Eisenbrand and J. Koenemann. Springer International Publishing, 2017, pp. 75–85. doi: 10.1007/978-3-319-59250-3_7 (cit. on p. 92).

[13] L. W. Beineke. "Characterizations of derived graphs". In: *Journal of Combinatorial Theory* 9.2 (1970), pp. 129–135. doi: 10.1016/s0021-9800(70)80019-9 (cit. on p. 33).

[14] J. Bensmail. "On the complexity of partitioning a graph into a few connected subgraphs". In: *Journal of Combinatorial Optimization* 30.1 (2015), pp. 174–187. doi: 10.1007/s10878-013-9642-8 (cit. on p. 56).

[15] M. D. Biha, H. L. Kerivin, and P. H. Ng. "Polyhedral study of the connected subgraph problem". In: *Discrete Mathematics* 338.1 (2015), pp. 80–92. doi: 10.1016/j.disc.2014.08.026 (cit. on pp. 33, 40, 41).

[16] H. L. Bodlaender. "Treewidth: Structure and Algorithms". In: *Structural Information and Communication Complexity: 14th International Colloquium, SIROCCO 2007, Castiglioncello, Italy, June 5-8, 2007. Proceedings*. Ed. by G. Prencipe and S. Zaks. Springer Berlin Heidelberg, 2007, pp. 11–25. doi: 10.1007/978-3-540-72951-8_3 (cit. on p. 15).

[17] V. Bonnici, R. Giugno, A. Pulvirenti, D. Shasha, and A. Ferro. "A subgraph isomorphism algorithm and its application to biochemical data". In: *BMC Bioinformatics* 14.Suppl 7 (2013), S13. doi: 10.1186/1471-2105-14-s7-s13 (cit. on p. 18).

[18] *Boost C++ Libraries*. url: http://www.boost.org/ (cit. on pp. 69, 71, 74, 76).

[19] F. Brandenburg. "Pattern matching problems in graphs". Manuscript. 2000 (cit. on pp. 18, 27).

[20] A. Buluç, H. Meyerhenke, I. Safro, P. Sanders, and C. Schulz. "Recent Advances in Graph Partitioning". In: *Algorithm Engineering: Selected Results and Surveys*. Ed. by L. Kliemann and P. Sanders. Springer International Publishing, 2016, pp. 117–158. doi: 10.1007/978-3-319-49487-6_4 (cit. on p. 18).

[21] H. Bunke. "On a relation between graph edit distance and maximum common subgraph". In: *Pattern Recognition Letters* 18.8 (1997), pp. 689–694. doi: 10.1016/S0167-8655(97)00060-3 (cit. on p. 120).

[22] R. Carvajal, M. Constantino, M. Goycoolea, J. P. Vielma, and A. Weintraub. "Imposing Connectivity Constraints in Forest Planning Models". In: *Operations Research* 61.4 (2013), pp. 824–836. doi: 10.1287/opre.2013.1183 (cit. on p. 34).

[23] S. Chopra and M. R. Rao. "Facets of the k-partition polytope". In: *Discrete Applied Mathematics* 61.1 (1995), pp. 27–48. doi: 10.1016/0166-218x(93)e0175-x (cit. on p. 64).

[24] S. Chopra and M. R. Rao. "The partition problem". In: *Mathematical Programming* 59.1 (1993), pp. 87–115. doi: 10.1007/BF01581239 (cit. on pp. 57, 64, 75, 90).

[25] S. Chopra and M. R. Rao. *The Partition Problem I: Formulations, dimensions and basic facets*. Working Paper 89-27. Stern School of Business, New York University, 1989 (cit. on p. 57).

[26] *Color02 - computational symposium: Graph coloring and its generalizations. Available at*. http://mat.gsia.cmu.edu/COLOR02. 2002 (cit. on p. 84).

[27] S. Cook. "The Complexity of Theorem Proving Procedures". In: *Proceedings of the Third Annual ACM Symposium on Theory of Computing*. 1971, pp. 151–158 (cit. on pp. 18, 19).

[28] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. 3rd. The MIT Press, 2009 (cit. on pp. 69, 71, 74).

[29] M. Dawande, P. Keskinocak, and S. Tayur. *On the biclique problem in bipartite graphs*. Tech. rep. GSIA Working Paper 1996-04. Carnegie Mellon University, 1996 (cit. on p. 95).

[30] M. Deza and M. Laurent. "Applications of cut polyhedra I". In: *Journal of Computational and Applied Mathematics* 55.2 (1994), pp. 191–216. DOI: 10.1016/0377-0427(94)90020-5 (cit. on p. 64).

[31] M. Deza and M. Laurent. "Applications of cut polyhedra II". In: *Journal of Computational and Applied Mathematics* 55.2 (1994), pp. 217–247. DOI: 10.1016/0377-0427(94)90021-3 (cit. on p. 64).

[32] R. Diestel. *Graph Theory*. 4th ed. Vol. 173. Graduate texts in mathematics. Springer, 2012 (cit. on pp. 14, 106).

[33] B. Dilkina and C. P. Gomes. "Solving Connected Subgraph Problems in Wildlife Conservation". In: *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems: 7th International Conference, CPAIOR 2010, Bologna, Italy, June 14-18, 2010. Proceedings*. Ed. by A. Lodi, M. Milano, and P. Toth. Springer Berlin Heidelberg, 2010, pp. 102–116. DOI: 10.1007/978-3-642-13520-0_14 (cit. on p. 67).

[34] M. T. Dittrich, G. W. Klau, A. Rosenwald, T. Dandekar, and T. Müller. "Identifying functional modules in protein-protein interaction networks: an integrated exact approach". In: *Bioinformatics* 24.13 (2008), pp. i223–i231. DOI: 10.1093/bioinformatics/btn161 (cit. on p. 33).

[35] M. E. Dyer and A. M. Frieze. "On the complexity of partitioning graphs into connected subgraphs". In: *Discrete Applied Mathematics* 10.2 (1985), pp. 139–153. DOI: 10.1016/0166-218X(85)90008-3 (cit. on p. 56).

[36] J. Edmonds. "Paths, Trees, and Flowers". In: *Canadian Journal of Mathematics* (1965), pp. 449–467 (cit. on p. 23).

[37] H.-C. Ehrlich and M. Rarey. "Maximum common subgraph isomorphism algorithms and their applications in molecular science: a review". In: *Wiley Interdisciplinary Reviews: Computational Molecular Science* 1.1 (2011), pp. 68–79. DOI: 10.1002/wcms.5 (cit. on p. 19).

[38] L. Q. Eifler, K. B. Reid Jr, and D. P. Roselle. "Sequences with adjacent elements unequal". English. In: *Aequationes Mathematicae* 6.2-3 (1971), pp. 256–262. DOI: 10.1007/BF01819761 (cit. on p. 29).

[39] P. Elias, A. Feinstein, and C. E. Shannon. "A note on the maximum flow through a network". In: *IRE Trans. Information Theory* 2.4 (1956), pp. 117–119. DOI: 10.1109/TIT.1956.1056816 (cit. on p. 67).

[40] M. Fischetti et al. "Thinning out Steiner trees: a node-based model for uniform edge costs". In: *Mathematical Programming Computation* 9.2 (2017), pp. 203–229. doi: 10.1007/s12532-016-0111-0 (cit. on pp. 33, 68, 69).

[41] L. R. Ford and D. R. Fulkerson. "Maximal flow through a network". In: *Journal canadien de mathématiques* 8.0 (1956), pp. 399–404. doi: 10.4153/cjm-1956-045-5 (cit. on p. 67).

[42] B. A. Galler and M. J. Fisher. "An Improved Equivalence Algorithm". In: *Communications of the ACM* 7.5 (1964), pp. 301–303. doi: 10.1145/364099.364331 (cit. on p. 69).

[43] X. Gao, B. Xiao, D. Tao, and X. Li. "A survey of graph edit distance". In: *Pattern Analysis and Applications* 13.1 (2010), pp. 113–129. doi: 10.1007/s10044-008-0141-y (cit. on p. 117).

[44] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1979 (cit. on pp. 18, 19, 90, 94, 95).

[45] M. C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Vol. 57. Annals of Discrete Mathematics. North-Holland Publishing Co., 2004 (cit. on pp. 35, 69).

[46] V. Grimm, T. Kleinert, F. Liers, M. Schmidt, and G. Zöttl. "Optimal price zones of electricity markets: a mixed-integer multilevel model and global solution approaches". In: *Optimization Methods and Software* 0.0 (2017), pp. 1–31. doi: 10.1080/10556788.2017.1401069 (cit. on pp. 67, 78).

[47] P. Groche, E. Bruder, and S. Gramlich, eds. *Manufacturing Integrated Design*. Springer International Publishing, 2017. doi: 10.1007/978-3-319-52377-4 (cit. on pp. 9–11).

[48] P. Groche, C. Ludwig, W. Schmitt, and D. Vucic. "Herstellung multifunktionaler Blechprofile". In: *Werkstatttechnik online: wt, Springer VDI Verlag, Düsseldorf* 99.10 (2009), pp. 712–720 (cit. on p. 11).

[49] P. Groche, J. Ringler, and T. A. Schreehah. "Bending-Rolling combinations for strips with optimized cross section geometries". In: *CIRP Annals - Manufacturing Technology, Elsevier, Manchester* 58/1 (2009), pp. 263–266 (cit. on p. 11).

[50] P. Groche et al. "Future trends in cold rolled profile process technology". In: *Confederation of British Metalforming: cbm* 19 (2010), pp. 16–19 (cit. on p. 10).

[51] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*. English. Vol. 2. Algorithms and Combinatorics. Springer, 1988 (cit. on p. 93).

[52] U. Günther. “Integral Sheet Metal Design by Discrete Optimization”. PhD thesis. Technische Universität Darmstadt, 2010 (cit. on pp. 10, 12, 115).

[53] U. Günther, W. Hess, B. M. Horn, and H. Lüthen. “A holistic topology and shape optimization approach with an application to steel profiles”. In: *Structural and Multidisciplinary Optimization* (2017). DOI: 10.1007/s00158-017-1809-y (cit. on p. 10).

[54] D. J. Haglin and S. M. Venkatesan. “Approximation and intractability results for the maximum cut problem and its variants”. In: *IEEE Transactions on Computers* 40.1 (1991), pp. 110–113. DOI: 10.1109/12.67327 (cit. on p. 64).

[55] W. Hess. “Geometry Optimization with PDE Constraints and Applications to the Design of Branched Sheet Metal Products”. PhD thesis. Technische Universität Darmstadt, 2010 (cit. on pp. 10, 115).

[56] C. Hojny, I. Joormann, H. Lüthen, and M. Schmidt. *Mixed-Integer Programming Techniques for the Connected Max-k-Cut Problem*. to appear. 2018 (cit. on pp. 64, 86).

[57] C. Hojny, H. Lüthen, and M. E. Pfetsch. *On the Size of Integer Programs with Bounded Coefficients or Sparse Constraints*. 2018. eprint: http://www.optimization-online.org/DB_HTML/2017/06/6056.html (cit. on p. 89).

[58] C. Hojny and M. E. Pfetsch. “Polytopes associated with symmetry handling”. In: *Mathematical Programming* (2018). DOI: 10.1007/s10107-018-1239-7 (cit. on p. 84).

[59] J. E. Hopcroft and J. K. Wong. “Linear Time Algorithm for Isomorphism of Planar Graphs (Preliminary Report)”. In: *Proceedings of the Sixth Annual ACM Symposium on Theory of Computing*. STOC ’74. ACM, 1974, pp. 172–184. DOI: 10.1145/800119.803896 (cit. on p. 26).

[60] J. Hopcroft and R. Tarjan. “Algorithm 447: Efficient Algorithms for Graph Manipulation”. In: *Communications of the ACM* 16.6 (1973), pp. 372–378. DOI: 10.1145/362248.362272 (cit. on pp. 72, 81).

[61] B. M. Horn, H. Lüthen, M. E. Pfetsch, and S. Ulbrich. “Geometry and Topology optimization of Sheet Metal Profiles by Using a Branch-And-Bound Framework”. In: *Materials Science & Engineering Technology* 48 (1 2017), pp. 27–40 (cit. on p. 10).

[62] *IBM ILOG CPLEX Optimization Studio*. URL: http://www-03.ibm.com/software/products/de/ibmilogcpleoptistud/ (cit. on p. 84).

[63] O. F. Inc. *The On-Line Encyclopedia of Integer Sequences*. 2017. URL: `http://oeis.org` (cit. on p. 30).

[64] T. Januschowski and M. E. Pfetsch. "The maximum k-colorable subgraph problem and orbitopes". In: *Discrete Optimization* 8.3 (2011), pp. 478–494. DOI: `10.1016/j.disopt.2011.04.002` (cit. on p. 54).

[65] R. G. Jeroslow and J. K. Lowe. "Modelling with integer variables". In: *Mathematical Programming Studies* 22 (1984), pp. 167–184 (cit. on p. 92).

[66] D. S. Johnson. "The NP-completeness column: An ongoing guide". In: *Journal of Algorithms* 6.1 (1985), pp. 145–159. DOI: `10.1016/0196-6774(85)90025-2` (cit. on p. 33).

[67] M. Jünger, G. Reinelt, and W. R. Pulleyblank. "On partitioning the edges of graphs into connected subgraphs". In: *Journal of Graph Theory* 9.4 (1985), pp. 539–549. DOI: `10.1002/jgt.3190090416` (cit. on p. 56).

[68] T. Junttila and P. Kaski. "Engineering an efficient canonical labeling tool for large and sparse graphs". In: *Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments and the Fourth Workshop on Analytic Algorithms and Combinatorics*. Ed. by D. Applegate, G. S. Brodal, D. Panario, and R. Sedgewick. SIAM, 2007, pp. 135–149 (cit. on p. 116).

[69] D. Justice and A. Hero. "A binary linear programming formulation of the graph edit distance". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 28.8 (2006), pp. 1200–1214. DOI: `10.1109/TPAMI.2006.152` (cit. on p. 118).

[70] V. Kaibel. *Extended Formulations in Combinatorial Optimization*. Optima 85. 2011 (cit. on p. 95).

[71] V. Kaibel, M. Peinhardt, and M. E. Pfetsch. "Orbitopal fixing". In: *Discrete Optimization* 8.4 (2011), pp. 595–610. DOI: `10.1016/j.disopt.2011.07.001` (cit. on p. 83).

[72] V. Kaibel and M. E. Pfetsch. "Packing and partitioning orbitopes". In: *Mathematical Programming* 114.1 (2008), pp. 1–36. DOI: `10.1007/s10107-006-0081-5` (cit. on p. 83).

[73] V. Kaibel and S. Weltge. "Lower Bounds on the Sizes of Integer Programs without Additional Variables". In: *Mathematical Programming* (2014). DOI: `10.1007/s10107-014-0855-0` (cit. on p. 93).

[74] R. M. Karp. "Reducibility Among Combinatorial Problems". In: *Complexity of Computer Computations*. Ed. by R. E. Miller and J. W. Thatcher. Plenum Press, 1972, pp. 85–103 (cit. on p. 19).

[75] D. G. Kirkpatrick and P. Hell. “On the Completeness of a Generalized Matching Problem”. In: *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing*. STOC ’78. ACM, 1978, pp. 240–245. DOI: 10.1145/800133.804353 (cit. on p. 18).

[76] T. Kloks and D. Kratsch. “Finding all minimal separators of a graph”. In: *STACS 94: 11th Annual Symposium on Theoretical Aspects of Computer Science Caen, France, February 24–26, 1994 Proceedings*. Ed. by P. Enjalbert, E. W. Mayr, and K. W. Wagner. Springer Berlin Heidelberg, 1994, pp. 759–768. DOI: 10.1007/3-540-57785-8_188 (cit. on p. 69).

[77] D. E. Knuth. *The Art of Computer Programming, Volume 4, Fascicle 2: Generating All Tuples and Permutations*. Addison-Wesley Professional, 2005 (cit. on p. 112).

[78] B. H. Korte, L. Lovász, and R. Schrader. *Greedoids*. Algorithms and Combinatorics 4. Springer-Verlag, 1991 (cit. on pp. 32, 41).

[79] J. B. Kruskal. “On the shortest spanning subtree of a graph and the traveling salesman problem”. In: *Proceedings of the American Mathematical Society* 7.1 (1956), pp. 48–48. DOI: 10.1090/s0002-9939-1956-0078686-7 (cit. on p. 76).

[80] R. E. Ladner. “On the Structure of Polynomial Time Reducibility”. In: *Journal of the ACM* 22.1 (1975), pp. 155–171. DOI: 10.1145/321864.321877 (cit. on p. 15).

[81] J. Lerouge, Z. Abu-Aisheh, R. Raveaux, P. Héroux, and S. Adam. “Exact Graph Edit Distance Computation Using a Binary Linear Program”. In: *Structural, Syntactic, and Statistical Pattern Recognition: Joint IAPR International Workshop, S+SSPR 2016, Mérida, Mexico, November 29 - December 2, 2016, Proceedings*. Ed. by A. Robles-Kelly, M. Loog, B. Biggio, F. Escolano, and R. Wilson. Springer International Publishing, 2016, pp. 485–495. DOI: 10.1007/978-3-319-49055-7_43 (cit. on p. 118).

[82] V. I. Levenshtein. *Binary codes capable of correcting deletions, insertions, and reversals*. Tech. rep. 8. 1966, pp. 707–710 (cit. on p. 117).

[83] G. Levi. “A note on the derivation of maximal common subgraphs of two directed or undirected graphs”. In: *CALCOLO* 9.4 (1973), p. 341. DOI: 10.1007/BF02575586 (cit. on p. 120).

[84] L. A. Levin. “Universal sorting problems”. In: *Problems of Information Transmission* 9 (1973), pp. 265–266 (cit. on p. 19).

[85] I. Ljubić et al. “An Algorithmic Framework for the Exact Solution of the Prize-Collecting Steiner Tree Problem”. In: *Mathematical Programming* 105.2 (2006), pp. 427–449. DOI: 10.1007/s10107-005-0660-x (cit. on p. 33).

[86] L. Lovász and M. D. Plummer. *Matching Theory*. Vol. 29. Annals of Discrete Mathematics 121. North-Holland, Amsterdam and Akadémiai Kiadó, Budapest, 1986 (cit. on p. 57).

[87] S. J. Maher et al. *The SCIP Optimization Suite 4.0*. eng. Tech. rep. 17-12. ZIB, 2017 (cit. on pp. 64, 84).

[88] F. Margot. "Symmetry in Integer Linear Programming". In: *50 Years of Integer Programming 1958-2008: From the Early Years to the State-of-the-Art*. Ed. by M. Jünger et al. Springer Berlin Heidelberg, 2010. Chap. 17, pp. 647–686. doi: 10.1007/978-3-540-68279-0_17 (cit. on p. 83).

[89] E. Q. V. Martins. "On a multicriteria shortest path problem". In: *European Journal of Operational Research* 16.2 (1984), pp. 236–245. doi: 10.1016/0377-2217(84)90077-8 (cit. on p. 65).

[90] C. McCreesh, S. N. Ndiaye, P. Prosser, and C. Solnon. "Clique and Constraint Models for Maximum Common (Connected) Subgraph Problems". In: *Principles and Practice of Constraint Programming: 22nd International Conference, CP 2016, Toulouse, France, September 5-9, 2016, Proceedings*. Ed. by M. Rueher. Springer International Publishing, 2016, pp. 350–368. doi: 10.1007/978-3-319-44953-1_23 (cit. on p. 120).

[91] B. D. McKay and A. Piperno. "Practical graph isomorphism, II". In: *Journal of Symbolic Computation* 60 (2014), pp. 94–112. doi: 10.1016/j.jsc.2013.09.003 (cit. on p. 84).

[92] C. E. Miller, A. W. Tucker, and R. A. Zemlin. "Integer programming formulations and traveling salesman problems". In: *Journal of Association for Computing Machinery* 7 (1960), pp. 326–329 (cit. on p. 95).

[93] M. E. Pfetsch and T. Rehn. *A Computational Comparison of Symmetry Handling Methods for Mixed Integer Programs*. 2015. eprint: http://www.optimization-online.org/DB_HTML/2015/11/5209.html (cit. on p. 83).

[94] B. Piva and C. C. de Souza. "Polyhedral study of the maximum common induced subgraph problem". In: *Annals of Operations Research* 199.1 (2012), pp. 77–102. doi: 10.1007/s10479-011-1019-8 (cit. on p. 120).

[95] S. Poljak and Z. Tuza. "Maximum cuts and largest bipartite subgraphs". In: *Combinatorial Optimization*. Vol. 20. DIMACS Series in Discrete Mathematics and Theoretical Computer Science. DIMACS/AMS, 1993, pp. 181–244 (cit. on p. 64).

[96] P. Prosser. *Exact Algorithms for Maximum Clique: a computational study*. 2012. eprint: https://arxiv.org/abs/1207.4616 (cit. on p. 120).

[97] K. Riesen. *Structural Pattern Recognition with Graph Edit Distance*. Advances in Computer Vision and Pattern Recognition. Springer International Publishing, 2015. DOI: 10.1007/978-3-319-27252-8 (cit. on p. 117).

[98] K. Riesen, S. Emmenegger, and H. Bunke. "A Novel Software Toolkit for Graph Edit Distance Computation". In: *Graph-Based Representations in Pattern Recognition: 9th IAPR-TC-15 International Workshop, GbRPR 2013, Vienna, Austria, May 15-17, 2013. Proceedings*. Ed. by W. G. Kropatsch, N. M. Artner, Y. Haxhimusa, and X. Jiang. Springer Berlin Heidelberg, 2013, pp. 142–151. DOI: 10.1007/978-3-642-38221-5_15 (cit. on p. 118).

[99] A. Sanfeliu and K.-S. Fu. "A distance measure between attributed relational graphs for pattern recognition". In: *IEEE Transactions on Systems, Man, and Cybernetics* SMC-13.3 (1983), pp. 353–362. DOI: 10.1109/TSMC.1983.6313167 (cit. on p. 117).

[100] T. J. Schaefer. "The Complexity of Satisfiability Problems". In: *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing*. STOC '78. ACM, 1978, pp. 216–226. DOI: 10.1145/800133.804350 (cit. on p. 19).

[101] W. Schmitt. *Laserschweißen von Mehrkammerprofilen aus spaltprofilierten Halbzeugen*. Diplomarbeit, Technische Universität Darmstadt. 2008 (cit. on p. 12).

[102] U. Schöning. "Graph Isomorphism is in the Low Hierarchy". In: *Proceedings of the 4th Annual Symposium on Theoretical Aspects of Computer Science*. STACS '87. Springer-Verlag, 1987, pp. 114–124 (cit. on p. 15).

[103] R. Sedgewick and K. Wayne. *Algorithms*. Pearson Education, 2011 (cit. on p. 65).

[104] *SteinLib Testsets*. URL: http://steinlib.zib.de/showset.php?I080 (cit. on p. 84).

[105] P. Turán. "Egy gráfelméleti szélsőértékfeladatról (On an extremal problem in graph theory)". In: *Matematikai és Fizikai Lapok* 48 (1941), pp. 436–452 (cit. on p. 26).

[106] K. Wagner. "Über eine Eigenschaft der ebenen Komplexe". In: *Mathematische Annalen* 114.1 (1937), pp. 570–590. DOI: 10.1007/BF01594196 (cit. on p. 24).

[107] R. A. Wagner and M. J. Fischer. "The String-to-String Correction Problem". In: *J. ACM* 21.1 (1974), pp. 168–173. DOI: 10.1145/321796.321811 (cit. on p. 117).

[108] Y. Wang, A. Buchanan, and S. Butenko. “On imposing connectivity constraints in integer programs”. In: *Mathematical Programming* (2017), pp. 1–31. doi: 10.1007/s10107-017-1117-8 (cit. on pp. 32, 34, 39, 40, 46, 51, 54).

[109] M. Yannakakis. “Node-Deletion Problems on Bipartite Graphs”. In: *SIAM Journal on Computing* 10.2 (1981), pp. 310–327. doi: 10.1137/0210022 (cit. on p. 95).

[110] F. F. C. Yao. “Graph 2-isomorphism is NP-complete”. In: *Information Processing Letters* 9.2 (1979), pp. 68–72. doi: 10.1016/0020-0190(79)90130-3 (cit. on p. 18).

[111] V. N. Zemlyachenko, N. M. Korneenko, and R. I. Tyshkevich. “Graph isomorphism problem”. In: *Journal of Mathematical Sciences* 29.4 (1985), pp. 1426–1481. doi: 10.1007/bf02104746 (cit. on p. 20).

[112] Z. Zeng, A. K. H. Tung, J. Wang, J. Feng, and L. Zhou. “Comparing Stars: On Approximating Graph Edit Distance”. In: *PVLDB* 2.1 (2009), pp. 25–36 (cit. on p. 118).

[113] G. M. Ziegler. *Lectures on Polytopes*. 1st ed. Vol. 152. Graduate Texts in Mathematics. Springer-Verlag, 1995. doi: 10.1007/978-1-4613-8431-1 (cit. on pp. 57, 95, 97).

Wissenschaftlicher Werdegang

11/2012 – 03/2017	Wissenschaftlicher Mitarbeiter am Fachbereich Mathematik der Technischen Universität Darmstadt, Arbeitsgruppe Diskrete Optimierung
10/2012	Abschluss Master of Science in Mathematik
04/2007 – 10/2012	Studium der Mathematik an der Technischen Universität Berlin
10/2006 – 03/2007	Studium der Elektrotechnik an der Technischen Universität Berlin
06/2006	Abitur an der Humboldt-Oberschule Berlin