FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Combinatorial Optimization Approaches for Graph Construction Problems

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktor der Technischen Wissenschaften

by

Dipl.-Ing Dipl.-Ing Benedikt Klocker, BSc
Registration Number 0926194

to the Faculty of Informatics

at the TU Wien

Advisor: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Günther Raidl
Second advisor: Ao.Univ.Prof. Dr. Herbert Fleischner

The dissertation has been reviewed by:

Hao Li	Markus Leitner

Vienna, 17th December, 2019

Benedikt Klocker

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.ac.at

Erklärung zur Verfassung der Arbeit

Dipl.-Ing Dipl.-Ing Benedikt Klocker, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 17. Dezember 2019

Benedikt Klocker

Acknowledgements

First and foremost I want to thank my supervisor Günther Raidl for his comprehensive support and his valuable advice in every aspect of my research. You gave me guidance whenever I needed it and your constructive feedback was very appreciated. I also want to express my deep gratitude to my co-supervisor Herbert Fleischner for providing me with many interesting topics and for collaborating with me on those. A special thanks goes to Hao Li and Markus Leitner for their effort to review this thesis and their highly appreciated feedback.

Furthermore, I want to thank all my coworkers for providing such a positive work environment. It was always nice to discuss different topics with you, not only research-related ones, but also many others.

For the financial support I thank the Austrian Science Fund (FWF) which financed my project position under grant P27615 and the Vienna Graduate School on Computational Optimization, grant W1260, for financing my travel expenses to conferences and workshops.

I am very grateful to my family and my friends for their lifelong support which made me what I am today. Especially I want to thank my parents for giving me the opportunity to follow my dreams and for always supporting me. Finally, my biggest thanks goes to the love of my life, Stefanie, for being always there for me, for being patient with me when I was working too long, for helping me whenever I needed it, and for showing me that mathematics and computer science are not the only important things in life. Without you I would not be where I am today.

Abstract

Many open problems in the field of graph theory are concerned with the existence or non-existence of graphs with certain properties. In this thesis we consider three such graph theoretic problems and develop algorithms to search for graphs with the wanted properties. Although we cannot use computer search to prove the non-existence of a graph with certain properties within a usually infinite set of graphs, we can apply our algorithms to prove the non-existence for all graphs up to a certain size. For some (sub-)problems we focus on developing a practically effective algorithm that only checks if a given graph satisfies certain properties. This can then be combined with an enumeration and filtering approach to search for such graphs. As tool set for our algorithms we use various techniques from the field of combinatorial optimization. Strictly speaking the problem of searching for a graph with certain properties is not an optimization problem, but many graph properties are defined via substructures that minimize or maximize an objective.

The first problem we consider is about the existence of uniquely hamiltonian planar graphs with minimum degree three. A prominent conjecture of Bondy and Jackson states that such graphs do not exist. To search for such graphs we consider two different kinds of approaches. The first approach tries to find uniquely hamiltonian graphs with minimum degree three and an embedding minimizing the number of crossings and the number of degree two vertices. We formalize an optimization problem for this purpose and propose a general variable neighborhood search (GVNS) for solving it heuristically. The different types of used neighborhoods also include an exponentially large neighborhood that is effectively searched by means of branch-and-bound. To check feasibility of neighbors we need to solve hamiltonian cycle problems, which is done in a delayed manner to minimize the computational effort. We compare three different configurations of the GVNS. Although our implementation could not find a uniquely hamiltonian planar graph with minimum degree three disproving Bondy and Jackson's conjecture, we were able to find uniquely hamiltonian graphs with one vertex of degree two and crossing number four for all graph orders from 10 to 100. In a second approach we are searching for planar graphs with minimum degree three that contain a stable fixed edge cycle (SFE-cycle) or equivalently a stable cycle with one vertex of degree two. We show that such a graph can be used to construct a uniquely hamiltonian planar graph with minimum degree three. For generating candidate graphs we use the program plantri and for checking if they contain an SFE-cycle we propose three approaches. Two of them are based on integer

linear programming (ILP) and the other is a cycle enumeration algorithm. To reduce the search space we prove several properties a minimum planar graph with minimum degree three containing an SFE-cycle must satisfy, the most significant being triangle freeness. Comparing the three algorithms shows that the enumeration is more effective on small graphs while for larger graphs the ILP-based approaches perform better. Finally, we use the enumeration approach together with plantri to prove that there does not exist a planar graph with minimum degree three that contains an SFE-cycle with 24 or fewer vertices. This verifies Bondy and Jackson's conjecture for all graphs with up to 25 vertices.

In the second part of this thesis we formulate an algorithm for finding smooth graphs, a special subclass of hamiltonian 4-regular graphs, with small independence numbers. To this end we formalize a family of satisfaction problems and propose a branch-and-bound based approach for solving them. Strong bounds are obtained by exploiting graph-theoretic aspects including new results obtained in cooperation with leading graph theorists. Based on a partial solution we derive a lower bound by computing an independent set on a partial graph and finding a lower bound on the size of possible extensions. The algorithm is used to test conjectured lower bounds on the independence numbers of smooth graphs and some subclasses of smooth graphs. In particular for the whole class of smooth graphs we test the lower bound of $2n/7$ for all smooth graphs with at least $n \geq 12$ vertices and can prove the correctness for all $12 \leq n \leq 24$. Furthermore, we apply the algorithm on different subclasses, such as all triangle free smooth graphs.

Finally, in the third part we are considering an extension of the minor concept to transitioned graphs, which arise in the context of the cycle double cover conjecture. This famous conjecture is strongly related to the compatible circuit decomposition (CCD) problem. A recent result by Fleischner et al. (2018) gives a sufficient condition for the existence of a CCD in a transitioned 2-connected eulerian graph, which is based on an extension of the definition of K_5-minors to transitioned graphs. Graphs satisfying this condition are called sup-undecomposable K_5 (SUD-K_5)-minor-free graphs. We formulate a generalization of this property by replacing the K_5 by a 4-regular transitioned graph H, which is part of the input. Furthermore, we consider the decision problem of checking for two given graphs if the extended property holds. We prove that this problem is $\mathcal{NP}$-complete but can be solved in polynomial time if the size of H is fixed. We then formulate an equivalent problem, present a mathematical model for it, and prove its correctness. This mathematical model is then translated into a mixed integer linear program (MILP) and a boolean satisfiability problem (SAT) for solving it in practice. Non-trivial symmetry breaking constraints are proposed, which improve the solving times of both models considerably. Compared to the MILP model the SAT approach performs significantly better. We used the faster SAT approach to further test graphs of graph theoretic interest and were able to get new insights. Among other results we found snarks with 30 and 32 vertices that do not contain perfect pseudo-matchings, that are spanning subgraphs consisting of K_2 and $K_{1,3}$ components whose contraction lead to SUD-K_5-minor-free graphs.

Kurzfassung

Viele ungelöste Probleme aus dem Gebiet der Graphentheorie beschäftigen sich mit der Existenz oder Nichtexistenz von Graphen mit vorgegebenen Eigenschaften. Wir behandeln in dieser Arbeit drei Probleme dieser Art und entwickeln Algorithmen, um nach Graphen mit den gewollten Eigenschaften zu suchen. Die Suche mittels Computer nach Graphen mit vorgegebenen Eigenschaften kann aufgrund des zumeist unendlich großen Suchraums die Nichtexistenz nicht beweisen. Allerdings können wir die Algorithmen verwenden um die Nichtexistenz für alle Graphen bis zu einer gewissen Größe zu zeigen. Für manche (Teil-)Probleme werden wir uns auf Algorithmen konzentrieren, welche für einen gegebenen Graphen überprüfen, ob dieser die vorgegebenen Eigenschaften hat oder nicht. Dies kann dann mit einer Enumeration von Graphen und darauffolgendem Filtern kombiniert werden. Als Basis für unsere Algorithmen verwenden wir verschiedene Techniken der kombinatorischen Optimierung. Die Suche nach Graphen mit vorgegebenen Eigenschaften stellt formal kein Optimierungsproblem dar, aber oftmals sind die gewollten Eigenschaften durch Substrukturen definiert, welche eine Zielfunktion minimieren oder maximieren.

Das erste Problem, dass wir behandeln, bezieht sich auf eindeutig hamiltonsche planare Graphen mit Minimalgrad drei. Eine berühmte Vermutung von Bondy und Jackson besagt, dass solche Graphen nicht existieren. Wir analysieren zwei Problemvarianten um nach solchen Graphen zu suchen. In der ersten Variante suchen wir nach in die Ebene eingebetteten eindeutig hamiltonschen Graphen mit Minimalgrad drei, welche die Anzahl der Überkreuzungen und die Anzahl der Knoten vom Grad zwei minimieren. Dazu formulieren wir ein Optimierungsproblem und beschreiben einen *General Variable Neighborhood Search (GVNS)* Ansatz um es heuristisch zu lösen. Unter den Nachbarschaften ist auch eine exponentiell große Nachbarschaft, welche mithilfe von *Branch-and-Bound* durchsucht wird. Um die Gültigkeit von Lösungen zu überprüfen müssen wir hamiltonsche Kreisprobleme lösen. Dies passiert verzögert zuerst mittels einer Heuristik und gegebenenfalls erst dann exakt um den Rechenaufwand zu minimieren. In unseren Tests vergleichen wir drei verschiedene Konfigurationen der GVNS. Unsere Implementierung konnte keinen eindeutig hamiltonschen planaren Graphen mit Minimalgrad drei finden, dafür jedoch für alle Knotengrade von 10 bis 100 eindeutig hamiltonsche Graphen mit nur einem Knoten vom Grad zwei und zwei Überkreuzungen.

In der zweiten Variante suchen wir nach planaren Graphen mit Minimalgrad drei welche einen *Stable Fixed Edge Cycle (SFE-Cycle)* enthalten, oder äquivalent einen stabilen

Kreis mit einem Knoten vom Grad zwei. Wir zeigen, dass ausgehend von so einem Graphen ein eindeutig hamiltonscher planarer Graph mit Minimalgrad drei konstruiert werden kann. Um Kandidatengraphen zu generieren verwenden wir das Programm plantri und um zu überprüfen ob sie einen *SFE-Cycle* beinhalten schlagen wir drei Lösungsmethoden vor. Zwei davon basieren auf *Integer Linear Program (ILP)* und der dritte auf einer Enumeration von Kreisen. Um den Suchraum zu verkleinern beweisen wir mehrere Eigenschaften, welche ein minimaler planarer Graph mit Minimalgrad drei, der einen *SFE-Cycle* besitzt, erfüllen muss. Die wichtigste Eigenschaft ist dabei, dass der minimale Graph dreiecksfrei sein muss. In unseren Tests schneidet der auf Enumeration basierende Algorithmus für kleinere Graphen besser ab und die auf ILP basierenden Algorithmen für größere Graphen. Um systematisch nach einem minimalen Gegenbeispiel zu suchen verwenden wir den enumerationsbasierten Algorithmus zusammen mit plantri. Mit den computationalen Tests kontte bewiesen werden, dass es keinen planaren Graphen mit Minimalgrad drei mit bis zu 24 Knoten gibt, der einen SFE-cycle beinhaltet. Dies impliziert, dass Bondy und Jackson's Vermutung für alle Graphen bis zu 25 Knoten wahr ist.

Im zweiten Teil dieser Arbeit formulieren wir einen Algorithmus um nach *smooth Graphs*, einer speziellen Teilklasse von 4-regulären hamiltonschen Graphen, mit kleiner Unabhängigkeitszahl zu suchen. Wir formulieren dazu eine Familie von Problemen und einen *Branch-and-Bound* Ansatz um diese zu lösen. Um starke Schranken zu erhalten nützen wir graphentheoretische Resultate im Bezug auf *smooth Graphs*. Basierend auf einer partiellen Lösung berechnen wir eine unabhängige Menge und verwenden diese um eine untere Schranke für die Unabhängigkeitszahl für alle möglichen Erweiterungen der partiellen Lösung zu berechnen. Wir verwenden dann den Algorithmus um verschiedene Vermutungen in Bezug auf unterschiedliche Teilklassen der *smooth Graphs*, wie zum Beispiel die dreicksfreien *smooth Graphs*, und ihre Unabhängigkeitszahlen zu überprüfen. Insbesondere haben wir für alle *smooth Graphs* mit $12 \leq n \leq 24$ Knoten verifiziert, dass sie eine Unabhängigkeitszahl von mindestens $2n/7$ haben.

Im dritten Teil der Arbeit modellieren wir eine Erweiterung des Konzepts von Minoren auf Graphen mit Durchgangssystemen, welche im Kontext der *Cycle Double Cover* Vermutung auftreten. Diese Vermutung steht in Beziehung mit dem *Compatible Circuit Decomposition (CCD)* Problem. Kürzlich bewiesen Fleischner et al. (2018) eine hinreichende Bedingung für die Existenz von einem CCD in einem 2-zusammenhängenden Eulerschen Graphen mit einem Durchgangssystem. Diese Bedingung basiert auf einer Erweiterung von K_5-Minoren auf Graphen mit einem Durchgangssystem. Graphen, welche diese Bedingung erfüllen, werden *sup-undecomposable* K_5 *(SUD-*K_5*)-minor free Graphs* genannt. Wir verallgemeinern diese Bedingung indem wir den K_5 durch einen beliebigen 4-regulären Graphen H mit einem Durchgangssystem, welcher Teil des Inputs ist, ersetzen. Weiters formulieren wir das Entscheidungsproblem ob zwei gegebene Graphen diese erweiterte Bedingung erfüllen. Wir zeigen, dass das Problem $\mathcal{NP}$-vollständig ist und in polynomieller Zeit gelöst werden kann, wenn die Größe des Graphen H fixiert ist. Dann formulieren wir ein mathematisches Modell für dieses Problem und beweisen

dessen Korrektheit. Dieses Modell wird schließlich verwendet um ein *Mixed Integer Linear Program (MILP)* und ein *boolean Satisfiability Problem* zu formulieren. Weiters entwickeln wir nichttriviale Bedingungen zur Brechung von Symmetrien, welche das Lösen von beiden Modellen bedeutend beschleunigen. Im Vergleich ist das Lösen des SAT Modells signifikant schneller als das Lösen des MILP Modells. Unter Benutzung des SAT Modells testeten wir graphentheoretisch interessante Instanzen und konnten neue Einblicke gewinnen. Unter anderem fanden wir *Snarks* mit 30 und 32 Knoten, welche keinen spannenden Teilgraphen bestehend aus K_2 und $K_{1,3}$ Komponenten, dessen Kontraktion zu einem *SUD-K_5-minor free Graph* führt, beinhalten.

Contents

Abstract vii

Kurzfassung ix

Contents xiii

1 Introduction 1
1.1 Structure of the Thesis 4

2 Methodology 7
2.1 Graph Theoretic Basics 7
2.1.1 Basic Definitions and Notations 7
2.1.2 Graph Isomorphisms 12
2.1.3 Perfect Pseudo-Matchings 13
2.1.4 Crossing Number and Planar Graphs 13
2.2 Decision Problems, Search Problems, and Combinatorial Optimization Problems 16
2.3 Computational Complexity 18
2.3.1 Enumeration 22
2.3.2 Branch-and-Bound 22
2.3.3 (Mixed) Integer Linear Programming 24
2.3.4 SAT Solving 31
2.4 Heuristic and Metaheuristic Approaches 33
2.4.1 Greedy Construction Heuristic 33
2.4.2 Local Search 35
2.4.3 Variable Neighborhood Search 36
2.4.4 Very Large Scale Neighborhood Search 38

3 Searching For Uniquely Hamiltonian Planar Graphs 41
3.1 Introduction 41
3.2 Metaheuristic Approach to Minimize the Number of Crossings and Vertices of Degree Two 43
3.2.1 Problem Description 44

3.2.2 General Variable Neighborhood Search with Delayed Feasibility Checking . . . 46
3.2.3 Computational Results . . . 51
3.2.4 Conclusions and Future Work . . . 55
3.3 Searching for Stable Cycles . . . 56
3.3.1 Reduction to Stable Fixed Edge Cycles . . . 57
3.3.2 Algorithmic Approaches . . . 60
3.3.3 Properties of a Minimum Counterexample . . . 65
3.3.4 Systematic Search for a Minimum Counterexample . . . 72
3.3.5 Computational Results . . . 72
3.3.6 Conclusion . . . 80

4 Finding Smooth Graphs with Small Independence Numbers 83
4.1 Introduction . . . 83
4.2 Problem Formulation . . . 84
4.3 Algorithmic Approach . . . 85
4.3.1 Solution Representation . . . 85
4.3.2 Core Algorithm . . . 85
4.4 Bounds and Other Useful Properties . . . 86
4.5 Checking Infeasibility . . . 89
4.5.1 Symmetry Breaking . . . 91
4.6 Computational Results . . . 93
4.6.1 Problem 1 . . . 93
4.6.2 Problem 2 . . . 94
4.6.3 Problem 3 . . . 94
4.6.4 Problem 4 . . . 94
4.7 Conclusion and Future Work . . . 95

5 Sup-Transition Minor Free Graphs 97
5.1 Introduction . . . 97
5.2 Problem Formulation . . . 101
5.2.1 Basic Definitions . . . 101
5.2.2 Problem Transformation . . . 107
5.3 Modeling . . . 112
5.3.1 Towards a Model . . . 112
5.3.2 The Model . . . 114
5.4 Mixed Integer Linear Programming Model . . . 129
5.5 SAT Model . . . 132
5.6 Symmetry Breaking . . . 135
5.6.1 Finding all Automorphisms and Stabilizers . . . 138
5.7 Systematically Checking PPMs of Snarks . . . 139
5.8 Computational Results . . . 140
5.8.1 Instance Sets . . . 140
5.8.2 Comparing the MILP with the SAT Model . . . 141

5.8.3 Characterizing all Snarks with Up to 32 Vertices 143
5.9 Conclusion and Future Work 144

6 Conclusions **147**

Index **151**

Acronyms **155**

Bibliography **157**

CHAPTER 1

Introduction

In the modern age of the internet where "everyone seems to be connected to everything", many aspects of social life and technological advancements can be viewed as deeply interconnected networks. Examples reach from modeling molecules in chemistry and communication networks in computer science to street networks and friendship relation graphs in social media. The field of graph theory mathematically formalizes such networks and provides the basic tools for analyzing them. Already in 1736 Leonhard Euler used graphs to prove that it is impossible to walk through the city of Königsberg and cross every of its seven bridges exactly once. Like in many fields of mathematics proving the existence of a structure can be done by providing an example, but disproving the existence needs a formalism and often heavy theoretical work. Formally a graph consists of entities that we call vertices and edges that connect two vertices, see Definition 2.1.1 in Chapter 2 for more details.

Many unsolved problems in the field of graph theory are concerned with the existence or non-existence of graphs with certain properties. To prove the non-existence one can use graph theoretical results to derive a contradiction from the assumption of the existence of such graphs. On the other hand, to prove the existence one can construct an example graph with the wanted properties. For some problems it may be possible to use well-known graphs as basic building blocks to construct more complex graphs by hand. Another approach is to use computers to search for example graphs. One advantage of using computers is that one can systematically search through all finitely many graphs up to a certain size. This can be used to either find an example graph or show the non-existence for all graphs with up to a certain number of vertices. Being able to verify a conjecture for small graphs helps to strengthen the belief that a conjecture might be true but, unfortunately, cannot be used as a proof for the non-existence, since there are in general infinitely many graphs.

Searching for a graph with certain properties algorithmically can be decomposed into two stages. The first stage constructs suitable candidate graphs and the second stage checks

for a candidate graph if it satisfies the wanted properties. Depending on the approach one can focus more on the first stage or the second. In the extreme case of only focusing on the first stage we get an algorithm that only constructs candidate graphs that already satisfy all wanted properties. If we want to focus only on the second stage, we can use in the first stage a precomputed set of candidate graphs or a third party enumeration algorithm to obtain a well-known class of candidate graphs.

In the context of these two stages we distinguish two kinds of problems. The first problem searches for a given order $n \in \mathbb{N}$ for a graph with n vertices that satisfies all wanted properties. We call this kind of problems *existence problems*, since they ask for the existence of a graph. Such problems can be solved by incorporating both stages. On the other hand, if we only focus on the second stage we consider the following kind of problems: Given a graph G, does G satisfy all the wanted properties? We call such problems *validation problems*. Existence problems can be theoretically viewed as decision problems but also as search problems, see Section 2.2 for more details. Clearly validation problems can also be viewed as decision problems. Interestingly, many graph theoretic properties can be expressed as searching for a substructure in a given graph and therefore, also most of the validation problems can also be interpreted as search problems.

In this thesis we are considering different existence and validation problems corresponding to open problems in graph theory. All problems that we are considering can also be interpreted as optimization problems. We use this fact and take advantage of the rich tool set available to solve combinatorial optimization problems (COPs). Therefore, most of the solving methods presented in this thesis are based on combinatorial optimization approaches.

The idea of using computer algorithms for solving graph theoretic problems is not new. There are algorithms for finding graphs with special structures, for example planar graphs [12] or snarks [14], and for generating conjectures [16]. The approach for generating planar graphs is used in Chapter 3.

This thesis is split into three parts. In the first part we are developing algorithms to search for uniquely hamiltonian planar graphs with minimum degree three. These are graphs that can be drawn without crossings on the plane and contain exactly one hamiltonian cycle, which is a cycle containing all vertices of the graph, see Section 2.1 for formal definitions. Bondy and Jackson [10] conjectured that no such graphs exist, i.e., every uniquely hamiltonian planar graph must have vertices of degree two. In 2014 Fleischner constructed non-planar uniquely hamiltonian graphs with minimum degree four [27]. This result, which was for the graph theory community quite surprising, leads to the guess that also Bondy and Jackson's conjecture is wrong. This motivates our work on algorithms for finding uniquely hamiltonian planar graphs with minimum degree three to either disprove Bondy and Jackson's conjecture or to verify it for small graphs.

We consider two different kinds of approaches to search for such graphs. The first approach searches heuristically for uniquely hamiltonian graphs drawn on the plane minimizing the number of crossings and the number of degree two vertices. We further restrict the

problem to only allow crossings between edges that are not part of the hamiltonian cycle. This helps us to combinatorially define the drawing in the plane by fixing a hamiltonian cycle with n vertices and bipartitioning the other edges. One partition represents then all edges drawn in the interior of the cycle and the other partition the edges in the exterior of the cycle. The number of crossings can then be easily calculated given the two partitions. Using this solution representation we develop a metaheuristic based on general variable neighborhood search (GVNS). Each neighborhood adds edges, removes them, changes the bipartition, or applies a combination of those three. As an interesting subproblem, we consider finding an optimal bipartition of a given set of edges. We formalize this as a very large scale neighborhood structure and find the optimal solution using a branch-and-bound procedure.

With the heuristic approach we can hope to find a uniquely hamiltonian planar graph with minimum degree three, but we are not able to verify Bondy and Jackson's conjecture for small graphs. To that end we also develop exact approaches to search for uniquely hamiltonian planar graphs with minimum degree three. We transform the problem of searching for a uniquely hamiltonian graph into a problem of searching for a graph containing a so called stable fixed edge cycle (SFE-cycle), which is a cycle C and an edge e such that no other cycle exists that contains e and all vertices of C. We then develop two exact approaches to check for a graph if it contains an SFE-cycle. One approach is based on integer linear program (ILP) and the other based on a cycle enumeration scheme together with an appropriate data structure for storing cycles. To verify Bondy and Jackson's conjecture for small graphs we study properties of a minimum counter example, i.e., a minimum graph according to the number of vertices and edges that is uniquely hamiltonian, planar, and has minimum degree three. Finally, we combine everything together with a planar graph generation program and are able to verify Bondy and Jackson's conjecture for graphs with up to 25 vertices.

In the second part of this thesis we develop an algorithm to construct smooth graphs with small independence numbers. Smooth graphs are a special subclass of 4-regular hamiltonian graphs, see Definition 4.2.1 in Section 4.2 for a formal definition. The independence number is the size of the largest set of vertices that are all pairwise not adjacent. We propose an algorithm to construct smooth graphs minimizing the independence number that is based on branch-and-bound. Starting with the hamiltonian cycle we iteratively partition the vertices such that each partition forms an additional cycle at the end. To prune partially constructed graphs we compute a lower bound for the independence number of any extension of the partial graph. The computation of the lower bounds is based on graph-theoretic results and uses a fast heuristic to compute an approximation for the independence number of the partial graph. The algorithm is then used to test the conjecture that every smooth graph with $n > 11$ vertices has independence number greater than or equal to $2n/7$, which we could verify for all graphs with $11 < n \leq 24$ vertices. Furthermore, we apply restrictions to the smooth graphs to check lower bounds on the independence number for some subclasses.

In the last part of this thesis we develop algorithmic approaches for checking if a

transitioned graph contains a sup-$(H, \mathcal{S})$-transition-minor for some other transitioned graph $(H, \mathcal{S})$. This terminology was introduced by Fleischner et al. [33] for $H = K_5$ and is an extension of the minor relation to transitioned graphs. In their work they proved that a transitioned 2-connected eulerian graph that is sup-undecomposable K_5 (SUD-K_5)-minor-free has a compatible circuit decomposition (CCD). This result solves the CCD problem for this class of graphs. The CCD problem is strongly related to the famous cycle double cover (CDC) conjecture. To verify the existence of SUD-K_5-minors we generalize the problem to sup-$(H, \mathcal{S})$-minors for any transitioned graph $(H, \mathcal{S})$ and develop a mathematical model that can be used to decide whether such a transition-minor exists or not. Because of the complex nature of the definition of a sup-$(H, \mathcal{S})$-minor this model is non-trivial. We use the model to develop an approach based on a mixed integer linear program (MILP) and a boolean satisfiability problem (SAT). To test the approaches we use instances based on the class of snarks, which is motivated by the CDC conjecture.

1.1 Structure of the Thesis

In Chapter 2 the graph theoretic definitions and notations on which this thesis is based are presented. Furthermore, we review some fundamental problem families and some basic definitions of complexity theory. Moreover, we introduce some selected solving methods for COPs. We focus only on the methods that are also used in this work, i.e. certain exact approaches but also (meta)-heuristics.

Chapter 3 is dealing with algorithms to search for uniquely hamiltonian planar graphs with minimum degree three. In this context we consider a GVNS-based metaheuristic to find uniquely hamiltonian graphs embedded in the plane minimizing the number of crossings and the number of degree two vertices. This algorithm and the computational results have been published in:

> B. Klocker, H. Fleischner, and G. R. Raidl. Finding uniquely hamiltonian graphs of minimum degree three with small crossing numbers. In *Hybrid Metaheuristics: 10th International Workshop, HM 2016*, volume 9668 of *LNCS*, pages 1–16. Springer, 2016.

Moreover, a presentation on this work has been given:

> B. Klocker. Heuristic approaches for finding uniquely hamiltonian graphs of minimum degree three with small crossing numbers. Austrian Workshop on Metaheuristics 11, Graz, Austria, 2016.

Furthermore, we consider two exact approaches to search for planar graphs with minimum degree three that contain SFE-cycles, which could be used to construct uniquely hamiltonian planar graphs with minimum degree three. This work has been submitted to:

B. Klocker, H. Fleischner, and G. R. Raidl. A lower bound for the smallest uniquely hamiltonian planar graph with minimum degree three. Technical Report AC-TR-19-007, Algorithms and Complexity Group, TU Wien, 2019. Submitted to Applied Mathematics and Computation.

Moreover, a presentation of preliminary results has been given:

B. Klocker. Searching uniquely hamiltonian planar graphs with minimum degree three. Graph Theory Workshop on How to Span a Graph, Bucharest, Romania, 2018.

In Chapter 4 we present a branch-and-bound approach for finding smooth graphs with small independence numbers. This work has been published in:

B. Klocker, H. Fleischner, and G. R. Raidl. Finding smooth graphs with small independence numbers. In *MOD 2017: Machine Learning, Optimization, and Big Data – Third International Conference*, volume 10710 of *LNCS*, pages 527–539. Springer, 2018.

Chapter 5 is dedicated to transition minors. We develop a mathematical model to check for a transitioned graph if it contains a sup-transition minor of some other transitioned graph. Based on the model we propose and compare a MILP approach and a SAT approach. The mathematical model together with the MILP approach has been submitted to:

B. Klocker, H. Fleischner, and G. R. Raidl. A Model for Finding Transition-Minors. Technical Report AC-TR-18-009, Algorithms and Complexity Group, TU Wien, 2018. Submitted to Discrete Applied Mathematics.

Furthermore, the SAT approach together with symmetry breaking improvements for the MILP and the SAT approaches have been published in:

B. Klocker, H. Fleischner, and G. Raidl. A SAT Approach for Finding Sup-Transition-Minors. In *Learning and Intelligent Optimization*, LNCS. Springer, 2019. To appear.

Some computational results concerning perfect pseudo-matchings (PPMs) in snarks have also been submitted to Ars Combinatoria:

H. Fleischner, B. Bagheri Gh., and B. Klocker. Perfect pseudo-matchings in cubic graphs. Technical report, Algorithms and Complexity Group, TU Wien, 2019. Submitted to ARS COMBINATORIA.

Finally, we also want to mention the following publication, which resulted from an industry collaboration:

> B. Klocker and G. R. Raidl. Solving a weighted set covering problem for improving algorithms for cutting stock problems with setup costs by solution merging. In *Computer Aided Systems Theory – EUROCAST 2017, Part I*, volume 10671 of *LNCS*, pages 355–363, Gran Canaria, Spain, 2018. Springer.

Its topic is not related to graph theory or graph construction, but to combinatorial optimization in general. Because of the only loose relation to the core topic of this thesis, we decided to not include further details in this thesis.

Finally, we conclude this thesis in Chapter 6 and propose some ideas for future research.

CHAPTER 2

Methodology

In this chapter we first present the graph theoretic basics on which this thesis builds. Then we cover three types of problems: decision problems, search problems, and optimization problems with their subclass of combinatorial optimization problems (COPs). As we will see, we can interpret the former two as special cases of optimization problems. Furthermore, all problems we are considering in this thesis can be interpreted as COPs. To solve such problems we consider different types of solution approaches. In this section we only focus on solution approaches relevant for the work presented in this thesis. To that end we briefly review different exact solution approaches, such as complete enumeration, branch-and-bound, (mixed) integer programming, and boolean satisfiability problem (SAT) solving. Furthermore, we present basic heuristic techniques such as greedy construction heuristics and local search as well as metaheuristics such as variable neighborhood search and very large scale neighborhood search (VLSN).

2.1 Graph Theoretic Basics

In this section we cover graph theoretic concepts used in this work. We start with basic definitions and notations and then cover advanced topics, which we need later on. Our notation is based on [11], [22], and [80].

2.1.1 Basic Definitions and Notations

First we introduce the most general term of graph we are using in this work, the undirected multigraphs, which we simply call graphs.

Definition 2.1.1. We denote by $\mathcal{P}_2(X)$ the set of all unordered pairs of elements in a set X, i.e. $\mathcal{P}_2(X) = \{S \subseteq X \mid |S| = 2\}$. A *graph* is a triple $G = (V, E, \psi)$, where V is the finite vertex set, E the finite edge set and $\psi : E \to \mathcal{P}_2(V)$ a function that maps each edge e to its two incident vertices, which we also call the *end vertices* of e. We

abbreviate $e = uv$ if $e \in E$ is incident to $u, v \in V$, i.e. $\psi(e) = \{u, v\}$. Note that this is just a notation and cannot be interpreted as an equality relation in the general case. If $e = uv$ and $f = uv$ for $e, f \in E$ with $e \neq f$ we call e, f *multiple edges*. To distinguish between different graphs we also use the notation $G = (V_G, E_G, \psi_G)$ or we simply write $V(G) := V_G$ and $E(G) := E_G$. The number of vertices $|V(G)|$ of a graph G is also called the *order* of G.

Note that in contrast to the notation of [11] and [22] we do not allow loops in our multigraphs. This definition often varies between authors. Another work that does not allow loops in multigraphs is [18].

Definition 2.1.2. Let $G = (V, E, \psi)$ be a graph. We say a vertex $v \in V$ and an edge $e \in E$ are *incident* if $e = vw$ for some $w \in V$. Furthermore, a vertex $v \in V$ is *adjacent* to a vertex $w \in V$ if there is an edge $e = vw$. Similarly, an edge $e_1 \in E$ is *incident* to an edge $e_2 \in E$ if both are incident to a common vertex $v \in V$. For a vertex $v \in V$ we write $E(v) = \{e \in E \mid v \in \psi(e)\}$ for the set of all edges incident to v and $N(v) = \{v' \in V \mid \exists e \in E : e = vv'\}$ for the set of all vertices adjacent to v, which we call the neighbors of v. The *degree* $d(v)$ of a vertex $v \in V$ is defined by $d(v) := |E(v)|$. The *minimum degree* of a graph G is denoted by $\delta(G) := \min_{v \in V} d(v)$. Analogously, the *maximum degree* of a graph G is defined by $\Delta(G) := \max_{v \in V} d(v)$. A graph is called *$k$-regular* if $\delta(G) = \Delta(G) = k$. A 3-regular graph is also called a *cubic graph*.

Parts of this work focus only on simple undirected graphs.

Definition 2.1.3. A graph is called *simple* if it has no multiple edges. For simple graphs $G = (V, E, \psi)$ we can interpret $E \subseteq \mathcal{P}_2(V)$, which defines the function ψ implicitly. We simply write $G = (V, E)$, in this case the notation $e = uv$ can now be read as an equality relation if we interpret uv as $\{u, v\}$.

The following graphs are important examples of simple graphs.

Example 2.1.1. The *complete graph* $K_n = (V(K_n), E(K_n))$ over n vertices is defined by $V(K_n) = \{1, \dots, n\}$ and $E(K_n) = \{e \subseteq V \mid |e| = 2\}$. In a complete graph there is exactly one edge between any pair of vertices. Therefore, the complete graph K_n is a simple graph that maximizes the number of edges over n vertices.

Other examples are the *complete bipartite graphs* $K_{n,m}$. A *bipartite graph* is a graph whose vertices can be partitioned into two sets such that the vertices in each set are all pairwise not adjacent, see also the definition of an independent set in Definition 2.1.13. The complete bipartite graph $K_{n,m} = (V(K_{n,m}), E(K_{n,m}))$ is now defined by

$$V(K_{n,m}) = V_1 \cup V_2 = \{1, \dots, n\} \cup \{n+1, \dots, n+m\},$$

$$E(K_{n,m}) = \{uv \mid u \in V_1, v \in V_2\}.$$

In this sense the complete bipartite graph $K_{n,m}$ is a simple graph that maximizes the number of edges in a bipartite graph if one vertex partition has n vertices and the other m.

Basic operations on graphs are vertex and edge deletions.

Definition 2.1.4. Let $G = (V, E, \psi)$ be a graph. For a subset $V_0 \subseteq V$ we denote by $G - V_0$ the graph after removing all vertices V_0 and all edges that are incident to at least one of the vertices in V_0 from G. For singletons we just write $G - v$ instead of $G - \{v\}$. Furthermore, if we want to focus on the remaining vertices $Y := V \setminus V_0$ and not on the deleted vertices we call the subgraph $G - V_0$ the *subgraph of G induced by Y* and denote it by $G[Y]$.

The graph obtained after removing a set of edges $E_0 \subseteq E$ from G is denoted by $G - E_0$ or in the case of a singleton $G - e$ instead of $G - \{e\}$.

Another basic relation is the notion of subgraph.

Definition 2.1.5. Let G be a graph. Any graph obtained after deleting edges and vertices from G is called a *subgraph* of G. Formally a subgraph can be described by $(G - E_0) - V_0$ for some $V_0 \subseteq V$ and $E_0 \subseteq E$. If we want to remove a whole subgraph H from G we also write $G - H$ instead of $G - V(H)$.

We continue with basic objects inside a graph.

Definition 2.1.6. Let $G = (V, E, \psi)$ be a graph. A *walk* in G is a sequence $W := v_0, e_1, v_1, \dots, v_{\ell-1}, e_\ell, v_\ell$ of vertices $v_i \in V$ and edges $e_i \in E$ such that $e_i = v_{i-1}v_i$ for all $i \in \{1, \dots, \ell\}$. The integer ℓ is then called the *length* of W. A walk W is called *closed* if $v_\ell = v_0$. Furthermore, a walk W is called a *trail* if all edges are different, i.e. $e_i \neq e_j$ for all $i \neq j$. A closed trail is called a *circuit*.

A *path* is a simple graph P whose vertices can be arranged in a linear sequence such that two vertices are adjacent if and only if they are consecutive in the sequence. The two vertices of degree one in a path, i.e. the vertex at the beginning and the end of the linear sequence, are also called *end vertices*. Furthermore, a *cycle* is a simple graph C whose vertices can be arranged in a cyclic sequence such that two vertices are adjacent if and only if they are consecutive in the cyclic sequence.

Note that some authors define paths and cycles in terms of trails where all vertices are different, except the first and the last in the case of a cycle. Those two definitions are not equivalent since one path interpreted as a subgraph may be represented as two different trails, depending on at which vertex the trail starts. One cycle of length ℓ may be represented as 2ℓ different trails depending on where the trail starts and in which direction it goes. This differentiation is important if we want to count paths or cycles in which case we always use our definition in the sense of subgraphs. To represent a path

or a cycle we can use the notation of any trail corresponding to them, or in the case of simple graphs we can also just use a sequence of vertices.

With the help of paths we can now define connected graphs.

Definition 2.1.7. A graph $G = (V, E, \psi)$ is called *connected* if any two vertices $v_1, v_2 \in V$ can be connected by a path in G, i.e., there exists a path with v_1 and v_2 as end vertices. The maximal connected subgraphs of a graph G are called the connected components of G.

Furthermore, we can define edge and vertex connectivity.

Definition 2.1.8. Let $G = (V, E, \psi)$ be a graph. Let V be partitioned into two non-empty sets X and $Y = V \setminus X$, we denote by $E[X, Y] \subseteq E$ all edges with one end vertex in X and the other in Y. Such a set $E[X, Y]$ is then called an *edge cut* of G. If $|E[X, Y]| = 1$ then this edge $e \in E[X, Y]$ is called a *bridge* of G. A graph without a bridge is called *bridgeless*. Note that a bridgeless graph may not be connected. The graph G is called *k-edge-connected* if there does not exist any edge-cut with less than k edges. Furthermore, the *edge connectivity* $\lambda(G)$ of G if $|V(G)| \geq 2$ is defined as the maximum k for which G is still k-edge-connected.

A set of vertices $V_0 \subseteq V$ is called a *vertex cut*, if $G - V_0$ is disconnected. If v is a vertex cut, then we call v a *cut vertex* of G, which is sometimes also called *articulation point* of G. The *vertex-connectivity* or just *connectivity* $\kappa(G)$ of G is the minimum size of a vertex set $S \subseteq V$ such that S is a vertex cut or $G - S$ contains only one vertex. The graph G is called *k-vertex-connected* or just *k-connected* if $\kappa(G) \geq k$.

Going into more detail we can distinguish different types of edge cuts.

Definition 2.1.9. An edge cut $E[X, Y]$ is called *essential* if the induced subgraphs $G[X]$ and $G[Y]$ both contain at least one edge. If $G[X]$ and $G[Y]$ both contain a cycle we call the edge cut $E[X, Y]$ a *cyclical edge cut*.

Finally, a graph G is called *essentially k-edge connected* if it has no essential edge cut with fewer than k edges. Analogously, it is called *cyclically k-edge connected* if it has no cyclical edge cut with fewer than k edges.

Clearly every cyclical edge cut is also an essential edge cut and therefore we get the following lemma.

Lemma 2.1.1. *An essentially k-edge connected graph is also cyclically k-edge connected.*

We also get the other direction if the graph has a high enough minimum degree as Fleischner mentioned in [28].

Lemma 2.1.2. *A graph G with $\lambda(G) \geq \frac{k}{2} + 1$ is essentially k-edge connected if and only if it is cyclically k-edge connected.*

Proof. We only need to show that every essential edge cut $E[X, Y]$ with $|E[X, Y]| < k$ is also a cyclical edge cut. Assume that such a cut is not a cyclical edge cut. That means w.l.o.g that $G[X]$ does not contain any cycles and therefore $E(G[X]) \leq V(G[X]) - 1 = |X| - 1$. Summing up over the degrees of the vertices in X we get

$$|X| \left(\frac{k}{2} + 1\right) \leq \sum_{v \in X} d(v) = |E[X, Y]| + 2|E(G[X])| < k + 2|X| - 2.$$

Since X contains an edge we know $|X| \geq 2$ and therefore we get a contradiction by

$$k - 2 > |X| \left(\frac{k}{2} - 1\right) \geq k - 2.$$

□

In this work we often focus on special types of cycles, such as dominating or hamiltonian cycles.

Definition 2.1.10. Let $G = (V, E, \psi)$ be a graph. A cycle C in G is called *dominating* if for each edge $e = uv \in E$ either u or v is in $V(C)$. A subgraph H of G is called *spanning* if $V(H) = V(G)$. We also call a spanning subgraph a *factor* of G. A *k-factor* of G is a k-regular factor of G. Furthermore, a cycle C in G is called *hamiltonian* if each vertex of G is visited in C, i.e. C is a spanning subgraph of G. A graph is called *hamiltonian* if it contains a hamiltonian cycle.

In the way that hamiltonian cycles visit all vertices eulerian tours visit all edges.

Definition 2.1.11. Let $G = (V, E, \psi)$ be a connected graph. A *tour* of G is a closed walk in G that visits every edge G at least once. An *eulerian tour* is a tour of G that visits every edge of G exactly once, i.e. it is a tour and a trail. A graph that contains an eulerian tour is called an *eulerian graph*.

Eulerian graphs can be easily characterized by the following notion.

Definition 2.1.12. A graph G is called *even* if all vertices have even degrees.

The connection between even graphs and eulerian graphs is stated in the following theorem, see for example [11].

Theorem 2.1.3. *A graph is eulerian if and only if it is connected and even.*

Some authors such as Fleischner allow eulerian graphs to be disconnected and define the eulerian graphs to be what we call even graphs, but this theorem states that for connected graphs those two definitions coincide.

Another property of graphs we are using is the independence number.

Definition 2.1.13. Let $G = (V, E, \psi)$ be a graph. An *independent set* of G is a vertex set $I \subseteq V$ such that there is no edge $e = ij$ for $i, j \in I$ in G, i.e. all vertices of I are pairwise not adjacent. The *independence number* $\alpha(G)$ of a graph G is the size of the largest independent set of G.

Related to independent sets is the notion of graph colorability.

Definition 2.1.14. A *k-vertex-coloring* or alternatively a *k-coloring* of a graph G is a function $c : V(G) \to \{1, \dots, k\}$ such that for each edge $e = vw \in E(G)$ the two end vertices have different colors, i.e. $c(v) \neq c(w)$. In this context the colors are the integers $\{1, \dots, k\}$ but could be replaced by any set of colors with cardinality k.

A graph G is *k-colorable* if there exists a k-coloring for G. Furthermore, $\chi(G)$ denotes the *chromatic number* of G, which is the smallest $k \in \mathbb{N}$ for which G is k-colorable.

We can also color edges, which leads to edge-colorings.

Definition 2.1.15. A *k-edge-coloring* of a graph G is a function $c : E(G) \to \{1, \dots, k\}$ such that for each edge $e = vw \in E(G)$ all incident edges have different colors, i.e.

$$c(e) \neq c(e') \quad \forall e = vw \in E(G), \forall e' \in (E(v) \cup E(w)) \setminus \{e\}\,.$$

A graph G is *k-edge-colorable* if there exists a k-edge-coloring for G. Furthermore, $\chi'(G)$ denotes the *edge chromatic number* of G, which is the smallest $k \in \mathbb{N}$ for which G is k-edge-colorable. The edge chromatic number of G is also called the *chromatic index* of G.

2.1.2 Graph Isomorphisms

Most books define homomorphisms only on simple graphs, but we are using a more general definition on graphs. A definition of isomorphisms on graphs is given in [11], which is slightly different from our definition. Our definition does not consider an edge mapping as part of the isomorphism but ensures that such a mapping exist. This implies that our definition coincides with the classical definitions for the case of simple graphs as for example in [22].

Definition 2.1.16. Let $G = (V, E, \psi)$ and $G' = (V', E', \psi')$ be two graphs. A *homomorphism* between G and G' is a function $f : V \to V'$ such that for each pair of vertices $u, v \in V$ it holds

$$|\{e \in E \mid e = uv\}| = |\{e' \in E' \mid e' = f(u)f(v)\}|,$$

i.e. the function preserves the number of edges between mapped vertices.

We use now the definition of homomorphisms to define ismorphisms and automorphisms on graphs.

Definition 2.1.17. Let $G = (V, E, \psi)$ and $G' = (V', E', \psi')$ be two graphs. A homomorphism f between G and G' is a *isomorphism* if f is bijective and f^{-1} is also a homomorphism between G' and G. Two graphs are *isomorphic* if there exists an isomorphism between them. An isomorphism between a graph and itself is called an *automorphism*. The set of automorphisms on a graph G form with the composition operator a group, which we call $\mathrm{Aut}(G)$, the automorphism group of G.

2.1.3 Perfect Pseudo-Matchings

We introduce here a generalization of perfect matchings. Definitions of perfect matchings can be found in many standard books such as [80], but the terminology of perfect pseudo-matchings (PPMs) is uncommon outside the research areas of this work.

Definition 2.1.18. Let $G = (V, E, \psi)$ be a graph. A *pseudo-matching* M is a subgraph of G in which all connected components are either isomorphic to the K_2 or $K_{1,3}$. Connected components of a pseudo-matching that are isomorphic to the $K_{1,3}$ are called *claws*. A *perfect pseudo-matching (PPM)* is a pseudo-matching that is also a spanning subgraph of G, i.e. contains all vertices of G. A *matching* M is a pseudo-matching where all connected components are isomorphic to the K_2, i.e. a 1-regular subgraph of G. A matching can equivalently be interpreted as a set of edges that are all pairwise not incident, but we are using the definition as a subgraph.

In the context of 3-regular graphs PPMs are strongly related to dominating cycles in the same way as perfect matchings are related to hamiltonian cycles: The edge complement of a dominating/hamiltonian cycle induces a PPM/perfect matching. Note that in the other direction it is not true since the edge complement of a PPM/perfect matching may not even induce a cycle, in general it just induces a 2-regular graph, i.e. a collection of cycles.

2.1.4 Crossing Number and Planar Graphs

To formally define crossing numbers we follow the definitions of [80].

Definition 2.1.19. Let $G = (V, E, \psi)$ be a graph. A *curve* is the image of a continuous function from $[0, 1]$ to $\mathbb{R}^2$. A *drawing* of G in the plane is a function f that maps each vertex $v \in V$ to a point in $\mathbb{R}^2$ and each edge $e = uv \in E$ to a curve in $\mathbb{R}^2$ that connects the point $f(u)$ with $f(v)$. W.l.o.g. we can restrict our drawings such that no three curves have a common point, no vertex image $f(v)$ is part of a curve $f(e)$ if v is not an endpoint of e and two different curves have at most one common point, i.e. $|f(e_1) \cap f(e_2)| \leq 1$ if $e_1 \neq e_2$. This can always be achieved by slightly moving the curves. A common point of two curves, which is not a common endpoint of the curves is called a *crossing*.

We can define a graph structure on the images of a drawing. The edge set is the set of curves, the vertex set the set of points $f(v)$ for $v \in V$. For an edge $e = uv$ the curve $f(e)$ has endpoints $f(u)$ and $f(v)$. We call this graph the associated graph of the drawing f.

Note that in [80] they only allow polygonal curves in a drawing, but we omit that for simplicity, since the definitions are equivalent. We can now use the definition of a drawing to define the crossing number of a graph and planar graphs.

Definition 2.1.20. The *crossing number* $\nu(G)$ of a graph G is the minimum number of crossings of any drawing of G in the plane.

A graph G is called *planar* if it has crossing number $\nu(G) = 0$. In this case a drawing with no crossings is called a *planar embedding* and the associated graph structure of the drawing a *plane graph*.

Each drawing f of a planar graph G with no crossings defines formally different plane graphs, but all are isomorphic to G.

Definition 2.1.21. A *region* is an open subset of $\mathbb{R}^2$ where any two points can be connected by a curve. For a plane graph $G = (V, E, \psi)$ the maximal regions of $\mathbb{R}^2 \setminus (V \cup \bigcup_{e \in E} e)$ are called the *faces* $F(G)$ of G. Since all edges $e \in E$ are compact sets in $\mathbb{R}^2$ their union is bounded and therefore there is exactly one unbounded face, which is called the *outer face*. All other faces are called *inner faces*.

With the definition of faces we can define the dual graph of a plane graph.

Definition 2.1.22. Let $G = (V, E, \psi)$ be a plane graph with $\delta(G) \geq 2$. The *dual graph* of G is a graph $G^* = (V^*, E^*, \psi^*)$ whose vertices V^* correspond to the faces of G, i.e. each vertex is a point inside a face of G. For an edge $e \in E$ that borders the faces F_1 and F_2 the dual graph contains an edge e^* that connects the vertex corresponding to F_1 with the vertex corresponding to F_2. The curve e^* can be drawn in such a way that it crosses the edge e exactly once and no other edges of G or of G^*, see Figure 2.1 for two examples.

Note that we restricted the definition of dual graphs to plane graphs with $\delta(G) \geq 2$. This restriction ensures that every plane graph has a dual that has no self-loops and therefore is again a graph. If we would allow self loops in graphs we could lift this restriction, as it is done in most graph theory books, such as [80].

The definition of the dual graph is not unique, since we have some freedom where to place vertices and how to draw the curves, but regardless how we draw them the resulting graph structure is always isomorphic and therefore we simply talk of the dual graph.

For a connected plane graph with $\delta(G) \geq 2$ one can verify that $G^{**} = G$, i.e. G is the dual graph of its dual graph G^*. A 2-edge-cut in G corresponds to two parallel edges in G^* and vice versa. Therefore, the dual graph G^* of G is simple if and only if G is 3-edge-connected.

Note that two isomorphic plane graphs may have non-isomorphic dual graphs. Furthermore, a planar graph G may have different plane drawings f and f' such that the two

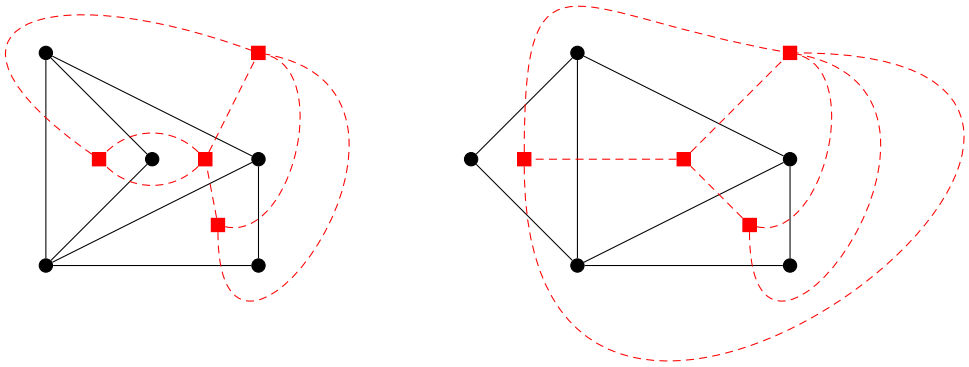


Figure 2.1: Two plane drawings of a planar graph whose associated plane graphs have non-isomorphic dual graphs. The plane graphs are drawn with black circled vertices and black solid edges and the associated dual graphs are drawn with red squared vertices and red dashed edges.

associated plane graphs have non-isomorphic duals. An example of such a graph G is given in Figure 2.1 and the fact that the two dual graphs are non-isomorphic can easily be seen by the sets of vertex degrees $\{3,3,4,4\} \neq \{3,3,3,5\}$. If G is simple and 3-connected, a theorem by Whitney shows that this cannot happen [11].

Theorem 2.1.4. *Let G be a simple 3-connected planar graph, then the duals of the associated graphs of all planar embeddings of G are all isomorphic.*

With that theorem we can define *the* dual graph of a simple 3-connected planar graph as the dual graph of the associated graph of any planar embedding of G. The proof of the following theorem is left as an exercise to the reader.

Theorem 2.1.5. *Let G be a simple 3-connected planar graph, then the dual graph of G is also 3-connected.*

A classical result for planar graphs is the following, see for example [80].

Theorem 2.1.6 (Euler's Formula)**.** *For a connected plane graph $G = (V, E, \psi)$ with $v = |V|$, $e = |E|$, and f the number of faces of G, it holds*

$$v - e + f = 2.$$

Note that Euler's Formula shows that the number of faces of a plane graph depend only on the number of vertices and edges of the graph. Therefore, the plane graph associated with every planar embedding must have the same number of faces. We can now define the *number of faces* $|F(G)|$ of a planar graph G to be the number of faces of any plane graph associated with any planar embedding of G. In this case Euler's Formula is also valid for planar graphs.

2.2 Decision Problems, Search Problems, and Combinatorial Optimization Problems

In this work we are generally considering two kinds of problems. Problems of the form "Does there exist a graph G with properties $\phi(G)$?", which we call existence problems, and problems of the form "Given a graph G, does G have properties $\phi(G)$?", which we call validation problems. Existence problems may be impossible to solve by a computer if the answer is "No". Therefore, we replace them by a problem of the form "Given an integer $n \in \mathbb{N}$, does there exist a graph G with properties $\phi(G)$ with n vertices?". Considering this replaced form both kinds of problems are formally decision problems. We formally define a decision problem similarly to Korte and Vygen [58].

Definition 2.2.1. A *decision problem* is a pair (X, Y) where X is the set of inputs and $Y \subseteq X$ the set of inputs for which the answer to the decision problem is "Yes".

Note that Korte and Vygen additionally restricted the input set X to be a language decidable in polynomial time. This would formally restrict our inputs to being strings, but for us inputs are most of the time a set of graphs or integers. We only consider decision problems for which there exists a string encoding of the set X such that the encoded elements correspond to a language decidable in polynomial time. For a definition of a language decidable in polynomial time we refer the interested reader to [58].

Example 2.2.1. As mentioned before the two kinds of problems we are considering are both decision problems. An existence problem can be translated to a decision problem (X, Y) with

$$X \subseteq \mathbb{N}, Y := \{n \in X \mid \exists \text{ graph } G \text{ satisfying } \phi(G) \text{ with } n \text{ vertices}\}.$$

Furthermore, a validation problem can be translated to a decision problem (X, Y) with X being an infinite set of graphs and $Y := \{G \in X \mid \phi(G)\}$.

Note that in order to check if there exists a graph with properties $\phi(G)$ one approach is to enumerate candidate graphs and check for all of them if they satisfy $\phi(G)$. In this sense validation problems often arise as subproblems of existence problems. This search for a graph $\phi(G)$ can also be viewed as a form of a so called search problem. The difference to the interpretation of a decision problem is that in the case of a search problem we do not just get a "Yes" or "No" but in the case of a "Yes" we get a witness, in our case a graph G satisfying $\phi(G)$.

But not only existence problems can be viewed as search problems, also many validation problems, in fact all validation problems we are considering, can be viewed as search problem. Most properties $\phi(G)$ in graph theory ask for the existence of an object, such as a cycle, or a set of vertices. In this case we can interpret the validation problem as searching for such an object. Let us first formally define what a search problem is similarly to Schrijver [70].

Definition 2.2.2. A *search problem* is a relation $R \subseteq X \times Y$ for some input set X and search space Y. The problem is now to find for an input $x \in X$ an element $y \in Y$ such that $(x, y) \in R$ or return "No" if no such $y \in Y$ exists.

Note that Schrijver defined search problems on some alphabet Σ and uses $X = Y = \Sigma^*$, but we omit this restriction for the same reasons as above in the case of the decision problem. Schrijver also mentions that a decision problem can be interpreted as a special case of a search problem $R \subseteq (X, Y)$ with a singleton $Y = \{\emptyset\}$ such that the yes instances are exactly the ones where $(x, \emptyset) \in R$.

Example 2.2.2. Our existence problems can be formulated as search problems with $X \subseteq \mathbb{N}$, Y a set of graphs, and

$$R := \{(n, G) \in (X, Y) \mid \phi(G) \wedge |V(G)| = n\}.$$

A representation of a validation problem as a search problem depends on the properties ϕ.

For example consider the *hamiltonian cycle problem*, which decides if a graph G contains a hamiltonian cycle. We can formulate this problem as a search problem with X a set of graphs, Y the set of all cycles in the graphs of X, and

$$R := \{(G, C) \in (X, Y) \mid C \text{ is a hamiltonian cycle of } G\}.$$

Our approaches for solving search problems and decision problems are based on combinatorial optimization approaches, which can be seen as a generalization of search problems. Given a search problem $R \subseteq X \times Y$ and a fixed instance $x \in X$. Then there is a set of feasible solutions $Y_x := \{y \in Y \mid (x, y) \in R\}$, which we are searching for. To find such solutions we can give all solutions $y \in Y$ an objective value in such a way that the objective values of solutions in Y_x have smaller objective values than all the other solutions and then try to minimize the objective value. This can be done by defining the objective function $f : Y \to \mathbb{R}$ by

$$f(y) = \begin{cases} 0, & \text{if } (x, y) \in R \\ 1, & \text{otherwise.} \end{cases}$$

But we can do even more, we can rate how infeasible a solution outside of Y_x is. In this way we can steer a possible search into a direction of solutions that are close to being in Y_x. This is basically the reason why search problems can often be solved effectively by a combinatorial optimization approach. Furthermore, we can, if we want, add different objectives to solutions in Y_x to find an even better solution as soon as we found one in Y_x. Let us formally define a COP as it is done in [39].

Definition 2.2.3. An *instance of an optimization problem* is a pair (F, c) with F being the set of feasible solutions and $c : F \to \mathbb{R}$ the cost function or objective function. A *optimization problem* is a set of instances and searches for a given instance (F, c) for a

globally optimal solution, that is a feasible solution $f \in F$ with $c(F) \leq c(y)$ for all $y \in F$. An optimization problem is a *combinatorial optimization problem (COP)* if the set of feasible solutions F is finite for each instance.

Note that as we defined optimization problems we only allow minimization problems. This can be extended by also allowing maximization problem in which case a globally optimal solution $f \in F$ is given if $c(F) \geq c(y)$ for all $y \in F$. We can transform a minimization problem into a maximization problem and vice versa by multiplying the objective function by -1. We also call an optimization problem that minimizes the objective a *minimization problem* and one that maximizes the objective a *maximization problem*.

Example 2.2.3. As an example we consider how to usefully turning a search problem into a COP. We consider the hamiltonian cycle problem as mentioned in Example 2.2.2. Let X be the set of possible input graphs. Then we defined for each graph $G \in X$ an instance of an optimization problem. The set of feasible solutions F is the set of all cycles C in G. Furthermore, the objective value of a cycle C is its length $f(C) = \ell(C)$ and we want to maximize the objective.

All in all we get a combinatorial maximization problem. If we found a globally optimal solution for an instance we can check if its length equals $|V(G)|$ in which case we found a hamiltonian cycle or else it is a "No" instance. Note that by trying to maximize the cycle length we also guide the search into a direction of long cycles and therefore possible hamiltonian cycles.

Another famous example of a COP, which can be seen as generalization of the hamiltonian cycle problem, is the following, see [58].

Problem 2.2.1 (traveling salesperson problem (TSP))**.** Given are a complete graph K_n together with edge costs $c_e \in \mathbb{R}_{\geq 0}$ for each $e \in E(K_n)$. Find a hamiltonian cycle C minimizing its total cost

$$c(C) := \sum_{e \in E(C)} c_e.$$

2.3 Computational Complexity

In this section we present the complexity class $\mathcal{NP}$, which is important in classifying the asymptotic complexity of many problems that we are considering. Before we can define $\mathcal{NP}$, we define $\mathcal{P}$ as in [58].

Definition 2.3.1. The set $\mathcal{P}$ consists of all decision problems for which there exist a polynomial-time algorithm.

Note that we did not define formally what a polynomial-time algorithm is. Since this definition requires heavy formalism based on Turing machines we refer the interested

reader to [58] or any other introductory book to complexity theory. To define $\mathcal{NP}$ we use a similar definition as in [58] noting again as for Definition 2.2.1 that we chose for simplicity to operate on arbitrary sets instead of languages.

Definition 2.3.2. A decision problem (X, Y) is in $\mathcal{NP}$ if there exists a polynomial p and a decision problem (X', Y') in $\mathcal{P}$ with $X' \subseteq X \times C$ for some set C, such that

- $\forall (x, c) \in X' : \text{length}(c) \leq p(\text{length}(x))$,
- $x \in Y \iff \exists c \in C : (x, c) \in Y'$.

By length we mean here the length of an encoding of the elements in X and in C. An element $c \in C$ with $(x, c) \in Y'$ is called a *certificate* for x. A polynomial-time algorithm for solving the problem (X', Y'), i.e. checking if $(x, c) \in Y'$, is called a *certificate-checking algorithm*.

It is easy to see that $\mathcal{P} \subseteq \mathcal{NP}$, but it is still an open problem if this subset relation is proper. This problem whether $\mathcal{P} = \mathcal{NP}$ is one of the most famous and important open problems in theoretical computer science.

Example 2.3.1. Let us consider the hamiltonian cycle problem as in Example 2.2.2 interpreted as a decision problem. We can use as set C the set of cycles in the graphs that we already used in the definition of the associated search problem. We define

$$X' := \{(G, C) \mid C \text{ is a cycle of } G\}, Y' := \{(G, C) \mid C \text{ is a hamiltonian cycle of } G\}.$$

Clearly checking for a given graph G and a cycle C, if C is a hamiltonian cycle of G can be done in polynomial time. Furthermore, a cycle of a graph G is a subgraph of G and therefore can be encoded in linear space of G. This shows that the hamiltonian cycle problem is in $\mathcal{NP}$.

Note that we used in Example 2.3.1 as possible certificates the same set as we defined our search space in the search problem variant. This can be done for any search problem $R \subseteq X \times Y$ if the size of any $y \in Y$ with $(x, y) \in R$ is polynomially bounded by the size of x and if checking if $(x, y) \in R$ can be done in polynomial time. All search problems satisfying these properties have corresponding decision problems in $\mathcal{NP}$.

The same argumentation works for COPs (F, c), if the size of any $f \in F$ is bounded polynomially by the size of the instance and if computing $c(f)$ can be done in polynomial time for any $f \in F$.

Note that the definition of $\mathcal{NP}$ is asymmetric in the sense of "Yes" and "No" instances. Therefore, one may consider the complement of an $\mathcal{NP}$ problem, which may not be in $\mathcal{NP}$ anymore.

Definition 2.3.3. Let $P = (X, Y)$ be a decision problem. The *complement* of P is the decision problem $(X, Y \setminus X)$, i.e. the "Yes" instances of the complement are exactly the "No" instances of the original problem and vice versa. The class *co-*$\mathcal{NP}$ consists of the complements of all problems in $\mathcal{NP}$.

Next we want to define $\mathcal{NP}$-complete problems, which are in some sense the hardest of all $\mathcal{NP}$ problems. Before we can define $\mathcal{NP}$-completeness, we need to define polynomial transformations.

Definition 2.3.4. Let $P_1 = (X_1, Y_1)$ and $P_2 = (X_2, Y_2)$ be two decision problems. A *polynomial transformation* from P_1 to P_2 is a function $f : X_1 \to X_2$ computable in polynomial time such that $f(x_1) \in Y_2$ if and only if $x_1 \in Y_1$.

Note that polynomial transformations are also called Karp reductions, see [70], or many-one reductions, see [20]. The idea of polynomial transformations is to express that problem P_1 is not harder than problem P_2 in the sense of a polynomial hierarchy. This implies especially that, if $P_2 \in \mathcal{P}$ and there is a polynomial transformation from P_1 to P_2 then P_1 must also be in $\mathcal{P}$, the same holds also for $\mathcal{NP}$. We can now define $\mathcal{NP}$-completeness.

Definition 2.3.5. A decision problem P is $\mathcal{NP}$*-complete*, if

- $P \in \mathcal{NP}$, and
- for all problems $P' \in \mathcal{NP}$ there exist a polynomial transformation from P' to P.

Note that formally search problems and optimization problems cannot be in $\mathcal{NP}$ or $\mathcal{NP}$-complete since they are no decision problems. To be able to measure the complexity of search and optimization problems we define being $\mathcal{NP}$-hard. To do that we need to generalize polynomial transformations to so called polynomial reductions, see [58].

Definition 2.3.6. Let P and P' be two decision, search, or optimization problems. A *polynomial reduction* from P' to P is an algorithm that uses an oracle for P to solve P' in polynomial time. An oracle is a constant time solver for the given problem P and is assumed to be given. If P is a decision problem (X, Y) then an oracle returns in constant time if a given instance $x \in X$ is in Y or not. Furthermore, if P is a search problem $R \subseteq X \times Y$ then an oracle returns for a given $x \in X$ in constant time either a $y \in Y$ with $(x, y) \in R$ or "No" if no such y exists. Finally, if P is an optimization problem, then an oracle returns for a given instance $I = (F, c)$ in constant time a globally optimal solution $f \in F$.

With that we can define what an $\mathcal{NP}$-hard problem is.

Definition 2.3.7. Let P be a decision, search, or optimization problem. Then, P is called $\mathcal{NP}$*-hard* if there exists a polynomial reduction from P' to P for all $P' \in \mathcal{NP}$.

Note that all $\mathcal{NP}$-complete problems are also $\mathcal{NP}$-hard but not the other way around. One common way of proving $\mathcal{NP}$-completeness for a problem P is to find a polynomial transformation from an already known $\mathcal{NP}$-complete problem Q to P. Similarly one can prove $\mathcal{NP}$-hardness for a problem P by finding a polynomial reduction from an already known $\mathcal{NP}$-hard problem Q to P.

Clearly this strategy only works as long as we already know one $\mathcal{NP}$-complete problem. The first problem that could be proven to be $\mathcal{NP}$-complete was the SAT, which we are presenting in the following, see also [58].

Definition 2.3.8. Let X be a set of boolean variables. A *truth assignment* for X is a function $T : X \to \{\texttt{TRUE}, \texttt{FALSE}\}$ that assigns each variable a truth value. The set of all *literals* over X is given by

$$L := X \cup \{\bar{x} \mid x \in X\}$$

where $\bar{x}$ represents the logical negation of the variable x. Therefore, a truth assignment T for X can be uniquely extended to L by defining $T(\bar{x}) = \texttt{TRUE}$ if and only if $T(x) = \texttt{FALSE}$. A *clause* over X is now represented by a set of literals over X and can be interpreted as a logical disjunction. Therefore, we say a clause is *satisfied* by a truth assignment T for X if at least one of the literals of the clause are assigned to TRUE. Finally, a family of clauses is *satisfiable* if there exists a truth assignment T for X that satisfies all clauses of the family.

A finite family of clauses over some set X can be interpreted and written as a boolean formula in conjunctive normal form (CNF). In this sense we call a finite family of clauses also CNF.

Example 2.3.2. Let $X = \{x_1, x_2, x_3\}$, then the family

$$\{\{x_1, \bar{x_2}\}, \{x_1, x_3\}, \{\bar{x_1}, \bar{x_2}, \bar{x_3}\}\}$$

can be written as

$$(x_1 \vee \neg x_2) \wedge (x_1 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3).$$

Furthermore, the truth assignment $T : x_1 \mapsto \texttt{TRUE}, x_2 \mapsto \texttt{TRUE}, x_3 \mapsto \texttt{FALSE}$ satisfies all three clauses. Therefore, the given family of clauses is satisfiable.

Problem 2.3.1 (boolean satisfiability problem (SAT))**.** Given a CNF Δ, is Δ satisfiable?

Cook [19] used in 1971 the expressiveness of boolean formulas in CNF to prove the following important theorem.

Theorem 2.3.1. *SAT is $\mathcal{NP}$-complete.*

2.3.1 Enumeration

The most basic approach for solving a COP given an instance (F, c) is enumerating all feasible solutions $f \in F$, computing for each of them $c(f)$, and remembering the solution with the smallest found cost. This kind of approach is sometimes called *enumeration*, *total enumeration*, or *exhaustive search*, see for example [58].

Example 2.3.3. As an example consider the TSP, see Problem 2.2.1. A simple enumeration approach for solving the TSP for a complete graph K_n with costs $(c_e)_{e \in E(K_n)}$ would be to enumerate all permutations π of $V(K_n) = \{1, \dots, n\}$. We can associate with each permutation a cycle in K_n that visits the vertices in the order $\pi(1), \pi(2), \dots, \pi(n)$. Then we compute of each such cycle the total costs and remember the cycle with the lowest costs.

Note that two permutations π_1 and π_2 lead to the same cycle in K_n if there is a $k \in \{1, \dots, n-1\}$ such that $\pi_1(i) = \pi_2(i+k)$ for all $i \in \{1, \dots n-k\}$ and $\pi_1(i) = \pi_2(i+k-n)$ for all $i \in \{n-k+1, \dots, n\}$. To avoid this duplication we can only consider permutations π with $\pi(1) = 1$. But there are still pairs of permutations π_1 and π_2 that lead to the same cycle. Namely, if $\pi_1(i) = \pi_2(n - i + 2)$ for $i \geq 2$ and $\pi_1(1) = \pi_2(1) = 1$, which corresponds to visiting the same cycle in opposite directions. One way to avoid also those duplicates is to consider only permutations π with $\pi(1) = 1$ and $\pi(2) < \pi(n)$.

Techniques of avoiding to generate the same feasible solution multiple times, such as the one explained in Example 2.3.3, are called *symmetry breaking*. They are important for well performing enumeration approaches but are also used in many other approaches.

2.3.2 Branch-and-Bound

To avoid enumerating all feasible solutions of the search space one technique is the *branch-and-bound* approach. The main idea is to use a divide and conquer approach to iteratively split the problem into subproblems building up a search tree. At each vertex of the search tree a lower bound for the best possible cost that can be achieved in this subproblem is calculated. If this cost is larger than the cost of the currently best known solution to the original problem, then we can drop this subproblem, since it cannot contain a global optimum.

Let for the following (F, c) be an instance of an optimization problem we want to solve. To present an abstract branch-and-bound approach as it is done in [58] we represent subproblems formally as subsets of the search space F, i.e. sets of feasible solutions. In a concrete implementation subproblems are not represented as sets of feasible solutions, since this would require enumerating all feasible solutions. Often subproblems are represented by partial solutions and correspond to the set of all feasible solutions that can be constructed by starting from this partial solution. We require now two problem specific components.

- A method *branch* that formally partitions a subproblem $F_0 \subseteq F$ into at least two non-empty disjoint subproblems $F_1, \ldots, F_k$ with $\bigcup_{i=1}^{k} F_i = F_0$.
- A method *bound* that computes a lower bound $L(F_0)$ for the costs of all solutions in a given subproblem $F_0 \subseteq F$, i.e. it must hold $L(F_0) \leq c(f)$ for all $f \in F_0$.

For a given *branch* and a *bound* method the branch-and-bound algorithm is given in Algorithm 2.1.

Algorithm 2.1: Branch-and-Bound

Input: An instance (F, c) of a COP
Output: A globally optimal solution $f^* \in F$

Initialize the tree $T := (\{F\}, \emptyset)$ to the tree with only one vertex F;
Mark the vertex F as active;
Set the upper bound $U := \infty$ or apply a heuristic to get a valid upper bound U;
while *There exists an active vertex* $F_0 \in V(T)$ **do**
    Mark F_0 as non-active;
    Apply *branch* on F_0 to get a partition $F_0 = F_1 \dot{\cup} \ldots \dot{\cup} F_k$;
    for $i \leftarrow 1$ **to** k **do**
        if $X_i = \{f\}$ *for some* $f \in F$ *and* $c(f) < U$ **then**
            Set $U := c(f)$ and $f^* = f$;
        else
            Apply *bound* to get a lower bound $L := L(F_0)$;
            if $L < U$ **then**
                Add vertex F_i to T, i.e. $T := (V(T) \cup \{F_i\}, E(T) \cup \{\{F, F_i\}\})$;
                Mark F_i as active;
            end
        end
    end
end
return f^*

Note that the performance of the branch-and-bound in general depends on the order in which the subproblems are processed, i.e. how an active vertex $F_0 \in V(T)$ gets chosen. This design decision is called *search strategy*. As search strategy one can use depth-first-search or breadth-first-search, but often combinations of them are used together with heuristic decisions.

Example 2.3.4. A naive branch-and-bound approach for the TSP based on the same idea as the enumeration approach presented in 2.3.3 can be defined as follows. Subproblems correspond to partial solutions, i.e. an injective function $\pi^{(k)} : \{1, \ldots, k\} \to \{1, \ldots, n\}$. Such a partial solution $\pi^{(k)}$ represents the subproblem of all permutations that coincide with $\pi^{(k)}$ on $\{1, \ldots, k\}$. The starting problem, the whole search space can then be

interpreted as the subproblem corresponding to the function $\pi^{(1)} : 1 \mapsto 1$, which only maps the first vertex to itself.

The branching procedure can now be described as extending a partial solution by one mapping. Formally, given a partial solution $\pi^{(k)}$ and still unmapped indices $I := \{1, \ldots, n\} \setminus \left\{\pi^{(k)}(j) \mid j \in 1, \ldots, k\right\} = \{i_1, \ldots, i_{n-k}\}$, we consider all $n - k$ possible extensions of the form

$$\pi_j^{(k+1)} := \pi^{(k)} \cup (k+1, i_j)$$

for $j \in \{1, \ldots, n-k\}$.

As a trivial lower bound $L(\pi^{(k)})$ we simply use the already fixed edges

$$L(\pi^{(k)}) := \sum_{j=1}^{k-1} c_{\pi(j)\pi(j+1)}.$$

This is a lower bound since by definition of the TSP adding more edges e to the partial solution would only increase the costs since $c_e \geq 0$ for all $e \in E$.

The performance of the branch-and-bound procedure strongly depends on the quality of the lower bounds. If they are weak then the branch-and-bound approach degenerates to a total enumeration approach.

2.3.3 (Mixed) Integer Linear Programming

A powerful tool for solving COPs are integer linear programs (ILPs) formulations and existing solving techniques. Since integer linear programming is based on linear programming, we first introduce linear programming. This section is based on [81] and [4].

Linear Programming

Definition 2.3.9. A *linear program (LP)* over $n \in \mathbb{N}$ variables is specified by a cost vector $\mathbf{c} \in \mathbb{R}^n$, disjoint finite index sets M_1, M_2, M_3, for each index $i \in M_1 \cup M_2 \cup M_3$ a constraint coefficient vector $\mathbf{a}_i \in \mathbb{R}^n$ and a scalar $b_i \in \mathbb{R}$, and disjoint subsets N_1 and N_2 of $\{1, \ldots, n\}$. A solution to the LP is given by values for the variables $\mathbf{x} \in \mathbb{R}^n$ solving

$$\begin{aligned}
&\text{minimize } \mathbf{c}'\mathbf{x} \\
&\text{s.t. } \mathbf{a}_i'\mathbf{x} \geq b_i && \forall i \in M_1, \\
&\quad \mathbf{a}_i'\mathbf{x} \leq b_i && \forall i \in M_2, \\
&\quad \mathbf{a}_i'\mathbf{x} = b_i && \forall i \in M_3, \\
&\quad x_j \geq 0 && \forall j \in N_1, \\
&\quad x_j \leq 0 && \forall j \in N_2.
\end{aligned}$$

The function $\mathbf{x} \mapsto \mathbf{c}'\mathbf{x}$ is called *objective function* or *cost function*. A vector $\mathbf{x}$ satisfying all the constraints but not necessarily minimizing the objective function is called a *feasible solution*. The set of all feasible solutions is also called the *feasible region*.

Note that we only defined the LP as a minimization problem, but we also consider a maximization problem as a valid LP, which can be turned into a minimization problem by multiplying the cost vector $\mathbf{c}$ with -1.

The above definition of LP is nice for modeling given problems but quite text intensive for handling and using it in a theoretical manner as we do in the following. Therefore, we propose a much simpler representation, which we call reduced LP to avoid confusion.

Definition 2.3.10. A *reduced LP* is specified by an $m \times n$ constraint matrix $A \in \mathbb{R}^{m \times n}$, a vector $\mathbf{b} \in \mathbb{R}^m$, and a cost vector $\mathbf{c} \in \mathbb{R}^n$. The reduced LP is then specified by

$$\begin{aligned} &\text{minimize } \mathbf{c}'\mathbf{x} \\ &\text{s.t. } A\mathbf{x} \geq \mathbf{b}. \end{aligned}$$

Note that the inequality $A\mathbf{x} \geq \mathbf{b}$ is considered element-wise.

The feasible region of a reduced LP can be described by $\{\mathbf{x} \in \mathbb{R}^n \mid Ax \geq \mathbf{b}\}$. We call such a set for any matrix A and vector $\mathbf{b}$ a *polyhedron*.

By using the rows of the matrix A as vectors $\mathbf{a}_i$, it is easy to see that every reduced LP is also an LP. To see that every LP can be transformed to a reduced LP we have to specify how to transform constraints associated with indices from M_2 and M_3 and the constraints of the form $x_j \geq 0$ and $x_j \leq 0$. For constraints of the form $\mathbf{a}_i'\mathbf{x} \leq b_i$ we can simply multiply $\mathbf{a}_i$ and $\mathbf{b}$ with -1 and get a new equivalent constraint $-\mathbf{a}_i'\mathbf{x} \geq -b_i$. An equality constraint $\mathbf{a}_i'\mathbf{x} = b_i$ can be replaced by two constraints $\mathbf{a}_i'\mathbf{x} \geq b_i$ and $-\mathbf{a}_i'\mathbf{x} \geq -b_i$. Last but not least, constraints of the form $x_j \geq 0$ or $x_j \leq 0$ can be transformed by adding a new row with the j-th unit vector or -1 times the j-th unit vector to A together with a 0 to $\mathbf{b}$. Note that this also shows that the feasible region of any LP can be described by a polyhedron.

We describe now in the following the idea of a solution method, called *Simplex method*, which is commonly used for solving LPs in practice. A detailed description of the method is out of scope for this work, and we refer the interested reader to [4]. One key property for this algorithm is that the feasible region of an LP is always convex. Furthermore, because of the convexity of the objective function every local optimum is also a global optimum.

Lemma 2.3.2. *The feasible region of an LP is convex.*

Proof. It is enough to show it for a reduced LP with constraint matrix A. Let $\mathbf{x}$ and $\mathbf{y}$ be two feasible solutions. Then we get for any $\lambda \in [0, 1]$ that

$$A\left(\lambda\mathbf{x} + (1-\lambda)\mathbf{y}\right) = \lambda A\mathbf{x} + (1-\lambda)A\mathbf{y} \geq \lambda\mathbf{b} + (1-\lambda)\mathbf{b} = \mathbf{b}$$

and therefore $\mathbf{z} = \lambda\mathbf{x} + (1-\lambda)\mathbf{y}$ is a feasible solution. □

Definition 2.3.11. A feasible solution $\mathbf{x}$ of an LP with cost vector $\mathbf{c}$ is called a *local optimum* if there exists an $\epsilon > 0$ such that for any feasible solution $\mathbf{y}$ with $\|\mathbf{x}-\mathbf{y}\| \leq \epsilon$ it holds that $\mathbf{c}'\mathbf{x} \leq \mathbf{c}'\mathbf{y}$.

Theorem 2.3.3. *Every local optimum* $\mathbf{x}$ *of an LP is also a global optimum.*

Proof. Let ϵ be as in the definition of the local optimum $\mathbf{x}$ and $\mathbf{y} \neq \mathbf{x} \in \mathbb{R}^n$ any feasible solution not equal to $\mathbf{x}$. Since the feasible region is convex, see Lemma 2.3.2, we know that $\mathbf{z} = \mathbf{x} + (\mathbf{y}-\mathbf{x})\frac{\epsilon}{\|\mathbf{y}-\mathbf{x}\|}$ is a feasible solution and $\|\mathbf{x}-\mathbf{z}\| \leq \epsilon$. Therefore, we get

$$\mathbf{c}'\mathbf{x} \leq \mathbf{c}'\left(\mathbf{x} + (\mathbf{y}-\mathbf{x})\frac{\epsilon}{\|\mathbf{y}-\mathbf{x}\|}\right) = \mathbf{c}'\mathbf{x} + \mathbf{c}'(\mathbf{y}-\mathbf{x})\frac{\epsilon}{\|\mathbf{y}-\mathbf{x}\|},$$

which implies $\mathbf{c}'(\mathbf{y}-\mathbf{x}) \geq 0$ and thus $\mathbf{c}'\mathbf{x} \leq \mathbf{c}'\mathbf{y}$. □

The next important result is that it is enough to check the extreme points of the feasible region for a local/global optimum in an LP.

Definition 2.3.12. A point $x \in P$ of a polyhedron P is called an *extreme point* of P if there are no $y, z \in P \setminus \{x\}$ such that $x = \lambda y + (1-\lambda)z$ for some $\lambda \in [0,1]$.

The following theorem is a combination of Theorem 2.8 and Corollary 2.3 in [58].

Theorem 2.3.4. *For any LP exactly one of the following holds.*

1. *The feasible region is empty,*
2. *the optimal cost is* $-\infty$*, i.e. there exist feasible solutions with arbitrary low costs,*
3. *the nonempty feasible region has no extreme points and there exists an optimal solution, or*
4. *there exists an optimal solution that is an extreme point.*

We want to avoid case 3 in Theorem 2.3.4 to be able to only search for extreme points. This can be done by transforming the LP into a so called standard form.

Definition 2.3.13. An LP of the form

$$\begin{aligned} \text{minimize } & \mathbf{c}'\mathbf{x} \\ \text{s.t. } & A\mathbf{x} = \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \end{aligned}$$

with $A \in \mathbb{R}^{m \times n}$ and $\mathbf{b} \in \mathbb{R}^m$ is said to be in *standard form*.

See [58] for the proof of the following theorem.

Theorem 2.3.5. *Every LP can be transformed into an equivalent LP in standard form.*

For LPs in standard form we get that they contain an extreme point if they are non-empty, see Corollary 2.2 in [58].

Theorem 2.3.6. *An LP in standard form has either an empty feasible region or its feasible region has at least one extreme point.*

We can describe now the basic procedure of the Simplex method. Given an LP, we transform it into an equivalent LP in standard form. Then we search for an extreme point, which must exist by Theorem 2.3.6, using an auxiliary LP for which we already know an extreme point, see [58] for more details. Having an extreme point, we try to improve it by moving along edges of the geometric border of the feasible region. If we cannot find any edge such that moving along this edge improves the current costs, we know that the current extreme point is a local optimum and therefore a global optimum by Theorem 2.3.3. If at some point there is an infinitely long edge that improves the costs, then we know that the optimal cost is $-\infty$.

Note that although the simplex method is commonly used in practice to solve LPs it has an exponential worst case running time. The ellipsoid method, see [58], on the other hand has polynomial running time but often performs worse than the simplex method for practical applications. There are other approaches for solving LPs based on interior-point methods. In contrast to the Simplex method that moves along the border of a polyhedron to find an optimal solution, interior point methods move in the interior of the polyhedron to find an optimal solution.

Mixed Integer Linear Programs

Since LPs can be solved in polynomial time we cannot directly use them to solve $\mathcal{NP}$-hard problems as long as $\mathcal{P} = \mathcal{NP}$ is not proved. By adding integrality constraints we get the much more powerful concept of mixed integer programs, see [81].

Definition 2.3.14. A *mixed integer linear program (MILP)* is given by a $m \times n$ matrix $A \in \mathbb{R}^{m \times n}$, a $m \times p$ matrix $G \in \mathbb{R}^{m \times p}$, a vector $\mathbf{c} \in \mathbb{R}^n$, a vector $\mathbf{h} \in \mathbb{R}^p$, and a vector $\mathbf{b} \in \mathbb{R}^m$. The problem is then to find a vector $\mathbf{x} \in \mathbb{R}^n$ and a vector $\mathbf{y} \in \mathbb{N}^p$ solving

$$\begin{aligned} \text{minimize } & \mathbf{c}'\mathbf{x} + \mathbf{h}'\mathbf{y} \\ \text{s.t. } & A\mathbf{x} + G\mathbf{y} \geq \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0}, \mathbf{y} \geq \mathbf{0}, \mathbf{y} \text{ is integral} \end{aligned}$$

If $n = 0$, i.e. all variables are integer, we call it *integer linear program (ILP)*.

Note that in a MILP in contrast to an LP the $\mathbf{y}$ variables are only allowed to be integers. We use here a slightly different definition of MILP than in [81] to stay consistent with our definition of LPs. As in the case of LPs we can also allow maximization problems, constraints of the form $\mathbf{a}'\mathbf{x} + \mathbf{g}'\mathbf{y} \leq \mathbf{b}$ or $\mathbf{a}'\mathbf{x} + \mathbf{g}'\mathbf{y} = \mathbf{b}$ and have unbounded or only non-positive variables and transform them into the above form of a MILP. Therefore, we also speak of a MILP if we do maximize or have such kind of constraints or variables.

To denote the feasible region for a MILP we use the shorter notation

$$\{\mathbf{z} \in \mathbb{R}^n \times \mathbb{Z}^p \mid A\mathbf{z} \geq \mathbf{b}\}.$$

This is possible since we can rewrite the inequalities $\mathbf{x} \geq 0$ and $\mathbf{y} \geq 0$ to be part of the matrix A.

The expressiveness of MILPs can be seen in the fact that solving a general ILP is $\mathcal{NP}$-hard, see [81], which implies that every problem in $\mathcal{NP}$ can be polynomially reduced to solving an ILP.

LP-based Branch-and-Bound

In the following we describe a general approach for solving MILPs based on branch-and-bound, see [81] for more details. Note that the approach described here is just a basic method and competitive solvers implement highly non-trivial additional improvements. For a competitive solver, see for example IBM ILOG CPLEX Optimizer[1] or Gurobi Optimizer[2].

As described in Section 2.3.2 we need to specify a *branch* method and a *bound* method to fully specify a branch-and-bound procedure. The idea of the branching is to split the search space into two halves by adding a new constraint and its complement to the MILP. Therefore, subproblems are represented by the base MILP plus additionally added constraints. In each node of the branch-and-bound procedure, corresponding to a subproblem represented by a MILP, we solve the so called LP relaxation.

Definition 2.3.15. Given a MILP

$$\begin{aligned} \text{minimize } & \mathbf{c}'\mathbf{x} + \mathbf{h}'\mathbf{y} \\ \text{s.t. } & A\mathbf{x} + G\mathbf{y} \geq \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0}, \mathbf{y} \geq \mathbf{0}, \mathbf{y} \text{ is integral} \end{aligned}$$

the *LP relaxation* is the linear program

$$\begin{aligned} \text{minimize } & \mathbf{c}'\mathbf{x} + \mathbf{h}'\mathbf{y} \\ \text{s.t. } & A\mathbf{x} + G\mathbf{y} \geq \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0}, \mathbf{y} \geq \mathbf{0} \end{aligned}$$

relaxing the integrality condition of $\mathbf{y}$.

[1] https://www.ibm.com/analytics/cplex-optimizer (accessed 09/2019)
[2] https://www.gurobi.com (accessed 09/2019)

Theorem 2.3.7. *Given a MILP, the optimal value z^{LP} of the LP relaxation is a lower bound of the corresponding MILP. This lower bound is called* dual bound.

Proof. The only difference between the MILP and its LP relaxation is, that the feasible region of the LP relaxation is a superset of the feasible region of the MILP. Therefore, every optimal solution of the MILP is a feasible solution of the LP relaxation with the same objective value. Thus, an optimal solution of the LP relaxation has costs at least as low as any optimal solution of the MILP. □

Consider now some subproblem corresponding to some MILP. Then we solve the LP relaxation using for example the Simplex method. If the LP relaxation is infeasible, we know that the subproblem has no feasible solution and can drop this subproblem. This case is also called *pruning by infeasibility*. Otherwise, let $(\mathbf{x}^*, \mathbf{y}^*)$ be an optimal solution of the LP relaxation.

If all variables in $\mathbf{y}^*$ are integral we got an optimal solution of the subproblem. We can therefore update the global upper bound U in Algorithm 2.1 if applicable and then drop this subproblem. This is also called *pruning by optimality*. Otherwise, let y_i^* be any non-integral variable. We can then split the problem into two by adding to one problem the constraint $y_i \leq \lfloor y_i^* \rfloor$ and to the other the constraint $y_i \geq \lceil y_i^* \rceil$. Clearly any feasible integer solution to the subproblem is either feasible in the first or in the second of those two new problems giving us a formal partition of the solutions of the subproblem as it is necessary for the *branch* method. In contrast to the formal condition that subproblems are non-empty in the definition of branch-and-bound this may not be the case for this procedure. But since the current solution $(\mathbf{x}^*, \mathbf{y}^*)$ of the LP relaxation is infeasible in both problems, this is enough to avoid infinite branching loops. The choice of the variable y_i^* strongly influences the performance of the branch-and-bound. A reasonable choice would be to use the most fractional variable, i.e., the variable with the largest distance to the next integer value, but there also exist more sophisticated approaches.

The *bound* method can simply be described as using the dual bound, i.e. the optimal objective value of the LP relaxation. If the dual bound is larger than or equal to the current global upper bound (see U in Algorithm 2.1) then this subproblem gets also dropped, which is called *pruning by bound* in this case.

The performance of this branch-and-bound approach strongly depends on the quality of the dual bounds. Note that different MILP representations of the same feasible region may result in different LP relaxations. Therefore, it is important to choose a good representation of the MILP that results in strong dual bounds. In this context we introduce a relation between formulations of the feasible area of a MILP, see [81].

Definition 2.3.16. A polyhedron $P \subseteq \mathbb{R}^{n+p}$ is a *formulation* for a set $Z \subseteq \mathbb{R}^n \times \mathbb{Z}^p$ if $Z = P \cap (\mathbb{R}^n \times \mathbb{Z}^p)$. Given two formulations P_1 and P_2 of a set X, we say that P_1 is *at least as strong* as P_2 if $P_1 \subseteq P_2$, and *stronger* as P_2 if $P_1 \subsetneq P_2$. Furthermore, if neither $P_1 \subseteq P_2$ nor $P_2 \subseteq P_1$ we say the two formulations are *incomparable*.

Branch-and-Cut

As we saw already in the previous section there can be different formulations of the same feasible region for a MILP. One way to improve the strength of a formulation is to add valid cuts, see [81].

Definition 2.3.17. An inequality $\mathbf{a}'\mathbf{z} \leq b_0$ with $\mathbf{a} \in \mathbb{R}^{n+p}$ and $b_0 \in \mathbb{R}$ is a *valid inequality* for a set $Z \subseteq \mathbb{R}^n \times \mathbb{Z}^p$ if $\mathbf{a}'\mathbf{z} \leq b_0$ for all $\mathbf{z} \in Z$.

Given a formulation P of a set Z, then we can search for a valid cut $\mathbf{a}'\mathbf{z} \leq b_0$ for Z with $\{\mathbf{z} \in \mathbb{R}^{n+p} \mid \mathbf{a}'\mathbf{z} \leq b_0\} \not\supseteq P$ and add it to P resulting in the new formulation $P' := P \cap \{\mathbf{z} \in \mathbb{R}^{n+p} \mid \mathbf{a}'\mathbf{z} \leq b_0\}$, which is stronger than P. The task to find good valid inequalities, which do not blow up the model and strengthen it substantially, is non-trivial, for more details see [81].

Given an optimal solution to the LP relaxation of a MILP, we call a valid inequality a *cut* if the current solution does not satisfy the inequality. The problem of finding a new cut for a given solution of the LP relaxation is called *separation problem*. The technique of iteratively adding cuts to strengthen the formulation by solving the separation problem is called *cutting plane method*. *Branch-and-cut* can be seen as incorporating the cutting plane method into the branch-and-bound procedure presented previously. At each node of the branch-and-bound tree, before we solve the LP relaxation, we search for cuts to add to the current formulation. If we cannot find anymore suitable cuts we solve the LP relaxation of the final formulation and proceed with pruning or branching as described in the branch-and-bound procedure.

Until now, we defined branch-and-cut to only add cuts at the nodes as it is also done in [81]. But we are also interested in another use case of branch-and-cut in which we dynamically add inequalities that cut away unwanted integer solutions. This can be helpful in situations where the original formulation of the MILP has a lot of constraints $\mathcal{C}$, maybe even exponential many, and generating all of them would blow up the model significantly. In this case we formally consider at the beginning only a small subset $\mathcal{C}_0 \subseteq \mathcal{C}$ of the constraints, resulting in a larger feasible region $Z_0 \supseteq Z$.

Consider now the original formulation P of Z, which is described by $\mathcal{C}$, and the new formulation $P_0 \supseteq P$ of Z_0 described by $\mathcal{C}_0$. We start now the branch-and-cut procedure working on the formulation P_0. At each node if the solution of the LP relaxation is not in P we search for a violated constraint $C \in \mathcal{C}$ and add it to the formulation. The problem of finding such a violated constraint is again called the separation problem. Note that this procedure works without generating all the constraints describing P, only the logic of them has to be incorporated into solving the separation problem, which is strongly problem specific. The constraints in $\mathcal{C} \setminus \mathcal{C}_0$ are often called *lazy constraints*.

A famous example for using lazy constraints are the subtour inequalities in the context of the TSP, see for example [3]. Exponential many inequalities ensure that for every proper subset V_0 of the vertex set V at least two edges between the vertices of V_0

and $V \setminus V_0$ are used. As it turns out in many situations dynamically generating those constraints in a branch-and-cut procedure result in a better performance than a more compact formulation, such as the Miller-Tucker-Zemlin (MTZ) formulation. Important for the effectivity of the branch-and-cut procedure is that in most practical cases only few constraints out of the exponential many get really generated and added.

2.3.4 SAT Solving

As mentioned in Section 2.2 SAT is $\mathcal{NP}$-complete and heavily studied in literature. This focus led to performant solvers for solving SAT instances in practice. This section is based on the book [6].

Using the expressiveness of boolean formulas one approach of solving any decision problem is to reduce it to SAT and use a powerful SAT solver to solve it. Therefore, we can see SAT as a modeling language with performant solvers. In this sense SAT can be compared to MILP as a modeling language, although MILP solvers can solve discrete optimization problems and SAT solvers can only solve decision problems. Note that there exist extensions of SAT such as the maximum satisfiability problem (MAX-SAT) that are also able to model optimization problems, see [6].

In this section we present the basic solving approach of most modern SAT solvers. There are many different solution approaches for solving a SAT instance, but one technique based on conflict driven clause learning (CDCL) is especially successful and is used nowadays in most competitive SAT solvers. This can be seen as an extension of the *Davis–Putnam–Logemann–Loveland (DPLL)* algorithm, which we present first.

The basic idea of DPLL is to systematically try out truth values for variables, resulting in a search tree, and use unit resolution to detect infeasibility of a branch as early as possible.

Definition 2.3.18. Given two clauses C_i and C_j and a variable P with $P \in C_i$ and $\bar{P} \in C_j$, the clause $(C_i \setminus \{P\}) \cup (C_j \setminus \{\bar{P}\})$ is called a *P-resolvent* or simply *resolvent* that is obtained by *resolving* C_i and C_j. If $|C_i| = 1$ or $|C_j| = 1$ this technique is called *unit resolution*.

It is easy to see that if C_i and C_j are satisfied then any resolvent of them must also be satisfied. The technique of resolving is used to learn new clauses for a given set of clauses. Whenever our CNF contains a clause with only one literal we can fix the truth value of this literal and apply unit resolution to all clauses that contain the negated literal. We can then do this iteratively if more clauses with only one literal arise. Before we can describe the algorithm DPLL, we need one more notation.

Definition 2.3.19. Let Δ be a CNF and L a literal. Then we define the *conditioning* operator by

$$\Delta|L = \left\{ C \setminus \{\bar{L}\} \mid C \in \Delta, L \notin C \right\}.$$

It is easy to see that $\Delta|L$ is equivalent to $\Delta \wedge L$, i.e. fixing the literal L to true in Δ.

Let UNIT_RESOLUTION denote the procedure of applying iteratively all possible unit resolutions to a CNF Δ and returning a pair $(\mathcal{I}, \Gamma)$, where $\mathcal{I}$ is a set of the fixed literals and Γ is the CNF obtained after conditioning Δ on all literals in $\mathcal{I}$. The basic DPLL algorithm is now shown in Algorithm 2.2.

Algorithm 2.2: DPLL

Input: A CNF Δ
Output: Either a feasible set of literals satisfying Δ or UNSATISFIABLE

```
1  (I, Γ) := UNIT_RESOLUTION(Δ);
2  if Γ = ∅ then
3  |  return I
4  else if ∅ ∈ Γ then
5  |  return UNSATISFIABLE
6  else
7  |  Select a literal L from Γ;
8  |  if R = DPLL(Γ|L) ≠ UNSATISFIABLE then
9  |  |  return R ∪ I ∪ {L}
10 |  else if R = DPLL(Γ|L̄) ≠ UNSATISFIABLE then
11 |  |  return R ∪ I ∪ {L̄}
12 |  else
13 |  |  return UNSATISFIABLE
14 end
```

One of the drawbacks of the DPLL algorithm is that it does not consider the reason why a CNF is unsatisfiable. This leads to the fact that the same contradiction may be produced for different branches of the search tree, since the contradiction does not depend on all the previously fixed variables. Therefore, modern SAT solver incorporate conflict driven clause learning (CDCL) to learn conflicts also for other branches of the search tree. Whenever UNIT_RESOLUTION returns an empty Γ the solvers derive a conflict clause and add this conflict clause globally to the CNF Δ. This leads to the point that we do not need the search tree anymore, since we can simply only consider the current branch of fixed literals and whenever we learn a new clause undo some decisions and start again with the extended CNF.

Note that an empty Δ reached through unit resolution may lead to multiple different conflict clauses and the choice of which to use are non-trivial details of the different implementations. One possible choice is to use the clause consisting of the negations of all the literals fixed for the current branch that were used during the unit resolution process to obtain the contradiction.

Another implementation detail that may highly impact the performance of the solver is the selection of the next literal L of Γ to fix. Furthermore, other additional features occurring in many solvers are deletion of learned clauses after some time and restarts to

avoid being stuck in a difficult part of the search area, see [6] for more details. Moreover, some solvers may also include more advanced resolution techniques additionally to unit resolution.

2.4 Heuristic and Metaheuristic Approaches

In the previous section we saw methods for solving COPs in an exact way. That means that the algorithms guarantee to return an optimal solution when the algorithm terminates and one exists. For $\mathcal{NP}$-hard optimization problems all exact approaches need exponential running time as long as $\mathcal{P} = \mathcal{NP}$ is not proven. This often results in long running times in practice, especially for large instances. The idea of heuristics is to compute a promising solution in a short time. The solution may not be optimal and there are in general no quality guarantees in contrast to approximation algorithms, but in practice the objective value may frequently be close to the optimal one.

A technique for developing heuristics is the use of metaheuristics. Sörensen and Glover define metaheuristics as follows [76].

Definition 2.4.1. A *metaheuristic* is a high-level problem-independent algorithmic framework that provides a set of guidelines or strategies to develop heuristic optimization algorithms.

Note that we also use the term *metaheuristic* to describe problem specific algorithms that are based on a metaheuristic framework. One important principle of metaheuristics is the balance of *diversification* and *intensification* of the search, see [8]. Those terms are based on a theoretical model of the search space as metric space based on a similarity measure. Similar solutions that can be derived from each other by only small changes are close together in the search space and completely different solutions are far away from each other. In this context intensification means finding the best solution in a small subarea of the search space while diversification means diversifying the search over the whole search space searching in completely different areas.

Many metaheuristics got developed over time, see [35], and therefore we only present those metaheuristics here that we use in this work. In this section we present first one of the most basic ingredients of many metaheuristics, the greedy construction heuristic, which is a problem specific heuristic. Then we present the local search heuristic followed by variable neighborhood search based metaheuristics and very large scale neighborhood search (VLSN).

2.4.1 Greedy Construction Heuristic

Many metaheuristics such as variable neighborhood search based metaheuristics need a starting solution, which they can then subsequently improve. For that one normally uses problem specific construction heuristics, one common scheme of them is called *greedy*

construction heuristic. The basic idea is to build up a partial solution and iteratively add new components that look the most promising until we get a complete solution.

The problem specific ingredients of a greedy construction heuristic is a representation of partial solutions, a starting partial solution S_0, for each partial solution S a set $E(S)$ of possible extensions and a rating function $r(e, S)$ for each extension $e \in E(S)$. As we assume minimization, a lower rating is better. The pseudo-code for a general greedy construction heuristic with those ingredients is given in Algorithm 2.3.

Algorithm 2.3: Greedy Construction Heuristic

Input: An instance of a combinarial optimization problem
Output: A partial solution S that cannot be extended anymore

```
S := S_0;
while E(S) ≠ ∅ do
    e := arg min_{e∈E(S)} r(e, S);
    S := S extended with e;
end
return S
```

If a partial solution is already a feasible solution to the problem we call it complete. It is important to note that the design of the greedy construction heuristic has to be chosen in such a way that a partial solution is always complete whenever there is no possible extension for it. Otherwise, no feasible solution can be returned.

Example 2.4.1. Let us consider an instance of the TSP, i.e. a complete graph K_n with edge costs c_e for each $e \in E(K_n)$. There are many different possibilities how to design a construction heuristic for the TSP, see [66]. We consider in this example the so called *nearest neighbor heuristic*.

The idea of the heuristic is to incrementally create a path and add in each iteration an unvisited vertex that is the nearest to the current end of the path. The considered paths start at an arbitrary vertex $v_0 \in V(K_n)$. As partial solutions we use paths in K_n starting at v_0. The starting solution S_0 is the path of length zero starting and ending at v_0. For a path S an extension can be described by an unvisited vertex and therefore we can define $E(S) := V(K_n) \setminus V(S)$. Furthermore, we define the rating $r(v, S)$ for a path S connecting v_0 with w and a vertex $v \in E(S)$ to be $r(v, S) := c_{vw}$. A path S connecting v_0 with w gets extended by a vertex v by adding the vertex v and the edge vw to S.

Note that with the above described configuration the greedy construction heuristic returns only a path and we need to close the path by finally adding the edge between the two end vertices to get a hamiltonian cycle.

2.4.2 Local Search

The technique of *local search* is used in many metaheuristics for intensifying the search, but it can also be used as a standalone improvement heuristic. The main idea is to try to iteratively improve a solution by doing small modifications. To formalize this idea we use the concept of neighborhoods.

Definition 2.4.2. Given an instance (F, c) of a COP, a *neighborhood* of a solution $f \in F$ is a set of solutions $N(f) \subseteq F$. A *neighborhood structure* N is then a function that maps every solution to its neighborhood, i.e. $N : F \to 2^F, f \mapsto N(f)$.

Often neighborhood structures are described by move operations. A *move* is a description how to modify a solution to get to its neighbors. The neighborhood $N(f)$ of a solution f is then defined by the solutions obtained by applying all possible moves to the solution f.

Example 2.4.2. Consider again an instance of the TSP. A popular neighborhood structure for the TSP is the so called k-*opt* neighborhood structure for some fixed $k \in \mathbb{N}$, $k \geq 2$, see [66]. Given a hamiltonian cycle C we describe a move by removing k edges and adding k edges in such a way that the result is again a hamiltonian cycle. The neighborhood $N(C)$ associated with this move consists then of all hamiltonian cycles that differ from C in at least two and at most k edges, i.e.

$$N(C) := \{C' \text{ hamiltonian cycle of } K_n \mid C \neq C' \wedge |E(C') \setminus E(C)| \leq k\}.$$

Given a neighborhood structure N the standalone local search procedure is given in Algorithm 2.4

Algorithm 2.4: Local Search

Input: An instance (F, c), a starting solution $f_0 \in F$, and a neighborhood structure N
Output: A solution $f \in F$ with $c(f) \leq c(f_0)$

```
f := f_0;
while ∃f' ∈ N(f) with c(f') < f do
    f := f'
end
return f
```

An important implementation detail for a local search procedure is how we search through the current neighborhood. There are three established variants. The first is called *first improvement* and enumerates through the neighborhood until it finds a $f' \in N(f)$ with $c(f') < f$ and returns it. In the second variant, which is called *best improvment*, we always iterate through the whole neighborhood and search for a solution f' with the lowest costs, i.e., a best neighbor. Last but not least, in *random improvement* we randomly select a neighbor f' and test if $c(f') < f$.

Note that the third variant differs from the other two because we cannot directly use it in Algorithm 2.4 since we never know if there exists a $f' \in N(f)$ with $c(f') < f$ as long as we do not find such one randomly. Therefore, local search with random improvement discards worse solutions but continues with the next iteration until a termination criterion is satisfied, e.g. a certain number of successive unsuccessful iterations has been reached.

First and best improvement have the advantage that they can prove that a solution cannot be improved by this neighborhood. We call such a solution a local optimum, similarly as we defined it for the continuous LP problem, see Definition 2.3.11.

Definition 2.4.3. Given a neighborhood structure N, a solution f is called a *local optimum* with respect to N if $c(f) \leq c(f')$ for all $f' \in N(f)$.

Theorem 2.4.1. *Using local search with a neighborhood structure N and with first improvement or best improvement returns a local optimum with respect to N.*

2.4.3 Variable Neighborhood Search

Theorem 2.4.1 in the previous section can also be considered a weakness of local search since the cost of a local optimum may be much worse than the cost of the global optimum. In this sense we say that local search gets stuck in a local optimum. Therefore, many metaheuristics are designed to overcome this weakness of getting stuck in a local optimum. This section is based on the book chapter [42].

One simple approach is to use multiple different neighborhood structures instead of only one. Given neighborhood structures $N_1, N_2, \ldots, N_k$ the *variable neighborhood descent (VND)* approach systematically searches through all of them until none of the neighborhoods yield an improvement on the current solution. A pseudo-code for this approach is given in Algorithm 2.5.

Algorithm 2.5: VND

Input: An instance (F, c), a starting solution $f_0 \in F$, and neighborhood structures $N_1, \ldots, N_k$
Output: A solution $f \in F$ with $c(f) \leq c(f_0)$

```
f := f_0, i := 1;
while i ≤ k do
    f' := arg min_{x∈N_i(f)} c(x);
    if c(f') < c(f) then
        f := f', i := 1;
    else
        i := i + 1;
    end
end
return f
```

One important design decision for VND is the order of the neighborhood structures. Every time an improvement is found the neighborhood index is traditionally reset to 1, this leads to the fact that neighborhood structures with smaller index get potentially searched through much more often than the ones with larger indices. Therefore, the order of the neighborhood structures has a high impact on the performance of the algorithm.

For multiple neighborhood structures $N_1, \ldots, N_k$ we say a solution f is a *local optimum* with respect to $N_1, \ldots, N_k$ if it is a local optimum with respect to N_i for all $i \in \{1, \ldots, k\}$. With this definition the following holds.

Theorem 2.4.2. *Applying VND with neighborhood structures $N_1, \ldots, N_k$ results in a local optimum with respect to $N_1, \ldots, N_k$.*

Note that VND normally uses best improvement or next improvement to search through one neighborhood. Therefore, the neighborhoods need to be small such that enumerating through them can be done in reasonable time. In this sense VND is only focusing on intensification within a small area of the search space and still gets stuck in local optima, although this time with respect to all neighborhood structures $N_1, \ldots, N_k$.

One approach to allow much larger neighborhood structures that cannot be enumerated in reasonable time is called *reduced variable neighborhood search (RVNS)*. This approach can be described by simply replacing the best improvement in line 3 of Algorithm 2.5 with random improvement, i.e. selecting a random neighbor instead of searching for the best neighbor. This random improvement step is in this context called *shaking*. If we do not find an improvement in all k neighborhood structures in this case it might just be bad luck and therefore we repeat the whole procedure many times until some stopping criterion reached. See Algorithm 2.6 for a pseudo-code.

There can be many different kinds of stopping criteria including a time limit, a reached solution quality, a total number of iterations, or a number of iterations in which no improvement was found.

Since we have no limit in the size of neighborhoods for the RVNS approach we can even use one neighborhood structure in which every neighborhood $N(f)$ is the whole search space F. If we have such a neighborhood or at least neighborhood structures that satisfy $\bigcup_{i=1}^{k} N_i(f) = F$ for every $f \in F$, we get a nice theoretical property. Namely, that if we let the algorithm run indefinitely, i.e. use no stopping criterion, then we find a globally optimal solution with probability 1. Unfortunately the algorithm cannot prove optimality and stop, therefore we never know if the current found solution is a global optimum or not.

In terms of intensification versus diversification RVNS strongly focuses on diversification. To balance this and add an intensification step one extension of RVNS is called *basic variable neighborhood search (BVNS)*, which applies a local search after each shaking. Therefore, we need another neighborhood structure $\mathcal{N}$ additionally to $N_1, \ldots, N_k$. Then we can describe BVNS by adding a local search step with the neighborhood structure $\mathcal{N}$

Algorithm 2.6: RVNS

Input: An instance (F, c), a starting solution $f_0 \in F$, and neighborhood structures $N_1, \ldots, N_k$
Output: A solution $f \in F$ with $c(f) \leq c(f_0)$

```
1  f := f_0;
2  while Stopping criterion is not met do
3  |  i := 1;
4  |  while i ≤ k do
5  |  |  f' := Random solution in N_i(f) ;                    // Shaking
6  |  |  if c(f') < c(f) then
7  |  |  |  f := f', i := 1;
8  |  |  else
9  |  |  |  i := i + 1;
10 |  |  end
11 |  end
12 end
13 return f
```

after line 5 in Algorithm 2.6, i.e. $f' := \texttt{LOCAL\_SEARCH}(f', \mathcal{N})$. Note that we only use the result of the local search if it is better than the current solution f.

Finally, we can extend now BVNS by replacing the local search step through a whole VND step based on multiple neighborhood structures $\mathcal{N}_1, \ldots, \mathcal{N}_k$. This algorithm is then called *general variable neighborhood search (GVNS)*, see Algorithm 2.7.

2.4.4 Very Large Scale Neighborhood Search

As we saw in the previous section choosing small neighborhoods for local search or VND results in being stuck in a local optimum that may be far worse than the globally optimal solution. Another approach besides BVNS or GVNS to avoid this is using a much larger neighborhood structure. Of course this still may get stuck in a local optimum, but hopefully this local optimum is then already globally optimal or close to it. The class of *very large scale neighborhood search (VLSN)* algorithms are based on large, often exponentially large, neighborhoods that cannot be enumerated in reasonable time, see [63]. Instead of enumerating it such a large neighborhood is then either restricted to some smaller set of especially promising neighbors, searched through heuristically, or searched through using a more sophisticated exact approach to find the best neighbor. Often, problem specific solution approaches based on LP, MILP, or dynamic programming (DP) are used to search through the large neighborhood.

One specific metaheuristic, which is part of the class of VLSN algorithms, is the so called *large neighborhood search (LNS)* [63]. The neighbors of an LNS neighborhood are specified by all solutions that are reachable by a specified destruction and repairing

Algorithm 2.7: GVNS

Input: An instance (F, c), a starting solution $f_0 \in F$, shaking neighborhood structures $N_1, \ldots, N_k$, and VND neighborhood structures $\mathcal{N}_1, \ldots, \mathcal{N}_k$
Output: A solution $f \in F$ with $c(f) \leq c(f_0)$

```
1  f := f_0;
2  while Stopping criterion is not met do
3  |  i := 1;
4  |  while i ≤ k do
5  |  |  f' := Random solution in N_i(f) ;                    // Shaking
6  |  |  f' := VND(f', N_1, ..., N_k);
7  |  |  if c(f') < c(f) then
8  |  |  |  f := f', i := 1;
9  |  |  else
10 |  |  |  i := i + 1;
11 |  |  end
12 |  end
13 end
14 return f
```

method. In each iteration one (often randomly) chosen destruction is applied and then a repair method is applied to get again a complete feasible solution.

In Chapter 3 we propose a combination of a GVNS and VLSN approach, by incorporating a large neighborhood in the GVNS framework. Such combinations of metaheuristics are called *hybrid metaheuristics*, see [64, 7]. The class of hybrid metaheuristics does not only contain combinations of two or more metaheuristics but also combinations of metaheuristics with exact approaches.

CHAPTER 3

Searching For Uniquely Hamiltonian Planar Graphs

In this chapter we present two kinds of approaches to search for uniquely hamiltonian planar graphs with minimum degree three. After a general introduction we present in the first part a heuristic to search for uniquely hamiltonian graphs minimizing the crossing number and the number of degree two vertices. Then, in the second part, we search for planar graphs containing stable cycles from which we can construct uniquely hamiltonian graphs.

3.1 Introduction

A lot of research in graph theory focuses on hamiltonian cycles. The problem of finding hamiltonian cycles is well studied from a theoretical [37] and practical point-of-views as a special case of the TSP [3]. An interesting topic in graph theory is the question of how many hamiltonian cycles a given graph has. A simpler version of this question only asks if there is exactly one hamiltonian cycle in a given graph.

In this chapter we are only concerned with undirected simple graphs and just write graph for this type of graphs.

Definition 3.1.1. A graph $G = (V, E)$ is *uniquely hamiltonian* if and only if it contains exactly one hamiltonian cycle.

One type of problem in the area of uniquely hamiltonian graphs is to determine for a given class of graphs if it contains a uniquely hamiltonian graph. Already in 1946 Smith proved that every edge of a 3-regular graph is contained in an even number of hamiltonian cycles, which was published by Tutte [79]. Clearly, this result implies that 3-regular graphs can never be uniquely hamiltonian. This was then further improved by

Thomason [77] who showed that all graphs where all vertices have odd degrees cannot be uniquely hamiltonian. As an implication we get that every uniquely hamiltonian graph must contain at least two vertices of even degree.

The aforementioned early results show that vertex degrees play an important role for uniquely hamiltonian graphs. Bondy and Jackson [10] provided an upper bound of $c\log_2(8n)+3$ with $c \approx 2.41$ for the smallest vertex degree $\delta(G)$ for a uniquely hamiltonian graph G. Abbasi and Jamshed [1] could improve the upper bound to $c\log_2(n)+2$ with $c \approx 1.71$. Naturally, the question arises how tight this upper bound is.

Already in 1980 Entringer and Swart [23] found uniquely hamiltonian graphs with minimum degree three. Their graphs have only two vertices of degree four and all other vertices have degree three, which is as close as we can get to 3-regular uniquely hamiltonian graphs by Thomason's theorem. Sheehan conjectured in 1975 that there do not exist 4-regular uniquely hamiltonian graphs. If we allow parallel edges, Fleischner [26] constructed infinitely many 4-regular uniquely hamiltonian multigraphs. This construction could not be extended to simple graphs and therefore Sheehan's conjecture is still an open problem.

Another question raised by Bondy [9] asks whether there exists a uniquely hamiltonian graph with minimum degree four. This question got answered by Fleischner [27] who constructed an infinite family of uniquely hamiltonian graphs that only have vertices of degree four and 14. The big gap between the best known minimum degree $\delta(G) = 4$ and the upper bound $\delta(G) \leq c\log_2(n)+2$ remains an open problem.

If we consider only planar graphs Bondy and Jackson [10] provided a much smaller upper bound of $\delta(G) \leq 3$ by showing that a planar uniquely hamiltonian graph has at least two vertices of degree two or three. Clearly, there exist planar uniquely hamiltonian graphs with $\delta(G) = 2$, for example the cycle graphs C_n. Bondy and Jackson conjectured that every planar uniquely hamiltonian graph contains a vertex of degree two, i.e. $\delta(G) \leq 2$ for planar uniquely hamiltonian graphs, which would close the gap for the case of planar graphs.

Since the construction of uniquely hamiltonian graphs with minimum degree four by Fleischner was quite surprising for the scientific community, one may guess that there exist also planar uniquely hamiltonian graphs with minimum degree three. Unfortunately, the techniques used by Fleischner to construct uniquely hamiltonian graphs with minimum degree four cannot directly be applied to planar graphs. This motivates the computer-aided search for such graphs on which we focus in this chapter.

The following definition is useful for both types of approaches that we are going to consider.

Definition 3.1.2. Let G be a graph. A *fixed edge cycle (FE-cycle)* is a pair (C, e) where C is a cycle of G and e an edge occurring in C. An FE-cycle is called *uniquely hamiltonian* if C is hamiltonian and there is no other hamiltonian cycle containing the edge e. Furthermore, a graph that contains a uniquely hamiltonian FE-cycle is called a *fixed edge uniquely hamiltonian graph (FEUHG)*.

Note that an FEUHG may contain multiple FE-cycles with different fixed edges. The importance of FEUHGs is that we can transform them to uniquely hamiltonian graphs. A similar transformation was first mentioned by Leder in his master thesis [59].

Proposition 3.1.1. *Let G be an FEUHG. Then there exists a uniquely hamiltonian graph G' with $|V(G')| = 2|V(G)| - 2$ and $\delta(G') \geq \delta(G)$. Furthermore, if G is planar then G' is also planar.*

Proof. Let (C, e) with $e = uv$ be a uniquely hamiltonian FE-cycle of G. We construct G' by starting with two copies G^* and G^{**} of $G - e$. We then identify the vertices $u^* \in V(G^*)$ with $u^{**} \in V(G^{**})$ and $v^* \in V(G^*)$ with $v^{**} \in V(G^{**})$ to glue the two copies together. The hamiltonian cycle C naturally implies a hamiltonian cycle C' in G' by using two copies of $C - e$ and connecting them. It is also stable since if there would be another hamiltonian cycle C'' in G' then it must be different in one of the two copies. Considering the edges of the cycle in this copy together with the edge e gives us then a cycle in G that is different to C and also a hamiltonian cycle of G containing e, which is a contradiction.

Clearly, $|V(G')| = 2|V(G)| - 2$ holds. All vertices in the two copies have the same degrees as in the original graph except for the identified vertices u^* and v^*, but the degrees of those two can only increase and therefore we have $\delta(G') \geq \delta(G)$. Furthermore, if G is planar we can find a planar embedding such that e is on the outer face and then removing e, adding the copy of the embedding and identifying u^* and v^* with their copies can be done without any crossings. Therefore, G' is also planar. □

Note that Leder connected the vertices u^* with u^{**} and v^* with v^{**} using new edges instead of identifying them, which results in a slightly larger graph with the advantage that all vertices in the copies have the same degrees as in the original graph. This is especially important for regular graphs, since it is regularity-preserving.

3.2 Metaheuristic Approach to Minimize the Number of Crossings and Vertices of Degree Two

In this section we transform the problem of finding a uniquely hamiltonian planar graph with minimum degree three into a bi-objective optimization problem that minimizes the crossing number and the number of vertices with degree two. To solve this problem approximately we propose a GVNS heuristic. The GVNS framework was already successfully applied to other graph theoretical problems, see, e.g., [16].

As the search space increases exponentially with increasing number of vertices a heuristic could be beneficial compared to an exact approach, for exact approaches see Section 3.3. One of our proposed neighborhood structures is exponentially large and we use a branch-and-bound procedure to find the best neighbor in its neighborhoods. This embeds the idea of a VLSN [63] in our GVNS. We will see that the bottleneck in our algorithm,

which consumes most of the running time, are the expensive unique hamiltonicity checks. To reduce the number of such checks we keep infeasible, not uniquely hamiltonian, solutions formally in our neighborhood. Only after finding the best neighbor we apply a Lin-Kerninghan Heuristic (LKH) [44] and in a later step we use Concorde[1] to check for feasibility.

In the next section we describe some transformations of the problem and formally state the resulting optimization problem. In Section 3.2.2 we describe our GVNS framework and the neighborhood structures in detail. The comparison of three different configurations and the experimental results are presented in Section 3.2.3. Finally, we conclude with Section 3.2.4 and propose some further research ideas.

3.2.1 Problem Description

In graph theory an important and challenging open question is whether or not a *uniquely hamiltonian planar graph with minimum degree three (UHPG3)* exists [10]. Bondy and Jackson [10] conjectured that every planar uniquely hamiltonian graph contains a vertex of degree two, i.e., that no UHPG3 exists. So far, however, neither could a UHPG3 be found nor could it be proven that none exists.

Using Proposition 3.1.1 we see that to disprove Bondy and Jackson's conjecture, it would be enough to find a planar FEUHG with minimum degree three. We concentrate now on the optimization problem variant of finding a graph G with a given number of nodes $n = |V|$ that *as far as possible* corresponds to a UHPG3. To this end, we relax the conditions that the graph must be planar and must have minimum degree three and instead minimize the deviations from these properties.

To our knowledge, the problem of finding graphs that as far as possible correspond to a UHPG3 has so far not been considered in a more systematical, and in particular computational way.

More specifically, we consider the bi-objective optimization problem to find a *FEUHG* G together with a drawing in the plane and minimize

- the number of edge crossings and
- the number of vertices with degree two.

To obtain a single-objective optimization problem we linearly combine these two objectives with corresponding weights α and β.

Since the problem of computing the crossing number of a graph is $\mathcal{NP}$-hard (see [34]), it makes sense to approximate this value. We do this by allowing only crossings between edges that are not part of one fixed uniquely hamiltonian FE-cycle of the graph. This relaxation allows us to fix a hamiltonian FE-cycle in advance and then only concentrate

[1] http://www.math.uwaterloo.ca/tsp/concorde/ (accessed 01/2016)

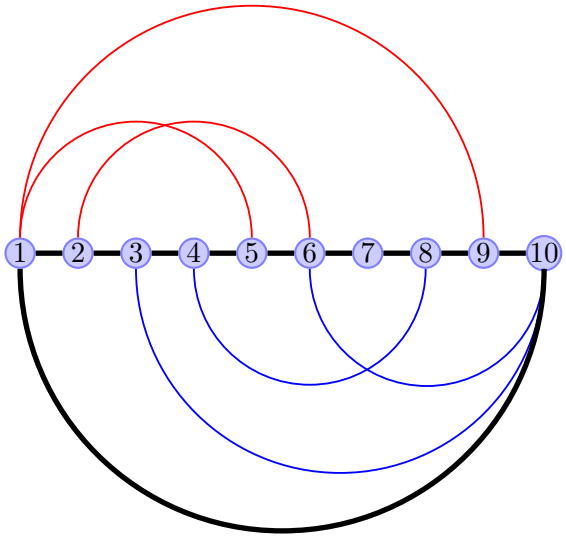


Figure 3.1: Example solution with two crossings and no nodes of degree two. Edges of the hamiltonian cycles are bold and chords are colored red and blue.

on the problem of adding additional edges (*chords*) between vertices that are no neighbors on the cycle. Since the added chords are not allowed to cross with an edge of the hamiltonian cycle, there are only two possibilities how to draw a chord: either outside the cycle or inside the cycle. We encode this two states, inside or outside, of a chord by two colors.

Let us assume without loss of generality that $V = \{1, \ldots, n\}$ and our predefined FE-cycle (C, e) visits the nodes in the natural order from 1 to n before getting back to 1. Furthermore, we assume without loss of generality that $e = \{1, 2\}$. If we fix the chords and their colors it is easy to derive the minimal number of crossings. To see this, we draw the nodes from 1 to n on one line, such that the hamiltonian cycle consists of this line together with a half circle connecting the vertices n and 1. We draw now every chord as a half circle, either above the line or below the line depending on the color of the chord. Figure 3.1 illustrates this construction with an example. For this construction we get that two chords (i, j) and (k, ℓ) with $1 \leq i < j \leq n$ and $1 \leq k < \ell \leq n$ cross if and only if their corresponding half circles cross. This is the case if and only if the two chords have the same color and

$$i < k < j < \ell \text{ or } k < i < \ell < j \tag{3.1}$$

holds. Notice that in any drawing of the graph, two chords, which are on the same side of the graph and satisfy (3.1), cross at least once. This implies the optimality of our construction.

All together we get the following optimization problem.

Problem 3.2.1. Given an integer $n \in \mathbb{N}$, let the vertex set be $V = \{1, \ldots, n\}$ and the FE-cycle (C, e) as described above. Furthermore, let

$$\mathcal{H} = \{\{i, j\} \mid i = 1, \ldots, n-2,\ j = i+2, \ldots, n\} \setminus \{\{1, n\}\}$$

be the set of all possible chords. A candidate solution is represented by (H, c), where $H \subseteq \mathcal{H}$ is the subset of selected chords and $c : H \to \{0, 1\}$ specifies their coloring by assigning either 0 or 1.

$$\begin{aligned} \underset{\substack{H \subseteq \mathcal{H} \\ c: H \to \{0,1\}}}{\text{minimize}} \quad & \alpha \cdot \sum_{\{i,j\} \in H} |\{\{k, \ell\} \in H : c(\{i, j\}) = c(\{k, \ell\}),\ i < k < j < \ell\}| \\ & + \ \beta \cdot \left(n - \left| \bigcup_{\{i,j\} \in H} \{i, j\} \right| \right) \end{aligned} \tag{3.2}$$

$$\text{s.t. there is no hamiltonian cycle in the graph } (V, E) \text{ with } E = C \cup H \text{ containing the edge } e = \{1, 2\} \text{ and at least one chord.} \tag{3.3}$$

Note that fixing the fixed edge $e = \{1, 2\}$ eliminates some symmetries. Be aware that a vertex has degree three or more if and only if it is incident to at least one chord in H. Therefore, the union in the second part of the objective expression (3.2) contains exactly all vertices of degree three or more. In the following we refer to Constraint (3.3) also simply as *unique hamiltonicity*.

The problem of checking if a given (planar) graph contains a hamiltonian cycle is $\mathcal{NP}$-complete. Therefore, the problem of checking if a given (planar) graph contains no hamiltonian cycle is in co-$\mathcal{NP}$. If $co - \mathcal{NP} \neq \mathcal{NP}$ this would also imply that the latter problem is not in $\mathcal{NP}$. Thus, checking only feasibility in our model is already a hard problem.

3.2.2 General Variable Neighborhood Search with Delayed Feasibility Checking

In this section we present our algorithmic approach to tackle the optimization problem described in Section 3.2.1. As mentioned checking only feasibility of an instance in our model is already a hard problem. Therefore, the use of heuristics, instead of an exact algorithm, appears appropriate. We propose a GVNS as framework to solve the problem heuristically [41] and combine it with Concorde for checking feasibility w.r.t. unique hamiltonicity. Remember that a solution is represented by the set of chords H and the associated coloring c for each chord in H. We start with the empty initial solution $H = \emptyset$, i.e., just the cycle C without any chords, as this is always a feasible solution.

The GVNS contains a VND for locally improving candidate solutions in systematic ways according to five different types of neighborhood structures, and a parameterized shaking neighborhood structure for diversifying the search. In the following we describe these neighborhood structures and the corresponding search algorithms.

VND Neighborhoods

The following types of neighborhoods and respective algorithms to search them are considered within the VND. From those neighborhoods different specific configurations are considered in the computational tests. All these neighborhoods are searched in a best-improvement fashion, and ties are broken randomly.

Changing color of k chords [cchol(k)]: This neighborhood consists of all solutions where the colors of k chords are flipped for some parameter $k \geq 1$. The size of the neighborhood is therefore m^k where m is the number of chords in the current solution. We apply this neighborhood structure for $k = 1$ and $k = 2$ since the neighborhood is relatively small and easy to search. Since only the colors of chords are changed, the structure of the solution graph stays the same, and therefore the unique hamiltonicity is still valid for each neighbor of a feasible incumbent solution. To calculate the objective gain of a neighbor incrementally we simply count the number of crossings with the old color and with the new color and take the difference.

Removing k chords and adding ℓ new chords [remadd(k, ℓ)]: This neighborhood consists of all solutions where exactly k chords are removed from the current solution and ℓ new chords are added. The size of the neighborhood is therefore $m^k(M-m)^\ell$ where m is the number of chords in the current solution and M is the number of possible chords ($M = (n-2)(n-1)/2 - 1$). For $\ell > 0$ the neighborhood may contain solutions that are infeasible as they are not uniquely hamiltonian. Instead of checking feasibility immediately for each neighbor, which can be time-expensive, we first evaluate all neighboring solutions according to our objective function, filter out any solutions that are not better than our incumbent, and sort all better solutions according to their objective function gain. Only then we consider these solutions according to decreasing gain, check the feasibility of each w.r.t. unique hamiltonicity, and immediately return with the first, and thus best, feasible solution. To calculate the objective gain of a neighbor incrementally we count the crossings of the removed edges and the crossings of the added edges and take the difference. Additionally, we have to check all vertices incident to removed and added edges if their degree decreased to two or increased to three or higher. In case there are multiple, i.e., equally good, best neighbors, all of these are checked for feasibility and one is chosen at random. This random tie breaking turned out to be crucial for the performance. The procedure of checking unique hamiltonicity gets described later on in this section. In case of $\ell = 0$ we do not need this check as a solution obtained from a feasible solution by just removing chords cannot become infeasible. The values used for k and ℓ gets described later.

Computing optimal crossings [compcross]: This is an exponentially large neighborhood consisting of all solutions that have the same underlying graph as the current solution but different colors on the chords. The size of this neighborhood is thus $2^m - 1$, where m is the number of chords in the current solution. Instead of a naive enumeration

we search this neighborhood in an effective way by a branch-and-bound procedure to obtain a best possible coloring of the chords. The crossings of the current solution can be used as a good initial upper bound, which is in many cases already tight. In every level of the search tree we assign one color to one chord. The sequence of chords is predefined randomly and used to determine which chord is colored next. As a branching strategy, we use depth-first search. To compute a local lower bound we use the crossings of the currently assigned chords and add for every not assigned chord the minimum of crossings to the assigned chords for the two colors. As already mentioned, finding the optimal crossings is an $\mathcal{NP}$-hard problem, but with the described branch-and-bound procedure it can be computed relatively fast compared to the effort that is needed to check for unique hamiltonicity.

Split crossings [splitcross]: This neighborhood is a special subset of the neighborhood remadd$(2, 2)$ that tries to resolve a crossing by splitting. More precisely for every crossing pair of chords $\{i, j\}$ and $\{k, \ell\}$ with $i < k < j < \ell$ we construct a new solution by removing these two edges and adding the chords $\{i, \ell\}$ and $\{k, j\}$ instead. There are two exceptions: if $j = k + 1$ the edge $\{k, j\}$ would be no chord and therefore this case is skipped. Similarly, case $i = 1$ and $\ell = n$ is excluded since these two vertices are already connected in the original hamiltonian cycle. If the chords $\{i, \ell\}$ or $\{k, j\}$ already exist in the current solution, this neighbor is skipped. If the newly added edges do not generate other crossings and the new graph is still uniquely hamiltonian we get a solution with one crossing less. The size of this neighborhood is equal to the number of crossings in the current solution and is therefore typically a small subset of remadd$(2, 2)$.

Merge crossings [mergecross]: This neighborhood is a special subset of the neighborhood remadd$(1, 2)$ that tries to resolve a crossing by merging the crossing point into one of the neighboring vertices. We generate up to four new solutions for every pair of crossing chords $\{i, j\}$ and $\{k, \ell\}$ with $i < k < j < \ell$ by applying the following operations:

1. Remove $\{i, j\}$ and add $\{i, k\}$ and $\{k, j\}$.
2. Remove $\{i, j\}$ and add $\{i, \ell\}$ and $\{j, \ell\}$.
3. Remove $\{k, \ell\}$ and add $\{i, k\}$ and $\{i, \ell\}$.
4. Remove $\{k, \ell\}$ and add $\{k, j\}$ and $\{j, \ell\}$.

Cases where the edges to be added do not correspond to valid chords or where they already exist in the solution are skipped again.

VND Neighborhood Selection

In our experiments in Section 3.2.3, we consider three different configurations of VND neighborhood, which are shown in Table 3.1. The VND considers the stated specific neighborhoods in the listed order.

Table 3.1: Three different neighborhood structure sets used to compare with each other.

slim set	medium set	thick set
1. cchol(1)	1. cchol(1)	1. cchol(1)
2. cchol(2)	2. cchol(2)	2. cchol(2)
3. remadd(1, 0)	3. remadd(1, 0)	3. remadd(1, 0)
4. remadd(0, 1)	4. remadd(0, 1)	4. remadd(0, 1)
	5. splitcross	5. splitcross
	6. mergecross	6. mergecross
	7. compcross	7. compcross
	8. remadd(1, 1)	8. remadd(1, 1)
	9. remadd(2, 1)	9. remadd(2, 1)
		10. remadd(2, 2)

It is important to notice that all neighborhoods choose the candidates randomly if their improvements are the same. This implies that the slim neighborhood set is still capable of reaching all feasible solutions. The neighborhoods that do not require rechecking unique hamiltonicity are listed before the expensive neighborhoods that require rechecking. The only exception to this rule is the compcross neighborhood, which may also need significant time for larger graphs as it solves an $\mathcal{NP}$-hard problem by branch-and-bound.

Shaking

To diversify the search, the GVNS applies the following shaking operation parameterized by $k \in \{3, \ldots, \lfloor \frac{n}{2} \rfloor\}$. Considering the chords in an order of non-increasing number of crossings, k chords are deleted from the current solution. If there are less than k chords in the current solution we delete them all. Be aware that there are in general multiple chords with the same number of crossings (most of them have no crossings in a good solution), and they are then considered in a random order. Thus, there also is a significant randomization involved. Since we do not know in advance how many chords a solution has, it is so far not guaranteed that in principle our GVNS can reach every possible solution from an incumbent solution. Therefore, we add as last shaking operation the removal of all chords. In other words this last shaking operation corresponds to a complete restart of the GVNS, and we trust on the VND to add chords again.

Thus, our shaking neighborhoods do not contain the complete solution space but rather a cone of all solutions we can get by removing chords from the current solution. Just removing chords in the shaking has the advantage that we never violate unique hamiltonicity, and thus we always get a feasible solution without overhead. It would be difficult to generate a random solution in the complete feasible solution space since we would have to check for unique hamiltonicity until we find a solution satisfying it. This would cost a lot of time, which is not our intention behind shaking. However, our overall approach guarantees that every feasible solution can in principle be found by descending from one

of the solutions generated by shaking. This can easily be seen by the fact that every solution can be constructed by adding chords from the empty solution.

Checking Unique Hamiltonicity

In this section we describe the procedure we apply to check if a given solution is uniquely hamiltonian, i.e., satisfies (3.3). The running time of this procedure is crucial for the success of the algorithm since it is the bottleneck of the GVNS as we will see in the experimental results in Section 3.2.3.

As we only check unique hamiltonicity for neighbors that would improve the current solution we can descend to the neighbor whenever the condition is satisfied. This means that the number of procedure calls where the condition is satisfied corresponds to the number of local improvements. However, it is possible that any number of neighbors get checked before one is found that satisfies condition (3.3). Therefore, we do not have a better bound on the number of negative procedure calls than the current neighborhood size.

Since the hamiltonian cycle problem is a special case of the well studied TSP, there already exist a lot of practically well performing algorithms to approach the hamiltonian cycle problem. To model the hamiltonian cycle problem as a TSP one simply assigns all pairs of nodes corresponding to edges in the graph, i.e., E, unit costs and all other pairs of nodes larger costs. The question whether or not a hamiltonian cycle exists is then equivalent to the question whether or not a tour with costs $|V|$ exists. If we want to fix a subset of edges $E' \subseteq E$, then we can give them zero costs. The question if a hamiltonian cycle containing all edges in E' exists is then equivalent to the question if a tour with costs $|V| - |E'|$ exists. Thus, we can also model a fixed edge hamiltonian cycle problem.

To check condition (3.3), more specifically we need to check if a hamiltonian cycle containing edge $(1, 2)$ and edge e for any chord $e \in H$ exists. This means, we have to solve $|H|$ TSPs before we know for sure that condition (3.3) is satisfied. If at some point we find a hamiltonian cycle we can stop and know that the condition is not satisfied.

Clearly, solving $|H|$ TSPs would be very time-expensive. Fortunately, we can apply an improvement in our case that takes into account that the considered candidate solution graph is a neighbor of our current solution for which we already know that it satisfies condition (3.3). Remember that the only situation where we have to recheck the condition is for neighbors where we removed k chords and added $\ell > 0$ new chords. In this situation it is sufficient to check if there exists a hamiltonian cycle containing the edge $(1, 2)$ and at least one of the *newly added* chords. Therefore, we only have to solve ℓ TSPs to perform this check.

As we already mentioned there may be many more negative procedure calls than positive ones and therefore we need a solver that is able to find hamiltonian cycles fast. Thus, we decided to use a heuristic to find hamiltonian cycles. We use Helsgaun's version of the LKH, which is faster than exact approaches but can still solve many problems to

optimality [44]. If one run of the LKH does not find a hamiltonian cycle we apply ten further runs of the heuristic. If it still does not find a hamiltonian cycle we assume that the graph does not contain anyone. To avoid that the algorithm returns an infeasible solution as an optimal solution at the end we use Concorde to check for hamiltonian cycles. As Concorde needs much more time than the LKH, we only call Concorde whenever a neighbor would lead to a new best solution and the LKH did not find a cycle in any run.

This means it may happen that an infeasible neighbor gets visited if it is no new best solution. In this case LKH calls, where we assume that the current solution is feasible, are not correct anymore. This is no problem since as soon as the search visits a neighbor that would be a new best solution, Concorde gets applied and shows that the neighbor is infeasible. As we will see in Section 3.2.3 this situation almost never happens for small vertex degrees.

Further Improvements. To further improve the running time of checking unique hamiltonicity we exploit the fact that only small parts of the graph change when doing local improvements. Obviously the same hamiltonian cycles may frequently appear in the investigated graphs having only small differences. The idea is now to store found hamiltonian cycles in an appropriate data structure that allows us to check for a new graph if it contains a cycle from the data structure quickly. If searching through this data structure can be done reasonably fast, this will improve the overall running time. We can represent a cycle or a whole graph by the set of its edges. Thus, we need a data structure that stores sets and can compute subset queries quickly. This problem is known as the containment query problem [17]. It is the complementary problem of the better known subset query problem, which furthermore corresponds to the well-known partial match problem [48].

For our purposes we used a trie data structure presented in [5]. Note that we only store the set of chords used in the hamiltonian cycle. Checking if a subset exists in such a data structure needs exponential time in the worst case. Nevertheless, it is still much faster to search in this data structure than searching a hamiltonian cycle in practice, as we will see in Section 3.2.3. There would also be more sophisticated data structures for storing sets (see for Example [43]). In our case we do not need such complex data structures since our practical tests indicated that the simple data structure's time consumption is almost neglectable in comparison to the effort for finding hamiltonian cycles in the remaining cases.

3.2.3 Computational Results

Our GVNS approach is implemented in C++ and compiled with g++ 4.8.4. We used the LKH implementation. For heuristically searching for hamiltonian cycles we use LKH[2]. The Concorde implementation[3] uses CPLEX[4] 12.6.2 for solving. All tests were performed

[2]http://www.akira.ruc.dk/~keld/research/LKH/ (accessed 01/2016)
[3]http://www.math.uwaterloo.ca/tsp/concorde/ (accessed 01/2016)
[4]https://www.ibm.com/analytics/cplex-optimizer (accessed 01/2016)

Table 3.2: Results for the three different configurations.

	slim set			medium set			thick set		
n	*best*	*avg.*	$t[s]$ *med.*	*best*	*avg.*	$t[s]$ *med.*	*best*	*avg.*	$t[s]$ *med.*
10	**0.50**	**0.50**	56.83	**0.50**	**0.50**	37.77	**0.50**	**0.50**	**21.33**
15	**0.50**	**0.50**	5.45	**0.50**	**0.50**	**4.92**	**0.50**	**0.50**	15.77
20	**0.50**	**0.50**	13.23	**0.50**	**0.50**	**12.29**	**0.50**	**0.50**	21.71
25	**0.50**	**0.50**	50.25	**0.50**	**0.50**	**48.92**	**0.50**	**0.50**	82.42
30	**0.50**	**0.50**	234.60	**0.50**	**0.50**	**75.03**	**0.50**	**0.50**	301.92
35	**0.50**	**0.50**	**139.81**	**0.50**	**0.50**	209.68	**0.50**	**0.50**	442.44
40	**0.50**	**0.50**	369.86	**0.50**	**0.50**	**277.22**	**0.50**	0.60	426.32
45	**0.50**	0.55	1020.10	**0.50**	**0.53**	714.29	**0.50**	0.70	1510.86
50	**0.50**	0.74	144.20	**0.50**	**0.56**	1007.63	**0.50**	0.81	321.41
60	**0.50**	0.93	19.52	**0.50**	**0.82**	43.22	**0.50**	0.94	99.78
70	**0.50**	0.97	35.76	**0.50**	**0.76**	238.53	**0.50**	1.00	306.56
80	**0.50**	0.96	71.46	**0.50**	**0.90**	68.08	1.00	1.20	564.71
90	0.75	1.04	128.52	**0.50**	**0.95**	130.56	1.00	1.46	1260.14
100	1.00	1.00	183.33	**0.50**	**0.96**	173.95	**0.50**	1.66	570.27

on a single core of an Intel Xeon E5540 processor with 2.53 GHz and 10 GB RAM. The input of our algorithm is simply the number of vertices $n \in \mathbb{N}$. For all tests we used the weights $\alpha = 0.25$ and $\beta = 1$ (see (3.2)). This implies that all solution graphs with an objective value smaller than 1 have minimum degree three.

As the instances we used different n values between 10 and 100. We ran all three configurations (see Table 3.1) for every instance 20 times with different seed values and a maximal execution time of 3600 seconds per run. In Table 3.2 we see the results for the three different configurations for different instances n. The columns *best* contain the best value found in all 20 runs for one configuration. The columns *avg.* contain the averages of the results over the 20 runs and the columns $t[s]$ *med.* contain the medians of the times until the best solution was found in each run in seconds. For every instance and every of the three column types we marked the best value of the three configurations by displaying it bold.

As we can see, on average, the medium set found better solutions than the other two configurations. Note that the running times until the best solution was found are only comparable if the corresponding solutions are equally good. Therefore, it makes no sense to compare the time medians for the instances with $n \geq 45$. To verify that the medium solution performs better than the other two solutions we applied a Wilcoxon signed-rank test. We use a p-value of 5% for the significance value. As a result, we get that the medium set computed for the instances $n = 50, 60, 70$ significantly better results than the slim set and for all instances with $n \geq 40$ significantly better results than the thick set. For all other instances the difference was not significant. We also compared the

slim set with the thick set and interestingly the slim set computed for the instances $n = 40, 45, 80, 90, 100$ significantly better solutions than the thick set.

To compare the running times until the best solution was found we also applied the test for the running times, but only for the instances with $n \leq 40$. If we compare the medium set and the slim set there is only for the instance $n = 30$ a significant difference, where the medium set is significantly faster than the slim set. If we compare the slim or the medium set with the thick set, we get that both are for the instances with $15 \leq n \leq 35$ significantly faster than the thick set. For $n = 10$ the thick set is the fastest on average and compared to the slim set it is also significantly faster.

From these tests we get the intuition that the neighborhood remadd(2,2) is toolarge and not beneficial for graphs of medium size or larger sizes. Only for graphs with size around $n = 10$ the neighborhood is small enough to be beneficial. We need, however, more neighborhoods than only adding and removing chords, as in the slim set, to find good solutions for larger instances. It is also interesting that the slim set has no significant speed gain compared to the medium set. From the *best* column of the medium set we see that we found a solution with an objective of 0.5 for all given instances. This means that these solution graphs have minimum degree three and exactly two crossings in the drawing. We also applied some test runs for all other n values between 10 and 100 and found a solution with an objective of 0.5 for every $n \in \{10, \ldots, 100\}$.

Note that the solutions of our problem, as we stated it, are not uniquely hamiltonian, they are FEUHG. That means to get a uniquely hamiltonian graph we need to duplicate it as described in Proposition 3.1.1, but then the crossings get also duplicated. Therefore, all our solutions with an objective of 0.5 induce uniquely hamiltonian graphs with minimum degree three that have embeddings with four crossings. Therefore, our results impose the following question.

Question 3.2.1. Does there exist a uniquely hamiltonian graph with minimal degree three and a crossing number smaller than four?

Table 3.3 contains the number of performed local search improvements for each neighborhood and each instance with the medium set configuration. The numbers are averaged over all 20 runs with 20 different seeds. The column name $ra(x, y)$ stands for remadd(x, y), *splitcr* for splitcrossing, *mergecr* for mergecrossing and *compcr* for compcrossing.

Since the shake operations only remove chords, the neighborhood remadd$(0, 1)$, which only adds one chord, is applied after shaking several times. This explains why this neighborhood is used much more often than the other neighborhoods.

To find out which part of the algorithm consumes the most time we can identify three subroutines that have an exponential worst case running time. The first one is Concorde, which gets applied whenever the search finds a new best solution for which the LKH did not find a second hamiltonian cycle. The second one is the solver of the containment query problem, which uses a trie data structure to check if a cycle that was already found

Table 3.3: Number of improvements found in the different neighborhood structures for the medium set configuration.

n	*cchol*(1)	*cchol*(2)	*ra*(1, 0)	*ra*(0, 1)	*splitcr*	*mergecr*	*compcr*	*ra*(1, 1)	*ra*(2, 1)
10	2667	60	0	314 452	677	197	0	3898	4
15	6551	3872	18	279 279	522	856	3126	3202	204
20	3904	1887	19	202 046	1599	1371	1732	2211	128
25	2676	1179	26	133 997	1293	1287	1311	1675	86
30	1828	646	34	92 814	1201	1152	895	1334	63
35	1257	501	23	63 717	1013	958	550	1054	44
40	999	375	24	48 623	844	828	482	869	38
45	756	248	32	36 387	769	770	284	794	30
50	595	193	20	28 707	661	640	232	656	26
60	365	107	25	18 612	553	564	99	559	17
70	264	75	15	12 953	398	410	77	390	12
80	201	53	17	9572	330	336	50	328	9
90	147	37	17	7184	258	283	39	263	8
100	123	29	10	5891	221	229	27	212	5

is contained in the current candidate. The third one is the branch-and-bound procedure to calculate the optimal colors of the chords such that they have a minimal number of crossings.

Table 3.4 lists running time information and additional information for these three subroutines and the LKH subroutines with the medium set configuration. The column *HC-Checks* contains the number of graphs for which we had to test the unique hamiltonicity constraint and the *UHG* column contains the number of graphs that satisfied the unique hamiltonicity constraint. The column *calls* contains the number of Concorde calls and the column *rate* contains the number of graphs that got discarded by a containment query relative to the overall number of discarded graphs. The columns $t[s]$ contain the overall time used for all corresponding calls in seconds. The time columns represent median values and the other columns represent average values over all different seeds. As we can see the three mentioned subroutines are in total extremely fast compared to the LKH subroutines, which have to get applied often. Therefore, the LKH subroutines are clearly the bottleneck of the whole GVNS. Another interesting fact is that Concorde never found a hamiltonian cycle in any of the runs. This means that whenever LKH did not find a second hamiltonian cycle the graph was in fact uniquely hamiltonian.

The fact that the rates of the containment queries increase with increasing n can be explained as follows. First of all, for two solutions that are similar it is more likely that they both contain the same hamiltonian cycle than for solutions that are completely different. This implies that as long as the search is concentrated in a local area the containment queries are effective. Only the shaking methods guide the search out of such

Table 3.4: Hard subproblem statistics for the medium set configuration.

	LKH			Concorde		Containment Queries		B&B
n	*HC-Checks*	*UHG*	$t[s]$	*calls*	$t[s]$	*rate*	$t[s]$	$t[s]$
10	1 467 893	255 790	1356	7	0	4.8%	3	0
15	1 607 901	258 962	2480	10	0	5.5%	4	0
20	1 456 778	200 372	3230	14	0	7.5%	7	0
25	1 151 036	137 859	3421	17	0	10.1%	8	0
30	929 408	99 379	3490	20	1	12.3%	9	0
35	761 117	70 547	3512	24	1	14.5%	10	0
40	653 680	55 017	3518	26	1	17.2%	11	0
45	570 683	42 607	3521	30	1	19.3%	12	0
50	493 993	34 275	3523	33	2	21.3%	13	0
60	425 704	23 800	3523	40	3	28.5%	14	0
70	323 338	16 864	3528	46	5	30.6%	14	1
80	277 038	12 984	3525	52	9	34.2%	13	1
90	246 591	10 072	3515	60	11	39.1%	17	2
100	204 408	8324	3510	65	14	41.3%	21	4

a local area. The simple fact that for larger n the search in the local neighborhoods needs longer implies that it can do less shaking than for smaller n. Therefore, the containment queries are more effective for larger n.

We want to mention that we tested the algorithm also for $n > 100$. For these instances the running times of the subproblems exploded and the GVNS could do only few iterations in reasonable time, which lead to poor quality solutions.

3.2.4 Conclusions and Future Work

In this section we presented a new optimization problem for finding uniquely hamiltonian graphs of minimum degree three with small crossing numbers. We proposed a GVNS framework to solve the problem heuristically. We applied different neighborhood structures including a large one and different configurations, which we compared in experimental tests. The bottleneck of the proposed algorithm is checking unique hamiltonicity for every neighbor, to stay in the feasible area. With an implementation of this framework we were able to find uniquely hamiltonian graphs of minimum degree three with only four crossings for many different instances, which naturally gives rise to the question if we can do better.

Future work may be to develop an exact algorithm for the proposed problem and compare the two algorithms for small instances. One problem of the proposed heuristic is that the constraint of unique hamiltonicity is completely independent of the objective function. If we could measure how promising a graph is according to the unique hamiltonicity

constraint, we could better guide the search. Furthermore, it would be interesting to test if other heuristics than the LKH or other variants of the LKH for checking unique hamiltonicity perform better.

3.3 Searching for Stable Cycles

In this section we present approaches to systematically search through planar graphs up to a certain order to either find a counterexample to the conjecture of Bondy and Jackson or to establish a new lower bound for the order of a UHPG3. A lower bound of 18 vertices was proven recently by Goedgebeur et al. [36] who developed a construction procedure for generating all non-isomorphic graphs with a given number of hamiltonian cycles, which works especially well for a small number of cycles like in the case of uniquely hamiltonian graphs. As we will see in Theorem 3.3.16, we improve this lower bound to 25 vertices.

Instead of directly searching for uniquely hamiltonian planar graphs we focus in this section on searching for a planar graph with minimum degree three containing a so-called stable fixed edge cycle (SFE-cycle). In Section 3.3.1 we show that such a graph would imply the existence of a UHPG3. We then can prove strong properties for a minimum planar graph with minimum degree three that contains an SFE-cycle. This helps us to search for such a graph more effectively.

Our overall approach is to use the tool plantri [12] to generate representants of all isomorphism classes of planar graphs with minimum degree three of some given order. Then we check for each of these graphs if they contain an SFE-cycle or not. To that end we formulate three problem variants that decide for a given input graph if it contains an SFE-cycle, a dominating SFE-cycle, or a dominating SFE-cycle with some additional properties, respectively. The third problem variant is used for checking if a graph is a potential minimum counterexample. We prove in Section 3.3.3 that the stable cycle of a minimum counterexample must satisfy those additional properties. In Section 3.3.3 we also prove some properties a graph must satisfy to be a minimum counterexample, one of which is the strong property of triangle freeness. To effectively generate such candidate graphs we use plantri to construct dual graphs with minimum degree four.

For solving the three problem variants we propose an ILP-based approach and an enumeration scheme. Using both approaches helps us to verify the results and allows us comparing the performances for different graph sizes. As already mentioned the main result of this section is a new lower bound of 25 vertices for the smallest UHPG3.

In the following we focus only on planar graphs. In Section 3.3.1 we present the reductions from uniquely hamiltonian to stable cycles and the first two of the three problems that we consider. Two different solution approaches for solving the problems are then presented in Section 3.3.2. In Section 3.3.3 we focus on properties a minimum counterexample must satisfy and formulate the third problem variant. We then apply our algorithms presented

in Section 3.3.2 on different instance groups and summarize the results in Section 3.3.5. Finally, we conclude with Section 3.3.6 and propose some further research ideas.

3.3.1 Reduction to Stable Fixed Edge Cycles

In this section we reduce the search for uniquely hamiltonian planar graphs with minimum degree three to the search for planar graphs with minimum degree three with an SFE-cycle.

We start with defining what we mean by an SFE-cycle continuing Definition 3.1.2.

Definition 3.3.1. Let G be a graph. An FE-cycle (C, e) is called *maximal* if there is no FE-cycle (C', e) in G with $V(C') \supsetneq V(C)$. Moreover, an FE-cycle (C, e) is called *dominating* if C is edge-dominating in G, i.e. for all edges $e = vw \in G$ either v or w is in $V(C)$. Finally, a maximal FE-cycle (C, e) is called a *stable fixed edge cycle (SFE-cycle)* if there is no other FE-cycle (C', e) with $C' \neq C$ and $V(C') = V(C)$.

We call a graph containing an SFE-cycle a *stable fixed edge graph (SFE-graph)*.

In the following result we present the transformation by Leder [59] that transforms a graph containing a dominating SFE-cycle into an FEUHG preserving planarity.

Theorem 3.3.1. *Let G be a graph with minimum degree three containing a dominating SFE-cycle (C, e). Then there exists an FEUHG G' with minimum degree three whose uniquely hamiltonian FE-cycle satisfies $|V(G')| \leq 2|V(G)| - |V(C)|$. Furthermore, if G is planar, then G' is also planar.*

Proof. We apply iteratively a planarity preserving transformation that constructs from a graph G with a dominating SFE-cycle (C, e) a new graph G' with a dominating SFE-cycle (C', e') such that the number of unvisited vertices by the cycle is reduced by one. The order of the new graph is at most increased by one per iteration. Let $w \in V(G) \setminus V(C)$ be an unvisited vertex. If w has no neighbors of degree three, we can just remove the vertex w and define $G' = G \setminus v$ and $(C', e') = (C, e)$.

In the other case let w_1 be a neighbor of degree three. If w_1 is the only neighbor of degree three we connect w_1 with any other neighbor w_2 of w and define $G' = G \setminus v \cup (w_1, w_2)$ and $(C', e') = (C, e)$. Any cycle in G' that contains the edge (w_1, w_2) can be translated to a cycle in G containing the vertex w. Therefore, (C, e) being stable implies that (C', e') is stable.

If w has exactly two neighbors of degree three w_1 and w_2 we can connect those two and remove the vertex w, i.e. $G' = G \setminus v \cup (w_1, w_2)$, and $(C', e') = (C, e)$. Again any cycle in G' that contains the edge (w_1, w_2) corresponds to a cycle in G containing w and therefore this implies that (C', e') is stable.

The most interesting case is if w has at least three neighbors of degree three. Let e_1 and e_2 be the other two edges incident to w_1 that are not incident to w. Let $x_1 \neq w$ and $x_2 \neq w$ be the other end vertex of e_1 and e_2. We remove now the vertex w and connect

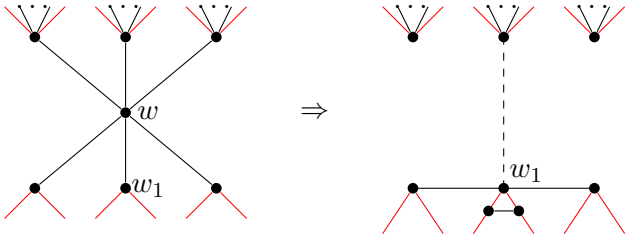


Figure 3.2: Transformation used for removing unvisited vertices. The dominating cycle is visualized by the red edges. The dashed edge is used if there is only one degree three neighbor and the two new vertices are only inserted if there are three or more degree three neighbors.

all other neighbors of w with degree three to w_1. To preserve the stability of the cycle we need to add two more vertices v_1 and v_2 placed on the edges e_1 and e_2 and connect them, see Figure 3.2. Formally we define

$$G' = G \setminus \{w, e_1, e_2\} \cup \{v_1, v_2, (v_1, v_2), (w_1, v_1), (w_1, v_2), (v_1, x_1), (v_2, x_2)\} \bigcup_{w' \in N(w): w' \neq w_1, \deg(w')=3} (w', w_1)$$

and $C' = C \setminus \{(w_1, x_1), (w_1, x_2)\} \cup \{(w_1, v_1), (w_1, v_2), (v_1, x_1), (v_2, x_2)\}$. If the fixed edge was e_1 or e_2 we define as fixed edge $e' = \{w_1, v_1\}$ respectively $e' = \{w_1, v_2\}$, otherwise $e' = e$.

A cycle in G' that uses exactly one edge between two neighbors of w can be transformed to a cycle in G that visits w. Furthermore, a cycle in G' that uses two edges between two neighbors of w, i.e. visits w_1 in between those two edges, and still visits the new vertices v_1 and v_2 can again be transformed to a cycle in G that visits w. Therefore, one can verify that the stability of (C, e) in G and its implied maximality imply that the new FE-cycle (C', e') is also stable.

Note that the transformations in all three cases above are planarity preserving, see also Figure 3.2. Furthermore, the first cases only remove one vertex and only the last case removes one vertex and adds two new vertices, which results in $|V(G')| \leq |V(G)| + 1$. Applying this transformation in sequence to all unvisited vertices $V(G) \setminus V(C)$ in G gives us then a graph G' and an SFE-cycle (C', e') that is hamiltonian. We note now that a hamiltonian SFE-cycle is nothing else than a uniquely hamiltonian FE-cycle. □

Proposition 3.1.1 and Theorem 3.3.1 together give us that if we find a planar graph with minimum degree three having a dominating SFE-cycle this would disprove the conjecture by Bondy and Jackson. To search for such graphs we want to develop algorithms that can check for a graph if it contains a dominating SFE-cycle, i.e. we want to solve the following problem.

Problem 3.3.1. Given a graph G, does G contain a dominating SFE-cycle?

But we can go one step further and even relax the dominating condition. To do that we need to restrict us to 2-connected graphs, which is not really a restriction since every graph with minimum degree 3 with a dominating cycle must be 2-connected. The next theorem presents a planarity-preserving transformation from a 2-connected SFE-graph with minimum degree three to a graph with minimum degree three that contains a dominating SFE-cycle.

Proposition 3.3.2. *Given a 2-connected graph G with minimum degree three that contains an SFE-cycle C. Then there exists a 2-connected graph G' with minimum degree three that contains a dominating SFE-cycle and $|V(G')| \leq |V(G)|$. If C is not dominating then $|V(G')| < |V(G)|$. Furthermore, if G is planar then G' is planar.*

Proof. If C is dominating we can choose $G' = G$ and are done. Otherwise, let H be a non-trivial connected component of $G \setminus V(C)$. We contract now H in G to one vertex w. The 2-connectedness of G implies now that w has degree at least two. If the degree is two we can replace w and its two incident edges by one edge and get again a graph with minimum degree three. Furthermore, every cycle in this new graph that contains this new edge can be transformed to a cycle in G that contains a path through the connected component of H. If the degree of w is larger than three we get also a graph with minimum degree three and again every cycle that contains w in the new graph can be transformed to a cycle in the original graph containing a path through the connected component of H. This implies that the FE-cycle (C, e) is in both cases also stable in the new graph.

Applying the above procedure for every non-trivial connected component of $G \setminus V(C)$ we get a graph where (C, e) is dominating and still stable. Since we only contracted connected subgraphs in the above procedure, we get that $V(G') < V(G)$ and that G' is planar if G is planar. □

Let us consider now the following conjecture.

Conjecture 3.3.1. There does not exist a 2-connected planar SFE-graph with minimum degree three.

Proposition 3.1.1, Theorem 3.3.1, and Proposition 3.3.2 imply the following theorem.

Theorem 3.3.3. *Conjecture 3.3.1 is equivalent to Bondy and Jackson's conjecture.*

To search for such graphs we need to check if a given graph is an SFE-graph, i.e. solve the following problem.

Problem 3.3.2. Given a 2-connected graph G, is G an SFE-graph?

3.3.2 Algorithmic Approaches

In this section we present two approaches, one based on ILP and one based on cycle enumeration. For the ILP-based approach we present a basic method and some algorithmic improvements for solving it. The algorithmic improvements and also the cycle enumeration approach use a data structure for storing and querying the found cycles, which we also present in this section. Nevertheless, the problem can also be solved with the basic ILP approach, which does not use the data structure. This is important for verifying the results independently.

Basic ILP Approach

We simply check for each edge $e_0 \in E(G)$ if there is any SFE-cycle with the fixed edge e_0. To do that we search for a maximal cycle containing the edge e_0, avoiding all cycles found until now. Whenever we find an FE-cycle we check if it is stable or not. This procedure is repeated until no new FE-cycle is found.

An ILP model is used for finding new cycles and another one for checking if a cycle is stable. We denote the set of all already found cycles for the edge e_0 by $\mathcal{C}$. Moreover, let w be a fixed end vertex of e_0. The following ILP model is for finding new cycles.

$$\max \sum_{v \in V(G)} x_v \tag{3.4}$$

$$\text{s.t.} \sum_{e \in E(v)} y_e = 2x_v \qquad \forall v \in V(G) \tag{3.5}$$

$$\sum_{e \in E[V', V(G) \setminus V']} y_e \geq 2x_v \qquad \forall \emptyset \neq V' \subseteq V(G) \setminus \{w\}, v \in V' \tag{3.6}$$

$$y_{e_0} = 1 \tag{3.7}$$

$$\sum_{v \in V(G) \setminus V(C)} x_v \geq 1 \qquad \forall C \in \mathcal{C} \tag{3.8}$$

$$y_e, x_v \in \{0, 1\} \qquad \forall e \in E(G), \forall v \in V(G) \tag{3.9}$$

The boolean x-variables indicate if a vertex is part of the cycle or not and the boolean y-variables if an edge is part of the cycle or not. To ensure that each found cycle is maximal we maximize the number of vertices in the cycle, see (3.4). We use here a classical approach for modeling cycles in an undirected version: For eliminating subtours we use cut constraints originally proposed for the TSP by Dantzig et al. [21], see (3.6). As it is standard with this approach the exponential many subtour elimination constraints get added to the program dynamically in a branch-and-cut manner using a fast max-flow algorithm for separating cuts in the LP-relaxation. Constraint (3.7) ensures that e_0 is part of the cycle and constraints (3.8) guarantee that we find a new maximal cycle that was not found before.

Note that for similar problems models based on directed edge variables frequently result in better performance than the undirected variant, since the directed variant has a tighter LP-relaxation [81]. However, as we will see in Section 3.3.5 the graph sizes we are mainly concerned with are so small that the ILP models get solved fast, and our experiments indicated that the overhead of a directed model does not pay off in our case.

For each cycle C we find with this model we solve the following ILP model to check if the cycle is stable. This time we do not need the x-variables since we already know which vertices must be part of the cycle. Therefore, we can use any model for finding hamiltonian cycles, such as the following. Let $G' = G[V(C)]$ be the graph induced by the vertex set $V(C)$.

$$\max 0 \tag{3.10}$$

$$\text{s.t.} \sum_{e \in E(v) \cap E(G')} y_e = 2 \qquad \forall v \in V(C) \tag{3.11}$$

$$y_{e_0} = 1 \tag{3.12}$$

$$\sum_{e \in E[V', V(C) \setminus V']} y_e \geq 2 \qquad \forall \emptyset \neq V' \subseteq V(C) \setminus \{w\} \tag{3.13}$$

$$\sum_{e \in E(G') \setminus E(C)} y_e \geq 2 \tag{3.14}$$

$$y_e \in \{0, 1\} \qquad \forall e \in E(G') \tag{3.15}$$

Note that this model has no meaningful objective function as we are interested just in the fact whether or not a feasible solution exists. We use again cut constraints for eliminating subtours, see (3.13). To ensure that the cycle is different from the already found cycle C we ensure with Constraint (3.14) that the new cycle contains edges that do not appear in the old cycle.

This basic approach needs to solve many ILP models to check if a given graph contains an SFE-cycle. Although the basic approach has an overhead for each ILP model it needs to solve, it is still important since we can use it to check our results and especially to verify the correctness of the enumeration approach.

To improve this basic approach we want to add in an advanced approach constraints like (3.8) dynamically. This results in cycles that may not be maximal. The following data structure helps us to store cycles and detect non-maximal cycles.

Data Structure For Storing Cycles

To be able to check if a new found cycle C is a potential maximal cycle for some fixed edge we need to check if there exists for each edge $e \in E(C)$ an already found cycle C' that contains e and $V(C) \subsetneq V(C')$. To that end we store each cycle we find in an appropriate

data structure. Furthermore, the data structure helps us to search for SFE-cycles in the end.

The above query problem is an extension of the containment query problem, which is a widely studied problem [17, 72, 5]. For every known data structure for these problems it holds that either the query time is in $\Omega(N)$ or storage size is in $2^{\Omega(n)}$ where N is the number of stored sets and n is the alphabet size, which is in our case the number of vertices of the graph. If this must hold is an open question, although some theoretical lower bounds for computation times or storage size got already established [48].

Note that we already considered the containment query problem in the context of finding uniquely hamiltonian planar graphs with small crossing numbers, see the end of Section 3.2.2. This time we are dealing with an extension of this problem and therefore we focus on a simple solution based on the Hasse diagram [68]. Furthermore, since we build up the data structure during the execution of our algorithm we have to balance the query times and the size of the data structure. Because we are searching for maximal cycles and in the case of Problem 3.3.1 for dominating cycles the vertex complements of such cycles tend to be small sets. Therefore, we focus on the vertex complements $V(G) \setminus V(C)$ in our data structure. The worst case size of the Hasse diagram is $\mathcal{O}(N^2)$, which is in our case normally much smaller than $2^{\Omega(n)}$, especially if we only consider dominating cycles.

Additionally, to the Hasse diagram of all vertex complements of found cycles we store a hash map mapping vertex complements X and edges e to the number of found cycles $m(X, e)$ with vertex complement X containing the edge e. If $m(X, e) \geq 2$ we know that cycles with the vertex complement X cannot be SFE-cycles for the edge e. Furthermore, we use a special value of $m(X, e) = \infty$ if there exists a cycle with a smaller vertex complement $Y \subsetneq X$ that contains e, in this case no cycle with the vertex complement X can be a maximal FE-cycle for the edge e.

For each new cycle C we first check if its vertex complement X is already in the Hasse diagram, if not we add it. Then we adapt $m(X, e)$ for all edges e in the cycle. Furthermore, we need to search for all $Y \subsetneq X$ inside the Hasse diagram and set $m(X, e) = \infty$ for all e where $m(Y, e) \neq 0$. Note that we only need to search for direct neighbors in the Hasse diagram since if $Y_1 \subsetneq Y_2 \subsetneq X$ we know that $m(Y_1, e) \neq 0$ implies $m(Y_2, e) = \infty \neq 0$. Last but not least, we need to search for all $Z \supsetneq X$ inside the Hasse diagram and set $m(Z, e) = \infty$ for all edges $e \in E(C)$.

Whenever we have a vertex complement X for which all values of $m(X, e)$ for all edges e are either 0 or ∞ we can remove it, since the corresponding cycles are all not relevant anymore. This helps us to keep the size of the data structure small, which is relevant if we find many not maximal cycles. After we added all found cycles we can use this data structure to check if there is any SFE-cycle by simply checking if for any vertex complement X and edge e we have $m(X, e) = 1$.

Advanced ILP Approach

We use now the previously presented data structure to improve the performance of our ILP approach. First of all we do not want to search for cycles for each edge e separately, since we can reuse a cycle for all its edges. To that end we split the approach into two phases, the first phase searches for maximal cycles and the second phase checks if one of them is an FE-cycle for some fixed edge e. All cycles we find in the first phase get added to the data structure. The second phase searches then the data structure for cycles C with $m(X(C), e) = 1$ and checks if (C, e) is an SFE-cycle in the same way as we presented for the basic ILP approach.

To search for new maximal cycles for arbitrary fixed edges we add new boolean variables z_e for $e \in E(G)$ to our model that describe that the edge e is a potential fixed edge. We then replace the constraints (3.7) by the following constraints where $\mathcal{C}$ represents the set of all so far found cycles:

$$z_e \leq y_e \qquad \forall e \in E(G) \tag{3.16}$$

$$\sum_{e \in E} z_e \geq 1 \tag{3.17}$$

$$\sum_{v \notin V(C)} x_v \geq z_e \qquad \forall C \in \mathcal{C}, \forall e \in E(C) : m(X(C), e) \geq 1. \tag{3.18}$$

Constraints (3.16) ensure that a fixed edge must always be part of the cycle and Constraint (3.17) enforces that there is at least one fixed edge. Furthermore, constraints (3.18) guarantee that the newly found cycle is a new maximal cycle for the fixed edge e if z_e is active.

To further speed up the approach we treat constraints (3.18) as lazy constraints and solve the model only once instead of restarting the solver after each found cycle for the updated model. That means whenever we find a new cycle, i.e., an integer solution to the current model, we add for all edges e with $m(X(C), e) = 1$ the respective Constraint (3.18) dynamically. Formally this results in the end in an infeasible model but by collecting all the cycles gathered during the execution we get a complete collection of all potential maximal cycles. We can then execute phase two as explained above to check if any of the gathered cycles in the data structure is an SFE-cycle for some edge.

Note that since we do not have a fixed edge e in the model, we have to specify what we use as vertex w in constraints (3.6). This vertex should always be part of the cycle. To that end we enumerate over all vertices in $V(G)$, starting with the first as w. Then we apply the first phase and afterwards we forbid this vertex in all following runs by forcing $x_v = 0$ and chose the next vertex as w. The successive runs are much faster than the first ones since the model gets easier to solve and fewer cycles get found with each forbidden vertex. After all this we apply phase two to the current data structure.

If we are searching for dominating cycles we can improve this by only looking at an edge $e \in E(G)$ and using the fact that one of the ends of this edge must be part of the cycle.

Therefore, we only need to apply the first phase two times. Furthermore, we will see in Section 3.3.3 that if we search for a minimum counterexample we can identify vertices that must always be part of the cycle, see Proposition 3.3.6, and therefore can use such a vertex as w and have to apply the first phase only once.

Cycle Enumeration Approach

In the above ILP approach we directly searched for maximal cycles. The idea of the alternative cycle enumeration approach is to enumerate all cycles in the graph and let our data structure do the work of filtering out the maximal ones. To speed up the algorithm we try to identify as early as possible if a cycle may be maximal and do not add it to the data structure if this is not the case.

The basic idea is to build a path recursively and try all possibilities to extend it until we can close the path to a cycle. We keep track of the two ends of the path and all the edges in the graph that are only incident to unvisited vertices or ends. Then for the end vertex with fewer remaining incident edges we try every such edge as an extension of the path. We call an edge a closing edge if it connects the two end vertices, and we call the resulting cycle the corresponding cycle to this closing edge. Whenever we find a closing edge, we can add the corresponding cycle to our data structure. However, before we close a cycle we try all non-closing edges and check if at least one of them leads to at least one cycle. In this case we know that the corresponding cycle of the current closing edge cannot be maximal.

To further speed up this algorithm we use five different states for vertices: `unvisited`, `visited`, `end`, `fixed`, and `dropped`. The vertices within the path are `visited` and the two end vertices have state `end`. For a vertex that is not in the path and has at some point only one incident edge left we know that it cannot be part of the cycle. We set the state to `dropped` and remove its incident edge.

If we search for dominating cycles we know that one of the two end vertices of each edge must be part of the cycle. Whenever we drop a vertex we can mark its only remaining unvisited adjacent vertex as `fixed`. Furthermore, if at some point a fixed vertex gets dropped we know that this path can never lead to a dominating cycle. To check for a closing edge if the corresponding cycle is dominating we keep during the whole algorithm track of the number of edges for which none of the two end vertices is part of the current path. A closing edge corresponds to a dominating cycle if and only if this number is zero at this point.

What remains is to specify with which vertex we start the algorithm. To do that we can iterate over all possible starting vertices in the same way as described for the advanced ILP approach and apply the algorithm to each of them. Whenever we used a start vertex and applied the algorithm we forbid it by setting its status to `dropped` in the following runs.

3.3.3 Properties of a Minimum Counterexample

In this section we discuss some properties a minimum planar, 2-connected SFE-graph with minimum degree 3, i.e. a minimum counterexample to Conjecture 3.3.1, must have. By minimum we mean w.r.t. the following relation:

$$G = (V, E) \leq G' = (V', E') \Leftrightarrow |V| < |V'| \vee (|V| = |V'| \wedge |E| \leq |E'|)$$

For the whole section let $G = (V, E)$ be a minimum counterexample and let (C, e) be the SFE-cycle. The following notations will be useful.

Definition 3.3.2. A vertex v of G is called a *small vertex* if its degree $d(v) \leq 3$ and if it is not incident to the fixed edge e. Otherwise, a vertex v is called a *large vertex*. If a large vertex has a degree larger than three it is called a *really large vertex* and otherwise it is called a *fixed large vertex*.

A vertex in $V(C)$ is called a *cycle vertex* and a vertex in $V \setminus V(C)$ is called an *outer vertex*.

We write $V^{\mathrm{l}}(G)$ for the set of all large vertices in G and $V^{\mathrm{s}}(G)$ for the set of all small vertices in G. Furthermore, we write $V^{\mathrm{lc}}(G)$, $V^{\mathrm{lo}}(G)$, $V^{\mathrm{sc}}(G)$, $V^{\mathrm{so}}(G)$ for the set of all large cycle vertices, large outer vertices, small cycle vertices, and small outer vertices, respectively.

The following statement about the maximal FE-cycle C follows directly from Proposition 3.3.2.

Corollary 3.3.4. *C is dominating.*

Furthermore, we can prove the following connectedness result.

Proposition 3.3.5. *G is 3-connected.*

Proof. Assume G is not 3-connected. Hence, there are two vertices v and w such that $G \setminus \{v, w\}$ is not connected. Let V_1 be the vertex set of a component of $G \setminus \{v, w\}$ that does not contain the fixed edge. Let $H = G[V_1 \cup \{v, w\}]$ be the subgraph of G induced by $V_1 \cup \{v, w\}$. We consider the graph $G' = (V', E') = (V(H), E(H) \cup \{vw\})$. Since G is 2-connected we know that the degree of v and w in G' is at least two. If the degree of v in G' is two we can consider the other neighbor $x \neq w$ of v in G'. We get that $V_1 \setminus x$ is a nonempty component of $G \setminus \{x, w\}$. It is nonempty since otherwise x would have only the neighbors v and w in G, which contradicts the minimum degree of 3.

Therefore, we can choose w.l.o.g. the vertices v and w in such a way that the vertex set V_1 is minimal and therefore v and w have degree 3 or higher in G'.

Now, C together with the edge vw induce a cycle C' in G'. The stability of the FE-cycle (C, e) implies the stability of the FE-cycle (vw, C') in G'. But this is a contradiction to G being a minimum counterexample. □

Proposition 3.3.6. *Every neighbor of a large vertex is in $V(C)$.*

Proof. Assume a neighbor w of a large vertex v is not in $V(C)$. Then we can remove the edge vw and still have the same SFE-cycle. If w has now degree 2 we remove it and replace it with an edge. Furthermore, if v was a fixed large vertex and has degree two now, we remove it and replace it with an edge, which is now the new fixed edge. This leads to a contradiction of G being a minimum counterexample. □

Proposition 3.3.7. *Every edge between large vertices is in $E(C)$.*

Proof. An edge between large vertices that is not in $E(C)$ could be removed, which would result in a smaller graph that is planar, 2-connected, still has minimum degree 3, and still contains an SFE-cycle. This, however, contradicts the assumption that G is a minimum counterexample. □

Corollary 3.3.8. *There is no large vertex with at least three large vertex neighbors in G and there is no cycle consisting only of large vertices in G.*

Proof. If there would be a large vertex with at least three large vertex neighbors, then there is at least one large neighbor whose connecting edge is not part of the cycle C, which is a contradiction to Proposition 3.3.7.

Furthermore, if there would be a cycle only of large vertices in G, then by Proposition 3.3.7 all its edges must be part of C, which means that it equals C. Let $v \in V(C)$ be a large vertex in this case. Then there exists one edge $e = vw$ incident to v that is not in $E(C)$. Since v is large we know by Proposition 3.3.6 that $w \in V(C)$ and therefore by our assumption that w is a large vertex. But this is a contradiction to Proposition 3.3.7. □

Lemma 3.3.9. *The number of small vertices $|V^{\mathrm{s}}(G)|$ is equal to*

$$\sum_{v \in V^{\mathrm{lc}}(G)} (d(v) - 2) + \sum_{v \in V^{\mathrm{lo}}} d(v) + 2\,|\{vw \in E(G) \setminus E(C) : v, w \in V^{\mathrm{sc}}(G)\}| + 4|V^{\mathrm{so}}(G)|.$$

Proof. Each small cycle vertex w has exactly one incident edge vw that is in $E(G) \setminus E(C)$. This means that counting the small cycle vertices is the same as counting the half edges that are in $E(G) \setminus E(C)$ and are connected to a small cycle vertex. There are now three possibilities for v. Either it is in $V^{\mathrm{o}}(G)$, in $V^{\mathrm{lc}}(G)$, or in $V^{\mathrm{sc}}(G)$.

A vertex v in $V^{\mathrm{lc}}(G)$ has $d(v) - 2$ incident edges that are not part of the cycle. By Proposition 3.3.6 and Proposition 3.3.7 those edges must be incident to a small cycle vertex. Therefore, each large cycle vertex v is connected to $d(v) - 2$ small cycle vertices via edges in $E(G) \setminus E(C)$.

Moreover, an outer vertex v must be connected to $d(v)$ small cycle vertices, clearly via edges in $E(G) \setminus E(C)$, by Proposition 3.3.6 and Corollary 3.3.4.

Adding this up over all outer vertices and large cycle vertices gives us the number of small cycle vertices that are either connected to a large cycle vertex or to an outer vertex via an edge in $E(G) \setminus E(C)$. The remaining small cycle vertices must now all be connected to exactly one other small cycle vertex via an edge in $E(G) \setminus E(C)$. Therefore, the number of such remaining cycle vertices equals

$$2\,|\{vw \in E(G) \setminus E(C) : v, w \in V^{\text{sc}}(G)\}|\,.$$

What remains is now to add $|V^{\text{so}}|$ to the sum to get the number of all small vertices, which results in the theorem's statement. □

For the next statement we introduce the notion of very small vertices.

Definition 3.3.3. A small vertex is called *very small* if all its neighbors are small.

Furthermore, we need the notion of a maximum pseudo-matching based on Definition 2.1.18.

Definition 3.3.4. A *maximum pseudo-matching* in a graph G is a pseudo-matching M in G that maximizes the number of vertices $|V(M)|$.

Lemma 3.3.10. *Let G^{vs} be the graph induced by all very small vertices in G and M^{vs} a maximum pseudo-matching in G^{vs}. Then it holds that*

$$2\,|\{vw \in E(G) \setminus E(C) : v, w \in V^{\text{sc}}(G)\}| + 4|V^{\text{so}}(G)| \geq 2|V(G^{\text{vs}})| - |V(M^{\text{vs}})|.$$

Proof. We begin by defining a pseudo-matching M in G^{vs}. For each very small outer vertex v for which all three neighbors are also very small we add the claw with center v to the pseudo-matching M. Furthermore, for each very small outer vertex v that has at least one very small neighbor and at least one not very small neighbor we add an edge between v and one of its very small neighbors to M. Last but not least, we add for all very small cycle vertices that are connected to other very small cycle vertices via an edge e in $E(G) \setminus E(C)$ this edge e to M.

Note now that

$$X := 2\,|\{vw \in E(G) \setminus E(C) : v, w \in V^{\text{sc}}(G)\}| + 4|V^{\text{so}}(G)|$$

is equal to the number of small vertices that are not connected to a large vertex by any edge in $E(G) \setminus E(C)$. By the construction of our pseudo-matching M there are three different types of very small vertices in $V(G^{\text{vs}}) \setminus V(M) = V_1 \cup V_2 \cup V_3$. Let V_1 be all very small cycle vertices that are connected to another small but not very small cycle vertex via an edge in $E(G) \setminus E(C)$. Furthermore, let V_2 be all very small outer vertices that are connected to three small but not very small cycle vertices. Finally, let V_3 be all very small cycle vertices that are connected to a very small outer vertex that has exactly two very small neighbors and the edge connecting the other very small neighbor is part of M.

We define now for each vertex in $v \in V(G^{\text{vs}}) \setminus V(M)$ an additional small cycle vertex w_v that is not very small but also not connected to a large vertex by any edge in $E(G) \setminus E(C)$. Furthermore, those vertices w_v are all different for different vertices $v \in V(G^{\text{vs}}) \setminus V(M)$.

For $v \in V_1$ we define w_v to be the small neighbor of v via an edge in $E(G) \setminus E(C)$. For $v \in V_2$ we define w_v to be any neighbor of v and for $v \in V_3$ we define w_v to be the third neighbor of the outer vertex that is not a very small vertex.

With that we get that

$$\begin{aligned} X &\geq |V(G^{\text{vs}}) \cup \{w_v : v \in V(G^{\text{vs}}) \setminus V(M)\}| \\ &= |V(G^{\text{vs}})| + |V(G^{\text{vs}} \setminus V(M)\}| = 2|V(G^{\text{vs}})| - |V(M)|. \end{aligned}$$

The Lemma follows now from the fact that M^{vs} is a maximum pseudo-matching and therefore by definition $|V(M)| \leq |V(M^{\text{vs}})|$. □

Theorem 3.3.11. *Let G^{vs} be defined as in Lemma 3.3.10 and $C^{\text{vs}}_{\text{odd}}$ be the number of connected components in G^{vs} with an odd number of vertices. Then*

$$|V^{\text{s}}(G)| \geq \sum_{v \in V^1(G)} (d(v) - 2) + |V(G^{\text{vs}})| + C^{\text{vs}}_{\text{odd}},$$

holds, which is equivalent to

$$|E(G)| \leq |V^{\text{s}}(G)| + |V(G)| - \frac{1}{2}(|V(G^{\text{vs}})| + C^{\text{vs}}_{\text{odd}}).$$

Proof. The first statement follows from Lemma 3.3.9, Lemma 3.3.10, and the fact that the connected components of a pseudo-matching always have an even number of vertices and therefore

$$|V(M^{\text{vs}})| \leq |V(G^{\text{vs}})| - C^{\text{vs}}_{\text{odd}}.$$

To see the equivalence with the second statement we do the following equivalence transformations.

$$\begin{aligned} & |V^{\text{s}}(G)| \geq \sum_{v \in V^1(G)} (d(v) - 2) + |V(G^{\text{vs}})| + C^{\text{vs}}_{\text{odd}} \\ \Leftrightarrow\ & |V^{\text{s}}(G)| \geq \sum_{v \in V(G)} (d(v) - 2) - |V^{\text{s}}(G)| + |V(G^{\text{vs}})| + C^{\text{vs}}_{\text{odd}} \\ \Leftrightarrow\ & 2|V^{\text{s}}(G)| \geq 2|E(G)| - 2|V(G)| + |V(G^{\text{vs}})| + C^{\text{vs}}_{\text{odd}} \\ \Leftrightarrow\ & |E(G)| \leq |V^{\text{s}}(G)| + |V(G)| - \frac{1}{2}(|V(G^{\text{vs}})| + C^{\text{vs}}_{\text{odd}}). \end{aligned}$$

□

An important property of a minimum counterexample, which helps much in searching for counterexamples, is the following.

Theorem 3.3.12. *G is triangle-free.*

Proof. Assume G contains a triangle T. We distinguish whether G can get embedded into the plane such that T is a 3-face in this embedding or not. If there is no such embedding, we know that the removal of T would split G into two components. Since G is 3-connected, there must be an edge from every vertex of T to each component. This implies that all vertices of T are large, which is a contradiction to Corollary 3.3.8.

In the other case we know that there is an embedding of G such that T is a 3-face. Let u, v, w be the three vertices of the 3-face T. We distinguish now two cases.

1. T does not contain the fixed edge e. We distinguish again three cases.

   a) If all three vertices are small, we can contract them all into one vertex of degree three. Every maximal FE-cycle in the original graph corresponds now to a maximal FE-cycle in the new graph and vice versa. This implies that the corresponding cycle $(\tilde{C}, e)$ in the new graph is again an SFE-cycle with the same fixed edge e as in the original graph. This is a contradiction to G being a minimum counterexample.

   b) If one vertex is large and the other two are small, we say w.l.o.g. the large vertex is u and we can shrink the two small vertices v and w together. There are now again two cases. If the cycle C in the original graph contains the path v, u, w, then the new graph has an SFE-cycle that visits all vertices of C (including the new merged vertex instead of w and x) except v. If the cycle C contains the path v, w, then (C, e) transforms into an SFE-cycle containing the same vertices as C (except the new merged vertex instead of v, w).

   c) If two vertices are large and one is small, we get by Proposition 3.3.7 that the edge between the two large vertices is used by the cycle C. If we remove now this edge there is clearly no cycle containing all vertices of C anymore. Moreover, C transforms into a cycle containing all vertices except one of the large vertices, call it v. This cycle must be stable since every cycle that contains all vertices of C except v can be transformed into a cycle containing all vertices in the original graph.

   d) If all vertices of the triangle are large, we get by Corollary 3.3.8 a direct contradiction.

2. If T contains the fixed edge, w.l.o.g. let vw be the fixed edge. In this case v and w are large vertices by definition and therefore by Corollary 3.3.8 we get that u is a small vertex and by Proposition 3.3.6 that u is in C. Let x be the third neighbor of u, then the edge ux must be part of C. We merge now the triangle to one new vertex z and consider the to C corresponding cycle $\tilde{C}$ together with the fixed edge xz. Every cycle containing the edge xz corresponds to a cycle in the original graph containing all three vertices of T and the edge vw. This implies that $(\tilde{C}, \tilde{e} = xz)$ is

an SFE-cycle in the new graph, which is a contradiction to G being a minimum counterexample.

□

The next theorem shows an even stronger connectedness of a minimum counterexample.

Definition 3.3.5. A graph G is called *essentially k-edge connected* if there does not exist a set of at most $k-1$ edges whose removal would split G into components such that at least two of the components are non-trivial, i.e. contain more than one vertex.

Theorem 3.3.13. *G is essentially 4-edge connected.*

Proof. Assume G contains three edges e_1, e_2, e_3 whose removal disconnect G into two non-trivial components G_1, G_2. Assume w.l.o.g. that the fixed edge e is not in G_1. Let $e_i = v_i w_i$ with $v_i \in G_1$ and $w_i \in G_2$. The three vertices v_1, v_2, and v_3 have degree at least two in G_1.

Since C is dominating and since G_1 and G_2 are both non-trivial we know that C must use two of the three cut edges. W.l.o.g. let C use the edges e_1 and e_2. The cycle C corresponds to a path P in G_1 going from v_1 to v_2.

If we consider the graph G' consisting of G_1 together with an edge connecting v_1 with v_2 (if they are not already connected) we can extend P with this new edge to an FE-cycle C' in G' by fixing the edge between v_1 and v_2. We distinguish now two cases, if v_3 is in $V(C')$ or not.

If v_3 is not in $V(C')$ we know that C' is stable since, if there would exist another FE-cycle with the same fixed edge that contains the same or more vertices we could remove the fixed edge from it and get a different path P' in G_1. We could then replace the path P in C with P' and get a contradiction to the stability of C. The graph G' may contain vertices of degree two. If v_1 or v_2 have degree two we can replace them in G' with an edge, which will then be the new fixed edge. Furthermore, if v_3 has degree two we can simply replace it with an edge since v_3 is not used in the cycle C'. Note that G' must contain at least 4 vertices since otherwise we could easily find a cycle that also contains v_3, which would be a contradiction to the maximality of C'. Therefore, after the removals described above the remaining graph is still not empty, it has minimum degree three, is planar, is smaller than G, and has an SFE-cycle, which is a contradiction.

The remaining case is that v_3 is in $V(C')$. Here G_1 must contain a path P_1 from v_2 to v_3 containing at least all the vertices of P. If there would not exist such a path we could contract G_2 in G to a vertex x and fix the edge from x to v_2. Then the cycle consisting of P and the edges v_1x and v_2x is again stable. This can be seen since the only other possibility would be to use the edges v_2x and v_3x but then the remaining path after removing those two edges would have the properties of P_1, which we assumed does not exist.

With the same argumentation we can prove the existence of a path P_2 from v_1 to v_3 in G_1 that contains at least all the vertices of P. We contract now the component G_1 in G to a vertex x and get a new graph G'. Clearly, the cycle C corresponds to a new cycle C' using the edges w_1x and w_2x. The fixed edge of C' is still the same fixed edge as the one of C (which was not in the contracted G_1). If there would exist now a second cycle containing the fixed edge and at least the same vertices as C' in G' this second cycle must use either w_1x and w_3x or it uses w_2x and w_3x, since otherwise it would contradict the stability of C. But if such a cycle exists we could replace the two edges with either P_1 or P_2 and would get a cycle in the original graph G, which would again contradict the stability of C. □

The following corollary summarizes now all properties for a minimum counterexample G if we do not know the FE-cycle (C, e). Since we do not know the fixed edge we define for the corollary that the small vertices are exactly the vertices of degree 3 and the large vertices are the vertices of degree larger than 3.

Corollary 3.3.14. *Let $G = (V, E)$ be a minimum planar, 2-connected SFE-graph with minimum degree 3. Furthermore, let G^{vs} be the graph induced by all very small vertices in G. Then the following properties hold.*

- *G is 3-connected.*
- *G is essentially 4-edge connected.*
- *G contains no triangles.*
- *There is no large vertex in G with at least three large vertex neighbors.*
- *There is no cycle of large vertices in G.*
- *Let C^{vs}_{odd} be the number of connected components in G^{vs} with an odd number of vertices, then it holds that*

$$|E(G)| \leq |V^s(G)| + |V(G)| - \frac{1}{2}(|V(G^{vs})| + C^{vs}_{odd}).$$

Corollary 3.3.14 summarizes all the properties a graph must satisfy, but we can also consider the properties a potential SFE-cycle must satisfy such as Corollary 3.3.4 and Proposition 3.3.6. To search for a minimum SFE-graph it is enough to solve for each graph satisfying the properties from Corollary 3.3.14 the following problem.

Problem 3.3.3. Given a graph G, does G contain a dominating SFE-cycle (C, e) containing all neighbors of large vertices of G?

Note that the algorithms presented in Section 3.3.2 can all also solve Problem 3.3.3. For the ILP-based approaches we can fix neighbors of large vertices by forcing the corresponding x-variables to one and for the enumeration approach we can set the status of neighbors of large vertices to `fixed` at the beginning of the algorithm.

3.3.4 Systematic Search for a Minimum Counterexample

In this section we describe how we use the results from Corollary 3.3.14 to do a computationally effective search for a minimum counterexample. We use plantri [12], a tool for generating exactly one representant of all isomorphism classes of planar graphs of a given order. It furthermore can effectively create only 3-connected graphs with a given minimum degree.

Our first approach was to directly generate representants of 3-connected planar graphs and then filter them by the properties from Corollary 3.3.14. This results in a bad performance since the filtering will be the bottleneck due to the restrictive condition of triangle freeness. To be able to incorporate triangle freeness into the generation process we generate representants of the dual graphs instead of the original graphs.

Note that for a 3-regular planar graph the dual is well-defined by Theorem 2.1.4 and again 3-regular by Theorem 2.1.5. We can then translate the triangle freeness of the original graphs to minimum degree four in the dual graph, which can be handled by plantri in a performant way. Note that there might still be triangles in the resulting graph that are not faces, but those would lead to a triangle of large vertices, which is also forbidden by the conditions in Corollary 3.3.14 and therefore get filtered out afterwards.

In order to generate all candidate graphs with up to n vertices we need to generate all dual graphs with up to $n-2$ vertices. This can be seen by the fact that $|E(G)| \geq 2|F(G)|$ in triangle free planar graphs and therefore

$$|F(G)| = |E(G)| - |V(G)| + 2 \geq 2|F(G)| - |V(G)| + 2 \Rightarrow |F(G)| \leq |V(G)| - 2.$$

Furthermore, we get an upper bound for the number of edges by

$$|E(G)| = |F(G)| + |V(G)| - 2 \leq |F(G)| + n - 2.$$

To summarize the total process given a maximum target graph size n, we let n^* iterate from 8 to $n-2$ and use plantri to generate all 3-connected planar graphs with minimum degree 4, n^* vertices, and at most $n^* + n - 2$ edges. Then we build the duals of all generated representants and filter them by the conditions in Corollary 3.3.14. The resulting graphs are all possible candidate graphs with size up to n. Then we can use any of our three approaches to check if one of them contains a candidate SFE-cycle by solving Problem 3.3.3.

Note that the step of building the duals can be parallelized in plantri by splitting the search space into junks, which are of reasonable similar sizes. Therefore, we can parallelize the whole process, which we heavily use to get our computational results.

3.3.5 Computational Results

In this section we present computational tests and results of our algorithms in a two-folded manner. On the one hand we use three structurally different classes of graphs

and test and compare our approaches on them. On the other hand, we use the best performing approach to systematically search for a minimum counterexample, which will computationally prove the main result of this section, Theorem 3.3.16.

All tests and computations are performed on machines with Intel Xeon E5-2640 v4 processors with 2.4GHz using at most 8GB RAM per thread. The tests performed on the instance sets are each done on a single core and for searching for a minimum counterexample we made use of up to 320 cores split over 16 machines in a parallelized fashion. Furthermore, the implementations are done in C++ and for the ILP approaches we use Gurobi Optimizer 8.1[5] to solve.

Instance Sets

We describe here three different instance sets I_1, I_2, and I_3 that we use for testing our algorithms. The instance sets I_1 and I_2 are for the problems 3.3.1 and 3.3.2. The third instance set is for Problem 3.3.3.

The first set I_1 consists of up to 5000 random 3-regular planar graphs for each graph order from 6 to 59. Note that for order 6 to 9 there are fewer than 5000 isomorphism classes and therefore we simply use a representant of each of them. For all other orders we sample 5000 different representants.

The second set I_2 gets constructed in the following way. For each order from 6 to 59 take up to 5000 different representants of random 3-regular planar graphs as before, then randomly insert into every edge of the graph a vertex of degree 2 with 2% probability. The probability was chosen in such a way that the resulting graphs with around 20 vertices contain SFE-cycles with a probability of around 50%, which was evaluated experimentally. For larger graphs up to 59 vertices the probability of containing an SFE-cycle increases but the probability of containing a dominating SFE-cycle decreases, see also Figure 3.4.

The third set I_3 consists for each n^* from 8 to 59 of up to 5000 random graphs that satisfy all properties of Corollary 3.3.14, i.e. are possible candidates for a minimum counterexample, and have n^* faces.

To generate the instance sets we used plantri for creating graphs with the desired properties and selected with a given probability the graphs. For the sets of I_3 we used plantri on the dual graphs with minimum degree four. For the orders 6–14 in the case of I_1 and I_2 and for the orders 8–17 in the case of I_3 we enumerated all representants and randomly chose 5000 of them. For all other orders we used a feature from plantri that can partition the search space into sets of similar size and enumerate through one of those sets effectively. Then we randomly select a set, generate graphs from this set and keep a graph with a small probability of $1/2\,000\,000$ in the case of I_1 and I_2 and $1/1\,000\,000$ in the case of I_3 until we have two graphs, or we enumerated the whole set. Then we start with another randomly selected set (same sets can get selected multiple times) until we have 5000 graphs not allowing duplicates.

[5] https://www.gurobi.com (accessed 09/2019)

Note that this procedure selects not uniform from the whole search space since the splitting of the search space by plantri is limited and there might be large sets in this partition. The probability of a graph depends on the order of enumeration of those sets, the graphs enumerated first have the highest selection probabilities and the graphs at the end the lowest. But for our purposes this biased selection is sufficient. To select truly uniformly for the three different search spaces would be non-trivial and presumably computationally far too expensive.

The instance sets and the source code of the implementations of our algorithms can be downloaded from https://www.ac.tuwien.ac.at/research/problem-instances.

Testing the Three Approaches on the Three Instance Sets

To verify the correctness of our approaches and to compare their performance we applied all three approaches on our instance sets. For instance sets I_1 and I_2 we test two problem variants, Problem 3.3.1, which searches for dominating SFE-cycles and Problem 3.3.2, which searches for any SFE-cycle. For graphs of the instance set I_3 we only consider the variant of Problem 3.3.3, which searches for potential SFE-cycles in a minimum counterexample. For each instance set, problem variant, and order n we use a running time limit of one hour to solve all the up to 5000 graphs of order n.

Figure 3.3 compares the average running times per graph of the three algorithms for the different graph orders and problem variants in seconds. Note that the graph orders in instance set I_3 denote the order of the dual graph and therefore the number of faces of the original graph. We only display data points for orders for which at least 30 graphs could get solved within the one hour time limit.

As we can see, our enumeration algorithm performs best on small graphs and does not scale well for larger graphs. Furthermore, for small graphs the advanced ILP approach performs better than the basic ILP approach. For larger graphs of the instance set I_1 and I_2 the basic ILP approach performs better than the advanced ILP approach. This can be explained by the fact that the basic ILP approach has to solve many more ILPs than the advanced ILP approach. For small graphs those ILP models are so easy to solve that the overhead of starting the solver outweighs the actual solving time. On the other hand, for larger graphs the models get harder to solve and the simplicity of the models in the basic ILP approach compared to the advanced ILP approach compensates the overhead of solving more models.

For the instance set I_3 the models stay easy to solve also for larger graphs since a lot of vertices can be fixed in the cycle due to the definition of Problem 3.3.3 and therefore the advanced ILP approach performs better than the simple ILP approach over all orders. Our enumeration performs better than the ILP approaches for searching for a minimum counterexample up to a dual graph order of 19. This will be important, when we want to search for a smallest counterexample.

Instance set I_2 is important to also test positive instances, i.e. graphs that contain SFE-cycles. Although the three algorithms could not solve all graphs in the given time

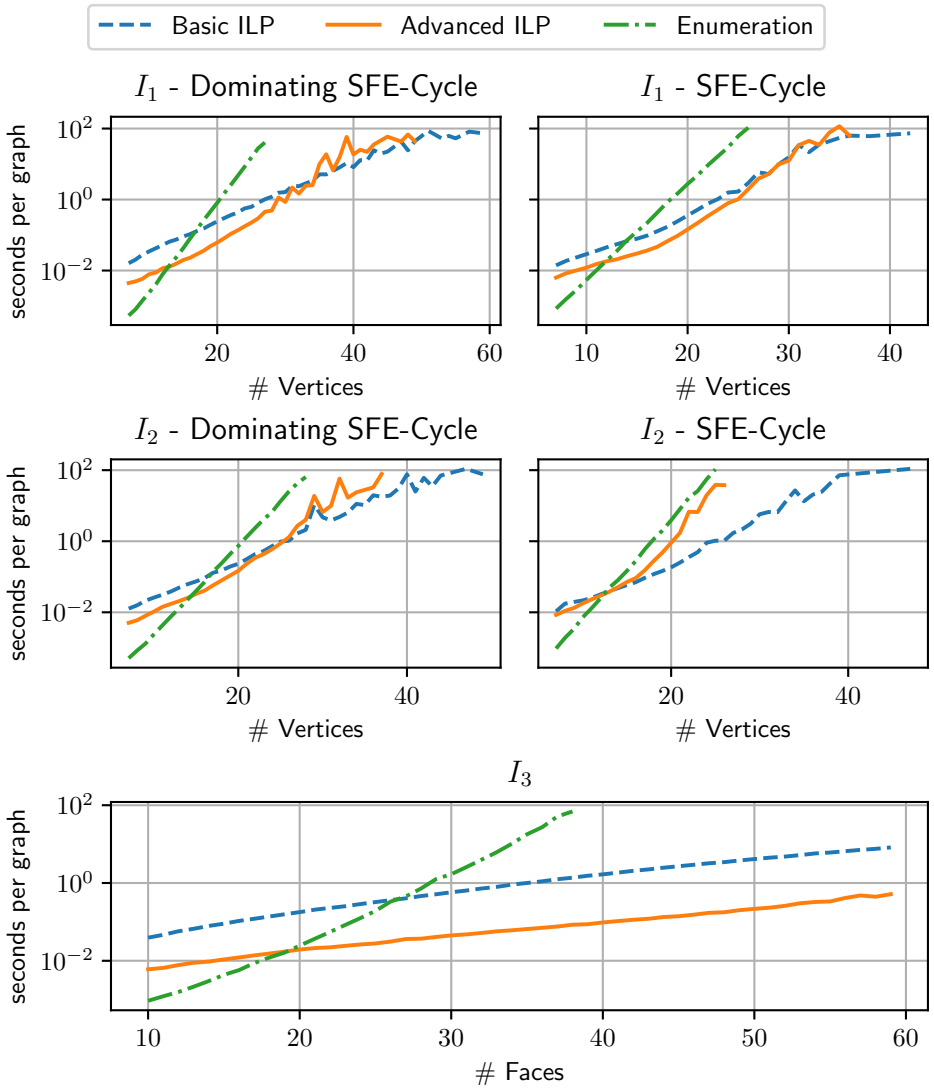


Figure 3.3: Average running time comparison of the three different algorithms on the three instance sets.

limit, we compared the results of the solved graphs between the three approaches and can conclude that for each of the tested graphs the approaches coincide in their decision between containing an SFE-cycle or not. It may be that the approaches find different SFE-cycles if they exist. In Figure 3.4 we see for the two different variants for instance set I_2 for each graph order how many graphs could get checked within the time limit and how many SFE-graphs were found by the basic ILP approach. The other two approaches were able to check only the same amount of graphs or less for each graph order.

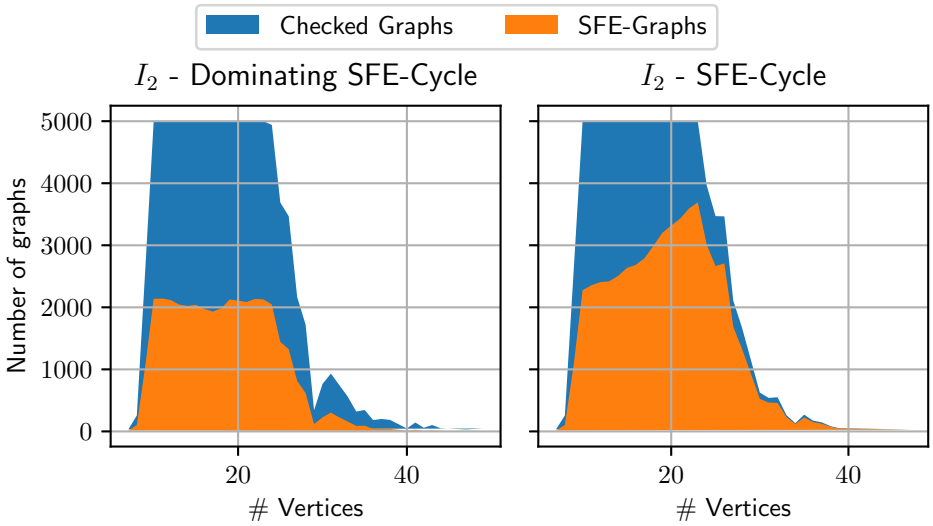


Figure 3.4: Number of tested graphs in instance set I_2 compared to number of found SFE-graphs.

To better understand the difference between the basic ILP approach and the advanced ILP approach we analyze the number of ILPs solved for each single graph. Figure 3.5 shows the number of ILPs solved per graph for each graph order and problem variant for the two different ILP approaches.

As we can see the basic ILP approach needs to solve many more models than the advanced approach. The difference is especially big for the instance set I_3 where the advanced approach needs to solve only up to 5 models per graph on average. The basic model needs to solve on average multiple hundred models per graph for the larger I_3 instances.

Searching for Minimum Counterexamples with up to 24 Vertices

In this subsection we describe the results of applying the approach described in Section 3.3.4 to search for a minimum counterexample. As we saw before the enumeration approach performs best for Problem 3.3.3 on graphs with up to 19 faces. We will see that the vast majority of candidate graphs with up to 24 vertices have 19 or fewer vertices and therefore we will use the enumeration approach for checking if any of the candidate graphs contain an SFE-cycle. Note that we also use the enumeration approach for graphs with more than 19 faces for simplicity. One could instead use the advanced ILP-approach for all graphs with 20 or more faces, which may be important if we want to go further and check graphs with more than 24 vertices.

Table 3.5 shows the results of our search. Each row lists the results for the generated graphs with a given number of vertices, which is displayed in the first column. Furthermore, the column *part.* gives the number of parallel threads used to generate the graphs, filter

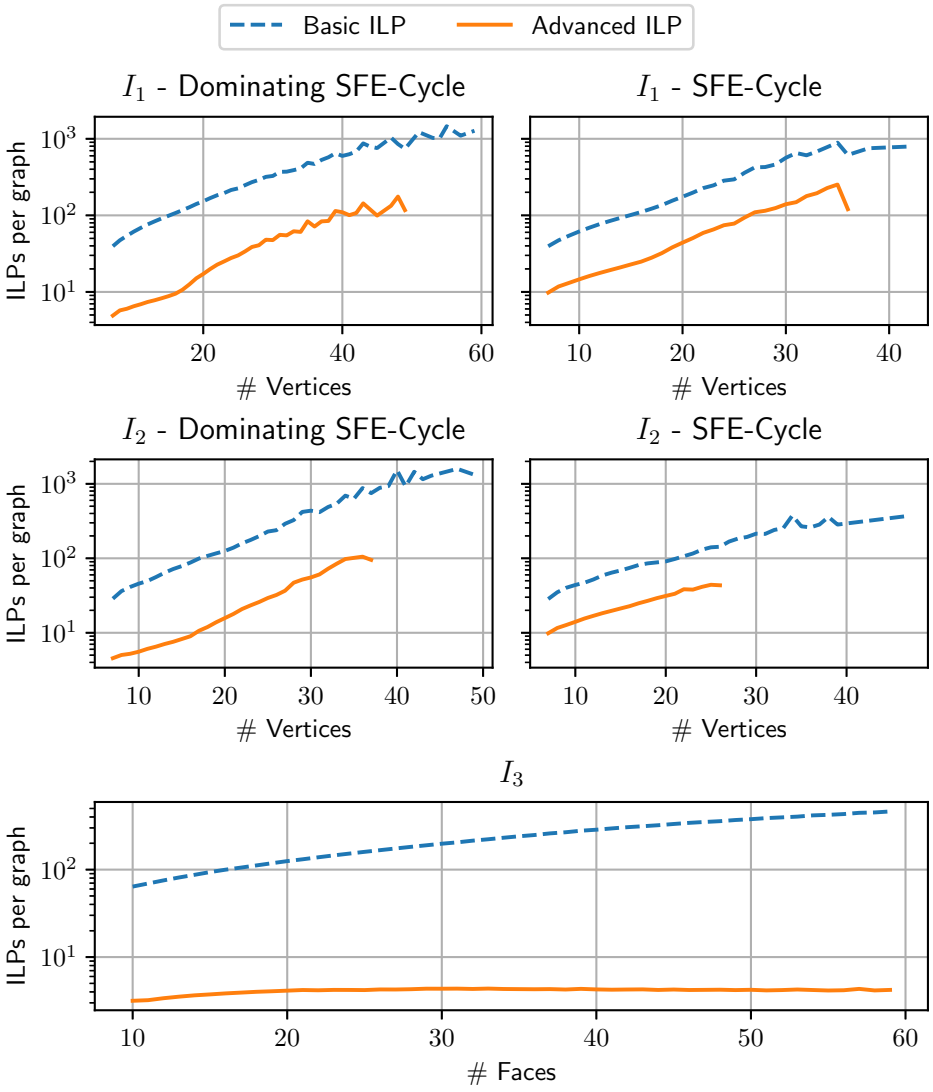


Figure 3.5: Number of ILPs solved per graph of the two different ILP approaches on the three instance sets.

them and check them for SFE-cycles. The third and fourth column show the number of graphs that got checked for SFE-cycles by solving Problem 3.3.3 and the number of graphs that were filtered out by the properties of Corollary 3.3.14.

Since we are interested in the running time of the slowest thread, this is given in seconds by column *max t[s]*. The next three columns list summed running times of all threads in hours. Column *total* gives the total CPU time of everything together, generation, filtering, and checking. Moreover, columns *gen.* and *check* give the total CPU times for

Table 3.5: Results for Candidate Graphs with up to 24 Vertices.

		# Graphs			Total CPU times [h]				
$\#F$	part.	checked	filtered	max t[s]	total	gen.	check	C/G	SFE
6	1	1	0	<1	<1	<1	<1	8.0	0
7	1	1	0	<1	<1	<1	<1	15.0	0
8	1	4	0	<1	<1	<1	<1	14.8	0
9	1	12	2	<1	<1	<1	<1	15.9	0
10	1	61	6	<1	<1	<1	<1	17.6	0
11	1	371	57	1	<1	<1	<1	18.6	0
12	1	2927	588	8	<1	<1	<1	21.0	0
13	1	24 957	6806	50	<1	<1	<1	23.9	0
14	32	228 793	78 750	184	<1	<1	<1	27.7	0
15	128	2 090 930	862 784	644	3	<1	3	29.2	0
16	512	14 750 689	7 746 354	1316	25	<1	24	25.1	0
17	512	45 728 911	57 554 063	2160	79	1	76	19.8	0
18	512	36 650 395	211 986 918	1486	71	2	67	16.2	0
19	512	6 061 407	283 793 146	328	27	13	14	14.2	0
20	512	201 991	145 836 208	1370	106	106	<1	13.2	0
21	512	1309	24 624 586	9464	1100	1100	<1	13.0	0
22	512	5	598 097	151 153	12 544	12 544	<1	20.8	0

generation and checking respectively. As we can see the highest running time is used to generate graphs with 22 faces although only 600 000 such graphs get generated in the end and only 5 of them get really checked. This shows that the bottleneck of the current approach is not solving Problem 3.3.3 but to quickly generate graphs satisfying the properties of Corollary 3.3.14.

The column *C/G* lists the average number of cycles the algorithm found per graph. Note that those cycles may not even be maximal and are simply added to the data structure for storing cycles to check at the end if some of them are SFE-cycles. Finally, the column *SFE* shows that none of the generated graphs contain any SFE-cycles.

Those computational results prove the following lemma.

Lemma 3.3.15. *There does not exist any planar graph with minimum degree three and at most 24 vertices that contains an SFE-cycle.*

Definition 3.3.6. A cycle C in a graph G is called *stable* if there exists no other cycle $C' \neq C$ with $V(C') \supseteq V(C)$.

With that we can finally prove our main theorem.

Theorem 3.3.16. *There does not exist any planar graph with minimum degree three having at most 25 vertices that contains a stable cycle. This implies that Bondy and Jackson's conjecture holds for graphs with up to 25 vertices.*

Proof. Clearly, every uniquely hamiltonian cycle is a stable cycle and therefore we only need to prove the first statement. Assume that there exists a planar graph G with minimum degree three having at most 25 vertices that contains a stable cycle C. W.l.o.g. let G be a minimum by the relation defined at the beginning of Section 3.3.3. There are now two cases, either G is 3-regular or it contains a vertex v with degree at least four.

In the first case we know that C cannot be uniquely hamiltonian and therefore there must exist a vertex u that is not part of C. W.l.o.g. we can assume that all neighbors of u are part of C since otherwise we can contract the whole component of vertices that are not part of C. Note that this implies that the neighbors of u are not connected by an edge since this would imply that C is not maximal. What we do now is remove the vertex u and connect two of its neighbors with a new edge. What remains is a planar graph containing a stable cycle with up to 24 vertices and one vertex w of degree two.

If, on the other hand, C is hamiltonian and G contains a vertex v with degree at least four we know that at least two of the incident edges to v are not part of the cycle C, let $e = vw$ be one of them. Because G is a minimum we know that w must have degree three since otherwise we could remove e and get a smaller graph with the same properties. We remove now the edge e from G and get a graph with up to 25 vertices and one vertex w of degree two containing a stable cycle.

We got in both cases a planar graph containing a stable cycle with exactly one vertex w of degree two having at most 25 vertices. In both cases we replace w and its two remaining incident edges by one new edge e' and get a new graph G', which has at most 24 vertices and minimum degree three. If w is not in $V(C)$ then C is also a stable cycle in G' and therefore also an SFE-cycle in G'. Otherwise, we can replace the edges e_1 and e_2 in C by the new edge e' and get a new cycle C' in G'. The stability of C in G implies now that (C', e') is an SFE-cycle in G'. Therefore, we get in both cases a contradiction to Lemma 3.3.15. □

Another interesting implication of Lemma 3.3.15 concerns uniquely hamiltonian planar graphs with vertex connectivity two.

Theorem 3.3.17. *There does not exist any planar graph with minimum degree three and vertex connectivity two having at most 46 vertices that contains a stable dominating cycle.*

Proof. We assume there exists such a graph G with a stable dominating cycle C. Let v, w be a 2-vertex cut of G and C be the smallest component of $G \setminus \{v, w\}$. Furthermore, let v, w be in such a way that $|V(C)|$ is as small as possible. Clearly, $|V(C)| \leq 22$. We

define now G' as the graph induced by the vertex set $V(C) \cup \{v, w\}$ and add the edge $e' = vw$ if it is not already part of G'.

Since C is dominating we know that C traverses v and w. We define C' by restricting C to G' and adding the edge vw to C', which gives us a dominating cycle in G'. The stability of C in G implies now the stability of the FE-cycle (C', e') in G'. Furthermore, since we chose v, w in such a way that $|V(C)|$ is as small as possible we know that the degree of v and w in G' is at least three. Therefore, G' has minimum degree three, which is a contradiction to Lemma 3.3.15. □

3.3.6 Conclusion

In this section we focused on systematically searching for uniquely hamiltonian planar graphs with minimum degree three. To this end we reduced the search to planar graphs with minimum degree three that contain an SFE-cycle. The fundamental approach was to generate candidate graphs using the external tool plantri and then check for each of the generated graphs if they contain a stable fixed edge cycle (SFE-cycle).

We formulated three different problem variants. They ask if a given graph contains an SFE-cycle, a dominating SFE-cycle, or a dominating SFE-cycle with some additional properties, respectively. We proved additional properties an SFE-cycle of a minimum counterexample must satisfy and use them in the problem variant three. To solve the problem variants we proposed three different approaches, two based on integer linear programming and one based on cycle enumeration.

To effectively search for a minimum planar SFE-graph with minimum degree three we proved several properties that a minimum counterexample must satisfy, see Corollary 3.3.14 for a summary. The property that helps the most in reducing the search space is triangle freeness.

We tested our three algorithms on a selection of random instances for the three different problem variants. Most of the time the enumeration approach performs better for small graphs and the ILP-based approaches for larger graphs. Finally, we generated all planar 3-regular triangle-free graphs with minimum degree three and up to 24 vertices using plantri to create the dual graphs. Then, we filtered the graphs using the properties of Corollary 3.3.14 and applied the enumeration approach to the remaining graphs to check if they contain an SFE-graph. The computations for searching for a minimum counterexample were heavily parallelized and used a total of 1.59 CPU years with a bottleneck in generating all planar 3-regular triangle-free graphs with minimum degree three and up to 24 vertices.

The computational results show that no planar SFE-graph with minimum degree three with up to 24 vertices exists. This implies the main result that no UHPG3 with up to 25 vertices exists and no UHPG3 with connectivity two with up to 46 vertices exists.

To be able to further increase the lower bound of 25 vertices in the future we have to focus on the fast generation of planar 3-connected triangle-free graphs with minimum

degree three as this was the bottleneck in our search for a minimum counterexample. Furthermore, if we can solve this bottleneck, the usage of the presented ILP-based approaches instead of the enumeration approach for the larger graphs will be crucial. If also those approaches are too slow, one could try to use constraint programming (CP) or SAT-based approaches and compare them to the ILP-based approaches. Last but not least, there might be additional properties that we can prove for a minimum counterexample leading to a more effective search. This might also help to solve the bottleneck of generating candidate graphs.

CHAPTER 4

Finding Smooth Graphs with Small Independence Numbers

In this chapter we consider different open problems regarding lower bounds for classes of smooth graphs, a subclass of 4-regular graphs. We propose and apply a branch-and-bound approach to search for smooth graphs with small independence numbers.

4.1 Introduction

In graph theory independent sets are well studied objects and the independence number of a graph is a central characteristic, which is strongly related to many important properties. One natural research subject is to find lower and upper bounds for the independence number for general graphs, see for Example [38], or for specific subclasses of graphs, see for Example [73].

In this chapter we focus on the independence number of smooth graphs, a subclass of 4-regular hamiltonian graphs. For a complete definition of smooth graphs see Definition 4.2.1 in Section 4.2. Smooth graphs and their independence numbers got already studied in depth from a graph-theoretic perspective by Fleischner, Sabidussi and Sarvanov [31, 29].

We are interested in lower bounds on the independence number of smooth graphs. In this context Sarvanov stated the following conjecture [69].

Conjecture 4.1.1. Every smooth graph G with $n > 11$ vertices has independence number $\alpha(G) \geq \frac{2}{7}n$.

Our main goal is to design an algorithm that can check lower bounds on the independence number for smooth graphs and either prove them for all graphs with a given number of vertices or disprove them by finding a graph with a smaller independence number.

By using Brooks' Theorem [15] we get a lower bound on the independence number for all 4-regular graphs.

Theorem 4.1.1. *Every 4-regular graph with n vertices that is not the K_5 is 4-colorable. This implies that every such graph has an independent set of size at least $n/4$.*

This property, together with the fact that we only consider graphs containing a hamiltonian cycle and therefore have an independence number of at most $n/2$, give us an interval of possible lower bounds.

We describe a branch-and-bound algorithm that heavily depends on the graph-theoretic results and bounds to search through the space of possible graphs in an effective way. The main idea is to use a heuristic to compute a large independent set together with the graph-theoretic bounds to detect infeasible subproblems as early as possible. For complete solutions we use an ILP model to compute their independence number and to check if they are feasible.

In the next section we formally define smooth graphs and state the problem framework. In Section 4.4 we infer some useful bounds and properties using already existing graph-theoretic results, and in Section 4.5 we describe how to use those bounds and properties to compute a usually tight bound on the independence number of a partial solution in order to detect infeasibility as early as possible. In Section 4.6 we present some computational results for four different problem variants. Finally, we conclude with Section 4.7 and propose promising future work.

4.2 Problem Formulation

Note that in contrast to Chapter 3 we are considering in this chapter general graphs that may contain multiple edges, as defined in Definition 2.1.1. We are interested in 4-regular hamiltonian graphs $G = (V, E)$ with a fixed hamiltonian cycle H. If we consider the graph $G - E(H)$ after removing the edges of the cycle H we get a 2-regular graph that consists of a set of cycles.

Definition 4.2.1. Let G be a 4-regular graph with a hamiltonian cycle H. The cycles of $G - E(H)$ are called *inner cycles* of G. Furthermore, G is called *smooth* if the inner cycles are "non-selfcrossing" in the sense that the cyclic order of its vertices agrees with their cyclic order in H.

An example for a smooth graph is given in Figure 4.1.

Based on Sarvanov's conjecture [69] we formulate the following decision or search problem.

Problem 4.2.1. Given $n \in \mathbb{N}$ as input, does there exist a smooth graph with n vertices and independence number smaller than $\frac{2}{7}n$?

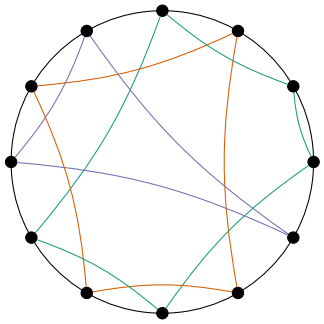


Figure 4.1: Smooth graph with twelve vertices and three inner cycles in different colors.

This problem can be generalized to the following family of problems by restricting the smooth graphs by some properties $\mathcal{P}$ and testing a lower bound of qn for some $q \in \left(\frac{1}{4}, \frac{1}{2}\right]$.

Problem 4.2.2 (Existence of Smooth Graphs with Small Independence Numbers (ESSI$(q, \mathcal{P})$))**.** Given $n \in \mathbb{N}$ as input, does there exist a smooth graph with n vertices that satisfies properties $\mathcal{P}$ and has independence number smaller than qn?

4.3 Algorithmic Approach

In this section we present a branch-and-bound approach that solves ESSI$(q, \mathcal{P})$, i.e. it checks for a given $n \in \mathbb{N}$ if there exists a smooth graph with n vertices and independence number smaller than qn that satisfies the conditions $\mathcal{P}$. The conditions of $\mathcal{P}$ can get added to the branch-and-bound approach in a problem-specific manner.

4.3.1 Solution Representation

If we assume that the hamiltonian cycle and therefore the order of the vertices in the hamiltonian cycle is given, every inner cycle of a smooth graph is already uniquely determined if we only know the set of its vertices. W.l.o.g. we assume the vertex set $V = \{1, \ldots, n\}$ to be ordered so that the hamiltonian cycle visiting the vertices in the order $1, \ldots, n$ is fixed. Therefore, we only have to partition the vertex set $\{1, \ldots, n\}$ into sets of size at least three and the result represents a smooth graph. For the rest of the algorithmic description section we use a partitioning of the ordered vertex set $\{1, \ldots, n\}$ into sets of size at least three as a solution representation.

4.3.2 Core Algorithm

The core algorithm is based on the branch-and-bound principle, see Section 2.3.2 The branching is done by assigning the next not yet assigned vertex in the order of the hamiltonian cycle to an already existing partition or to a new partition. The start solution is the solution where no vertex is assigned. After assigning a vertex to a partition

we check if the resulting partial solution satisfies all bounds and if there is a theoretical possibility to complete it to a solution that satisfies the wanted conditions. We call a partial solution that fails this check an infeasible partial solution. If the current partial solution is infeasible, we can cut off this branch and continue with the next partial solution. The infeasibility check of partial solutions is described in more detail in Section 4.5. Note that in contrast to the classical branch-and-bound procedure described in Section 2.3.2, which solves an optimization problem, we use here multiple lower bounds and do not compare them to our global best found objective, but to the given bound qn.

Whenever the branching reaches a complete solution, where all vertices are assigned to partitions, we compute its independence number and check the conditions $\mathcal{P}$. Note that computing the independence number is $\mathcal{NP}$-hard for the class of smooth graphs [31]. We compute it by solving the ILP

$$\begin{aligned} \max \ & \sum_{v \in V} x_v & & \\ \text{s.t. } & x_v + x_w \leq 1 & & \forall e \in E(G), e = vw \\ & x_v \in \{0,1\} & & \forall v \in V(G) \end{aligned}$$

As search strategy we use depth first search. Although for searching through the whole tree in order to obtain all feasible graphs, the search strategy is irrelevant since we are not reusing information of found solutions, it may be relevant for finding a feasible solution as fast as possible.

4.4 Bounds and Other Useful Properties

To reduce the search space for our problem we first derive some bounds and other properties for smooth graphs that may have an independence number smaller than qn. We mainly use the results of Fleischner, Sabidussi and Sarvanov to infer bounds and other properties [31, 29]. Those are then used for checking infeasibility and recognizing infeasible partial solutions as early as possible.

We consider the problem ESSI$(q, \mathcal{P})$ and we assume that the satisfaction properties $\mathcal{P}$ and the factor q are fixed. For the rest of this section we assume that G^* is a smooth graph with n vertices that satisfies the properties $\mathcal{P}$ and has independence number $\alpha(G^*) < qn$, i.e. G^* is a solution to the problem ESSI$(q, \mathcal{P})$. Let r^* be the number of inner cycles of G^*.

Fleischner and Sarvanov proved in [29] the following theorem.

Theorem 4.4.1. *Let G be a smooth graph with n vertices and r the number of inner cycles. Then the following holds.*

$$\alpha(G) \geq \frac{n - r}{3} \tag{4.1}$$

We use this theorem to compute a lower bound of r^*.

Corollary 4.4.2. *For G^* and r^* the following holds.*

$$r^* \geq n - 3\lceil qn \rceil + 3 \tag{4.2}$$

Proof. Since the independence number $\alpha(G^*)$ is integral we get from (4.1) that $\alpha(G^*) \geq \left\lceil \frac{n-r^*}{3} \right\rceil$.

$$\begin{aligned} \alpha(G^*) < qn \Rightarrow \left\lceil \frac{n-r^*}{3} \right\rceil < qn \Leftrightarrow \left\lceil \frac{n-r^*}{3} \right\rceil \leq \lceil qn \rceil - 1 \\ \Leftrightarrow \frac{n-r^*}{3} \leq \lceil qn \rceil - 1 \Leftrightarrow r^* \geq n - 3\lceil qn \rceil + 3 \end{aligned}$$

□

Inequality (4.1) can be strengthened if we exclude one special graph, which we call $G^{(2)}$. $G^{(2)}$ is defined for even n and is the unique simple smooth graph with only two inner cycles. $G^{(2)}$ is unique since the only possibility to being simple and having only two inner cycles is if all even vertices are in one inner cycle and all odd vertices are in another inner cycle. By excluding $G^{(2)}$ Fleischner and Sarvanov [29] proved the following stronger inequality.

Theorem 4.4.3. *Let G be a smooth graph with n vertices that is not isomorphic to $G^{(2)}$ and let r be the number of inner cycles. Then the following holds.*

$$\alpha \geq \frac{n - r + 1}{3} \tag{4.3}$$

Fleischner and Sarvanov stated this theorem with another equivalent condition. They proved Theorem 4.4.3 first for multigraphs and then showed that it also holds for simple graphs that have three consecutive vertices in different inner cycles. Putting this two conditions together we get that two consecutive vertices lie in different cycles, since the graph must be simple. Therefore, if three consecutive vertices never lie in three different inner cycles it must hold that vertex k and vertex $k+2$ always lie in the same inner cycle. This further implies that all even vertices form one inner cycle and so do all odd vertices. Therefore, the only graph that does not satisfy both conditions is $G^{(2)}$.

As before we can use this theorem to compute a stronger lower bound for r^*.

Corollary 4.4.4. *If G^* is not isomorphic to $G^{(2)}$ the following holds.*

$$r^* \geq n - 3\lceil qn \rceil + 4 \tag{4.4}$$

Proof. The proof is analogous to the proof of Corollary 4.4.2 by replacing (4.1) with (4.3). □

Another useful theorem is the following from [30].

Theorem 4.4.5 (Cycle-Plus-Triangles Theorem)**.** *Let G be a smooth graph where all inner cycles are triangles, i.e. have length three. Then G is 3-colorable.*

In [29] the following corollary of the cycle-plus-triangle theorem is stated.

Corollary 4.4.6. *Let G be a smooth graph with n vertices where all inner cycles have length smaller than or equal to four. Let r be the number of inner cycles and r_3 be the number of inner cycles of length three. Then the following holds.*

$$\alpha(G) \geq \frac{n - (r - r_3)}{3} \tag{4.5}$$

Let for the following corollary r_3^* be the number of inner cycles of length three of G^*.

Corollary 4.4.7. *G^* has either an inner cycle with length greater than four or the following holds.*

$$r^* \geq n - 3\lceil qn \rceil + 3 + r_3^* \tag{4.6}$$

Proof. The proof is analogous to the proof of Corollary 4.4.2 by replacing (4.1) with (4.5). □

Until now, we only provided lower bounds for r^*, but by using Theorem 4.4.5 we can also compute the following upper bound.

Corollary 4.4.8. *Let G^* and r^* be as described at the beginning of the section. Then $r^* < qn$ holds.*

Proof. We remove vertices for each inner cycle with length greater than three until every inner cycle has length three. For each removed vertex we connect the two neighbors in the inner cycle and the two neighbors in the hamiltonian cycle. The result is a smooth graph G' with $n' = 3r^*$ where all inner cycles are triangles. Removing vertices and adding edges can only decrease the independence number since every independent set in the transformed graph is also an independent set in the original graph. Therefore, we know $\alpha(G') \leq \alpha(G^*)$ and we can conclude the proof using Theorem 4.4.5 as follows.

$$qn > \alpha(G^*) \geq \alpha(G') = \frac{n'}{3} = r^*$$

□

4.5 Checking Infeasibility

To check if a given partial solution is infeasible, we use the bounds and properties from Section 4.4, and compute an as tight lower bound for the independence number of any completion of the partial solution as possible. Let S be a partial solution, i.e. S is a partitioning of a subset of the vertices of G.

To be able to use the lower bound from Corollary 4.4.4 for r, we need to exclude the graph $G^{(2)}$. To do this we check the conditions $\mathcal{P}$ for the unique graph $G^{(2)}$ and compute the independence number of it before we execute the branch-and-bound algorithm. Let r^{LB} be the lower bound for the number of inner cycles r, which we get from (4.4). Furthermore, let $r^{\mathrm{UB}} = \lfloor qn \rfloor$ be the upper bound for the number of inner cycles r, which we get from Corollary 4.4.8.

If $|S| > r^{\mathrm{UB}}$, the given partial solution is infeasible. Let $k = \sum_{P \in S} |P|$ be the number of fixed vertices in S and

$$\ell := \sum_{P \in S : |P| < 3} 3 - |P|$$

the number of vertices that are at least needed to complete all partitions of S. Furthermore, let $R_i := |\{P \in S : |P| \geq i\}|$ be the number of partitions in S with at least i vertices. Now we can show the following theorem.

Theorem 4.5.1. *Let S be a partial solution and r^{UB}, r^{LB}, k, ℓ and $(R_i)_{i \geq 3}$ be as described above. With that we can define the following value.*

$$m := \max\left[0, \min\left(5 - \max\left(3, \max_{P \in S} |P|\right), n - 3\lceil qn \rceil + 3 - R_4\right)\right].$$

If there exists a feasible completion of S, the following holds.

$$k + \ell + m + 3\max(0, r^{\mathrm{LB}} - |S|) \leq n \tag{4.7}$$

Proof. First of all every completion of S must complete all partitions $P \in S$ with $|P| < 3$, which implies that at least ℓ vertices must be added to the k existing ones. If $|S| < r^{\mathrm{LB}}$ we know that a completion of S with the desired properties must have at least r^{LB} different partitions and therefore $3(r^{\mathrm{LB}} - |S|)$ additional vertices must be added.

By Corollary 4.4.7 either the completion must contain a partition of size at least five or (4.6) must hold. To get a partition of size five we can add

$$\max(0, 5 - \max(3, \max_{P \in S} |P|))$$

additional vertices to the largest partition. Otherwise, to satisfy (4.6) we need to have $n - 3\lceil qn \rceil + 3$ many partitions of size at least four. We have at the moment R_4 many inner cycles with length at least four and therefore we need $\max(0, n - 3\lceil qn \rceil + 3 - R_4)$ many additional vertices to get enough inner cycles of length four.

Plugging everything together and considering that in total we have n vertices we get (4.7). □

If (4.7) is violated, we know that S is infeasible.

We covered now the cases where we can determine that S is infeasible without even computing an independent set. Now we compute an independent set on the partial graph of S, which is the graph induced by all fixed vertices $V_S = \bigcup_{P \in S} P$. By the branching rules we know that $V_S = \{1, \dots, k\}$ for some $k \leq n$.

The partial graph $G_S = (V_S, E_S, \psi_S)$ consists of the fixed vertices and all possible edges between those vertices. Since we do not know if a partition $P \in S$ with $|P| \geq 3$ is already complete or not, we also do not know if the vertices $\min(P)$ and $\max(P)$ are connected or not. We want that every independent set in G_S is also an independent set in G and therefore we have to add those edges to E_S. Therefore, E_S consists of the edges of the hamiltonian cycle between $1, \dots, k$ and the inner edges between the partitions i.e. edges of the form $e = i_k i_{k+1}$ or $e = i_1 i_\ell$ where $P = \{i_1, i_2, \dots, i_\ell\} \in S$ with $i_1 < i_2 < \cdots < i_\ell$ and $k \in \{1, \dots, \ell - 1\}$.

To compute an independent set on G_S we use the minimum-degree greedy algorithm [40]. In each iteration this algorithm adds a vertex with the minimum degree to the independent set and removes the vertex and all its neighbors from the graph. Besides good approximation ratios the greedy algorithm is also fast, it can be implemented in $\mathcal{O}(k)$ time.

Let I be the independent set found by the minimum-degree greedy on the graph G_S. Our goal is now to find a good lower bound on how many additional vertices can be added to I in each completion of S.

Theorem 4.5.2. *Let S be a partial solution and I an independent set on the graph G_S. Furthermore, let k, ℓ, m and r^{UB} be as described in Theorem 4.5.1 and let*

$$n_I^{\max} := |I \cap \{1, k\}| + |I \cap \{\min P : P \in S\}| + |I \cap \{\max P : P \in S\}|.$$

Then there exists for every completion G of S an independent set I_G with

$$|I_G| \geq |I| + \frac{\left\lceil n - k - n_I^{\max} - \min\left(r^{\mathrm{UB}} - |S|, \frac{n-k-\ell-m}{3}\right)\right\rceil}{3}. \tag{4.8}$$

Proof. Let G be an arbitrary completion of S. First of all we upper bound the number of inner cycles r of G. Clearly we know $r \leq r^{\mathrm{UB}}$. Furthermore, by using the same reduction as in the proof of Theorem 4.5.1 with r instead of r^{LB} we get

$$k + \ell + m + 3\max(0, r - |S|) \leq n \Rightarrow r \leq \frac{n - k - \ell - m}{3} + |S|. \tag{4.9}$$

Now we can compute a lower bound on the independence number of G. Let $V_I \subseteq V_G \setminus V_S$ be the set of all vertices in G that are not in V_S and are adjacent to one of the vertices in I. The vertices of V_I are either connected to I via the hamiltonian cycle, which is only possible if the vertex 1 or the vertex k is in I, or via an inner cycle, which is only

possible for the end vertices $\min P$ and $\max P$ of an inner cycle $P \in S$. Therefore, we can bound the size of V_I by

$$|V_I| \leq |I \cap \{1, k\}| + |I \cap (\{\min P : P \in S\}| + |I \cap \{\max P : P \in S\})| = n_I^{\max}.$$

We consider now the residual graph G^{rem} after removing the vertices V_S and V_I from G, which is a graph with $n - k - |V_I|$ vertices. We complete the independent set I by an algorithm that is similar to the minimum-degree greedy algorithm. Instead of always taking a vertex with the minimum degree we take the minimum remaining vertex, i.e. the first vertex in the order of the hamiltonian cycle that is not adjacent to any vertex in the independent set so far.

Let $I_0 = I$ be the start set and I_i the set after iteration i and let v_i be the vertex added in iteration i. Furthermore, let P_i be the partition in G of the vertex v_i and G_i be the remaining graph in iteration i, $G_0 = G^{\text{rem}}$. We distinguish two cases, the case if $v_i = \min(P_i)$ is the first vertex in P_i or not. Since we selected v_i as the first vertex in the order of the hamiltonian cycle that is still in G_{i-1} we know that the preceding neighbor of v_i in the hamiltonian cycle is not in G_{i-1} and therefore we obtain that the degree $d_{G_{i-1}}(v_i)$ of v_i in G_{i-1} is smaller than or equal to three. If $v_i \neq \min(P_i)$ we also know that one neighbor in the inner cycle containing v_i is a predecessor of v_i in the hamiltonian cycle and therefore it is also not in G_{i-1}, which gives us $d_{G_{i-1}}(v_i) \leq 2$. Summing up the removed vertices during the greedy algorithm over all iterations we get

$$n - k - |V_I| = \sum_{i=1}^{x} d_{G_{i-1}}(v_i) + 1 \leq x + 3(r - |S|) + 2(x - r + |S|)$$

$$\Rightarrow x \geq \frac{n - k - |V_I| - r + |S|}{3} \geq \frac{n - k - |V_I| - \min\left(r^{\text{UB}} - |S|, \frac{n-k-\ell-m}{3}\right)}{3}.$$

In total, we constructed a new independent set I_G with $|I|+x$ elements and therefore (4.8) holds. □

If $\mathcal{P}$ is not empty we can calculate problem specific bounds for those constraints and check them. To summarize this section Algorithm 4.1 describes the whole procedure for checking infeasibility.

4.5.1 Symmetry Breaking

Until now the branch-and-bound procedure considers many isomorphic graphs, such as all rotations alongside the hamiltonian cycle and their reversals. In this section we describe how we break those symmetries.

To this end we define the gap sequence of a complete solution.

Definition 4.5.1. Let S be a complete solution, i.e., a partitioning of the vertex set $V = \{1, \ldots, n\}$. Let $P_i \in S$ be the partition of vertex i and let g_i be the gap between

Algorithm 4.1: Checking Infeasibility

Input: n, q, $\mathcal{P}$ and a partial solution S
Output: Either `INFEASIBLE` or `POSSIBLY_FEASIBLE`

Compute r^{LB}, r^{UB}, k, ℓ, m;
if $|S| > r^{\mathrm{UB}}$ **then**
  return `INFEASIBLE`
end
if (4.7) *is not satisfied* **then**
  return `INFEASIBLE`
end
Construct G_S and apply minimum-degree greedy to get independent set I;
Compute $V_I^{\max}$;
if $|I| + \max\left(0, \left\lceil \frac{\lfloor n-k-n_I^{\max} - \min\left(r^{\mathrm{UB}}-|S|, \frac{n-k-\ell-m}{3}\right)\rfloor}{3} \right\rceil\right) \geq qn$ **then**
  return `INFEASIBLE`
end
if *Problem specific bound check for* $\mathcal{P}$ *fails* **then**
  return `INFEASIBLE`
end
return `POSSIBLY_FEASIBLE`

vertex i and its successor j in the partition P_i, i.e., let $j = \min\{j \in P_i : j > i\}$ if this set is not empty or $j = \min\{j \in P_i : j < i\}$ otherwise and $g_i = j - i$ if $j > i$ or $g_i = j + n - i$ otherwise. We call the sequence $(g_i)_{i=1}^n$ the *gap sequence* of S.

If two S have the same gap sequence they are not only isomorphic but also exactly the same according to the vertex labeling. We break those symmetries by ensuring that the gap sequence is minimal according to the lexicographical order under all rotations alongside the hamiltonian cycle and their reversals. Be aware that rotating alongside the hamiltonian cycle simply means shifting the gap sequence, but reversing the hamiltonian cycle is a non-trivial change in the gap sequence.

We can compute the gap sequence not only for complete solutions but also for partial solutions. In some cases the next gap is not yet known and instead of calculating a gap we can calculate a lower bound and an upper bound for the gap. With the lower and upper bounds we can check if there is a rotation that always leads to a smaller gap sequence. We can also compute lower and upper bounds for the reversed gap sequence and also check if reversing leads to a smaller gap sequence.

If we found a rotation or a reversed rotation that always leads to a smaller gap sequence, we can fathom the current branch and continue with the next one.

Table 4.1: Results for selected values of n for Problem 1 and Problem 2

	Problem 1		Problem 2	
n^{**}	$t[s]$	candidates	$t[s]$	candidates
8	< 1	1	< 1	1
11	< 1	3	< 1	1
15	< 1	5	< 1	0
18	94	2298	33	259
22	25 443	5795	5047	145
25	$> 5\,000\,000$	$> 330\,000$	4 868 324	160 556
29	$> 5\,000\,000$	> 1463	$> 5\,000\,000$	$> 60\,713$

4.6 Computational Results

In this section we present computational results for instances to four different problems. Our algorithm is implemented in C++ and compiled with g++ 4.8.4. To solve the ILP model for finding a maximum independent set we used Gurobi Optimizer 7.0.1[1]. All tests were performed on a single core of an Intel Xeon E5540 processor with 2.53 GHz and 2 GB RAM.

We consider four different variants of the problem. The first and original variant is with $q_1 = \frac{2}{7}$ and with an empty constraint set $\mathcal{P}_1 = \emptyset$. The second problem is also with $q_2 = \frac{2}{7}$ but with the additional constraint that all inner cycles have length at most four, i.e. $\mathcal{P}_2 = \{(R_5 = 0)\}$. The third problem is with $q_3 = \frac{5}{16}$ and $\mathcal{P}_3 = \{(\text{all inner cycles have length } 4)\}$. The fourth problem is with $q_4 = 0.334$ and $\mathcal{P}_4 = \{(\text{G contains no triangles})\}$. The motivations behind the choices of $\mathcal{P}$ are explained subsequently.

4.6.1 Problem 1

We tested the implementation for $n \in \{6, \ldots, 29\}$. The algorithm found for $n = 8$ one feasible solution and $n = 11$ two feasible solutions. For all larger n it could not find any feasible solutions. Furthermore, the algorithm was able to finish the branch-and-bound search for all $n \leq 24$, which proves that for $n = 8$ and $n = 11$ the found feasible solutions are the only ones and for all other $n \leq 24$ there does not exist any feasible solution. For $n > 24$ it could not finish the search within 5 000 000 seconds.

The interesting values of n are the ones where $2n/7$ is only a little bit larger than $\lfloor 2n/7 \rfloor$, since then it may be easier to find a graph with independence number $\lfloor 2n/7 \rfloor$. Therefore, we are especially interested in the values $n \equiv 1 \pmod 7$ and $n \equiv 4 \pmod 7$. Table 4.1 summarizes the results and running times for those values and compares them with the results of Problem 2. Column $t[s]$ shows the run time in seconds and column *candidates*

[1] https://www.gurobi.com (accessed 09/2019)

the number of complete solutions that got checked by the ILP solver.

4.6.2 Problem 2

Problem 2 is a more restricted variant of Problem 1 and was tested to check if the restriction helps speeding up the search. Especially the bound corresponding to the value m can be improved through this restriction. We tested again all inputs $n \in \{6, \ldots, 29\}$. For $n = 8$ and $n = 11$ the algorithms found one solution, the second solution of $n = 11$ contains an inner cycle of length five. For all larger n it also could not find any feasible solution.

Through the speedup compared to Problem 1 the algorithm was able to finish the search for all $n \leq 28$ and therefore proves for all $11 < n \leq 28$ that there does not exist a feasible solution. For $n = 29$ it could not finish the search within 5 000 000 seconds. Table 4.1 summarizes the results and running times and compares them with Problem 1.

4.6.3 Problem 3

Fleischner conjectured that smooth graphs only containing inner cycles of length four with at least 12 vertices have independence number at least $5n/16$. This was the motivation to consider this problem with $q_3 = \frac{5}{16}$. Our algorithm was able to disprove the conjecture by finding 36 smooth graphs with 20 vertices and independence number $6 < qn = 20 \cdot 5/16$ containing only inner cycles of length four. Furthermore, it could find feasible graphs with 24 vertices and independence number $7 < qn = 24 \cdot 5/16$.

Clearly we only have to consider values for n with $n \equiv 0 \pmod 4$. For $n = 8$ we found the same graph as in Problem 1 and 2, for $n = 12$ and $n = 16$ the algorithm could prove that there are no feasible graphs. For $n = 20$ it could finish the search and prove that the found 36 feasible graphs are the only ones but for $n = 24$ the search did not finish in under 5 000 000 seconds.

The run time until the first solution got found for $n = 20$ was 11 minutes and for $n = 24$ it was 11 hours. For $n = 28$ the algorithm could not finish in reasonable time and also did not find a feasible solution in the first 5 000 000 seconds run time.

4.6.4 Problem 4

For triangle-free smooth graphs it is proven that $4n/13$ is a valid lower bound for the independence number [49]. This raises the question if it is possible to reach this lower bound or if there exists a stronger lower bound. We use $q = 0.334$ since we want to check if there exist triangle-free smooth graphs with independence number smaller than or equal to $n/3$ and therefore we could use for q any value $1/3 + \varepsilon$ with a small $\varepsilon > 0$. The algorithm was not able to find a graph with independence number smaller than $n/3$ but it was able to find graphs with independence number $n/3$. It could solve the instances up to $n = 26$ in under 5 000 000 seconds.

4.7 Conclusion and Future Work

In this chapter we formalized a family of problems for finding smooth graphs with small independence numbers. We proposed an algorithm for solving problems of this family that is based on branch-and-bound. To increase the effectivity of the algorithm by computing good bounds, we used graph-theoretic results to obtain properties and bounds for the number of inner cycles and their sizes. Using those results we proposed a procedure for computing a strong lower bound on the independence number of partial solutions to detect infeasibility as early as possible. We applied our algorithm to four different problems and reported the results and the running times for different graph sizes. Doing this we could disprove one conjecture and find more support for other conjectures for small graphs.

Future work may be to compare different heuristics for computing independent sets for partial solutions. Furthermore, one idea could be to search for a minimum feasible graph, which may enable some reduction properties and therefore some stronger bounds, as we did it for uniquely hamiltonian graphs in Section 3.3.3. Additionally, it would be interesting to use a metaheuristic to solve our problems, which would allow searching larger smooth graphs with small independence numbers heuristically.

CHAPTER 5

Sup-Transition Minor Free Graphs

In this chapter we are concerned with searching for sup-transition minors, an extension of the minor concept to transitioned graphs. We are developing a mathematical model for modeling sup-transition minors. Based on the model we propose a MILP and a SAT approach. We improve the approaches with non-trivial symmetry breaking constraints and finally compare the two approaches and use them to derive new graph theoretic insights.

5.1 Introduction

The cycle double cover (CDC) conjecture is a longstanding famous conjecture in graph theory.

Definition 5.1.1. A *cycle double cover (CDC)* of a graph G is a multiset of cycles in G such that each edge of G is contained in exactly two cycles.

Conjecture 5.1.1 (*CDC conjecture*)**.** Every bridgeless graph contains a CDC.

This already over 40 years old conjecture is well studied and was originally posed by Szekeres [75] and Seymour [71]. Jaeger reduced the problem from general bridgeless graphs to the class of all snarks [45]. There are several similar definitions of snarks, but we use the same definition as Jaeger does.

Definition 5.1.2. A *snark* is a simple cyclically 4-edge connected cubic graph with chromatic index four.

Note that by Lemma 2.1.2 a snark is also essentially 4-edge connected.

Finding a CDC in a snark, or in general in a 3-regular graph, is correlated to the compatible circuit decomposition (CCD) problem, which is formulated on graphs with a transition system. A transition system is a collection of transitions, where each transition represents a set of two adjacent edges. For a formal definition see Definition 5.2.1 in Section 5.2.1. The CCD problem asks for a given 2-connected eulerian graph G and a transition system $\mathcal{T}$ if there exists a set of circuits in G such that each edge is contained in exactly one circuit and none of its circuits contain both edges of any transition T in $\mathcal{T}$.

One correlation between the CDC conjecture and the CCD problem can be seen via line graphs.

Definition 5.1.3. Let G be a graph. The *line graph* $L(G)$ of G is defined by $V(L(G)) := E(G)$ and for each vertex $v \in V$ and $e_1, e_2 \in E(v), e_1 \neq e_2$ we add an edge connecting e_1 with e_2 to $E(L(G))$. Note that if e_1 and e_2 are parallel edges then e_1 and e_2 are connected in $L(G)$ also with two parallel edges.

Theorem 5.1.1. *Consider for a 3-regular graph G its 4-regular line graph $L(G)$ together with two transitions per vertex as shown in Figure 5.1. If the line graph together with the given transition system contains a CCD, one can construct a CDC of the original graph G.*

Proof. Let $\mathcal{C}$ be the set of circuits in $L(G)$ corresponding to the CDC. Note that a all vertices in $L(G)$ have degree two and therefore cannot be used twice in a circuit $C \in \mathcal{C}$ since this would imply that C contains all edges of the transitions at this vertex. Therefore, all $C \in \mathcal{C}$ are cycles in $L(G)$. The vertices of a cycle in $L(G)$ form an edge sequence in G corresponding to a trail in G. The collection of those trails in G covers every edge in G exactly twice since the circuits in $\mathcal{C}$ cover every vertex in $L(G)$ exactly twice. We can split up the trails in G to cycles in G and get a CDC in G. □

Another correlation between the CCD problem and the CDC conjecture for 3-regular simple graphs can be seen by the following construction.

Theorem 5.1.2. *Let G be a 3-regular simple triangle-free graph and H the graph obtained from G after contracting each edge of a PPM M of G. Now we define a transition system on H by adding transitions between two edges if and only if their corresponding edges in G are adjacent, see Figure 5.2 for an illustration. Then, if H contains a CDC, one can construct from it a CDC in the original graph G containing the 2-factor $Q = E(G) - M$ as a subset.*

Proof. Consider a CDC $\mathcal{C}$ in H. Each $C \in \mathcal{C}$ can be extended to a circuit C' in G by using the contracted edges. With this extension the circuits $\{C' \mid C \in \mathcal{C}\}$ cover all contracted edges in M twice and all edges in Q once. But Q is a 2-factor and therefore consists of a set of cycles. Adding this set of cycles to $\{C' \mid C \in \mathcal{C}\}$ we get CDC of G that contains the cycles of Q as a subset. □

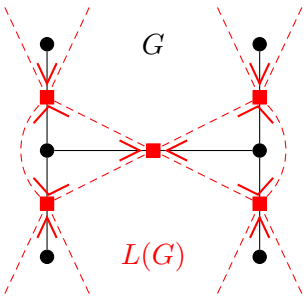


Figure 5.1: Transformation of a part of a 3-regular graph G into its line graph $L(G)$ and two transitions per vertex, represented by a red vee ($\vee$) between their two edges. The vertices of G are represented by black circles, the edges of G by black solid lines, the vertices of $L(G)$ by red squares, and the edges of $L(G)$ by red dashed lines.

Figure 5.2: Example contraction of parts of a PPM. The edges of the PPM getting contracted are drawn dashed. The transitions in the resulting graph are represented by a vee ($\vee$) between the two edges of the transition.

Note that triangle-freeness is needed to ensure that the graph does not contain self-loops after contracting a claw, but it may contain parallel edges. Using this theorem we get that to prove the existence of a CDC in a 3-regular simple triangle-free graph it suffices to find a PPM such that its contraction leads to a transitioned graph containing a CCD. Since the number of vertices of the contracted graph is at most half of the number of vertices of the original graph, it may be faster to find a CCD in the much smaller graph than a CDC in the original graph. On the other hand, if a snark contains a CDC one cannot conclude that it also contains a perfect matching whose contraction leads to a graph with a CCD; the Peterson graph is a counter example. If we allow PPMs, this direction is still an open problem.

Already in 1980 Fleischner [25] proved that every 2-connected planar eulerian graph has a CCD regardless of the structure of the transition system. This result was generalized by Fan and Zhang [24] who proved that whenever the graph has no K_5-minor it has a CCD. These two results both only use the graph structure and ignore the transition system. In order to include the structure of the transition system, Fleischner et al. [33] extended the definition of transition minors to transitioned graphs and proved that if a

transitioned 2-connected eulerian graph is sup-undecomposable K_5 (SUD-K_5)-minor-free, it contains a CCD, which is a generalization of Fan and Zhang's result. For a definition of SUD-K_5-minor-free graphs see Definition 5.2.7 and Example 5.2.1 in the next section.

This recent result of Fleischner et al. leads to the question of how to check for a graph if it contains a SUD-K_5-minor. This task is not easy because of the complex nature of the definition of a SUD-K_5-minor. Because of this difficulty Fleischner et al. could not provide a snark and a PPM whose contraction leads to a graph that is SUD-K_5-minor-free but has a K_5-minor, i.e. an example in the context of snarks where the new theorem is stronger than the old theorem.

In this chapter we analyze the problem of finding a SUD-K_5-minor and formulate a generalization that checks if a graph is sup-$(H, \mathcal{S})$-minor-free. Furthermore, we prove some complexity results for this problem concerning $\mathcal{NP}$-hardness and solvability in polynomial time if H is fixed. Then, we develop a mathematical model for testing the existence of sup-$(H, \mathcal{S})$-minors and prove its correctness. Based on the model, we propose a MILP and a SAT model. We then compare and use those two approaches to get further graph theoretic insights.

There is no literature yet that is concerned with finding SUD-K_5-minors, although the problem of finding K_5-minors is well analyzed. Robertson and Seymour proved that checking if a graph contains a K_5-minor can be done in polynomial time [67]. We use the same proof-idea to prove that checking whether a graph contains a SUD-K_5-minor can be done in polynomial time. The polynomial algorithm described in the proof of Robertson and Seymour is not practically applicable since its computation time has large constants and polynomial factors. A more practical algorithm was provided by Reed and Li who proved that checking if a graph contains a K_5-minor can be done in linear time [65]. This algorithm heavily depends on the fact that a 4-connected graph contains no K_5-minor if and only if it is planar and that checking planarity can be done in linear time. Since there is no known extension of planarity to transitioned graphs that would lead to a connection between planarity and the existence of a SUD-K_5-minors, this linear time approach cannot easily be extended to checking the existence of SUD-K_5-minors.

To improve the solving times of the MILP as well as the SAT model we propose a non-trivial symmetry breaking based on graph automorphisms of the two input graphs $(G, \mathcal{T})$ and $(H, \mathcal{S})$. The idea of breaking symmetries using automorphism groups has been studied in a general context, see e.g. [2], and in problem-specific contexts, see e.g. [47]. We extend the definition of automorphisms to transitioned graphs and propose problem specific symmetry breaking constraints based on a vertex mapping between the two input graphs $(G, \mathcal{T})$ and $(H, \mathcal{S})$.

In the next section we introduce the problem existence of sup-transition minors (ESTM), its equivalent problem existence of basic sup-reduced-transition minors (EBSRTM) and prove the complexity results. A mathematical model for EBSRTM is then presented in Section 5.3.2, which also includes the correctness proof of the model. Based on the mathematical model a MILP is presented in Section 5.4 and a SAT model in Section 5.5.

Then we discuss symmetry breaking constraints, which can be used in the MILP and in the SAT model, in Section 5.6. The computational results of the MILP and the SAT model are presented in Section 5.8. Finally, we conclude with Section 5.9 and propose possible future work.

5.2 Problem Formulation

Before we can formulate the problem we need some mathematical notations and graph theoretic definitions. For a partial function $\alpha\colon A \nrightarrow B$ we use the following notations. For a subset $X \subseteq A$ we write $\alpha[X] := \{b \in B \mid \exists a \in X : b = \alpha(a)\}$ for the image of X under α and for $a \in A$ we simply write $\alpha[a] := \alpha[\{a\}]$. Similarly, for a subset $Y \subseteq B$ we write $\alpha^{-1}[Y] := \{a \in A \mid \alpha(a) \in Y\}$ for the preimage of Y under α and for $b \in B$ we simply write $\alpha^{-1}[b] := \alpha^{-1}[\{b\}]$. Note that $\alpha^{-1}[b]$ may be empty if α is not surjective. If α is injective we denote by $\alpha^{-1}\colon B \nrightarrow A$ the inverse partial function of α and $\alpha^{-1}(b) = a$ if and only if $\alpha(a) = b$. Furthermore, we denote by $\mathrm{dom}(\alpha) = \alpha^{-1}[B] \subseteq A$ the domain of α.

5.2.1 Basic Definitions

We define a transition system as in [33] but use a different notation, which will be useful later on.

Definition 5.2.1 (Transition System)**.** Let G be a graph. A *transition system* of G is a set $\mathcal{T} \subseteq V \times \mathcal{P}_2(E)$ of *transitions* that satisfies the following. Each transition $T \in \mathcal{T}$ with $T = (v, \{e_1, e_2\})$ has to satisfy $\{e_1, e_2\} \subseteq E(v)$. We use the projections $\pi_1(T) := v$ and $\pi_2(T) := \{e_1, e_2\}$ to denote the values of T. Furthermore, we write $\mathcal{T}(v) := \{T \in \mathcal{T} \mid \pi_1(T) = v\}$ for the set of all transitions at vertex v. The transitions at a vertex v must all be edge-disjoint, i.e.

$$\pi_2(T_1) \cap \pi_2(T_2) = \emptyset \quad \forall v \in V, \forall T_1 \in \mathcal{T}(v), \forall T_2 \in \mathcal{T}(v) : T_1 \neq T_2$$

The graph G with a non-empty transition system $\mathcal{T}$ is called a *transitioned graph* and denoted by $(G, \mathcal{T})$. A *completely transitioned graph* is a transitioned graph where for each vertex each incident edge is in one transition of the vertex, i.e. $E(v) = \bigcup_{T \in \mathcal{T}(v)} \pi_2(T)$ for all $v \in V(G)$. For every subgraph H of G, $\mathcal{T}|_H = \{T \in \mathcal{T} \mid \pi_2(T) \subseteq E(H)\}$. Clearly, a connected completely transitioned graph is eulerian.

The following two definitions are adopted from [33].

Definition 5.2.2 (separator)**.** Let G be a graph. A vertex subset U is a *separator* of G separating G to G_1, G_2 if $E(G) = E(G_1) \cup E(G_2)$, $V(G_1) \cap V(G_2) = U$, and $E(G_1) \cap E(G_2) = \emptyset$. We call U a *t-separator* if $|U| = t$. We say a separator U *separating subgraphs* X_1, X_2 of G if U is a separator of G separating G to G_1, G_2 with $X_i \subseteq G_i$, $i = 1, 2$.

Definition 5.2.3 (bad-cut-vertex)**.** Let $(G, \mathcal{T})$ be a transitioned graph. A 1-separator $\{v\}$ separating G to G_1, G_2 is a *bad-cut-vertex* if $(v, E(v) \cap E(G_1)) \in \mathcal{T}(v)$ implying that $|E(v) \cap E(G_1)| = 2$.

Definition 5.2.4 (minor)**.** Let G be a graph. H is a *minor* of G if and only if H can be derived from G by deletion of vertices, deletion of edges and contraction of edges.

For our purposes we use the following equivalent representation of H. The vertices of H correspond to non-empty vertex-disjoint connected subgraphs of G. We can formalize this by a partial surjective function $\varphi\colon V(G) \twoheadrightarrow V(H)$, which maps vertices in G to vertices in H. Note that φ is a partial function, which means that there might be vertices in G that do not get mapped on vertices in H. Furthermore, the preimage $\varphi^{-1}[w] \subseteq V(G)$ for each $w \in V(H)$ must be connected in G.

The edges of H correspond to edges of G. We can again formalize this by a partial injective and surjective function $\kappa\colon E(G) \twoheadrightarrow E(H)$, which maps an edge in G to its corresponding edge in H. The end vertices of an edge $\kappa(e) \in E(H)$ correspond to the connected subgraphs that contain the end vertices of the edge e in G. Formally this means

$$\psi_H(\kappa(e)) = \varphi\left[\psi_G(e)\right] \quad \forall e \in \mathrm{dom}(\kappa). \tag{5.1}$$

Note that we also do not allow loops for minors, even if you could generate one by contracting some edges. Next we define a transition minor as in [33] but with a different notation.

Definition 5.2.5 (transition minor)**.** Let $(G, \mathcal{T})$ be a transitioned graph and H a minor of G with correspondence maps φ and κ. We define a transition system $\mathcal{S}$ on H as follows. We keep all transitions whose edges do not get deleted or contracted; formally that means:

$$\mathcal{S}' := \{(\varphi(\pi_1(T)), \kappa[\pi_2(T)]) \mid T \in \mathcal{T}, \pi_1(T) \in \mathrm{dom}(\varphi), \pi_2(T) \subseteq \mathrm{dom}(\kappa)\}. \tag{5.2}$$

The transitioned graph $(H, \mathcal{S}')$ is called a *reduced transition minor* of $(G, \mathcal{T})$.

If $w \in V(H)$ is a vertex of degree four and there exists a transition between two of the incident edges, we also want to add a transition between the other two edges. Formally we get all in all the following transition system

$$\mathcal{S} := \mathcal{S}'(w) \cup \{(w, E(w) \setminus \pi_2(T)) \mid w \in V(H), \deg(w) = 4, T \in \mathcal{S}'(w)\}. \tag{5.3}$$

We call the transition graph $(H, \mathcal{S})$ a *transition minor* of $(G, \mathcal{T})$.

The next two definitions are generalizations of the definitions of a SUD-K_5 and of SUD-K_5-minor-free from [33].

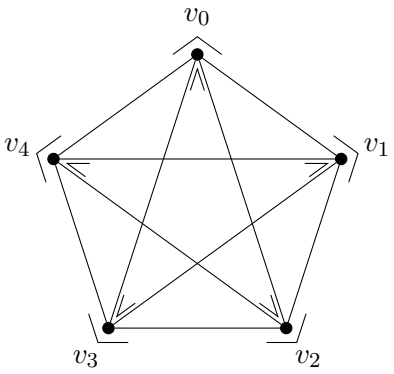


Figure 5.3: The completely transitioned graph UD-K_5 with transitions represented by a vee ($\vee$) between their two edges.

Definition 5.2.6 (sup-$(H, \mathcal{S})$)**.** Let $(H, \mathcal{S})$ be a completely transitioned 4-regular graph. A transitioned graph $(H', \mathcal{S}')$ is a *sup-*$(H, \mathcal{S})$ graph if the following holds.

The graph H' can be decomposed into $|E(H)|+|V(H)|$ connected edge-disjoint subgraphs

$$\{P_f \mid f \in E(H)\} \cup \{Q_w \mid w \in V(H)\}$$

as follows.

1. The graphs $\{Q_w \mid w \in V(H)\}$ are vertex-disjoint connected subgraphs of H'.
2. Each P_f for $f \in E(H)$ with $f = w_1w_2$ is a path in H' joining $V(Q_{w_1})$ and $V(Q_{w_2})$, and all $\{P_f \mid f \in E(H)\}$ are internally disjoint.
3. Let Q_w^+ be the subgraph of H' induced by $E(Q_w)$ and the four adjacent paths P_f for $f \in E(w)$. Furthermore, let $\mathcal{S}(w) = \{S_w^1 = (w, \{f_1, f_2\}), S_w^2 = (w, \{f_3, f_4\})\}$. Then the subgraph Q_w^+ has a bad 1-separator $\{u_w\}$ separating H_w^1 and H_w^2 such that $P_{f_1} \cup P_{f_2} \subseteq H_w^i$ and $P_{f_3} \cup P_{f_4} \subseteq H_w^{3-i}$ for some $i \in \{1, 2\}$.

Definition 5.2.7 (sup-$(H, \mathcal{S})$-minor-free)**.** Let $(H, \mathcal{S})$ be a completely transitioned 4-regular graph. A transitioned graph $(G, \mathcal{T})$ is *sup-*$(H, \mathcal{S})$*-minor-free* if and only if it does not have any eulerian transition minor $(H', \mathcal{S}')$ that is a sup-$(H, \mathcal{S})$ graph.

Example 5.2.1 (SUD-K_5)**.** The completely transitioned four-regular graph $(H, \mathcal{S})$, called *undecomposable* K_5 *(UD-*K_5*)*, is defined by $H = K_5$ and

$$\mathcal{S} = \{(v_i, \{v_{i-1}v_i, v_iv_{i+1}\}), (v_i, \{v_{i-2}v_i, v_iv_{i+2}\}) \mid i \in \mathbb{Z}_5\},$$

see Figure 5.3. With this notation a sup-$(H, \mathcal{S})$ graph is called a *sup-undecomposable* K_5 *(SUD-*K_5*)*. If a graph is sup-$(H, \mathcal{S})$-minor-free it is called a *SUD-*K_5*-minor-free graph*.

We are focusing in this chapter primarily on the following question.

Problem 5.2.1 (*existence of sup-transition minors (ESTM)*)**.** Given a transitioned graph $(G, \mathcal{T})$ and a completely transitioned 4-regular graph $(H, \mathcal{S})$, does there exist an eulerian transition minor of $(G, \mathcal{T})$ that is a sup-$(H, \mathcal{S})$ graph?

Note that ESTM is the inverse problem of asking if a transitioned graph $(G, \mathcal{T})$ is sup-$(H, \mathcal{S})$-minor-free. Next we prove the following two complexity theorems.

Theorem 5.2.1. *ESTM is $\mathcal{NP}$-complete.*

Theorem 5.2.2. *ESTM restricted to simple graphs is $\mathcal{NP}$-complete.*

Formally it would be enough to prove that ESTM is in $\mathcal{NP}$ and that ESTM restricted to simple graphs is $\mathcal{NP}$-hard. But since the $\mathcal{NP}$-hardness proof for ESTM restricted to simple graphs is based on the same idea as the $\mathcal{NP}$-hardness proof for the general ESTM, we first proof the more basic statement for the general ESTM.

We use the following lemmas to prove Theorem 5.2.1.

Lemma 5.2.3. *If a graph H is a minor of a graph G with $|V(H)| = |V(G)|$, then H is a subgraph of G.*

Proof. We cannot contract any edges in G to get H, since that would reduce the number of vertices. Therefore, we only remove edges from G to get H, which results in a subgraph of G. □

Lemma 5.2.4. *Let $(H, \mathcal{S})$ be a completely transitioned graph and $(H', \mathcal{S}')$ a sup-$(H, \mathcal{S})$ graph. Then H is a minor of H'.*

Proof. By contracting the paths P_f to one edge f and the connected subgraphs Q_w to one vertex w we get exactly H. Therefore, H is a minor of H'. □

Lemma 5.2.5. *The minor relation is transitive, i.e. if H is a minor of H' and H' is a minor of G, then H is also a minor of G.*

Proof. A minor can be constructed by a finite number of edge removals, vertex removals and edge contractions. If we apply the steps from G to H' and then the steps from H' to H we still only applied a finite number of steps and got from G to H. Therefore, H is also a minor of G. □

Proof of Theorem 5.2.1. First we prove that ESTM is in $\mathcal{NP}$. As a solution representation we use a transitioned graph $(H', \mathcal{S}')$ with at most $|V(G)|$ vertices together with the minor correspondence maps $\varphi\colon V(G) \twoheadrightarrow V(H')$ and $\kappa\colon E(G) \twoheadrightarrow E(H')$, the decomposition $\{P_f \mid f \in E(H)\} \cup \{Q_w \mid w \in V(H)\}$ of H' and the sequence of 1-separator vertices

$(u_w)_{w \in V(H)}$. The size of the solution representation is polynomial in the input size. Furthermore, checking for a given solution representation if $(H', \mathcal{S}')$ is by the embeddings φ and κ an eulerian transition minor of $(G, \mathcal{T})$ can be done in polynomial time. Last but not least, using the given decomposition and 1-separator vertices it can be checked in polynomial time if $(H', \mathcal{S}')$ is a sup-$(H, \mathcal{S})$-graph. Therefore, ESTM is in $\mathcal{NP}$.

To prove that ESTM is $\mathcal{NP}$-hard we define a polynomial-time reduction from the $\mathcal{NP}$-hard Hamiltonian cycle problem to ESTM. Let G be a simple graph for which we want to check if it contains a Hamiltonian cycle. We define a double cycle graph H with $n := |V(G)|$ vertices, i.e.

$$V(H) := \{1, ..., n\}.$$

The edges of H form two Hamiltonian cycles in H, i.e.

$$E(H) = \{e_1, ..., e_{2n}\}$$

with $\psi(e_i) = \psi(e_{i+n}) = \{i, i+1\}$ for $i < n$ and $\psi(e_n) = \psi(e_{2n}) = \{1, n\}$. Furthermore, we define a complete transition system $\mathcal{S}$ on H by adding transitions between all parallel edges of H, i.e.

$$\mathcal{S}(i) := \{(i, \{e_i, e_{i+n}\}), (i, \{e_{i-1}, e_{i-1+n}\})\} \quad \forall i > 1$$

and

$$\mathcal{S}(1) = \{(1, \{e_1, e_{n+1}\}), (1, \{e_n, e_{2n}\})\}.$$

Additionally, we define a transitioned graph $(G_2, \mathcal{T})$ by duplicating all edges in G and adding transitions between them, i.e.

$$\begin{aligned} V(G_2) &= V(G), \\ E(G_2) &= \{e, e' \mid e \in E(G)\}, \\ \mathcal{T}(v) &= \{(v, \{e, e'\}) \mid e \in E(v)\} \quad \forall v \in V(G_2). \end{aligned}$$

Together $(G_2, \mathcal{T})$ and $(H, \mathcal{S})$ form an instance of ESTM. What is left to prove is that $(G_2, \mathcal{T})$ has an eulerian transition minor that is a sup-$(H, \mathcal{S})$ graph if and only if G has a Hamiltonian cycle.

If $(G_2, \mathcal{T})$ has an eulerian transition minor $(H', \mathcal{S}')$ that is a sup-$(H, \mathcal{S})$ graph then we get by Lemma 5.2.4 that H is a minor of H' and by Lemma 5.2.5 that H is a minor of G_2. Since $V(H) = n = V(G_2)$ we get by Lemma 5.2.3 that H is a subgraph of G_2. This implies that G_2 has a Hamiltonian cycle and therefore G has one, too.

On the other hand, if G has a Hamiltonian cycle C, taking the edges in C and the duplicates of them we get a subgraph H' of G_2 that is isomorphic to H. By adding all transitions between two parallel edges in H' to a transition system $\mathcal{S}'$ on H' we get that $(H', \mathcal{S}')$ is an eulerian transition minor of G_2 and that the transition system $\mathcal{S}'$ corresponds to the transition system $\mathcal{S}$ on H. Therefore, $(H', \mathcal{S}')$ is trivially a sup-$(H, \mathcal{S})$ graph. □

Proof of Theorem 5.2.2. In Theorem 5.2.1, we already proved that ESTM is in $\mathcal{NP}$, which implies that it is also in $\mathcal{NP}$ if we restrict it to simple graphs. Similar to the proof of Theorem 5.2.1 we prove hardness by reducing the Hamiltonian cycle problem to ESTM. Let G be an instance of a Hamiltonian cycle problem. Instead of replacing edges in G by double edges as we did in the proof of Theorem 5.2.1, we replace this time each edge e by a subgraph A_e. Each such subgraph A_e contains a K_4 of which two vertices are connected to the one end vertex of e and the other two edges are connected to the other end vertex of e, see Figure 5.4. Let G' be the resulting graph after replacing all edges e in G by the respective subgraphs A_e. We call the vertices in G' that correspond to a vertex in G original vertices.

Furthermore, let H be a cycle of length $n = |V(G)|$ and let H' be the result after replacing all edges e in H by the subgraph A_e. We define the transition systems $\mathcal{T}$ and $\mathcal{S}$ on G' and H' by adding for each subgraph A_e the transitions as shown in Figure 5.4. The obtained graph H' is 4-regular and $(H', \mathcal{S})$ is completely transitioned. All in all we get that $((G', \mathcal{T}), (H', \mathcal{S}))$ is an instance of ESTM restricted to simple graphs.

We prove now that G has a Hamiltonian cycle if and only if there exists an eulerian transition minor of $(G', \mathcal{T})$ that is a sup-$(H', \mathcal{S})$ graph, i.e. that $((G', \mathcal{T}), (H', \mathcal{S}))$ is a positive instance of ESTM.

$\Rightarrow$: Assume G has a Hamiltonian cycle C. We construct a corresponding subgraph H'' in G' by adding for each edge $e \in E(C)$ the subgraph A_e to H''. Since the length of the cycle C is also n we get that H'' is isomorphic to H'. Furthermore, the transitions in H'' correspond to the transitions in H' and therefore H'' is trivially a sup-$(H', \mathcal{S})$ graph. On the other hand, due to its construction H'' together with all transitions in H'' is an eulerian transition minor of G'.

$\Leftarrow$: Let $(H'', \mathcal{S}')$ be an eulerian transition minor of $(G', \mathcal{T})$ and a sup-$(H', \mathcal{S})$ graph. Note that there exists a Hamiltonian path in each A_e, $e \in E(H')$ from v_1 to v_2 and therefore H' has a Hamiltonian cycle. The graph H' has $|V(H)| + 4|E(H)| = n + 4n = 5n$ vertices and therefore the Hamiltonian cycle is of length $5n$. By definition of a sup-$(H', \mathcal{S})$ graph the cycle in H' corresponds to a cycle in H'' with at least the same number of vertices. Therefore, H'' has a cycle of length at least $5n$. Since H'' is a minor of G', we get that also G' contains a cycle C' of length at least $5n$.

Let $(v_1, ..., v_N)$ be the vertex sequence of C' with $N \geq 5n$. By construction of G' five consecutive vertices in C' contain at least one original vertex. Thus, C' contains at least n original vertices, i.e., all original vertices. Furthermore, if v_i and v_j are original vertices and all vertices between them in C', i.e. v_k for $i < k < j$, are not original, then v_i and v_j must both be part of a common subgraph A_e, and therefore they are connected by e in G. Thus, if we remove all non-original vertices from the vertex sequence $(v_1, ..., v_N)$ we get a new vertex sequence in G of length n in which all consecutive vertices are connected. Therefore, G is a Hamiltonian graph.

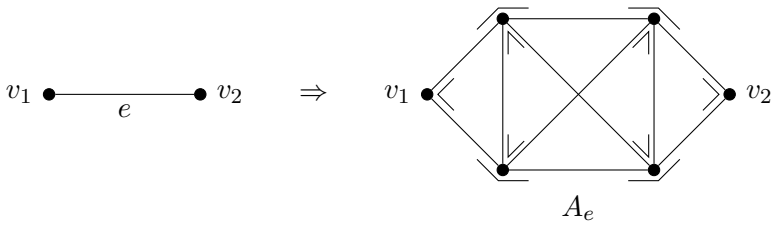


Figure 5.4: The graph A_e, which replaces an edge e in G and H. The transitions are represented by a vee ($\vee$) between their two edges.

□

5.2.2 Problem Transformation

Definition 5.2.8 (basic-sup-$(H, \mathcal{S})$)**.** Let $(H, \mathcal{S})$ be a completely transitioned 4-regular graph. A transitioned graph $(H', \mathcal{S}')$ is a *basic-sup-*$(H, \mathcal{S})$ graph if the following holds.

The graph H' can be decomposed into $|V(H)|$ many connected vertex-disjoint subgraphs

$$\{R_w \mid w \in V(H)\}$$

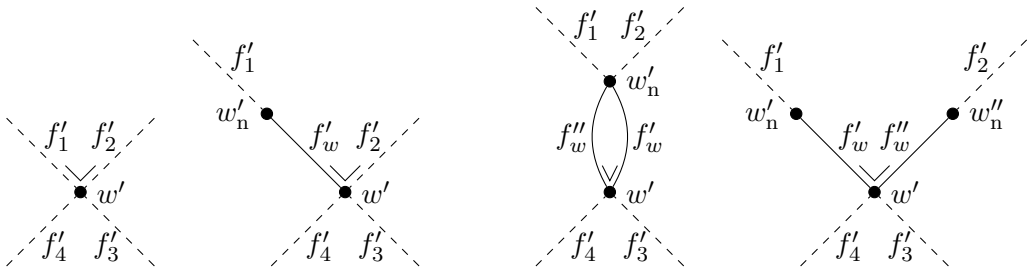
and $|E(H)|$ many edges $\{f' \mid f \in E(H)\}$. For each edge $f = w_1w_2 \in E(H)$ the edge f' connects the subgraphs R_{w_1} and R_{w_2}. For a vertex $w \in V(H)$ let R_w^+ be the subgraph of H' induced by $E(R_w)$ and the four adjacent edges f' for $f \in E(w)$.

There exists an ordering of the outgoing edges of w by

$$E(w) = \{f_1, f_2, f_3, f_4\}$$

such that the following holds.

- The two transitions in $\mathcal{S}(w)$ are $S_w^1 = (w, \{f_1, f_2\})$ and $S_w^2 = (w, \{f_3, f_4\})$.
- There exists a transition S'_w in $\mathcal{S}'$ such that the form of R_w and the transition S'_w satisfy one of the four possibilities (see Figure 5.5):
  1. R_w is only one vertex w' and $S'_w = (w', \{f'_1, f'_2\}) \in \mathcal{S}'(w')$
  2. R_w is a K_2 with two vertices w' and w'_{n}, where w'_{n} is of degree two with two incident edges f'_1 and f'_w. Moreover, $S'_w = (w', \{f'_w, f'_2\}) \in \mathcal{S}'(w')$.
  3. R_w is a cycle of length two, i.e. two vertices w' and w'_{n} and two parallel edges f'_w and f''_w connecting them. Furthermore, w'_{n} has degree four and is incident to f'_1 and f'_2 and $S'_w = (w', \{f'_w, f''_w\}) \in \mathcal{S}'(w')$.



case 1 case 2 case 3 case 4

Figure 5.5: The four possibilities for R_w and R_w^+ including the transition S'_w. The solid edges represent the edges of R_w and the dashed edges the four additional edges of R'_w. The transition is represented by a vee ($\vee$) between its two edges.

4. R_w consists of a vertex w' and two vertices w'_n and w''_n and two edges f'_w and f''_w connecting w' with w'_n and w' with w''_n. Furthermore, w'_n is incident to f'_1, w''_n is incident to f'_2, w' is incident to f'_3 and f'_4, and $S'_w = (w', \{f'_w, f''_w\}) \in \mathcal{S}'(w')$.

Note that we did not specify the order of the edges f_1, f_2, f_3, and f_4 except that there must be transitions $(w, \{f_1, f_2\})$ and $(w, \{f_3, f_4\})$ in $\mathcal{S}(w)$. Therefore, we could always interchange f_1 and f_2 with f_3 and f_4 in the second condition. The condition only has to hold for one of the two possibilities.

Definition 5.2.9 (basic-sup-$(H, \mathcal{S})$-reduced-minor-free)**.** Let $(H, \mathcal{S})$ be a completely transitioned 4-regular graph. A transitioned graph $(G, \mathcal{T})$ is *basic-sup-$(H, \mathcal{S})$-reduced-minor-free* if and only if it does not have any reduced transition minor $(H', \mathcal{S}')$ that is a basic-sup-$(H, \mathcal{S})$ graph.

Problem 5.2.2 (*existence of basic sup-reduced-transition minors (EBSRTM)*)**.** Given a transitioned graph $(G, \mathcal{T})$ and a completely transitioned 4-regular graph $(H, \mathcal{S})$, does there exist a reduced transition minor of $(G, \mathcal{T})$ that is a basic-sup-$(H, \mathcal{S})$ graph?

Theorem 5.2.6. *EBSRTM is equivalent to ESTM.*

Proof. Let $((G, \mathcal{T}), (H, \mathcal{S}))$ be a positive instance of EBSRTM, i.e. there exists a reduced transition minor $(H', \mathcal{S}')$ of $(G, \mathcal{T})$ that is a basic-sup-$(H, \mathcal{S})$ graph. We have to prove that this instance is also a positive instance of ESTM. First of all the reduced transition minor can be extended to a transition minor $(H', \mathcal{S}'')$ of $(G, \mathcal{T})$ by adding opposite transitions for degree four vertices. For every vertex $w \in V(H)$ there exists a subgraph R_w of H' that is of one of the four forms described in Definition 5.2.8 and shown in Figure 5.5. In all four cases every vertex in R_w^+ has degree two or four. Since every vertex in H' occurs in one R_w, we get that H' is eulerian.

It remains to show that $(H', \mathcal{S}'')$ is a sup-$(H, \mathcal{S})$ graph. For each edge $f \in E(H)$ we use the graph induced by the single edge $f' \in E(H')$ as path P_f. For each vertex $w \in V(H)$ we use the subgraph R_w as Q_w. All defined subgraphs are connected and edge-disjoint since the graphs R_w are vertex-disjoint and the edges f' are all different and always connect two different subgraphs R_{w_1} and R_{w_2}.

By definition of f' the path P_f only consists of f' and it connects the subgraphs $R_{w_1} = Q_{w_1}$ and $R_{w_2} = Q_{w_2}$ if $f = w_1w_2$. The paths P_f are also internally disjoint since the edges f' are all different. Furthermore, by definition the graphs $Q_w = R_w$ are vertex-disjoint connected subgraphs. The subgraphs $Q_w^+ = R_w^+$ have in all four cases a bad 1-separator $\{u_w := w'\}$ separating H_w^1 and H_w^2 where H_w^2 is the graph induced by $\{f_3', f_4'\}$ (in Figure 5.5 the lower part of the graphs below the vertex w') and H_w^1 is the rest of R_w^+, i.e. the graph induced by $E(R_w^+) \setminus \{f_3', f_4'\}$. It is a bad 1-separator since by definition we get in all four cases

$$(w', E(w') \cap E(H_w^1)) \in \mathcal{S}''(w').$$

Furthermore, we have $\{f_1', f_2'\} \subseteq H_w^1$ and $\{f_3', f_4'\} \subseteq H_w^2$, which concludes the proof that $(H', \mathcal{S}'')$ is a sup-$(H, \mathcal{S})$ graph. Therefore, the instance $((G, \mathcal{T}), (H, \mathcal{S}))$ is also a positive instance of ESTM.

Now, let $((G, \mathcal{T}), (H, \mathcal{S}))$ be a positive instance of ESTM, we want to show that it is also a positive instance of EBSRTM. Let $(H', \mathcal{S}')$ be an eulerian transition minor of $(G, \mathcal{T})$ that is a sup-$(H, \mathcal{S})$ graph. By removing the transitions in $\mathcal{S}'$ that do not correspond to a transition in $\mathcal{T}$, see (5.3), we get a reduced transition minor $(H', \mathcal{S}'')$ of $(G, \mathcal{T})$.

For each vertex $w \in V(H)$ the subgraph Q_w^+ of H' has a bad 1-separator u_w separating H_w^1 and H_w^2. Without loss of generality let $S_w^1 := (u_w, E(u_w) \cap E(H_w^1)) \in \mathcal{S}'(u_w)$. If $S_w^1 \notin \mathcal{S}''(u_w)$ we know that $\deg(u_w) = 4$ and that $S_w^2 := (u_w, E(u_w) \setminus S_w^1 = E(u_w) \cap E(H_w^2)) \in \mathcal{S}''(u_w)$. In this case we exchange S_w^1 and H_w^1 with S_w^2 and H_w^2 so that we always have

$$S_w^1 := (u_w, E(u_w) \cap E(H_w^1)) \in \mathcal{S}''(u_w).$$

Furthermore, we can assume without loss of generality that the paths P_f only consist of single edges, since otherwise we could move some edges from P_f to one of the subgraphs Q_w until P_f only consists of one edge. Let $Q_w^1 := Q_w \cap H_w^1$ and $Q_w^2 := Q_w \cap H_w^2$. Now we contract all edges in Q_w^2 such that only the vertex u_w remains. Furthermore, we contract all edges in $E(Q_w^1) \setminus S_w^1$, i.e. all edges in Q_w^1 that are not in the transition S_w^1. Starting from Q_w and applying those contractions we call the resulting graph R_w. Applying these contractions to all graphs Q_w for $w \in E(H)$ we call the whole resulting graph H'', which is a minor of H'. Let $\mathcal{S}'''$ be the corresponding transition system of H'' such that $(H'', \mathcal{S}''')$ is a reduced transition minor of $(H', \mathcal{S}'')$. We still have $S_w^1 \in \mathcal{S}'''$ since we did not contract any of the edges incident to u_w. Let R_w^+ be the graph R_w together with the four adjacent paths P_f, each consisting of only one edge, for $f \in E(w)$. To see that R_w^+ has one of the four forms shown in Figure 5.5, we only have to distinguish different cases for the degree of the vertex u_w in Q_w^1:

1. If u_w has degree 0 in Q_w^1, this would imply that Q_w only consists of the vertex u_w and therefore R_w only consists of the vertex u_w, which results in case 1.
2. If u_w has degree 1 in Q_w^1, this would imply that R_w is a K_2, which results in case 2.
3. If u_w has degree 2 in Q_w^1, there are two possibilities. If u_w is not a cut vertex of Q_w^1, then R_w is a digon (two vertices and two parallel edges), which results in case 3. Otherwise, if u_w is a cut vertex of Q_w^1 then R_w is a path of length two, which results in case 4.

Note that the degree of u_w in Q_w^1 is at most two, since $\pi_2(S_w^1) = E(u_w) \cap E(H_w^1)$ and $|\pi_2(S_w^1)| = 2$. In each of the four cases the transition S' is exactly the transition drawn in the figure. Therefore, we just proved that $(H'', \mathcal{S}''')$ is a basic-sup-$(H, \mathcal{S})$ graph. Since $(H', \mathcal{S}'')$ is a reduced transition minor of $(G, \mathcal{T})$ and $(H'', \mathcal{S}''')$ is a reduced transition minor of $(H', \mathcal{S}'')$, we get by transitivity that $(H'', \mathcal{S}''')$ is a reduced transition minor of $(G, \mathcal{T})$. All in all we proved that the given instance $((G, \mathcal{T}), (H, \mathcal{S}))$ is also a positive instance of EBSRTM. □

With that result we can prove that EBSRTM and therefore also ESTM can be solved in polynomial time if $k = |V(H)|$ is fixed.

Theorem 5.2.7. *ESTM can be solved in polynomial time if $k = |V(H)|$ is fixed.*

Proof. By Theorem 5.2.6 it is enough to prove this for EBSRTM. The proof is similar to the one in [67].

Notice that every basic-sup-$(H, \mathcal{S})$ graph $(H', \mathcal{S}')$ satisfies $|V(H')| \leq 3|V(H)|$. This implies that for a fixed $k = |V(H)|$ there are only finite many possible graphs $(H', \mathcal{S}')$ that can be a basic-sup-$(H, \mathcal{S})$ graph for any graph $(H, \mathcal{S})$ with $|V(H)| = k$. We can formulate now an algorithm that first generates all possible graphs $(H', \mathcal{S}')$ and then checks for each of them if it is a basic-sup-$(H, \mathcal{S})$ graph and if it is a reduced transition minor of $(G, \mathcal{T})$. Since the size of the graphs H' and H are bounded, checking if $(H', \mathcal{S}')$ is a basic-sup-$(H, \mathcal{S})$ graph can be done in constant time. What is left is to prove that checking if $(H', \mathcal{S}')$ is a reduced transition minor of $(G, \mathcal{T})$ can be done in polynomial time in $m = |V(G)| + |E(G)|$ if $k' = |V(H')| \leq 3k$ is fixed. By definition $(H', \mathcal{S}')$ is a reduced transition minor of $(G, \mathcal{T})$ if and only if there exist partial mappings φ and κ as described in Definition 5.2.4 such that $\mathcal{S}'$ equals the set defined in (5.2).

Since κ must be a partial injective and surjective function and since each vertex of H' has at most degree four, there are at most

$$|E(G)|^{|E(H')|} \leq m^{2k'} \leq m^{6k}$$

many possibilities to define κ by choosing exactly one edge in G for each edge in H'. Since k is fixed this number is polynomial in m and therefore we can generate all those possibilities and check for each of them if there exists an adequate partial mapping φ

satisfying all needed properties. Therefore, let κ be a fixed partial injective and surjective function from $E(G)$ to $E(H')$ for the rest of this proof.

Let

$$V_0 := \bigcup_{e \in \operatorname{dom}(\kappa)} \psi_G(e) \subseteq V(G)$$

be the set of all vertices in G that are incident to a mapped edge of κ. Since κ is injective and surjective, we have $|\operatorname{dom}(\kappa)| = |E(H')| \leq 2k'$ and therefore $|V_0| \leq 2|\operatorname{dom}(\kappa)| \leq 4k'$. Since φ must satisfy (5.1), φ must be defined on each vertex in V_0 and only two values for $\varphi(v)$ are possible for each $v \in V_0$. Thus, the number of possible definitions of φ on V_0 is smaller than or equal to

$$2^{|V_0|} \leq 2^{4k'} \leq 2^{12k}$$

and is therefore constant in m. We can again generate all those possibilities and check for each of them if there exists an adequate extension of φ to the whole vertex set V. Let therefore be φ already fixed and defined on V_0.

By using κ and the partially defined φ on V_0 the transition set defined in (5.2) is already well-defined, and we can compute it and check if it equals the transition set $\mathcal{S}'$. If the two transition sets do not match we can discard this possibility. In the case when the two transition sets are equal we only have to check if we can extend φ to V in such a way that φ is a partial surjective function and that each preimage $\varphi^{-1}[w]$ is a connected vertex set in G for each $w \in V(H')$. Since φ as defined already on V_0 satisfies (5.1) we already know that it is surjective without extending it at all. Therefore, we only have to extend it in such a way that each preimage $\varphi^{-1}[w]$ is a connected vertex set in G for each $w \in V(H')$. The existence of such an extension is equivalent to the existence of disjoint spanning trees T_w in G for each $w \in V(H')$, such that the already defined sets $\varphi^{-1}[w] \cap V_0 \subseteq T_w$ for each $w \in V(H')$. Without loss of generality we can search for spanning trees T_w such that all leaf vertices of T_w are in V_0.

Let $w \in V(H')$ be an arbitrary vertex. For each incident edge f in $E(w)$ we know by (5.2) that exactly one vertex in G incident to $\kappa^{-1}(f)$ is mapped to w under the already defined part of φ on V_0. Therefore, $|\varphi^{-1}[w] \cap V_0| \leq \deg_{H'}(w) \leq 4$ for each $w \in V(H')$. Putting everything together we get that T_w has at most four leaf vertices for each $w \in V(H')$. Since T_w is a tree it holds that $|E(T_w)| = V(T_w) - 1$, which implies that T_w has at most two vertices that have degree three or larger in T_w. These vertices could be in $V \setminus V_0$. The number of possible selections of such vertices of degree three or larger in T_w outside of V_0 for all $w \in V(H')$ is bounded by $|V(G)|^{2(|V(H')|+1)} \leq m^{2k+2}$ and is therefore polynomial in m. We can again iterate through all such possible selections of vertices of degree three or larger in T_w and check for each of them if we can construct the trees T_w such that they satisfy all needed properties. Let $V_w \subseteq V \setminus V_0$ be the fixed selected set of vertices of degree three or larger in T_w outside of V_0 for each $w \in V(H')$.

We construct now all possible labeled simple trees with the vertices $V_w \cup (\varphi^{-1}[w] \cap V_0)$. Since those are at most six vertices, the number of such labeled trees is finite. We can again iterate through all combinations of labeled trees L_w for all $w \in V(H')$ and check if

the following holds for at least one of them. For each $w \in V(H')$ and its current labeled tree L_w we check if we can replace the edges of L_w by paths in G to get a tree T_w in G. Since the trees T_w in G must be vertex disjoint for each $w \in V(H')$ all those paths in G replacing the edges of the labeled trees must be vertex disjoint. Therefore, what remains to check is if there exist vertex disjoint paths in G connecting all vertex pairs $\psi_{L_w}(e)$ for each edge $e \in L_w$ and each $w \in V(H')$. Note that the number of such pairs can be bounded by

$$\sum_{w \in V(H')} |E(L_w)| = \sum_{w \in V(H')} |V(L_w)| - 1 \leq \sum_{w \in V(H')} 5 = 5|V(H')| = 5k' \leq 15k,$$

which is assumed to be constant. This problem can be solved for a bounded number of pairs in polynomial time in the size of G and therefore in m, as proven in [67]. This concludes our proof. □

5.3 Modeling

In this chapter we present a mathematical model for solving the problem EBSRTM that can be expressed as a MILP, cf. Section 5.4, or as a SAT model, cf. Section 5.5. First we present the model, and then we prove that the model really solves EBSRTM, since this is not obvious.

A naive model would need variables for representing the unknown graph $(H', \mathcal{S}')$ and constraints to ensure that $(H', \mathcal{S}')$ is a reduced transition minor of the given $(G, \mathcal{T})$ and that $(H', \mathcal{S}')$ is a basic-sup-$(H, \mathcal{S})$. Such a naive model would be quite large since we do not even know the size of $(H', \mathcal{S}')$. Furthermore, formulating the constraints would be nontrivial and likely need a lot of additional auxiliary variables. The model we present does not describe the graph $(H', \mathcal{S}')$ directly. It just consists of mappings between $(G, \mathcal{T})$ and $(H, \mathcal{S})$ and constraints to ensure that a valid intermediate graph $(H', \mathcal{S}')$ exists.

5.3.1 Towards a Model

Let $(G, \mathcal{T})$ and $(H, \mathcal{S})$ be given as the input. Let us assume now that there exists a reduced transition minor $(H', \mathcal{S}')$ of $(G, \mathcal{T})$ that is a basic-sup-$(H, \mathcal{S})$ graph, i.e. that the given instance is a yes instance of EBSRTM. This situation and the notation that we introduce in the following is visualized in Figure 5.6.

By the definition of a minor there exist partial mappings $\varphi' \colon V(G) \twoheadrightarrow V(H')$ and $\kappa' \colon E(G) \twoheadrightarrow E(H')$ where φ' is surjective and κ' is injective and surjective. By the definition of a basic-sup-$(H, \mathcal{S})$ graph we can define the following natural partial mappings. The vertex mapping $\varphi'' \colon V(H') \twoheadrightarrow V(H)$ maps each vertex in the subgraph R_w to w for each $w \in V(H)$. Since each R_w is non-empty, φ'' is surjective. The edge mapping $\kappa'' \colon E(H') \twoheadrightarrow E(H)$ maps each edge f' to its corresponding edge $f \in E(H)$ for each $f \in E(H)$ as defined in Definition 5.2.8. By definition this partial mapping is injective and surjective.

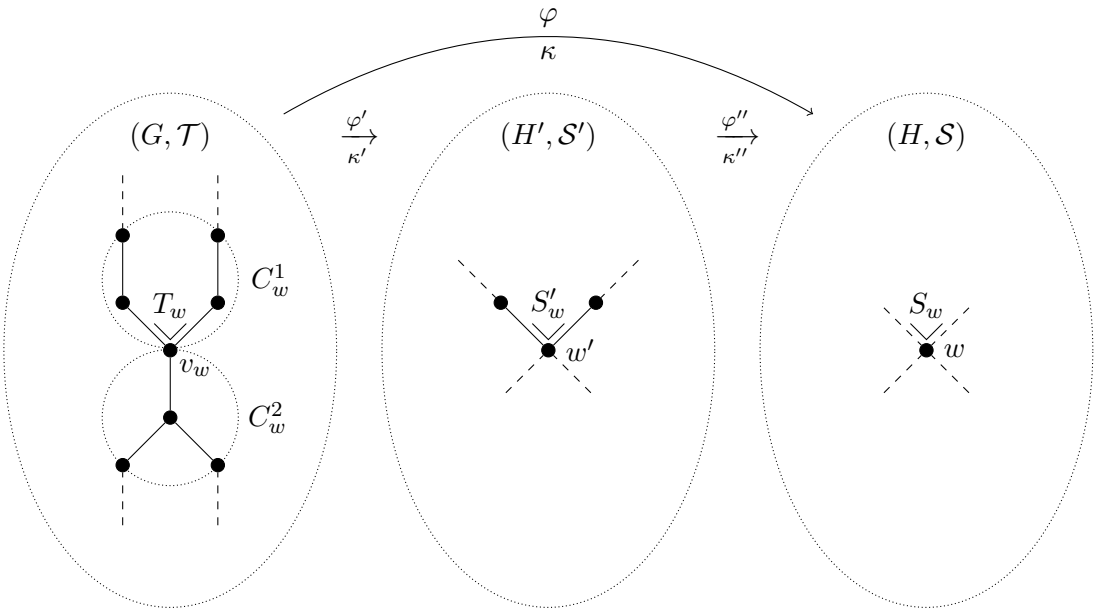


Figure 5.6: Exemplary visualization of the two input graphs together with the intermediate graph H'.

Using the above mappings we can define partial mappings from G to H by composing them. We define the two functions

$$\varphi := \varphi'' \circ \varphi' \colon V(G) \nrightarrow V(H),$$

and

$$\kappa := \kappa'' \circ \kappa' \colon E(G) \nrightarrow E(H).$$

Since φ' and φ'' are surjective also φ is surjective and since κ' and κ'' are injective and surjective also κ is injective and surjective.

We denote for each $w \in V(H)$ the transition $S_w := S_w^1 \in \mathcal{S}$ as defined in Definition 5.2.8. Each such transition S_w is associated via Definition 5.2.8 with a transition S'_w in $\mathcal{S}'$. Furthermore, by this definition we have $\pi_1(S'_w) =: w' \in \varphi''^{-1}[w]$. Moreover, since S'_w is in $\mathcal{S}'$ we get by definition of a reduced transition minor that

$$T_w := (v_w, \kappa'^{-1}[\pi_2(S'_w)]) \in \mathcal{T}$$

for some $v_w \in \varphi'^{-1}[w'] \subseteq \varphi^{-1}[w]$.

To model the fact that each vertex set $\varphi'^{-1}[x]$ must be connected in G for each $x \in V(H')$ and to model the four different possible structures of each R_w, we further introduce for each $w \in V(H)$ two trees C_w^1 and C_w^2. The two trees cover the connected subgraph $\varphi^{-1}[w]$ and share exactly one common vertex $v_w = \pi_1(T_w)$.

The tree C_w^2 represents a spanning tree of $\varphi'^{-1}[w']$ in G, where w' is as specified in Definition 5.2.8 for each $w \in V(H)$. The tree C_w^1 represents a spanning tree of $\{v_w\}$, $\{v_w\} \cup \varphi'^{-1}[w'_{\mathrm{n}}]$, or $\{v_w\} \cup \varphi'^{-1}[\{w'_{\mathrm{n}}, w''_{\mathrm{n}}\}]$ depending on the possible four cases for the form of R_w. Since the two trees share one vertex this ensures that the vertex set $\varphi^{-1}[w] = V(C_w^1) \cup V(C_w^2)$ is connected in G. We have to distinguish the two trees to formulate the constraint that the edges of T_w must be part of $C_w^1 \cup \kappa^{-1}[\pi_2(S_w)]$. Since we are only concerned with connectivity for the trees C_w^1 and C_w^2 we define them not as subtrees of G but as simple trees with vertices $V(C_w^i) \subseteq V(G)$ and for each edge in $E(C_w^i)$, which is represented as a set of two vertices, there must exist at least one edge in G whose end vertices are exactly those two vertices.

In case 3 of Definition 5.2.8 it may be that $\psi_G(\kappa'^{-1}(f'_w)) \neq \psi_G(\kappa'^{-1}(f''_w))$. In this case only one of those two edges is included in the tree C_w^1. Therefore, for this case we have to reformulate the condition that the edges of T_w must be part of $C_w^1 \cup \kappa^{-1}[\pi_2(S_w)]$. To do this we introduce a new partial injective mapping $\theta\colon E(G) \nrightarrow V(H)$ to our model, which only maps an edge to the vertex w if R_w is of the form of case 3 of Definition 5.2.8 and $\psi_G(\kappa'^{-1}(f'_w)) \neq \psi_G(\kappa'^{-1}(f''_w))$. With that mapping we can reformulate the condition such that the edges of T_w must be part of $C_w^1 \cup \kappa^{-1}[\pi_2(S_w)] \cup \theta^{-1}[w]$.

With the above motivation we are ready to formulate our model.

5.3.2 The Model

Let $(G, \mathcal{T})$ and $(H, \mathcal{S})$ be given as the input. We use the following additional notation in the model. If C_w^i is a simple tree with vertices in G we define

$$E_w^i := \left\{ e \in E(G) \mid \psi(e) \in E(C_w^i) \right\}$$

as the set of all edges in G whose end vertices are connected in C_w^i by an edge. The model is now defined as finding

1. a partial surjective function $\varphi\colon V(G) \nrightarrow V(H)$,
2. a partial injective and surjective function $\kappa\colon E(G) \nrightarrow E(H)$,
3. a partial injective function $\theta\colon E(G) \nrightarrow V(H)$,
4. for each $w \in V(H)$ a pair (T_w, S_w) of transitions with $T_w \in \mathcal{T}$ and $S_w \in \mathcal{S}(w)$,
5. for each $w \in V(H)$ two simple trees C_w^1 and C_w^2 with $V(C_w^i) \subseteq V(G)$ for $i = 1, 2$,

such that

$$E(C_w^i) \subseteq \psi_G[E(G)] \qquad \forall w \in V(H), \forall i \in \{1,2\} \tag{5.4}$$

$$\kappa(e) = f \Rightarrow \varphi[\psi_G(e)] = \psi_H(f) \qquad \forall e \in E(G), \forall f \in E(H) \tag{5.5}$$

$$V(C_w^1) \cup V(C_w^2) = \varphi^{-1}[w] \qquad \forall w \in V(H) \tag{5.6}$$

$$\{\pi_1(T_w)\} = V(C_w^1) \cap V(C_w^2) \qquad \forall w \in V(H) \qquad (5.7)$$

$$\pi_2(T_w) \subseteq \kappa^{-1}[\pi_2(S_w)] \cup \theta^{-1}[w] \cup E_w^1 \qquad \forall w \in V(H) \qquad (5.8)$$

$$\left(\kappa^{-1}[\pi_2(S_w)] \cap E(\pi_1(T_w))\right) \cup \theta^{-1}[w] \subseteq \pi_2(T_w) \qquad \forall w \in V(H) \qquad (5.9)$$

$$e \in \mathrm{dom}(\kappa) \wedge \kappa(e) \in \pi_2(S_w) \Rightarrow \psi_G(e) \cap V(C_w^1) \neq \emptyset \qquad \forall w \in V(H), \forall e \in E(G) \qquad (5.10)$$

$$e \in \mathrm{dom}(\kappa) \wedge \kappa(e) \in E(w) \setminus \pi_2(S_w) \Rightarrow \psi_G(e) \cap V(C_w^2) \neq \emptyset \qquad \forall w \in V(H), \forall e \in E(G) \qquad (5.11)$$

$$v \in V(C_w^1) \setminus \{\pi_1(T_w)\} \wedge \deg_{C_w^1}(v) = 1 \wedge v \notin \bigcup \psi_G[\theta^{-1}[w]] \Rightarrow E(v) \cap \kappa^{-1}[\pi_2(S_w)] \neq \emptyset \qquad \forall w \in V(H), \forall v \in V(G) \qquad (5.12)$$

$$E_{C_w^1}(\pi_1(T_w)) \subseteq \psi_G[\pi_2(T_w)] \qquad \forall w \in V(H) \qquad (5.13)$$

$$\theta(e) = w \Rightarrow \psi_G(e) \subseteq V(C_w^1) \qquad \forall e \in E(G), \forall w \in V(H) \qquad (5.14)$$

$$\theta(e) = w \Rightarrow \psi_G(e) \notin E(C_w^1) \qquad \forall e \in E(G), \forall w \in V(H) \qquad (5.15)$$

Constraints (5.4) ensure that the trees C_w^i are simple trees where each simple edge corresponds to at least one edge of G. Constraints (5.5) couple the edge map κ with the vertex map φ, similar to condition (5.1). Furthermore, conditions (5.6) guarantee that the two trees C_w^1 and C_w^2 together really cover $\varphi^{-1}[w]$ and together with constraints (5.7) this ensures that $\varphi^{-1}[w]$ is connected in G. Constraints (5.7) furthermore enforce that $\pi_1(T_w)$ gets mapped to w under φ. Constraints (5.8) ensure that the edges of a transition T_w get either mapped directly to edges of S_w (see f_1', f_2' in case 1 and f_2' in case 2 of Definition 5.2.8), or correspond to edges of C_w^1 (see f_w' in case 2 or f_w', f_w'' in case 3 or 4), or get mapped to w by θ (f_w'' in case 3 of Definition 5.2.8 if $\psi_G(\kappa'^{-1}(f_w')) \neq \psi_G(\kappa'^{-1}(f_w''))$). On the other hand, constraints (5.9) guarantee that the only edges that are incident to $\pi_1(T_w)$ and get mapped under κ to an edge in $\pi_2(S_w)$ are in $\pi_2(T_w)$. Furthermore, those constraints ensure that only edges in $\pi_2(T_w)$ may be mapped to w under θ.

An edge in G that gets mapped under κ to $\pi_2(S_w)$ must be incident to a vertex in C_w^1, which is ensured by conditions (5.10). This forces edges like f_1' or f_2' in Definition 5.2.8 to be incident to a vertex of the upper tree. On the other hand, conditions (5.11) guarantees that edges that get mapped under κ to $E(w) \setminus \pi_2(S_w)$ are incident to a vertex of C_w^2. This forces edges like f_3' and f_4' in Definition 5.2.8 to be incident to a vertex of the lower tree.

Constraints (5.12) are the most complex ones of the model. They are needed to avoid situations like one similar to case 4 of Definition 5.2.8 where f_2' is not incident to w_n'' but to w_n'. In this case a leaf vertex of the upper tree would have no incident edge in $\mathrm{dom}(\kappa)$. On the other hand, it may happen in case 3 if $\psi_G(\kappa'^{-1}(f_w')) \neq \psi_G(\kappa'^{-1}(f_w''))$ that one leaf vertex of the upper tree also has no incident edge but is still representing a valid case. But then we can ensure that this leaf vertex is incident to the edge that gets mapped to w by θ. Therefore, to exclude the unwanted situation described above, which is similar to case 4, but to include the wanted cases in case 3, we need constraints (5.12).

Constraints (5.13) guarantee that all edges in the upper tree that are incident to $\pi_1(T_w)$

are represented in G by edges in $\pi_2(T_w)$. Furthermore, constraints (5.14) enforce that if an edge e gets mapped by θ to a vertex w that both incident vertices of e are in the upper tree C_w^1 and (5.15) ensures that the edge e itself is not represented by an edge in C_w^1.

Theorem 5.3.1. *For the above model exists a valid solution if and only if* $((G,\mathcal{T}),(H,\mathcal{S}))$ *is a yes instance of EBSRTM.*

We split this theorem into two parts, proofing each direction independently. Theorem 5.3.1 then directly follows Proposition 5.3.2 and Proposition 5.3.4.

Proposition 5.3.2. *If* $((G,\mathcal{T}),(H,\mathcal{S}))$ *is a yes instance of EBSRTM there exists a valid solution in the above model.*

Proof. Let $((G,\mathcal{T}),(H,\mathcal{S}))$ be a yes instance of EBSRTM and $(H',\mathcal{S}')$ a reduced transition minor of $(G,\mathcal{T})$ that is a basic-sup-$(H,\mathcal{S})$ graph. In the first part of the proof we derive the partial mappings φ,κ,θ, the transitions S_w and T_w for $w \in V(H)$, and the trees C_w^i for $w \in V(H), i \in \{1,2\}$. In the second part we prove that the constraints (5.4)–(5.15) are satisfied. Let φ' and κ' be the correspondence functions of the minor H' of G, see the second part of Definition 5.2.4. W.l.o.g. we assume that $\mathrm{dom}(\varphi')$ is minimal, i.e. after removing any vertex from $\mathrm{dom}(\varphi')$ the conditions of Definition 5.2.4 are not satisfied anymore. We need the following property for the minimal φ'.

Lemma 5.3.3. *Every vertex* $v \in \mathrm{dom}(\varphi')$ *is either incident to an edge* $e \in \mathrm{dom}(\kappa')$ *or a cut vertex of* ${\varphi'}^{-1}[\varphi'(v)]$.

Proof. Assume there is a vertex $v \in \mathrm{dom}(\varphi')$ that is not incident to any edge in $\mathrm{dom}(\kappa')$ and is no cut vertex of ${\varphi'}^{-1}[\varphi'(v)]$. We prove now that removing the vertex v from φ' still leads to a valid vertex mapping for Definition 5.2.4, which then is a contradiction to the minimality of φ'. Let $\overline{\varphi'} = \varphi' \setminus \{(v,\varphi'(v))\}$ be the reduced mapping. We need to prove all conditions of Definition 5.2.4 for $\overline{\varphi'}$. First of all $\overline{\varphi'}^{-1}[\varphi'(v)]$ is non-empty since H' is four regular and therefore there exists an edge $f \in E(H')$ that is incident to $\varphi'(v)$. Then we have

$$\varphi'(v) \in \psi_H(f) = \varphi'[\psi_G({\kappa'}^{-1}(f))] = \overline{\varphi'}[\psi_G({\kappa'}^{-1}(f))],$$

which implies that $\overline{\varphi'}^{-1}[\varphi'(v)]$ is non-empty. Furthermore, $\overline{\varphi'}^{-1}[\varphi'(v)] = {\varphi'}^{-1}[\varphi'(v)] \setminus \{v\}$ is connected since v was not a cut vertex of ${\varphi'}^{-1}[\varphi'(v)]$. Last but not least (5.1) is also valid for $\overline{\varphi'}$ since v was not incident to any edges in $\mathrm{dom}(\kappa')$. □

Let the subgraphs R_w for $w \in V(H)$ and the edges f' of H' for $f \in E(H)$ be as described in Definition 5.2.8. We define a new surjective function $\varphi'' \colon V(H') \to V(H)$ and a partial injective and surjective function $\kappa'' \colon E(H') \nrightarrow E(H)$ by

$$\varphi''(x) = w \quad \forall x \in R_w, \forall w \in V(H)$$

and

$$\kappa''(f') = f \quad \forall f \in E(H).$$

Function φ'' is well-defined and surjective since the vertex sets of the non-empty subgraphs R_w partition the vertex set $V(H')$. The partial function κ'' is well-defined and injective since the edge f' is different for each $f \in E(H)$. This can be seen by the fact that $|\{f' \mid f \in E(H)\}| = |E(H)|$ in Definition 5.2.8. Furthermore, κ'' is surjective since there exists an edge f' for every $f \in E(H)$.

Now we can concatenate the given functions and get the surjective partial functions $\varphi\colon V(G) \twoheadrightarrow V(H)$ and $\kappa\colon E(G) \twoheadrightarrow E(H)$ by $\varphi = \varphi'' \circ \varphi'$ and $\kappa = \kappa'' \circ \kappa'$. Since both κ' and κ'' are injective κ is also injective. Moreover, since the four maps φ', φ'', κ', and κ'' are surjective the maps φ and κ are also surjective.

In the following we use the notation in Definition 5.2.8, which is possible since we know that $(H', \mathcal{S}')$ is a sup-$(H, \mathcal{S})$ graph. For each vertex $w \in V(H)$ let $S'_w \in \mathcal{S}'(w')$ be the transition at vertex w' defined in Definition 5.2.8. Since the transition S'_w is in the reduced transition minor $(H', \mathcal{S}')$ of $(G, \mathcal{T})$ there must exist a transition $T_w \in \mathcal{T}$ with $\kappa'[\pi_2(T_w)] = \pi_2(S'_w)$ and $\varphi'(\pi_1(T_w)) = \pi_1(S'_w) = w'$. Furthermore, for $w \in V(H)$, let $S_w := S^1_w = (w, \{f_1, f_2\})$ as specified in Definition 5.2.8. With this we defined for each vertex $w \in V(H)$ the pair (T_w, S_w). For a vertex $w \in V(H)$ we define the simple graph $G_w = (V_w, E_w)$ by $V_w := \varphi'^{-1}[V(R_w)]$ and

$$E_w := \left\{\psi_G(e) \mid e \in \kappa'^{-1}[E(R_w)] \vee \exists x \in V(R_w) : \psi_G(e) \subseteq \varphi'^{-1}[x]\right\}.$$

We know that $\varphi'^{-1}[x]$ is connected in G for every $x \in V(R_w)$, and for each edge $f^* \in E(R_w)$ with $\psi_{H'}(f^*) = \{x_1, x_2\}$ the edge $\kappa'^{-1}(f^*)$ connects $\varphi'^{-1}[x_1]$ with $\varphi'^{-1}[x_2]$. From that follows that G_w is connected. We can now define C^2_w to be a spanning tree of the vertex set $\varphi'^{-1}[w']$ in G_w.

To define C^1_w let V^1_w be the vertex set containing $v_w := \pi_1(T_w)$ and $\varphi'^{-1}[w'_{\mathrm{n}}]$ if the vertex w'_{n} exists together with $\varphi'^{-1}[w''_{\mathrm{n}}]$ if the vertex w''_{n} exists. The vertex set V^1_w is connected in G_w since $\varphi'^{-1}[w'_{\mathrm{n}}]$ is connected if existent, $\varphi'^{-1}[w''_{\mathrm{n}}]$ is connected if existent, the edge $\kappa'^{-1}(f'_w)$ connects v_w with $\varphi'^{-1}[w'_{\mathrm{n}}]$ if existent, and if $\varphi'^{-1}[w''_{\mathrm{n}}]$ exists it is connected to v_w by $\kappa'^{-1}(f''_w)$. We define now C^1_w to be a spanning tree of V^1_w in G_w. In case 3 of Definition 5.2.8 we require from C^1_w w.l.o.g. that v_w is a leaf vertex in C^1_w connected by $\psi_G(\kappa'^{-1}(f'_w))$. This is possible since $V^1_w \setminus \{v_w\} = \varphi'^{-1}[w'_{\mathrm{n}}]$ is still connected.

Furthermore, for a vertex $w \in V(H)$ in case 3 of Definition 5.2.8 we define $\theta(\kappa'^{-1}(f''_w)) = w$ if $\psi_G(\kappa'^{-1}(f''_w)) \neq \psi_G(\kappa'^{-1}(f'_w))$. If $\kappa'^{-1}(f''_w)$ is a parallel edge to $\kappa^{-1}(f'_w)$ and in the other three cases of Definition 5.2.8 no edge maps to w, i.e. $\theta^{-1}[w] = \emptyset$. By definition θ is injective since the edges $\kappa^{-1}(f''_w)$ are incident to two vertices in $\varphi'^{-1}[R_w]$ and the sets $\varphi'^{-1}[R_w]$ are disjoint for different vertices w. What remains is now to show that the constraints (5.4)–(5.15) hold, which we prove in the following.

(5.4) Constraints (5.4) are satisfied since C^1_w and C^2_w are subtrees of G_w and since $E(G_w) = E_w \subseteq \psi_G[E(G)]$ by definition.

(5.5) Let $e \in E(G)$ and $f \in E(H)$ with $\kappa(e) = f$, i.e. $\kappa''(\kappa'(e)) = f$. By definition of κ'' we get $\kappa'(e) = f'$ and by (5.1) we get $\psi'_H(f') = \varphi'[\psi_G(e)]$. Let $f = w_1w_2$ then we know by Definition 5.2.8 that f' connects the subgraphs R_{w_1} and R_{w_2} in H', i.e. $f' = x_1x_2$ with $x_1 \in V(R_{w_1})$ and $x_2 \in V(R_{w_2})$. By definition of φ'' we get $\varphi''(x_1) = w_1$ and $\varphi''(x_2) = w_2$ and plugging everything together we get

$$\varphi[\psi_G(e)] = \varphi''[\varphi'[\psi_G(e)]] = \varphi''[\psi'_H(f')] = \varphi''[\{x_1, x_2\}] = \{w_1, w_2\} = \psi_H(f).$$

(5.6) By the definition of C^1_w and C^2_w we have

$$V(C^1_w) \cup V(C^2_w) = V(G_w) = \varphi'^{-1}[V(R_w)] = \varphi'^{-1}[\varphi''^{-1}[w]] = \varphi^{-1}[w].$$

(5.7) Let $w \in V(H)$ then by definition of C^1_w and C^2_w we get (5.7) by the fact that $\varphi'^{-1}[w'_{\mathrm{n}}]$ and $\varphi'^{-1}[w''_{\mathrm{n}}]$ are disjoint to $\varphi'^{-1}[w']$ and because $\varphi'(\pi_1(T_w)) = w'$ by definition of T_w.

(5.8) Let $w \in V(H)$ and $e \in \pi_2(T_w)$. Then we know by the definition of T_w that $\kappa'(e) = f^* \in \pi_2(S'_w)$. In all four cases in the Definition 5.2.8 the edges of $\pi_2(S'_w)$ are either f'_i for some $f_i \in \pi_2(S_w)$, f'_w, or f''_w. If $f^* = f'_i$ for some $f_i \in \pi_2(S_w)$ we get $e = \kappa^{-1}(f_i) \in \kappa^{-1}[\pi_2(S_w)]$.

Next we show that $\psi_G(\kappa^{-1}(f'_w))$ is in $E(C^1_w)$ in the cases 2 to 4 of Definition 5.2.8. In case 2 the only edge between $\pi_1(T_w)$ and $\varphi'^{-1}[w'_{\mathrm{n}}]$ in C^1_w is $\psi_G(\kappa^{-1}(f'_w))$ and therefore it must be part of C^1_w since C^1_w is a spanning tree. In case 3 we directly forced $\psi_G(\kappa^{-1}(f'_w))$ to be a part of C^1_w. Furthermore, in case 4 both edges $\psi_G(\kappa^{-1}(f'_w))$ and $\psi_G(\kappa^{-1}(f''_w))$ must be part of C^1_w since otherwise the subgraphs $\varphi'^{-1}[w'_{\mathrm{n}}]$ and $\varphi'^{-1}[w''_{\mathrm{n}}]$ would not be connected. Therefore, if $f^* = f'_w$ we directly get $e \in E^1_w$. We already proved for the case 4 that $\psi_G(\kappa'^{-1}(f''_w)) \in E(C^1_w)$ and therefore also in case 4 if $f^* = f''_w$ we get $e \in E^1_w$.

The only remaining case is case 3 and if $f^* = f''_w$. Then either $\kappa'^{-1}(f''_w)$ is parallel to $\kappa'^{-1}(f'_w)$, which would imply again $\psi_G(\kappa'^{-1}(f''_w)) \in E(C^1_w)$ or $\theta(\kappa'^{-1}(f''_w)) = \theta(e) = w$ and therefore $e \in \theta^{-1}[w]$. All in all we just proved (5.8).

(5.9) Let $w \in V(H)$ and $e \in \kappa^{-1}[\pi_2(S_w)] \cap E(\pi_1(T_w))$. Let $f = \kappa(e) = \kappa''(\kappa'(e))$, then we know by definition of κ'' that $\kappa'(e) = f'$. Since e is incident to $\pi_1(T_w)$, we get by (5.1) that f' is incident to $\varphi'(\pi_1(T_w)) = w'$. We know now that $f \in \pi_2(S_w)$ and that f' is incident to w', which implies that $f' \in \pi_2(S'_w)$. This can easily be checked in all four cases of Definition 5.2.8. But since $\pi_2(S'_w) = \kappa'[\pi_2(T_w)]$ we get that $e \in \kappa'^{-1}[\pi_2(S'_w)] = \pi_2(T_w)$.

On the other hand, if $e \in \theta^{-1}[w]$ this means that we are in case 3 of Definition 5.2.8 and $e = \kappa'^{-1}(f''_w) \in \kappa'^{-1}[\pi_2(S'_w)] = \pi_2(T_w)$. All in all we proved (5.9).

(5.10) Let $w \in V(H)$, $e \in \mathrm{dom}(\kappa)$, and $f = \kappa(e) \in E(w)$. Then we know by definition of κ that $\kappa'(e) = f'$ and since f is incident to w we get that f' is incident to a vertex in R_w. If $f \in \pi_2(S_w)$, this implies by the notation of Definition 5.2.8 that $f = f_i$

with $i \in \{1,2\}$. In all four cases f'_i is either in $\pi_2(S'_w)$ or is incident to w'_n or w''_n for $i \in \{1,2\}$. If $f'_i \in \pi_2(S'_w)$ we get $e = \kappa'^{-1}(f'_i) \in \pi_2(T'_w)$, which implies that e is incident to $v_w \in V(C^1_w)$. On the other hand, if f'_i is incident to w'_n or w''_n, we also get that $e = \kappa'^{-1}(f'_i)$ is incident to a vertex in $\varphi'^{-1}[\{w'_\mathrm{n}, w''_\mathrm{n}\}] \subseteq V(C^1_w)$. With that we proved (5.10).

(5.11) With the same notation as for constraints (5.10) if $f \in E(w) \setminus \pi_2(S)$ we get $f' = f'_i$ with $i \in \{3,4\}$ and in all four cases of Definition 5.2.8 those edges are incident to w'. This implies that $e = \kappa'^{-1}(f')$ is incident to an edge in $\varphi'^{-1}[w'] = V(C^2_w)$ and therefore we showed (5.11).

(5.12) Let $w \in V(H)$, $v_w := \pi_1(T_w)$, and $v \in V(C^1_w) \setminus \{v_w\}$ with $|E_{C^1_w}(v)| = 1$ and $v \notin \bigcup \psi_G[\theta^{-1}[w]]$. Now we use Lemma 5.3.3 and obtain that $v \in V(C^1_w) \subseteq V_w = \varphi'^{-1}[V(R_w)]$ is either incident to an edge $e \in \mathrm{dom}(\kappa')$ or a cut vertex of $\varphi'^{-1}[\varphi'(v)]$. Since $v \in V(C^1_w) \setminus \{v_w\}$, we know $\varphi'(v) \neq w'$ and therefore that $\varphi'^{-1}[\varphi'(v)] \subseteq C^1_w$. If v would be a cut vertex of $\varphi'^{-1}[\varphi'(v)]$ this would imply that $\deg_{C^1_w}(v) \geq 2$, which is a contradiction to our assumption. Therefore, v is not a cut vertex of $\varphi'^{-1}[\varphi'(v)]$ and we get that v is incident to an edge $e \in \mathrm{dom}(\kappa')$. There are now two possibilities for $f' = \kappa'(e)$, either $f' \in \mathrm{dom}(\kappa'')$ or $f' \in \{f'_w, f''_w\}$. In the first case we know $f' \in \mathrm{dom}(\kappa'')$ and that f' is incident to $\varphi'(v) \subseteq \{w'_\mathrm{n}, w''_\mathrm{n}\}$. We can check in all four cases of Definition 5.2.8 that this implies either $f' = f'_1$ or $f' = f'_2$, which further implies $f \in \{f_1, f_2\} = \pi_2(S_w)$. In this case we are done since $e \in E(v) \cap \kappa^{-1}[\pi_2(S_w)] \neq \emptyset$.

On the other hand, the second case if $f' \in \{f'_w, f''_w\}$ implies $e \in \pi_2(T_w)$. Since $v \neq v_w$ we get that v is the other incident vertex of e, i.e. $e = vv_w$. Let $x = \varphi'(v)$, then $\psi_{H'}(f') = \varphi'[\psi_G(e)] = \{x, w'\}$. We know in the case $f' = f'_w$ that f'_1 or in the case that $f' = f''_w$ that f'_2 is incident to x; let f'_i denote this incident edge in both cases. From this we get that $e_i := \kappa'^{-1}(f'_i)$ is incident to a vertex in $\varphi'^{-1}[x]$. By the definition of $E(G_w) = E_w$ all edges in G_w that connect $\varphi'^{-1}[w']$ with $\varphi'^{-1}[x]$ are edges in $\psi_G[\kappa'^{-1}[E(R_w)]]$. Since $v_w \in \varphi'^{-1}[w']$ and $v \in \varphi'^{-1}[x]$ are vertices of the connected subtree C^1_w of G_w there must exist an edge $\tilde{e} \in \kappa'^{-1}[E(R_w)]$ that connects $\varphi'^{-1}[w']$ with $\varphi'^{-1}[x]$ such that $\psi_G(\tilde{e}) \in E(C^1_w)$. In all four cases of the Definition 5.2.8 the only two possibilities for $\tilde{e}$ are $\kappa'^{-1}(f'_w)$ and $\kappa'^{-1}(f''_w)$. If $\kappa'(\tilde{e}) = f'$ we get by the injectivity of κ' that $\tilde{e} = e$ and therefore $\psi_G(e) \in E(C^1_w)$, but then we know that there is no other edge in C^1_w that is incident to v, which implies $\varphi'^{-1}[x] = \{v\}$ and therefore that e_i is incident to v, which implies $e_i \in E(v) \cap \kappa^{-1}[\pi_2(S_w)]$.

The last case we need to check is if $\kappa'(\tilde{e}) \neq f'_i$. Since $\psi_{H'}(\kappa'(\tilde{e})) = \varphi'[\psi_G(\tilde{e})] = \{w', x\} = \psi_{H'}(f'_i)$ this can only happen in case 3 of Definition 5.2.8 and if $f' = f''_w$ and $\tilde{e} = \kappa'^{-1}[f'_w]$. But in this case we ensured that $w = \theta(\kappa'^{-1}(f''_w)) = \theta(\kappa'^{-1}(f')) = \theta(e)$, which implies $e \in \theta^{-1}[w]$, which is a contradiction since $v \in \psi_G(e) = \bigcup[\psi_G[\theta^{-1}[w]]]$. All in all we showed that (5.12) holds.

(5.13) Let $w \in V(H)$ and $\{v_w, v\} \in E_{C_w^1}(\pi_1(T_w))$. By definition of C_w^1 this implies that there exists an edge $e \in E(G)$ with $\psi_G(e) = \{v_w, v\}$. By definition of $E_w \supseteq E_{C_w^1}(v_w)$ either $e \in \kappa'^{-1}[E(R_w)]$ or $\psi_G(e) \subseteq \varphi'^{-1}[x]$ for some $x \in V(R_w)$. Since v_w is the only vertex of $\varphi'^{-1}[w']$ in C_w^1 we get that $\varphi'[v] \neq \varphi'[v_w]$ and therefore that $e \in \kappa'^{-1}[E(R_w)]$. Either $\kappa'(e) = f'_w$ or $\kappa'(e) = f''_w$ but in both cases we know that $e \in \pi_2(T_w)$. Therefore, we always get that $\{v_w, v\} = \psi_G(e) \in \psi_G[\pi_2(T_w)]$, which implies (5.13).

(5.14) Let $w \in V(H)$, and $e \in E(G)$ with $\theta(e) = w$. By definition of θ this implies that the structure of R_w equals the one of case 3 of Definition 5.2.8 and $e = \kappa'^{-1}(f''_w)$, which implies $e \in \pi_2(T_w)$. Since $\psi_{H'}(f''_w) = \{w', w''_{\mathrm{n}}\}$, we know that e is incident to a vertex in $\varphi^{-1}[w''_{\mathrm{n}}]$. Since e is incident to v_w and $\varphi^{-1}[w''_{\mathrm{n}}] \subseteq V(C_w^1)$ we get $\psi_G(e) \subseteq V(C_w^1)$, which finishes the proof for (5.14).

(5.15) These constraints follow directly from our definition of θ. We only define it in case 3 and only if $\psi_G(\kappa'^{-1}(f''_w)) \neq \psi_G(\kappa'^{-1}(f'_w))$. Since we also define C_w^1 in this case in such a way that v_w has degree 1 with the only incident edge $\psi_G(\kappa'^{-1}(f'_w))$, we know that $\psi_G(\kappa'^{-1}(f''_w))$, which is also incident to v_w, is not in $E(C_w^1)$.

□

Proposition 5.3.4. *If there exists a valid solution in the model given at the beginning of this section the instance $((G, \mathcal{T}), (H, \mathcal{S}))$ is a yes instance of EBSRTM.*

Proof. Let

$$(\varphi, \kappa, \theta, (T_w)_{w \in V(H)}, (S_w)_{w \in V(H)}, (C_w^1)_{w \in V(H)}, (C_w^2)_{w \in V(H)})$$

be a valid solution of the model. The proof consists of the following steps. We define a transitioned graph H', proof that H' is a minor of G, define a transition system $\mathcal{S}'$ on H' in such a way that $(H', \mathcal{S}')$ is a reduced transition minor of $(G, \mathcal{T})$ and finally prove that $(H', \mathcal{S}')$ is a basic-sup-$(H, \mathcal{S})$-graph.

Before we define the graph H', we introduce some notation that we use during the proof. For a vertex $w \in V(H)$ we define $v_w := \pi_1(T_w)$ and denote a possible extension of E_w^1 by

$$\overline{E_w^1} := E_w^1 \cup \theta^{-1}[w].$$

Lemma 5.3.5. *For an edge $e \in \overline{E_w^1}$ we get $\psi_G(e) \subseteq V(C_w^1)$.*

Proof. For $e \in E_w^1$ the lemma follows by definition of E_w^1 and for $e \in \theta^{-1}[w]$ it follows from (5.14). □

We describe now for each vertex $w \in V(H)$ a graph R_w. These graphs are then used to form the graph H'.

Definition 5.3.1 (R_w)**.** To describe R_w we distinguish the following four cases.

1. $\deg_{C_w^1}(v_w) = 0$: In this case R_w consists only of one vertex w' (see case 1 in Figure 5.5).
2. $\deg_{C_w^1}(v_w) = 1 \wedge |\kappa[\pi_2(T_w)]| = 1$: In this case R_w consists of two vertices w' and w'_{n} and an edge f'_w connecting them (see case 2 in Figure 5.5).
3. $\deg_{C_w^1}(v_w) = 1 \wedge |\kappa[\pi_2(T_w)]| = 0$: In this case R_w consists of two vertices w' and w'_{n} and two parallel edges f'_w and f''_w connecting them (see case 3 in Figure 5.5).
4. $\deg_{C_w^1}(v_w) = 2$: In this case R_w consists of three vertices w', w'_{n}, and w''_{n} and two edges f'_w connecting w' and w'_{n} and f''_w connecting w' and w''_{n} (see case 4 in Figure 5.5).

Note that $\deg_{C_w^1}(v_w) \leq 2$, which follows from (5.13) by

$$\deg_{C_w^1}(v_w) = |E_{C_w^1}(v_w)| \leq |\psi_G[\pi_2(T_w)]| \leq |\pi_2(T_w)| = 2.$$

To conclude that the above cases cover all possible cases we need to show that $\kappa[\pi_2(T_w)] \leq 1$ holds when $\deg_{C_w^1}(v_w) = 1$. In this case we know that there exists an edge $e \in E_w^1 \cap E(v_w)$, which implies by (5.13) that

$$\psi_G(e) \in E_{C_w^1}(v_w) \subseteq \psi_G[\pi_2(T_w)]$$

and therefore there must exist an edge $\tilde{e} \in \pi_2(T_w)$ with $\psi_G(\tilde{e}) = \psi_G(e) \subseteq V(C_w^1)$. Furthermore, this implies by (5.5), (5.6), and the fact that we have no loops in H that $\tilde{e} \notin \mathrm{dom}(\kappa)$. Since $|\pi_2(T_w)| = 2$, this gives us $\kappa[\pi_2(T_w)] \leq 1$.

We can now define the vertex set of the graph H' as the disjoint union of the vertex sets of all graphs R_w. Formally this means

$$V(H') := \biguplus_{w \in V(H)} V(R_w).$$

We define one more notation, which we use in the rest of the proof. In case 4 of Definition 5.3.1 we know that $\deg_{C_w^1} = 2$ and therefore the tree C_w^1 decomposes after removing the vertex v_w into two subtrees, which we denote by $C_w^{1,1}$ and $C_w^{1,2}$. Before we define edges between the subgraphs R_w we describe a partial vertex mapping $\varphi' \colon V(G) \nrightarrow V(H')$.

Definition 5.3.2 ($\varphi' \colon V(G) \nrightarrow V(H')$)**.** For a vertex $w \in V(H)$ we distinguish again the four cases (see Definition 5.3.1).

1. $\varphi'(v) := w' \quad \forall v \in V(C_w^2)$.

2. + 3. $\varphi'(v) := w' \quad \forall v \in V(C_w^2), \qquad \varphi'(v) := w'_{\mathrm{n}} \quad \forall v \in V(C_w^1) \setminus \{v_w\}.$

4. $\varphi'(v) := w' \quad \forall v \in V(C_w^2), \qquad \varphi'(v) := w'_{\mathrm{n}} \quad \forall v \in V(C_w^{1,1}),$
$\varphi'(v) := w''_{\mathrm{n}} \quad \forall v \in V(C_w^{1,2}).$

To prove that φ' is well-defined we need to show that no vertex maps to more than one vertex, that means that all occurring sets in the universal quantifiers are disjoint. For two vertices $w_1, w_2 \in V(H)$ with $w_1 \neq w_2$ and $i, j \in \{1, 2\}$ we get from (5.6) that

$$V(C_{w_1}^i) \cap V(C_{w_2}^j) \subseteq \varphi^{-1}[w_1] \cap \varphi^{-1}[w_2] = \emptyset.$$

Therefore, it remains to check that the universal quantifiers are disjoint within the cases of one vertex $w \in V(H)$. In the case 1 there is only one universal quantifier. In the cases 2 and 3 the two sets $V(C_w^2)$ and $V(C_w^1) \setminus \{v_w\}$ are disjoint since $V(C_w^1) \cap V(C_w^2) = \{v_w\}$ by (5.7). Furthermore, in case 4 we know by definition of $C_w^{1,1}$ and $C_w^{1,2}$ that its vertex sets are disjoint. Since for $i \in \{1, 2\}$ the tree $C_w^{1,i}$ is a subtree of C_w^1 that does not contain v_w, its vertex set is disjoint with $V(C_w^2)$ with the same argumentation as in the cases 2 and 3.

Lemma 5.3.6. *The above defined partial function φ' is surjective.*

Proof. We prove that for all $w \in V(H)$ all vertices of R_w are in the image of φ', this is enough since $V(H') = \biguplus_{w \in V(H)} V(R_w)$. In all four cases w' is in the image of φ' since $v_w \in V(C_w^2)$ by (5.7) and therefore $\varphi'(v_w) = w'$. In the cases 2 and 3 there exists a vertex in $v \in V(C_w^1) \setminus \{v_w\}$ since the degree of v_w in C_w^1 is one and therefore $\varphi'(v) = w'_{\mathrm{n}}$. In case 4 both $C_w^{1,1}$ and $C_w^{1,2}$ are by construction non-empty and therefore w'_{n} and w''_{n} are both in the image of φ'. □

Lemma 5.3.7. *For each vertex $x \in V(H')$ the vertex set $\varphi'^{-1}[x]$ is connected in G.*

Proof. We distinguish again the four cases. In all four cases $\varphi'^{-1}[w'] = V(C_w^2)$ is connected in G since C_w^2 is a tree and (5.4) holds. In the cases 2 and 3 we have $\varphi'^{-1}[w'_{\mathrm{n}}] = V(C_w^1) \setminus \{v_w\}$, which is connected since C_w^1 is a tree, the vertex v_w has degree one in C_w^1, which implies that after removing the vertex the result is still a tree, and (5.4). Furthermore, in case 4 we have $\varphi'^{-1}[w'_{\mathrm{n}}] = V(C_w^{1,1})$ and $\varphi'^{-1}[w''_{\mathrm{n}}] = V(C_w^{1,2})$, which are both vertex sets of connected subtrees of C_w^1. The connectedness in G follows again by (5.4). □

Lemma 5.3.8. *It holds that* $\mathrm{dom}(\varphi') = \mathrm{dom}(\varphi)$ *and for* $v_1, v_2 \in \mathrm{dom}(\varphi)$ *it holds that* $\varphi(v_1) \neq \varphi(v_2) \Rightarrow \varphi'(v_1) \neq \varphi'(v_2)$.

Proof. By the definition of φ', (5.6), and (5.7) we get

$$\mathrm{dom}(\varphi') = \bigcup_{w \in V(H)} V(C_w^1) \cup V(C_w^2) \overset{(5.6)}{=} \bigcup_{w \in V(H)} \varphi^{-1}[w] = \varphi^{-1}[V(H)] = \mathrm{dom}(\varphi).$$

The second statement follows from the fact that $\varphi'[\varphi^{-1}[w]] = V(R_w)$ for all $w \in V(H)$ and that $V(R_{w_1}) \cap V(R_{w_2}) = \emptyset$ for two different vertices $w_1, w_2 \in V(H)$. □

Using φ' we can now define edges that connect the subgraphs R_w. For each edge $f \in E(H)$ we add an edge f' to H' with

$$\psi_{H'}(f') := \varphi'[\psi_G(\kappa^{-1}(f))].$$

First of all since κ is injective and surjective the inverse function κ^{-1} is well-defined. Furthermore, we have to prove that $|\varphi'[\psi_G(\kappa^{-1}(f))]| = 2$. By using (5.5) we get $\varphi[\psi_G(\kappa^{-1}(f))] = \psi_H(f)$ and therefore $\psi_G(\kappa^{-1}(f)) \subseteq \mathrm{dom}(\varphi)$. By Lemma 5.3.8, we get $\psi_G(\kappa^{-1}(f)) \subseteq \mathrm{dom}(\varphi')$. Since $\psi_G(\kappa^{-1}(f))$ has two elements and $\varphi[\psi_G(\kappa^{-1}(f))] = \psi_H(f)$ has still two elements we get by Lemma 5.3.8 that $\varphi'[\psi_G(\kappa^{-1}(f))]$ has also two elements.

All in all we can define now the edge set of H' by

$$E(H') := \{f' \mid f \in E(H)\} \dot\cup \biguplus_{w \in V(H)} E(R_w).$$

The graph H' is therefore fully defined. To prove that H' is a minor of G we use the vertex mapping φ' and a surjective edge mapping $\kappa' \colon E(G) \twoheadrightarrow E(H')$, which we define in the following.

Definition 5.3.3 ($\kappa' \colon E(G) \twoheadrightarrow E(H')$)**.** To define the preimages of edges in R_w we distinguish for each vertex $w \in V(H)$ the four cases of Definition 5.3.1.

1. There are no edges in R_w.
2. From (5.13) we know $E_{C_w^1}(v_w) \subseteq \psi_G(\pi_2(T_w))$. Since $E_{C_w^1}(v_w)$ has one element it follows that there exists a unique edge $e \in \pi_2(T_w)$ such that $\psi_G(e) \in E_{C_w^1}(v_w)$, which implies $e \in E_w^1 \cap \pi_2(T_w)$. Using this edge e we define $\kappa'(e) := f'_w$.
3. In this case we have $|\kappa[\pi_2(T_w)]| = 0$, which is equivalent to $\kappa^{-1}[E(H)] \cap \pi_2(T_w) = \emptyset$. This implies $\kappa^{-1}[\pi_2(S_w)] \cap \pi_2(T_w) = \emptyset$. Using (5.8) we can derive

$$\begin{aligned}(5.8) \Rightarrow &\pi_2(T_w) \subseteq \kappa^{-1}[\pi_2(S)] \cup \theta^{-1}[w] \cup (E(v_w) \cap E_w^1) \\ \Rightarrow &\pi_2(T_w) \subseteq \underbrace{\left(\kappa^{-1}[\pi_2(S)] \cap \pi_2(T_w)\right)}_{\emptyset} \cup \theta^{-1}[w] \cup (E(v_w) \cap E_w^1).\end{aligned}$$

   Since θ is injective $\theta^{-1}[w]$ contains at most one edge and therefore we can write $\pi_2(T_w) = \{e_1, e_2\}$ with $e_1 \in E_w^1$ and e_2 is either in $\theta^{-1}[w]$ or in E_w^1. We define now $\kappa'(e_1) := f'_w$ and $\kappa'(e_2) := f''_w$.
4. Again by (5.13) we know $E_{C_w^1}(v_w) \subseteq \psi_G(\pi_2(T_w))$. Since $E_{C_w^1}(v_w)$ has in this case two elements it follows that $E_{C_w^1}(v_w) = \psi_G(\pi_2(T_w))$. Let $\pi_2(T_w) = \{e_1, e_2\}$. Since $\psi_G(\pi_2(T_w)) = E_{C_w^1}(v_w)$ contains two edges where one is connecting v_w to $C_w^{1,1}$ and

the other is connecting v_w to $C_w^{1,2}$ we can say w.l.o.g. that e_i connects v_w and a vertex in $V(C_w^{1,i})$ for $i \in \{1,2\}$. Using this notation we define $\kappa'(e_1) := f'_w$ and $\kappa'(e_2) := f''_w$.

Furthermore, for an edge $f \in E(H)$ we define $\kappa'(\kappa^{-1}(f)) := f'$, which is well-defined since κ is injective and surjective.

We have to prove now that κ' is well-defined, i.e. that no edge in $E(G)$ has more than one image under κ'. First of all we note that in all four cases only edges from $\overline{E_w^1} := E_w^1 \cup \theta^{-1}[w]$ get mapped to an edge f'_w or f''_w.

By Lemma 5.3.5 an edge $e \in \overline{E_w^1}$ always has $\psi_G(e) \subseteq V(C_w^1)$. For two different vertices $w_1, w_2 \in V(H), w_1 \neq w_2$ we always have $\overline{E_{w_1}^1} \cap \overline{E_{w_2}^1} = \emptyset$ since by (5.6) we get $V(C_{w_1}^1) \cap V(C_{w_2}^1) \subseteq \varphi^{-1}[w_1] \cap \varphi^{-1}[w_2] = \emptyset$.

Furthermore, by the definition of κ' an edge never gets mapped to both f'_w and f''_w for a vertex w. It remains to show that an edge in $\mathrm{dom}(\kappa)$ gets not mapped to any edges f'_w or f''_w for any $w \in V(H)$. By (5.5) we get for an edge $e \in \mathrm{dom}(\kappa)$ that $\varphi[\psi_G(e)] = \psi_H(\kappa(e))$ and therefore e is not in $\overline{E_w^1}$ since all edges in $\overline{E_w^1}$ are incident to two vertices in $V(C_w^1) \subseteq \varphi^{-1}[w]$, which would imply $\varphi[\psi_G(e)] = \{w\} \neq \psi_H(\kappa(e))$, which is a contradiction. Therefore, we get $\overline{E_w^1} \cap \mathrm{dom}(\kappa) = \emptyset$ for all $w \in V(H)$. This finishes the proof that κ' is well-defined.

Lemma 5.3.9. *κ' is injective and surjective*

Proof. In the definition of κ' we defined in all cases for each edge in $E(H')$ exactly one preimage that maps under κ' to this edge. This implies directly both, that κ' is injective and surjective. □

Lemma 5.3.10. *The graph H' is a minor of G.*

Proof. We already defined a partial vertex mapping $\varphi' \colon V(G) \nrightarrow V(H')$ and a partial edge mapping $\kappa' \colon E(G) \nrightarrow E(H')$. In Lemma 5.3.6 we showed that φ' is surjective, in Lemma 5.3.7 we showed that all preimages $\varphi'^{-1}[x]$ are connected in G for all $x \in V(H')$, and in Lemma 5.3.9 we showed that κ' is injective and surjective. The only remaining condition of H' being a minor of G is (5.1). To prove this we distinguish again the four cases as we did in Definition 5.3.3.

1. E_w^1 is empty
2. Let $e = \kappa'^{-1}(f'_w)$, then $e \in E_w^1 \cap \pi_2(T_w)$, which implies $\psi_G(e) = \{v_w, v\}$ for some $v \in V(C_w^1) \setminus \{v_w\}$. Therefore, we get

$$\varphi'[\psi_G(e)] = \{\varphi'(v_w), \varphi'(v)\} = \{w', w'_{\mathrm{n}}\} = \psi_{H'}(f'_w) = \psi_{H'}(\kappa'(e)).$$

3. Let e_1 and e_2 be as in case 3 of the definition of κ'. For $i \in \{1,2\}$ we have by Lemma 5.3.5 $\psi_G(e_i) = \{v_w, v\}$ with $v \in V(C_w^1)$ and $v \neq v_w$. This implies for $i \in \{1,2\}$ that

$$\varphi'[\psi_G(e_i)] = \{\varphi'(v_w), \varphi'(v)\} = \{w', w'_{\mathrm{n}}\} = \psi_{H'}(f'_w) = \psi_{H'}(f''_w) = \psi_H(\kappa'(e_i)).$$

4. Let e_1 and e_2 be as in case 4 of the definition of κ' . Then we get $\psi_G(e_i) = \{v_w, v_i\}$ with $v_i \in V(C_w^{1,i})$ and $v_i \neq v_w$ for $i \in \{1,2\}$. With that we can follow (5.1) for $i = 1, 2$.

$$\varphi'[\psi_G(e_1)] = \{\varphi'(v_w), \varphi'(v_1)\} = \{w', w'_{\mathrm{n}}\} = \psi_{H'}(f'_w) = \psi_H(\kappa'(e_1))$$

$$\varphi'[\psi_G(e_2)] = \{\varphi'(v_w), \varphi'(v_2)\} = \{w', w''_{\mathrm{n}}\} = \psi_{H'}(f''_w) = \psi_H(\kappa'(e_2))$$

Finally for $e \in \mathrm{dom}(\kappa)$ with $\kappa(e) = f$ we get

$$\psi_{H'}(\kappa'(e)) = \psi_{H'}(f') = \varphi'[\psi_G(\kappa^{-1}(f))] = \varphi'[\psi_G(e)].$$

All in all we proved (5.1) for all $e \in \mathrm{dom}(\kappa')$. □

To get a reduced transition minor of $(G, \mathcal{T})$ we define $\mathcal{S}'$ as in (5.2) using φ' and κ'. Therefore, we get by definition that $(H', \mathcal{S}')$ is a reduced transition minor of $(G, \mathcal{T})$.

In the second part we want to prove that $(H', \mathcal{S}')$ is a basic-sup-$(H, \mathcal{S})$-graph. We already defined the connected subgraphs R_w for each $w \in V(H)$, they are by definition all vertex disjoint. We also defined the edges $f' \in E(H')$ for each $f \in E(H)$.

Lemma 5.3.11. *For each edge $f \in E(H)$ with $\psi_H(f) = \{w_1, w_2\}$ the edge f' connects the subgraphs R_{w_1} and R_{w_2} of H'.*

Proof. Let $e = \kappa^{-1}(f) \in E(G)$ and $\psi_G(e) = \{v_1, v_2\}$ then by definition we get

$$\psi_{H'}(f') = \varphi'[\psi_G(\kappa^{-1}(f))] = \varphi'[\{v_1, v_2\}].$$

By using (5.5) we get $\varphi[\{v_1, v_2\}] = \psi_H(f) = \{w_1, w_2\}$ and therefore w.l.o.g. $v_i \in \varphi^{-1}[w_i]$, which implies by definition of φ' that $\varphi'(v_i)$ is defined and in R_{w_i}. Therefore, f' connects the vertices $\varphi'(v_1) \in V(R_{w_1})$ and $\varphi'(v_2) \in V(R_{w_2})$. □

For the rest of the proof let $w \in V(H)$ be fixed. Furthermore, let $\{f_1, f_2\} = \pi_2(S_w)$ and let $\{f_3, f_4\} = E_H(w) \setminus \{f_1, f_2\}$ be the remaining edges (note that w has degree four, since H is 4-regular). Since H is completely transitioned we get that $(w, \{f_3, f_4\})$ is also a transition in $\mathcal{S}(w)$. Let $e_i = \kappa^{-1}(f_i)$, which is well-defined since κ is surjective. By definition of κ' we get $\kappa'(e_i) = f'_i$. Before we prove the final lemma, we need the following lemmas.

Lemma 5.3.12. *The only edges in H' that are incident to at least one vertex in R_w are the edges f'_1, f'_2, f'_3, f'_4 and the edges f'_w and f''_w if they exist. Furthermore, for $i \in \{1,2,3,4\}$ the edges f'_i are incident to exactly one vertex x_i in R_w and $x_i = w'$ if $i \in \{3,4\}$.*

Proof. Let E'_w be the set of all edges that are incident to at least one vertex in R_w. Since f'_w and f''_w are by definition incident to two vertices of R_w if they exist and since $V(R_{w_1})$ and $V(R_{w_2})$ are disjoint for $w_1 \neq w_2$ we know that f'_w and f''_w are in E'_w if they exist and f'_{w_1} and f''_{w_1} are not in E'_w if $w_1 \neq w$. Furthermore, from Lemma 5.3.11 follows that f' is incident to a vertex of R_w if and only if $w \in \psi_H(f)$. Therefore, we get

$$E'_w \cap \{f' \mid f \in E(H)\} = \{f' \mid f \in E_H(w)\} = \{f'_1, f'_2, f'_3, f'_4\}.$$

Lemma 5.3.11 also gives us that the edges f_i are incident to exactly one vertex in R_w, since the other vertex must be in another subgraph R_{w_2} with $w_2 \neq w$ since we do not allow loops in H. Let $x_i \in R_w$ be the unique incident vertex of f'_i. By (5.11) we get $\psi_G(e_i) \cap V(C^2_w) \neq \emptyset$ for $i \in \{3,4\}$. Let $v_i \in \psi_G(e_i) \cap V(C^2_w)$ for $i \in \{3,4\}$, which implies $\varphi'(v_i) = w'$ and all in all we get

$$\psi_{H'}(f'_i) = \varphi'[\psi_G(\kappa^{-1}(f_i))] = \varphi'[\psi_G(e_i)] \ni \varphi'(v_i) = w'.$$

Therefore, we know that w' is incident to f'_i and since there exists exactly one vertex in R_w that is incident to f'_i we get $x_i = w'$ for $i \in \{3,4\}$. □

Lemma 5.3.13. *The edges f'_i are incident to w' if and only if $e_i \in \pi_2(T_w)$ for $i \in \{1,2\}$.*

Proof. If $e_i \in \pi_2(T_w)$ we get

$$\psi_{H'}(f'_i) = \varphi'[\psi_G(\kappa^{-1}(f_i))] = \varphi'[\psi_G(e_i)] \ni \varphi'(v_w) = w'.$$

For the other direction we first note that by (5.10) we get that e_i is incident to a vertex in $V(C^1_w)$ for $i \in \{1,2\}$. If f'_i is incident to w' we get

$$w' \in \psi_{H'}(f'_i) = \varphi'[\psi_G(\kappa^{-1}(f_i))] = \varphi'[\psi_G(e_i)],$$

which implies that e_i is incident to a vertex v that is in $\varphi'^{-1}[w'] = V(C^2_w)$. By (5.4) we get that e_i is only incident to one vertex in $\varphi^{-1}[w] = V(C^1_w) \cup V(C^2_w)$. Therefore, we know that e_i is incident to a vertex that is in $V(C^1_w)$ and in $V(C^2_w)$, which can only be v_w by (5.7). All in all we get $e_i \in \kappa^{-1}[\pi_2(S_w)] \cap E(\pi_1(T_w)) \subseteq \pi_2(T_w)$ by (5.9). □

What remains to prove that $(H', \mathcal{S}')$ is a sup-$(H, \mathcal{S})$-graph is the following lemma.

Lemma 5.3.14. *There exists a transition S'_w in $\mathcal{S}'$ such that the form of R_w and the transition S'_w satisfy one of the four possibilities (see Figure 5.5):*

1. *R_w is only one vertex w' and $S'_w = (w', \{f'_1, f'_2\}) \in \mathcal{S}'(w')$.*

2. *R_w is a K_2 with two vertices w' and w'_n, where w'_n is of degree two with two incident edges f'_1 and f'_w. Moreover, $S'_w = (w', \{f'_w, f'_2\}) \in \mathcal{S}'(w')$.*

3. *R_w is a cycle of length two, i.e. two vertices w' and w'_n and two parallel edges f'_w and f''_w connecting them. Furthermore, w'_n has degree four and is incident to f'_1 and f'_2 and $S'_w = (w', \{f'_w, f''_w\}) \in \mathcal{S}'(w')$.*

4. *R_w consists of a vertex w' and two vertices w'_n and w''_n and two edges f'_w and f''_w connecting w' with w'_n and w' with w''_n. Furthermore, w'_n is incident to f'_1, w''_n is incident to f'_2, w' is incident to f'_3 and f'_4, and $S'_w = (w', \{f'_w, f''_w\}) \in \mathcal{S}'(w')$.*

Proof. First we define the transition S'_w by $S'_w := (w', \kappa'[\pi_2(T_w)])$. In the next step we prove that S'_w is in $\mathcal{S}'$. By (5.7), which implies $v_w \in V(C^2_w)$, and the definition of φ' in all four cases we get $v_w \in \mathrm{dom}(\varphi')$ and $\varphi'(v_w) = w'$. By definition of $\mathcal{S}'$ it only remains to show

$$\pi_2(T_w) \subseteq \mathrm{dom}(\kappa'). \tag{5.16}$$

to imply that $S'_w \in \mathcal{S}'$. We show (5.16) together with the rest of the lemma by case distinction of the four cases of Definition 5.3.1.

1. In this case E^1_w is empty. By Lemma 5.3.12, we get that f'_i is incident to w' for $i \in \{1, 2\}$ and by Lemma 5.3.13 this implies that $e_i \in \pi_2(T_w)$ for $i \in \{1, 2\}$. Since $\pi_2(T_w)$ has two elements, we get $\pi_2(T_w) = \{e_1, e_2\} \subseteq \mathrm{dom}(\kappa')$. By definition of κ' this implies $\pi_2(S'_w) = \kappa'[\pi_2(T_w)] = \{\kappa'(e_1), \kappa'(e_2)\} = \{f'_1, f'_2\}$. With that we proved the lemma for case 1.

2. Let e be the only edge in $\pi_2(T_w) \cap \mathrm{dom}(\kappa)$. Since $e \in \mathrm{dom}(\kappa)$ we get $|\varphi(\psi_G(e))| \overset{(5.5)}{=} |\psi_H(\kappa(e))| = 2$, and therefore $\psi_G(e) \not\subseteq V(C^1_w)$. This implies by (5.14) that $e \notin \theta^{-1}[w]$ and that $e \notin E^1_w$. From (5.8) we then get $e \in \kappa^{-1}[\pi_2(S_w)]$. Therefore, $\kappa(e) \in \pi_2(S_w) = \{f_1, f_2\}$ and by exchanging f_1 and f_2 if needed we get $\kappa(e) = f_2$, which implies $e = e_2$.

   Let now $\tilde{e} := \kappa'^{-1}(f'_w)$. By definition of κ' this implies $\tilde{e} \in \pi_2(T_w)$. Furthermore, by definition of κ' we know $\tilde{e} \in E^1_w$, which implies $\tilde{e} \neq e_2$. Therefore, we found the two edges of $\pi_2(T_w) = \{\tilde{e}, e_2\}$ and both of them are in $\mathrm{dom}(\kappa')$ (note that $e_2 \in \mathrm{dom}(\kappa) \subseteq \mathrm{dom}(\kappa')$), which implies (5.16).

   For the edges of S'_w we get

$$\pi_2(S'_w) = \kappa'[\pi_2(T_w)] = \kappa'[\{\tilde{e}, e_2\}] = \left\{\kappa'(\tilde{e}), \kappa'(\kappa^{-1}(f_2))\right\} = \{f'_w, f'_2\}\,.$$

   It remains to show that w'_n has degree two and is incident to f'_1 and f'_w. By definition of f'_w it is incident to w'_n, so we have to show that the only other incident edge to w'_n is f'_1. By Lemma 5.3.12, we get that the only possible other incident edges are f'_1 and f'_2 and by Lemma 5.3.13 we get that f'_2 is incident to w' and therefore not to w'_n.

Assume f'_1 would not be incident to w'_{n}. By Lemma 5.3.12, we get that f'_1 is incident to w' and by Lemma 5.3.13 that $e_1 \in \pi_2(T_w)$. But then $f_1 = \kappa(e_1) \in \kappa[\pi_2(T)] \ni \kappa(e_2) = f_2$, which is a contradiction to the fact that $|\kappa[\pi_2(T)]| = 1$ and that κ is injective.

3. By definition of κ' in this case we get directly (5.16) and also $\kappa'[\pi_2(T_w)] = \{f'_w, f''_w\}$. By definition of R_w in this case it is a cycle of length two consisting of two parallel edges f'_w and f''_w connecting two nodes w' and w'_{n}. Therefore, it only remains to show that w'_{n} has degree four and is connected by f'_1 and f'_2. By Lemma 5.3.12, we get that the only other possible incident edges to w'_{n} are f'_1 and f'_2.

   To show that f'_i is incident to w'_{n} for $i \in \{1, 2\}$ we show that it is not incident to w', which is enough by Lemma 5.3.12. This is equivalent to showing that $e_i \notin \pi_2(T_w)$ for $i \in \{1, 2\}$ by Lemma 5.3.13. For $i \in \{1, 2\}$ the fact $e_i \notin \pi_2(T_w)$ follows directly from the fact that $e_i \in \mathrm{dom}(\kappa)$ and $|\kappa[\pi_2(T_w)]| = 0$ in this case. All in all we showed that the incident edges of w'_{n} are exactly f'_w, f''_w, f'_1, and f'_2.

4. By definition of κ' in this case we get directly (5.16) and also $\kappa'[\pi_2(T_w)] = \{f'_w, f''_w\}$. By definition of R_w in this case it consists of a vertex w' and two vertices w'_{n} and w''_{n} and two edges f'_w and f''_w connecting w' with w'_{n} and w' with w''_{n}. By Lemma 5.3.12, we get that f'_3 and f'_4 are incident to w' and therefore it only remains to show that f'_1 is incident to w'_{n} and f'_2 is incident to w''_{n}.

   As shown in case 4 of the definition of κ' we know that the edges of $\pi_2(T_w)$ are in E^1_w and therefore $e_i \notin \pi_2(T_w)$ for $i \in \{1, 2\}$. By Lemma 5.3.13, we get that f'_i is not incident to w' and therefore that f'_i is incident to w'_{n} or to w''_{n} for $i \in \{1, 2\}$.

   We know in this case that the two trees $C^{1,i}_w$ are both nonempty subtrees of C^1_w for $i \in \{1, 2\}$. If $C^{1,i}_w$ consists only of one vertex, we know that this vertex has degree 1 in C^1_w. On the other hand, if $C^{1,i}_w$ has more than one vertex we know that $C^{1,i}_w$ has at least two leaf vertices of degree 1 and at least one of them has also degree 1 in C^1_w. Therefore, we know that in any case there exists a vertex $v_i \in V(C^{1,i}_w)$ that has degree 1 in C^1_w for $i \in \{1, 2\}$.

   Assume that there exists an edge $e \in \theta^{-1}[w]$ then by (5.9) we know $e \in \pi_2(T_w)$ but we already know that both edges of $\pi_2(T_w)$ are in E^1_w, which is a contradiction to (5.15). Therefore, we know $\theta^{-1}[w] = \emptyset$ in this case.

   Putting everything together we get

$$v_i \in V(C^1_w) \setminus \{\pi_1(T_w)\} \wedge \deg_{C^1_w}(v_i) = 1 \wedge v \notin \bigcup \psi_G[\theta^{-1}[w]],$$

   which implies by (5.12) that $E(v_i) \cap \kappa^{-1}[\pi_2(S_w)] \neq \emptyset$. Since $E(v_1) \cap E(v_2) \cap \mathrm{dom}(\kappa) = \emptyset$ and the fact that $\kappa^{-1}[\pi_2(S_w)] = \{e_1, e_2\}$ we get by exchanging e_1 with e_2 if needed that $e_i \in E(v_i) \cap \kappa^{-1}[\pi_2(S_w)]$ for $i \in \{1, 2\}$. But this implies

$$\psi_{H'}(f'_i) = \varphi'[\psi_G(\kappa^{-1}(f_i))] = \varphi'[\psi_G(e_i)] \ni \varphi'(v_i) \quad \forall i \in \{1, 2\}$$

   and therefore f'_1 is incident to $\varphi'(v_1) = w'_{\mathrm{n}}$ and f'_2 is incident to $\varphi'(v_2) = w''_{\mathrm{n}}$.

□

With Lemma 5.3.14 we finished the proof that $(H', \mathcal{S}')$ is a sup-$(H, \mathcal{S})$-graph and all together that $(G, \mathcal{T})$ has a basic-sup-$(H, \mathcal{S})$-reduced transition minor. □

5.4 Mixed Integer Linear Programming Model

In this section we derive a MILP model from the mathematical model described in Section 5.3.2 in order to practically solve it. Let E_G^{S} denote the set of simple edges $\psi_G[E(G)]$.

To model the partial functions φ, κ, and λ we use boolean variables x_v^w for $v \in V(G)$ and $w \in V(H)$, which are true if and only if $\varphi(v) = w$, boolean variables y_e^f for $e \in E(G)$ and $f \in E(H)$, which are true if and only if $\kappa(e) = f$, and boolean variables z_e^w for $e \in E(G)$ and $w \in V(H)$, which are true if and only if $\theta(e) = w$. Furthermore, to describe the transitions T_w and S_w we use boolean variables a_T^w for $w \in V(H)$ and $T \in \mathcal{T}$ representing $T_w = T$ and boolean variables b_S^w for $w \in V(H)$ and $S \in \mathcal{S}(w)$ representing $S_w = S$.

To describe the trees C_w^i we use boolean variables $h_v^{i,w}$ for $v \in V(G)$, $i \in \{1, 2\}$, and $w \in V(H)$ that encode $v \in V(C_w^i)$ and boolean variables $t_{v_1,v_2}^{i,w}$ for $\{v_1, v_2\} \in \psi_G[E(G)]$, $i \in \{1, 2\}$, and $w \in V(H)$ that encode that the simple edge $\{v_1, v_2\}$ is in $E(C_w^i)$. Note that the variables $t_{v_1,v_2}^{i,w}$ are directed, i.e. for $\{v_1, v_2\} \in \psi_G[E(G)]$ there exist two variables $t_{v_1,v_2}^{i,w}$ and $t_{v_2,v_1}^{i,w}$. To eliminate subtours in the trees C_w^i, which form together a forest, we use a MTZ formulation using continuous variables u_v for $v \in V(G)$ [62].

The following first part of the constraints is concerned with ensuring the properties of the mappings φ, κ, λ, and θ, and the tree structure of the trees C_w^i.

$$\sum_{w \in V(H)} x_v^w \leq 1 \qquad \forall v \in V(G) \quad (5.17)$$

$$\sum_{v \in V(G)} x_v^w \geq 1 \qquad \forall w \in V(H) \quad (5.18)$$

$$\sum_{f \in E(H)} y_e^f \leq 1 \qquad \forall e \in E(G) \quad (5.19)$$

$$\sum_{e \in E(G)} y_e^f = 1 \qquad \forall f \in E(H) \quad (5.20)$$

$$\sum_{w \in V(H)} z_e^w \leq 1 \qquad \forall e \in E(G) \quad (5.21)$$

$$\sum_{e \in E(G)} z_e^w \leq 1 \qquad \forall w \in V(H) \quad (5.22)$$

$$\sum_{T \in \mathcal{T}} a_T^w = 1 \qquad \forall w \in V(H) \quad (5.23)$$

$$\sum_{S \in \mathcal{S}(w)} b_S^w = 1 \qquad \forall w \in V(H) \quad (5.24)$$

$$t^{i,w}_{v_1,v_2} + t^{i,w}_{v_2,v_1} \leq h^{i,w}_{v_j} \qquad \forall \{v_1, v_2\} \in E^{\mathrm{S}}_G, \forall w \in V(H), \forall i, j \in \{1,2\} \tag{5.25}$$

$$\sum_{v_1 \in N_G(v)} t^{i,w}_{v_1,v} = h^{i,w}_v - \sum_{T \in \mathcal{T}(v)} a^w_T \qquad \forall v \in V(G), \forall i \in \{1,2\}, \forall w \in V(H) \tag{5.26}$$

$$u_v \leq (|V_G| - |V_H|)(1 - \sum_{\substack{T \in \mathcal{T}(v) \\ w \in V(H)}} a^w_T) \qquad \forall v \in V(G) \tag{5.27}$$

$$u_{v_1} - u_{v_2} + 1 \leq (|V_G| - |V_H| + 1)(1 - \sum_{\substack{w \in V(H) \\ i \in \{1,2\}}} t^{i,w}_{v_1,v_2}) \qquad \forall \{v_1, v_2\} \in E^{\mathrm{S}}_G \tag{5.28}$$

Constraints (5.17), (5.19), and (5.21) ensure that φ, κ, and θ are partial functions. Furthermore, constraints (5.18) guarantee that φ is surjective, (5.20) ensure that κ is injective and surjective, and (5.22) guarantee that θ is injective. The fact that there should be exactly one transition $T_w \in \mathcal{T}$ and $S_w \in \mathcal{S}(w)$ for each $w \in V(H)$ is ensured by constraints (5.23) and (5.24). Note the difference that T_w can be any transition in $\mathcal{T}$ but S_w must be a transition at the vertex w, i.e. in $\mathcal{S}(w)$.

Constraints (5.25) couple the vertex variables h with the directed edge variables t for each tree C^i_w. To enforce the tree structure of all C^i_w we ensure that each vertex in the tree has exactly one incoming arc except the root vertex, which has no incoming arc and that there are no cycles. Together, this is enough to guarantee the tree structure. As root vertex for each tree C^i_w we use the vertex $v_w = \pi_1(T_w)$, which must be part of both trees by (5.7). Constraints (5.26) specify that each vertex in a tree except the root has exactly one incoming arc and that the root has no incoming arcs. To enforce connectivity we use the MTZ formulation, which is realized by the constraints (5.27) and (5.28). The second part of the MILP is concerned with ensuring constraints (5.4)–(5.15).

$$y^f_e \leq x^{w_1}_v + x^{w_2}_v \qquad \forall e \in E(G), \forall v \in \psi_G(e), \forall f \in E(H), \psi_H(f) = \{w_1, w_2\} \tag{5.29}$$

$$y^f_e \leq x^w_{v_1} + x^w_{v_2} \qquad \forall e \in E(G), \psi_G(e) = \{v_1, v_2\}, \forall f \in E(H), \forall w \in \psi_H(f) \tag{5.30}$$

$$h^{1,w}_v + h^{2,w}_v = x^w_v + \sum_{T \in \mathcal{T}(v)} a^w_T \qquad \forall v \in V(G), \forall w \in V(H) \tag{5.31}$$

$$\sum_{T \in \mathcal{T}(v)} a^w_T \leq x^w_v \qquad \forall v \in V(G), \forall w \in V(H) \tag{5.32}$$

$$a^w_T + b^w_S - 1 \leq \sum_{f \in \pi_2(S)} y^f_e + z^w_e \qquad \forall T \in \mathcal{T}, \forall e = v_1 v_2 \in \pi_2(T) \quad t^{1,w}_{v_1,v_2} + t^{1,w}_{v_2,v_1} \qquad \forall w \in V(H), \forall S \in \mathcal{S}(w), \tag{5.33}$$

$$a^w_T + b^w_S - 1 \leq 1 - \sum_{e \in E(\pi_1(T)) \setminus \pi_2(T)} y^f_e \qquad \forall T \in \mathcal{T}, \forall w \in V(H), \forall S \in \mathcal{S}(w), \forall f \in \pi_2(S) \tag{5.34}$$

$$a_T^w \leq 1 - \sum_{e \in E(G) \setminus \pi_2(T)} z_e^w \qquad \forall T \in \mathcal{T}, \forall w \in V(H) \tag{5.35}$$

$$b_S^w + \sum_{f \in \pi_2(S)} y_e^f - 1 \leq \sum_{v \in \psi_G(e)} h_v^{1,w} \qquad \begin{array}{r} \forall w \in V(H), \forall S \in \mathcal{S}(w), \\ \forall e \in E(G) \end{array} \tag{5.36}$$

$$b_S^w + \sum_{f \in E(w) \setminus \pi_2(S)} y_e^f - 1 \leq \sum_{v \in \psi_G(e)} h_v^{2,w} \qquad \begin{array}{r} \forall w \in V(H), \forall S \in \mathcal{S}(w), \\ \forall e \in E(G) \end{array} \tag{5.37}$$

$$\begin{aligned} b_S^w + h_v^{1,w} - 1 - \sum_{T \in \mathcal{T}(v)} a_T^w \leq \frac{1}{2} \sum_{v_2 \in N_G(v)} t_{v,v_2}^{1,w} \\ + t_{v_2,v}^{1,w} + \sum_{e \in E(v)} z_e^w + \sum_{\substack{e \in E_G(v) \\ f \in \pi_2(S)}} y_e^f \end{aligned} \qquad \begin{array}{r} \forall w \in V(H), \forall S \in \mathcal{S}(w), \\ \forall v \in V(G) \end{array} \tag{5.38}$$

$$a_T^w \leq 1 - t_{v,\pi_1(T)}^{1,w} - t_{\pi_1(T),v}^{1,w} \qquad \begin{array}{r} \forall T \in \mathcal{T}, \forall w \in V(H), \\ \forall v \in N(\pi_1(T)) \setminus \bigcup \psi_G[\pi_2(T)] \end{array} \tag{5.39}$$

$$z_e^w \leq \frac{1}{2} \sum_{v \in \psi_G(e)} h_v^{1,w} \qquad \forall e \in E(G), \forall w \in V(H) \tag{5.40}$$

$$z_e^w \leq 1 - t_{v_1,v_2} - t_{v_2,v_1} \qquad \forall e = v_1 v_2 \in E(G), \forall w \in V(H) \tag{5.41}$$

$$x_v^w \in \{0,1\} \qquad \forall v \in V(G), \forall w \in V(H) \tag{5.42}$$

$$y_e^f \in \{0,1\} \qquad \forall e \in E(G), \forall f \in E(H) \tag{5.43}$$

$$z_e^w \in \{0,1\} \qquad \forall e \in E(G), \forall w \in V(H) \tag{5.44}$$

$$a_T^w \in \{0,1\} \qquad \forall T \in \mathcal{T}, \forall w \in V(H) \tag{5.45}$$

$$b_S^w \in \{0,1\} \qquad \forall w \in V(H), \forall S \in \mathcal{S}(w) \tag{5.46}$$

$$h_v^{i,w} \in \{0,1\} \qquad \begin{array}{r} \forall v \in V(G), \forall i \in \{1,2\}, \\ \forall w \in V(H) \end{array} \tag{5.47}$$

$$t_{v_1,v_2}^{i,w} \in \{0,1\} \qquad \begin{array}{r} \forall \{v_1, v_2\} \in r[E(G)], \\ \forall i \in \{1,2\}, \forall w \in V(H) \end{array} \tag{5.48}$$

$$0 \leq u_v \leq |V_G| - |V_H| \qquad \forall v \in V(G) \tag{5.49}$$

By the index range of the variables $t_{v_1,v_2}^{i,w}$ constraints (5.4) are implicitly satisfied. Constraints (5.29) and (5.30) ensure constraints (5.5). Furthermore, constraints (5.31) and (5.32) together enforce (5.6) and (5.7). Constraint (5.8) are realized by (5.33) and constraints (5.9) by (5.34) and (5.35). Moreover, (5.36) ensure (5.10) and (5.37) ensures (5.11). Furthermore, (5.12) are guaranteed by (5.38) and (5.13) by (5.39). Last but not least (5.40) ensure (5.14) and (5.41) ensure (5.15).

5.5 SAT Model

As an alternative to the MILP model presented in Section 5.4 we present in this sction a SAT model based on the mathematical model given in Section 5.3.2.

Most constraints of the mathematical model can more or less directly be translated into SAT clauses. One critical aspect is how to model the tree C_w^i for $w \in V(H)$ and $i \in \{1,2\}$. Constraints (5.6) and (5.7) ensure that the subgraph C_w formed by C_w^1 and C_w^2 together is a tree and all trees C_w are disjoint for $w \in V(H)$. Combining all trees C_w for $w \in V(H)$ we obtain a forest and for each of the trees C_w we define a unique root $\pi_1(T_w)$ by (5.7). When modeling the forest in a directed fashion, we then only have to take care to avoid any cycles. There are different techniques in literature to model acyclicity in SAT models. Some of those techniques are summarized in [46]. We use the approach based on a transitive closure for ensuring acyclicity in our model. Our SAT model uses the following variables:

- x_v^w for $v \in V(G)$, $w \in V(H)$ represents $\varphi(v) = w$,
- y_e^f for $e \in E(G)$, $f \in E(H)$ represents $\kappa(e) = f$,
- z_e^w for $e \in E(G)$, $w \in V(H)$ represents $\theta(e) = w$,
- a_T^w for $w \in V(H)$, $T \in \mathcal{T}$ represents $T = T_w$,
- b_S^w for $w \in V(H)$, $S \in \mathcal{S}(w)$ represents $S = S_w$,
- $o_v^{i,w}$ for $v \in V(G)$, $w \in V(H)$, $i \in \{1,2\}$ represents $v \in V(C_w^i)$,
- $p_a^{i,w}$ for $a \in A(G)$, $w \in V(H)$, $i \in \{1,2\}$ represents $a \in A(C_w^i)$,
- t_{v_1,v_2} for $v_1, v_2 \in V(G)$ is the transitive closure relation of all $p_a^{i,w}$ variables.

The trees C_w^i are modeled as a directed rooted out-trees and the variables $p_a^{i,w}$ decide which directed arcs are part of the tree. Set $A(G)$ is the set of all directed arcs of edges in G when eliminating parallel edges, i.e.

$$A(G) := \{(v,w),(w,v) \mid e = vw \in E(G)\}.$$

So for every pair of adjacent vertices in G there are two arcs in opposite direction in $A(G)$. We write $A^{\text{in}}(v)$ for the ingoing arcs at v and $A^{\text{out}}(v)$ for the outgoing arcs at v. In the following we list all constraints of our SAT model. For simplicity, we present the constraints in the form of propositional logic formulas. To transform them into clauses we use De Morgan's law and the distributive property. One alternative would be to use Tseitin transformations [78], although for the constraints we are using the number of

resulting clauses using the naive transformation is still small and therefore this is not needed. In the following we use for a given $v \in V(G)$, $w \in V(H)$, and $i \in \{1,2\}$

$$\text{oneIn}(v,i,w) := \Big(\bigvee_{a \in A^{\text{in}}(v)} p_a^{i,w} \Big) \wedge \bigwedge_{\substack{a_1,a_2 \in A^{\text{in}}(v) \\ a_1 \neq a_2}} \left(\neg p_{a_1}^{i,w} \vee \neg p_{a_2}^{i,w} \right).$$

The basic structures as defined in the mathematical model are expressed by

$$\neg(x_v^{w_1} \wedge x_v^{w_2}) \qquad \forall v \in V(G), \forall w_1, w_2 \in V(H), w_1 \neq w_2 \qquad (5.50)$$

$$\bigvee_{v \in V(G)} x_v^w \qquad \forall w \in V(H) \qquad (5.51)$$

$$\neg(y_e^{f_1} \wedge y_e^{f_2}) \qquad \forall e \in E(G), \forall f_1, f_2 \in E(H), f_1 \neq f_2 \qquad (5.52)$$

$$\neg(y_{e_1}^{f} \wedge y_{e_2}^{f}) \qquad \forall e_1, e_2 \in E(G), e_1 \neq e_2, \forall f \in E(H) \qquad (5.53)$$

$$\bigvee_{e \in E(G)} y_e^f \qquad \forall f \in E(H) \qquad (5.54)$$

$$\neg(z_e^{w_1} \wedge z_e^{w_2}) \qquad \forall e \in E(G), \forall w_1, w_2 \in V(H), w_1 \neq w_2 \qquad (5.55)$$

$$\neg(z_{e_1}^{w} \wedge z_{e_2}^{w}) \qquad \forall e_1, e_2 \in E(G), e_1 \neq e_2, \forall w \in V(H) \qquad (5.56)$$

$$\neg(a_{T_1}^{w} \wedge a_{T_w}^{w}) \qquad \forall w \in V(H), \forall T_1, T_2 \in \mathcal{T}, T_1 \neq T_2 \qquad (5.57)$$

$$\bigvee_{T \in \mathcal{T}} a_T^w \qquad \forall w \in V(H) \qquad (5.58)$$

$$\neg(a_T^{w_1} \wedge a_T^{w_2}) \qquad \forall w_1, w_2 \in V(H), w_1 \neq w_2, \forall T \in \mathcal{T} \qquad (5.59)$$

$$\neg(b_{S_1}^{w} \wedge b_{S_2}^{w}) \qquad \forall w \in V(H), \forall S_1, S_2 \in \mathcal{S}(w), S_1 \neq S_2 \qquad (5.60)$$

$$\bigvee_{S \in \mathcal{S}(w)} b_S^w \qquad \forall w \in V(H) \qquad (5.61)$$

$$o_v^{i,w} \rightarrow \bigvee_{T \in \mathcal{T}(v)} a_T^w \vee \text{oneIn}(v,i,w) \qquad \forall v \in V(G), w \in V(H), i \in \{1,2\} \qquad (5.62)$$

$$\bigvee_{T \in \mathcal{T}(v)} a_T^w \rightarrow o_v^{i,w} \wedge \bigwedge_{a \in A^{\text{in}}(v)} \neg p_a^{i,w} \qquad \forall v \in V(G), w \in V(H), i \in \{1,2\} \qquad (5.63)$$

$$\neg o_v^{i,w} \rightarrow \bigwedge_{a \in A^{\text{in}}(v) \cup A^{\text{out}}(v)} \neg p_a^{i,w} \qquad \forall v \in V(G), w \in V(H), i \in \{1,2\} \qquad (5.64)$$

$$\neg(t_{v_1,v_2} \wedge t_{v_2,v_1}) \qquad \forall a = (v_1,v_2) \in A(G) \qquad (5.65)$$

$$t_{v_1,v_2} \wedge t_{v_2,v_3} \rightarrow t_{v_1,v_3} \qquad \forall a = (v_1,v_2) \in A(G), v_3 \in V(G) \qquad (5.66)$$

$$\bigvee_{w \in V(H), i \in \{1,2\}} p_a^{i,w} \rightarrow t_{v_1,v_2} \qquad \forall a = (v_1,v_2) \in A(G). \qquad (5.67)$$

Constraints (5.50), (5.52), (5.55), (5.57), and (5.60) ensure that φ, κ, θ, $w \mapsto T_w$, and $w \mapsto S_w$ are partial functions with the special restriction that $S_w \in \mathcal{S}(w)$. Furthermore, constraints (5.51) and (5.54) enforce that φ and κ are surjective. On the other hand, constraints (5.53), (5.56), and (5.59) ensure that κ, θ, and $w \mapsto T_w$ are injective. Note that the mathematical model does not state directly that $w \mapsto T_w$ should be injective,

but it does indirectly by constraints (5.6) and (5.7). Additionally, constraints (5.58) and (5.61) guarantee that there exists a T_w and a S_w for each $w \in V(H)$.

Constraints (5.62)–(5.64) characterize three types of vertices in G. The root vertices of the trees C_w^i, which are defined by the vertices of the transitions T_w by (5.7), do not have any ingoing arcs in C_w^i. Other vertices in C_w^i that are not roots have exactly one ingoing arc in C_w^i and vertices that are not in C_w^i have no ingoing or outgoing arc in C_w^i. Last but not least, constraints (5.65)–(5.67) ensure that the trees C_w^i have no cycles by using the transitive closure variables t_{v_1,v_2} similarly as it is described in [46]. Instead of having just one variable, which represents if a directed edge is part of the forest, we use in our case the disjunction $\bigvee_{w \in V(H), i \in \{1,2,\}} p_a^{i,w}$ for an arc a. With this we ensured all structural properties formulated in the mathematical model. What is left is to model constraints (5.4)–(5.15), which is achieved by

$$y_e^f \to (x_{v_1}^{w_1} \wedge x_{v_2}^{w_2}) \vee (x_{v_2}^{w_1} \wedge x_{v_1}^{w_2}) \qquad \forall e = v_1, v_2 \in E(G), \forall f = w_1 w_2 \in E(H) \tag{5.68}$$

$$o_v^{1,w} \vee o_v^{2,w} \leftrightarrow x_v^w \qquad \forall v \in V(G), \forall w \in V(H) \tag{5.69}$$

$$\bigvee_{T \in \mathcal{T}(v)} a_T^w \leftrightarrow o_v^{1,w} \wedge o_v^{2,w} \qquad \forall v \in V(G), \forall w \in V(H) \tag{5.70}$$

$$\bigvee_{\substack{T \in \mathcal{T} \\ e \in \pi_2(T)}} a_T^w \to \bigvee_{S \in \mathcal{S}(w)} \left(b_S^w \wedge \bigvee_{f \in \pi_2(S)} y_e^f \right) \vee z_e^w \vee p_{(v_1,v_2)}^{1,w} \vee p_{(v_2,v_1)}^{1,w} \qquad \forall e = v_1 v_2 \in E(G), \forall w \in V(H) \tag{5.71}$$

$$a_T^w \wedge b_S^w \to \neg \bigvee_{f \in \pi_2(S)} y_e^f \qquad \forall w \in V(H), \forall S \in \mathcal{S}(w), \forall T \in \mathcal{T}, \forall e \in E(\pi_1(T)) \setminus \pi_2(T) \tag{5.72}$$

$$a_T^w \to \neg z_e^w \qquad \forall w \in V(H), \forall T \in \mathcal{T}, \forall e \in E(\pi_1(T)) \setminus \pi_2(T) \tag{5.73}$$

$$b_S^w \wedge \bigvee_{f \in \pi_2(S)} y_e^f \to o_{v_1}^{1,w} \vee o_{v_2}^{1,w} \qquad \forall w \in V(H), \forall S \in \mathcal{S}(w), \forall e = v_1 v_2 \in E(G) \tag{5.74}$$

$$\left(b_S^w \wedge \bigvee_{f \in E(w) \setminus \pi_2(S)} y_e^f \right) \to o_{v_1}^{2,w} \vee o_{v_2}^{2,w} \qquad \forall w \in V(H), \forall S \in \mathcal{S}(w), \forall e = v_1 v_2 \in E(G) \tag{5.75}$$

$$b_S^w \wedge o_v^{1,w} \wedge \bigwedge_{v' \in N(v)} \neg p_{(v,v')}^{1,w} \wedge \bigwedge_{e \in E(v)} \neg z_e^w \to \left(\bigvee_{e \in E(v), f \in \pi_2(S)} y_e^f \vee o_v^{2,w} \right) \qquad \forall w \in V(H), \forall S \in \mathcal{S}(w), \forall v \in V(G) \tag{5.76}$$

$$a_T^w \to \neg p_{(\pi_1(T),v)}^{1,w} \wedge \neg p_{(v,\pi_1(T))}^{1,w} \qquad \forall w \in V(H), \forall T \in \mathcal{T}, \forall v \in N(\pi_1(T)) \setminus \bigcup r_G[\pi_2(T)] \tag{5.77}$$

$$z_e^w \to (o_{v_1}^{1,w} \wedge o_{v_2}^{1,w}) \qquad \forall w \in V(H), \forall e = v_1 v_2 \in E(G) \tag{5.78}$$

$$z_e^w \to (\neg p^{1,w}_{(v_1,v_2)} \wedge \neg p^{1,w}_{(v_2,v_1)}) \qquad \forall w \in V(H), \forall e = v_1v_2 \in E(G). \qquad (5.79)$$

Constraints (5.4) are already satisfied implicitly and constraints (5.5)–(5.8) are realized by constraints (5.68)–(5.71) respectively. Furthermore, constraints (5.9) are guaranteed by (5.72) and (5.73). All the other constraints (5.10)–(5.15) are modeled via (5.74)–(5.79) respectively.

5.6 Symmetry Breaking

The input graphs G and H, especially H, often have symmetries leading to symmetric solutions in our model. To avoid those we analyze the structure of the symmetries in G and H and incorporate symmetry breaking constraints into our model.

To formalize the concept of symmetries in transitioned graphs we extend the definition of homomorphisms on graphs to transitioned graphs. Note that our definition of a graph homomorphism in Definition 2.1.16 does not specify which edge of parallel edges get mapped to which edge, since it is only defined on vertices and ensures that the number of edges between two vertices is correct. Therefore, using such homomorphisms cannot eliminate edge symmetries. If we would want to also eliminate edge symmetries this would lead to more complex symmetry breaking constraints and would only help in cases where there are a lot of parallel edges.

We can extend the definition of homomorphisms to transitioned graphs by enforcing that it also preserves transitions in the following way.

Definition 5.6.1. A *homomorphism* between two transitioned graphs $(G, \mathcal{T})$ and $(H, \mathcal{S})$ is a vertex mapping $f : V(G) \mapsto V(H)$ such that f is a homomorphism between G and H and induces a mapping between the edges $g : E(G) \to E(H)$ according to the end vertex relation of f, i.e

$$f[\psi(e)] = \psi(g(e)) \quad \forall e \in E(G),$$

such that transitions are preserved, i.e.

$$(f(v), \{g(e_1), g(e_2)\}) \in \mathcal{S} \quad \forall T = (v, \{e_1, e_2\}) \in \mathcal{T}.$$

An *isomorphism* between $(G, \mathcal{T})$ and $(H, \mathcal{S})$ is a bijective homomorphism f between $(G, \mathcal{T})$ and $(H, \mathcal{S})$ whose inverse is a homomorphism between $(H, \mathcal{S})$ and $(G, \mathcal{T})$. Furthermore, an *automorphism* of a transitioned graph $(G, \mathcal{T})$ as an isomorphism from $(G, \mathcal{T})$ to itself, we write $\mathrm{Aut}(G, \mathcal{T})$ for the group of all automorphisms on $(G, \mathcal{T})$.

Given input graphs $(G, \mathcal{T})$ and $(H, \mathcal{S})$ we can use automorphisms to transform feasible solutions into other feasible solutions. More formally for any feasible solution and any pair of automorphisms $f \in \mathrm{Aut}(G, \mathcal{T})$ and $g \in \mathrm{Aut}(H, \mathcal{S})$ of which at least one is not the identity we can construct another feasible solution by replacing all vertices in G according to f and all vertices in H according to g. Since f and g preserve all edges and all transitions this is sufficient to get a new feasible solution.

Next we propose an approach how to eliminate some of those symmetries. Let S be a feasible solution with vertex mapping $\varphi : V(G) \nrightarrow V(H)$. We assume that $V(G)$ and $V(H)$ are totally ordered sets. We can define for any pair of automorphisms $f \in \mathrm{Aut}(G, \mathcal{T})$ and $g \in \mathrm{Aut}(H, \mathcal{S})$ a sequence $\alpha^{f,g} := (\alpha_w^{f,g})_{w \in V(H)}$ by

$$\alpha_w^{f,g} := \min f[\varphi^{-1}[g(w)]]$$

which is well-defined since φ is surjective. The sequence $\alpha^{f,g}$ contains the smallest vertex of each preimage of φ after applying the automorphisms f and g to the solution. The idea is to enforce that $\alpha := \alpha^{\mathrm{id}_{V(G)}, \mathrm{id}_{V(H)}}$ is lexicographically minimal compared to all $\alpha^{f,g}$ for all pairs of automorphisms $f \in \mathrm{Aut}(G, \mathcal{T})$ and $g \in \mathrm{Aut}(H, \mathcal{S})$, i.e.

$$\alpha \leq_{\mathrm{lex}} \alpha^{f,g} \quad \forall f \in \mathrm{Aut}(G, \mathcal{T}), \forall g \in \mathrm{Aut}(H, \mathcal{S}). \tag{5.80}$$

Note that there may be multiple different feasible solutions with the same sequence α and therefore this only eliminates some symmetries. Such different solutions with the same α may differ in the mapped edges or transitions, or differ in vertices in G that are not mapped by φ or are not the smallest vertices of the preimages of φ. But if H is simple this restriction eliminates all symmetries occurring only in H, i.e. if we only apply an automorphism in $\mathrm{Aut}(H, \mathcal{S}) \setminus \left\{\mathrm{id}_{V(H)}\right\}$ to a feasible solution satisfying (5.80) the resulting solution will not satisfy (5.80). To formalize (5.80) in such a way that it can be modeled in a MILP or a SAT formulation we have to expand the definition of a lexicographical ordering. Condition (5.80) is equivalent to

$$\begin{aligned}
&\forall w \in V(H), \forall f \in \mathrm{Aut}(G, \mathcal{T}), \forall g \in \mathrm{Aut}(H, \mathcal{S}) : \\
&\alpha_w \leq \alpha_w^{f,g} \vee \exists w' < w : \alpha_{w'} < \alpha_{w'}^{f,g} \\
\Leftrightarrow &(\forall v < \alpha_w : f(v) \notin \varphi^{-1}[g(w)]) \vee (\exists w' < w : \forall v \leq \alpha_w : f(v) \notin \varphi^{-1}[g(w')]).
\end{aligned}$$

This constraint is still complicated and results in a lot of constraints in SAT or MILP models. To avoid bloating the models we consider only the variant for the smallest vertex $w_0 := \min(V(H))$ of H. Then the condition can be simplified using orbits.

Definition 5.6.2. Let f be an automorphism on a transitioned graph $(G, \mathcal{T})$. The set

$$\mathrm{orb}(v) := \{v' \in V \mid \exists f \in \mathrm{Aut}(G, \mathcal{T}) : f(v) = v'\}$$

is called the *orbit* of $v \in V$. Orbits are the equivalence classes of the equivalence relation corresponding to $\mathrm{Aut}(G, \mathcal{T})$ in which two vertices are equivalent if there exists an automorphism mapping one vertex to the other.

Using the definition of orbits we can simplify our condition for the special case w_0

$$\begin{aligned}
&f(v) \notin \varphi^{-1}[g(w_0)] \ \forall v < \alpha_{w_0}, \forall f \in \mathrm{Aut}(G, \mathcal{T}), \forall g \in \mathrm{Aut}(H, \mathcal{S}) \\
\Leftrightarrow &v' \notin \varphi^{-1}[w] \ \forall v < \alpha_{w_0}, \forall v' \in \mathrm{orb}(v), \forall w \in \mathrm{orb}(w_0)
\end{aligned}$$

$$\begin{aligned}
&\Leftrightarrow \varphi(v') = w \rightarrow \alpha_{w_0} \leq v \;\forall v \in V(G), \forall v' \in \text{orb}(v), \forall w \in \text{orb}(w_0) \\
&\Leftrightarrow \varphi(v) = w \rightarrow \exists v'' \leq v' : \varphi(v'') = w_0 \;\forall v' \in V(G), \forall v \in \text{orb}(v'), \forall w \in \text{orb}(w_0) \\
&\Leftrightarrow \varphi(v) = w \rightarrow \exists v' \leq \min \text{orb}(v) : \varphi(v') = w_0 \;\forall v \in V(G), \forall w \in \text{orb}(w_0). \qquad (5.81)
\end{aligned}$$

Another specialization of (5.80) is if we only consider automorphisms on H, i.e. fix $f = id_{V(G)}$. In this case we simply have $\alpha_w^g := \alpha_w^{\text{id}_{V(G)},g} = \min \varphi^{-1}[g(w)] = \alpha_{g(w)}$, i.e. the α values are simple permutations of each other based on g. Therefore, the symmetry breaking condition holds if and only if

$$(\alpha_w)_{w \in V(H)} \leq_{\text{lex}} (\alpha_{g(w)})_{w \in V(H)} \quad \forall g \in \text{Aut}(H, \mathcal{S}). \qquad (5.82)$$

Note that $(\alpha_w)_{w \in V(H)} \leq_{\text{lex}} (\alpha_{g(w)})_{w \in V(H)}$ if and only if for the first vertex w for which $\alpha_w \neq \alpha_{g(w)}$, $\alpha_w < \alpha_{g(w)}$ holds. Since all values in α_w are different we know that $\alpha_w = \alpha_{g(w)}$ if and only if $w = g(w)$. Therefore, if w is the first value where they are different this implies that g fixes all $w' < w$, i.e. $g(w') = w'$ for all $w' < w$.

Definition 5.6.3. Let $S \subseteq V(G)$, then the *stabilizer* of $\text{Aut}(H, \mathcal{S})$ with respect to S is defined by $\text{Aut}_S(H, \mathcal{S}) := \{g \in \text{Aut}(H, \mathcal{S}) \mid \forall s \in S : g(s) = s\}$, which is a subgroup of $\text{Aut}(H, \mathcal{S})$. We can again define *stabilizer orbits* according to the automorphisms in the stabilizer, i.e. $\text{orb}_S(v) := \{v' \in V \mid \exists f \in \text{Aut}_S(G, \mathcal{T}) : f(v) = v'\}$.

With this definition we can reformulate (5.82) in the following way:

$$\begin{aligned}
&\alpha_w < \alpha_{g(w)} \quad \forall w \in V(H), \forall g \in \text{Aut}_{\{w' \in V(H) : w' < w\}}(H, \mathcal{S}) : g(w) \neq w \\
&\Leftrightarrow \alpha_w < \alpha_{w''} \quad \forall w \in V(H), \forall w'' \in \text{orb}_{\{w' \in V(H) : w' < w\}}(w) \setminus \{w\}.
\end{aligned}$$

The condition $\alpha_w < \alpha_{w''}$ can be expressed such that the statement is equivalent to

$$\varphi(v) = w'' \rightarrow \exists v' < v : \varphi(v') = w \qquad \begin{array}{r} \forall v \in V(G), \forall w \in V(H), \\ \forall w'' \in \text{orb}_{\{w' \in V(H) : w' < w\}}(w) \setminus \{w\}. \end{array} \qquad (5.83)$$

To model constraints (5.81) and (5.83) we use the inequalities

$$x_v^w \leq \sum_{v' \leq \min \text{orb}(v)} x_{v'}^{w_0} \qquad \forall v \in V(G), \forall w \in \text{orb}(w_0) \qquad (5.84)$$

$$x_v^{w''} \leq \sum_{v' < v} x_{v'}^w \qquad \begin{array}{r} \forall v \in V(G), \forall w \in V(H), \\ \forall w'' \in \text{orb}_{\{w' \in V(H) : w' < w\}}(w) \setminus \{w\} \end{array} \qquad (5.85)$$

for the MILP model and the constraints

$$x_v^w \rightarrow \bigvee_{v' \leq \min \text{orb}(v)} x_{v'}^{w_0} \qquad \forall v \in V(G), \forall w \in \text{orb}(w_0) \qquad (5.86)$$

$$x_v^{w''} \rightarrow \bigvee_{v' < v} x_{v'}^w \qquad \begin{array}{r} \forall v \in V(G), \forall w \in V(H), \\ \forall w'' \in \text{orb}_{\{w' \in V(H) : w' < w\}}(w) \setminus \{w\} \end{array} \qquad (5.87)$$

for the SAT model.

Figure 5.7: Construction example of the auxiliary graph for a part of a transitioned graph $(G, \mathcal{T})$. The newly added artificial vertices have color 2, which is drawn white and the original vertices have color 1, which is drawn black.

5.6.1 Finding all Automorphisms and Stabilizers

To add constraints (5.84)–(5.85) or (5.86)–(5.87) to our model we need to compute the automorphism group $\mathrm{Aut}(G, \mathcal{T})$, its orbits, the automorphism group $\mathrm{Aut}(H, \mathcal{S})$, its orbits, and the orbits $\mathrm{orb}_{\{w' \in V(H): w' < w\}}(w)$ of the stabilizers for each $w \in V(H)$.

The problem of computing a set of generators of the automorphism group of a simple graph is well studied. It is closely related to the famous graph isomorphism problem. Since no polynomial time algorithm is known for the graph isomorphism problem, which can be reduced to computing generators of the automorphism group of the graph, all proposed algorithms in literature require exponential time in general. Nevertheless, if we restrict the problem to graphs with bounded degree, like it is the case for the input graph H, which is always 4-regular, there are polynomial time algorithms, see [60]. On the other hand, there are well performing algorithms in practice, which can handle graphs with unbounded degree. See for example McKay and Piperno [61] where they solved the problem for graphs with several thousand vertices in reasonable time.

The algorithm of McKay and Piperno and also other algorithms in the literature working similarly get as an input a simple undirected graph $G = (V, E)$ with a vertex coloring $c : V \to \{1, \dots m\}$ and compute a generator of $\mathrm{Aut}^c(G)$, which is the subgroup of $\mathrm{Aut}(G)$ that preserves the colors given by c, i.e.

$$\mathrm{Aut}^c(G) := \{f \in \mathrm{Aut}(G) \mid c(f(v)) = c(v) \quad \forall v \in V(G)\}\,.$$

Since we need to compute automorphism groups of transitioned multigraphs, we need to transform our graphs in such a way that we can apply McKay's algorithm to it. Let $(G, \mathcal{T})$ be a transitioned graph. We construct an auxiliary graph $\mathrm{Aux}(G, \mathcal{T})$ by inserting in each edge $e = v_1v_2$ of G two vertices $w_e^{v_1}$ and $w_e^{v_2}$. This gives us immediately a simple graph. Furthermore, for each transition $t = (v, e_1, e_2) \in \mathcal{T}$ we add an edge between the vertices $w_{e_1}^v$ and $w_{e_2}^v$. We also define a coloring c on the auxiliary graph by coloring all original vertices with the color 1 and all artificially added vertices with the color 2. See Figure 5.7 for an example on how to construct the auxiliary graph for a part of a given transitioned graph $(G, \mathcal{T})$.

Theorem 5.6.1.

$$\mathrm{Aut}(G, \mathcal{T}) = \left\{f|_{V(G)} \mid f \in \mathrm{Aut}^c(\mathrm{Aux}(G, \mathcal{T}))\right\}$$

Proof. By adding the two artificial vertices with a second color between each edge we can associate with each automorphism in the auxiliary graph a vertex mapping and an edge mapping in the original graph. The edge mapping is defined by mapping an edge e_1 to an edge e_2 if the two artificial vertices on e_1 get mapped to the two artificial vertices on e_2 in the auxiliary graph. Furthermore, since there are edges between two added vertices $w^v_{e_1}$ and $w^v_{e_2}$ if and only if there is a transaction $(v, \{e_1, e_2\})$ in the original graph we also get that the mappings are transition-preserving. On the other hand, given a vertex mapping and an edge mapping as in the definition of an automorphism in a transitioned graph, we can use those to formulate an edge-preserving vertex mapping on the auxiliary graph. □

Theorem 5.6.1 shows us that we can use the auxiliary graph $\mathrm{Aux}(G, \mathcal{T})$ to get the automorphism group of a transitioned graph $(G, \mathcal{T})$ by using an algorithm to compute $\mathrm{Aut}^c(\mathrm{Aux}(G, \mathcal{T}))$. What remains is how to compute the orbits and the orbits of the stabilizers, which can be done with the Schreier-Sims algorithm [74]. To get the orbits of the stabilizers we may have to reorder our vertices (which in effect changes the needed stabilizers) according to the result of the Schreier-Sims algorithm. This is no problem for our model, since the order of the vertices is only relevant for the symmetry breaking and can therefore be adjusted.

5.7 Systematically Checking PPMs of Snarks

One application of solving ESTM with our MILP approach or our SAT approach as motivated in Section 5.1 is to verify if the contraction of a PPM in a snark is SUD-K_5-minor-free and therefore contains a CCD, which would imply the existence of a CDC in the snark. To that end we want to analyze all snarks with up to a certain size if they contain a PPM whose contraction leads to a planar, a K_5-minor-free, a SUD-K_5-minor-free, or a CCD-containing graph.

Fortunately, there exist already collections of all snarks with up to 36 vertices, see [13]. Therefore, we can iterate through all snarks with up to a certain size and apply the following algorithm to them. The algorithm enumerates all PPMs iteratively by ordering the vertices of the snark and always trying to add all possible edges or claws to the pseudo-matching that contain the smallest not yet visited vertex of the snark. Then it checks for each generated PPM if its contraction leads to a planar graph. If it does not find such a matching it checks for K_5-minor-free contractions, if this also is not the case it checks for SUD-K_5-minor-free contractions and otherwise it checks for CCD-containing contractions. Using this algorithm one can specify for each snark the type of the strongest matching found for this snark.

5.8 Computational Results

To compare the SAT and MILP model we implemented both in C++ using Glucose 4.1[1] to solve the SAT model and Gurobi Optimizer 8.1[2] to solve the MILP model. We also tested the impact of the symmetry breaking constraints for both models. To get the automorphism groups as described in Section 5.6.1 we used nauty 2.6[3] [61] and to get a strong generating set we used the implementation of the Schreier-Sims algorithm contained in the nauty program. All tests were performed on a single core of an Intel Xeon E5-2640 v4 processor with 2.40GHz and 8GB RAM.

5.8.1 Instance Sets

We consider four different instance sets S1, S2, G1, and G2 two of them represent graph theoretic use cases and one represents the most general case. As discussed in Section 5.1 there are two correlations between the CCD and CDC, one via the line graph of a 3-regular graph and the other via contractions of a PPM of a 3-regular graph. The first correlation is not interesting for us, since the line graph of a 3-regular graph is larger than the original graph, the second correlation, however, gives us in general a graph with at most half the number of vertices and is therefore the use case we test in S1 and S2. Note that we can restrict our instances to contractions of PPMs of snarks as discussed in Section 5.1. For both sets S1 and S2 we use a snark together with a PPM of the snark to build a transitioned graph as described already in Section 5.1. We contract all components of the matching and add then a transition between the two remaining edges of each original vertex of the snark, which gives us a 4-regular completely transitioned graph.

To generate instance set S1 we use all snarks with up to 26 vertices plus 1000 snarks with 28 vertices and compute for each of them three random perfect matchings. For each snark and each of its perfect matchings we build the resulting transitioned graph as described above and use it as $(G, \mathcal{T})$, and as transitioned graph $(H, \mathcal{S})$ we use the graph UD-K_5 as defined in Example 5.2.1. As source for the snarks with up to 28 vertices we used the lists published by Brinkmann et al. [13].

For instance set S2 we consider all possible PPMs of each snark with up to 22 vertices. Again we construct the transitioned graph $(G, \mathcal{T})$ as described above. Note that by the cyclically 4-edge connectedness and the 3-regularity of a snark it follows that a snark contains no cycles of length three and therefore that after contracting a K_2 or a claw there will be no loop. As $(H, \mathcal{S})$ we use again the UD-K_5 from Example 5.2.1. When we construct all possible PPMs of a snark we first compute its automorphism group using the nauty tool from McKay et al. [61], which we then also apply to filter out all PPMs that lead after contraction to isomorphic transitioned graphs.

[1] https://www.labri.fr/perso/lsimon/glucose (accessed 09/2019)
[2] https://www.gurobi.com (accessed 09/2019)
[3] http://pallini.di.uniroma1.it/ (accessed 09/2019)

Table 5.1: Computation results for instance set S1.

				MILP			MILP_{sym}			SAT		SAT_{sym}	
$\lvert V(C)\rvert$	$\lvert I\rvert$	$\lvert I_{\text{feas}}\rvert$	$\lvert I_{\text{inf}}\rvert$	$t_{\text{feas}}[s]$	$t_{\text{inf}}[s]$	$\lvert I_{\text{tl}}\rvert$	$t_{\text{feas}}[s]$	$t_{\text{inf}}[s]$	$\lvert I_{\text{tl}}\rvert$	$t_{\text{feas}}[s]$	$t_{\text{inf}}[s]$	$t_{\text{feas}}[s]$	$t_{\text{inf}}[s]$
10	4	4	0	**< 1**	-	0	**< 1**	-	0	**< 1**	-	**< 1**	-
18	8	8	0	3	-	0	4	-	0	**< 1**	-	**< 1**	-
20	24	24	0	2	-	0	2	-	0	**< 1**	-	**< 1**	-
22	124	121	3	4	2035	0	5	727	0	**< 1**	11	**< 1**	**1**
24	620	604	16	8	3600	15	6	2955	2	**< 1**	26	**< 1**	**2**
26	5188	5124	64	12	3600	64	9	3600	58	**< 1**	78	**< 1**	**4**
28	4000	3970	30	19	3600	30	14	3600	30	**< 1**	166	**< 1**	**7**

The instances of the first two instance sets always used for $(H, \mathcal{S})$ the graph UD-K_5. To also test the more general problem for a general $(H, \mathcal{S})$ we created the other instance sets G1 and G2. There we use a randomly generated completely transitioned 4-regular (multi-)graph with n vertices as $(G, \mathcal{T})$ and a random completely transitioned 4-regular (multi-)graph with k vertices as $(H, \mathcal{S})$. To randomly generate a 4-regular graph with l vertices we start with l vertices and no edges and add random edges until the graph is 4-regular. Then we randomly partition the four edges incident to each vertex into two partitions of size two to define the two transitions. The instance set G1 consists of 30 instances for all combinations of $9 \leq n \leq 15$ and $5 \leq k \leq 7$, which gives us 630 instances. Finally, the instance set G2 considers larger graphs using 30 instances for all combinations with $16 \leq n \leq 30$ and $6 \leq k \leq 10$.

5.8.2 Comparing the MILP with the SAT Model

We compare the running times of four algorithms for the given instances, solving the original MILP model, the MILP model with the symmetry breaking constraints (5.84)–(5.85), which we call MILP_{sym}, the SAT model, and the SAT model with the symmetry breaking constraints (5.86)–(5.87), which we call SAT_{sym}.

Table 5.1 lists the computational results for instance set S1 for all four algorithms. The instances are grouped by the number of vertices $|V(C)|$ of the snark C used for the generation, one column per group. Column $|I|$ contains the numbers of instances, $|I_{\text{feas}}|$ the numbers of feasible instances, and $|I_{\text{inf}}|$ the numbers of infeasible instances. The time columns $t_{\text{feas}}[s]$ and $t_{\text{inf}}[s]$ list median running times of all feasible instances respectively the infeasible instances in seconds rounded to integer. Furthermore, for the MILP approaches columns I_{tl} contain the numbers of instances that could not be solved within the CPU-time limit of 3600 seconds. The best running times of the groups of feasible instances and infeasible instances are marked bold.

As we can see the SAT based approaches outperform the MILP approaches considerably and the symmetry breaking constraints improve the running times for the infeasible

Table 5.2: Computation results for instance set S2.

				MILP			MILP_{sym}			SAT		SAT_{sym}	
$\lvert V(C)\rvert$	$\lvert I\rvert$	$\lvert I_{\text{feas}}\rvert$	$\lvert I_{\text{inf}}\rvert$	$t_{\text{feas}}[s]$	$t_{\text{inf}}[s]$	$\lvert I_{\text{tl}}\rvert$	$t_{\text{feas}}[s]$	$t_{\text{inf}}[s]$	$\lvert I_{\text{tl}}\rvert$	$t_{\text{feas}}[s]$	$t_{\text{inf}}[s]$	$t_{\text{feas}}[s]$	$t_{\text{inf}}[s]$
18	98	15	83	2	194	0	1	9	0	**0.04**	0.12	**0.04**	**0.04**
20	1116	416	700	3	468	6	2	24	0	**0.05**	0.28	**0.05**	**0.06**
22	10 694	4873	5821	4	1173	892	3	74	0	**0.06**	0.78	**0.06**	**0.08**

instances, especially for the SAT approaches but also for the MILP approaches.

To further compare the four approaches we applied a Wilcoxon signed-rank test for each pair of them using a p-value of 5%. The algorithm MILP_{sym} is significantly faster than MILP for the instance groups with $|V(C)| \geq 24$, for the infeasible but also for the feasible instances. The two SAT approaches are significantly faster than both MILP approaches for all instance groups except for $|V(C)| = 18$ and the infeasible instances of $|V(C)| = 22$ since those are too few to get a significant result. In fact the SAT approaches are faster on almost all instances except a few feasible instances. For the SAT approaches the variant without the symmetry breaking constraints is significantly faster on all feasible instance groups with $|V(C)| \geq 22$ although the difference in the values is only within hundredth of seconds. On the other hand, for the infeasible instance groups with $|V(C)| \geq 24$ the approach with the symmetry breaking constraints is significantly faster.

Table 5.2 shows the computational results for instance set S2. The columns are the same as in Table 5.1. The results are similar as for instance set S1, but this time MILP_{sym} can solve all instances within the time limit. Applying the Wilcoxon signed-rank test we get that MILP_{sym} is significantly faster than MILP except for the infeasible instance group with $|V(C)| = 18$. Both SAT approaches are significantly faster than the MILP approaches for all instance groups. This time SAT is not significantly faster than SAT_{sym} on the feasible instance groups, SAT_{sym} is even significantly faster than SAT for the feasible instance group with $|V(C)| = 22$. For the infeasible instances SAT_{sym} is significantly faster.

Table 5.3 shows the computation results for the instance set G1. We group the instances by the number of vertices of the input graphs G and H. We do not distinguish between feasible and infeasible instance groups in this table, since the running time characteristics are similar for both types of instances. Columns $t[s]$ show the median running time for all instances of the instance group. Again both SAT approaches could solve all instances within one hour and outperform the MILP approaches. This time the differences between the models with symmetry breaking constraints and without are smaller, since the probability that a random graph has symmetries is small. Now the SAT approaches are on all instances faster than the MILP approaches. Between the MILP model there are only few instance groups where there is a significant difference in the running times in favor of both models. The situation between the two SAT models is similar although there are slightly more instance groups where SAT_{sym} is significantly faster.

Table 5.3: Computation results for instance set G1.

					MILP		MILP_{sym}		SAT	SAT_{sym}
$\lvert V(G)\rvert$	$\lvert V(H)\rvert$	$\lvert I\rvert$	$\lvert I_{\text{feas}}\rvert$	$\lvert I_{\text{inf}}\rvert$	$t[s]$	$\lvert I_{\text{tl}}\rvert$	$t[s]$	$\lvert I_{\text{tl}}\rvert$	$t[s]$	$t[s]$
09	5	30	15	15	106	0	91	0	$<$ **1**	$<$ **1**
09	6	30	4	26	440	1	409	0	1	$<$ **1**
09	7	30	0	30	2059	11	2735	14	$<$ **1**	$<$ **1**
10	5	30	19	11	90	1	87	1	$<$ **1**	$<$ **1**
10	6	30	4	26	1939	12	1862	10	**1**	**1**
10	7	30	0	30	3600	16	3600	16	2	**1**
11	5	30	25	5	42	1	19	0	$<$ **1**	$<$ **1**
11	6	30	9	21	2777	14	3600	16	3	**2**
11	7	30	1	29	3600	22	3600	20	**3**	**3**
12	5	30	28	2	50	1	17	0	$<$ **1**	$<$ **1**
12	6	30	21	9	2204	13	2124	11	3	**2**
12	7	30	1	29	3600	30	3600	30	8	**7**
13	5	30	28	2	23	2	26	2	$<$ **1**	$<$ **1**
13	6	30	20	10	3600	17	2055	13	**4**	**4**
13	7	30	7	23	3600	30	3600	27	14	**13**
14	5	30	30	0	24	0	30	0	$<$ **1**	$<$ **1**
14	6	30	28	2	562	7	823	8	**2**	**2**
14	7	30	8	22	3600	29	3600	27	**27**	28
15	5	30	30	0	30	0	24	0	$<$ **1**	$<$ **1**
15	6	30	29	1	670	2	1475	11	3	**2**
15	7	30	18	12	3600	26	3600	27	**27**	30

All instances in all three instance sets S1, S2, and G1 could be solved within the time limit of one hour by both SAT approaches. Clearly the MILP approach already reached the time limit for most instances in G1 and therefore we tested only the SAT approaches on the instance set G2 to analyze the limits of them. Figure 5.8 shows the median running times of SAT_{sym} for different sizes of $|G|$ and $|H|$. As we can see the running time heavily depends on the size of H and not so strongly on the size of G. For $|H| = 10$ and $|G| \geq 20$ we run into the time limit of one hour in most of the instances. Similarly, as for instance set $G1$ also in $G2$ the running times for SAT_{sym} and SAT are similar.

5.8.3 Characterizing all Snarks with Up to 32 Vertices

Using SAT_{sym} we also implemented the framework described at the end of Section 5.7. We use Boost's implementation of the Boyer-Myrvold planarity test to check for planar graphs. Furthermore, we use a simple SAT model for checking if a graph contains a K_5-minor and another SAT model for checking if it has a CCD. Since the bottleneck of this framework are the solving times for checking SUD-K_5-minor-freeness, the running

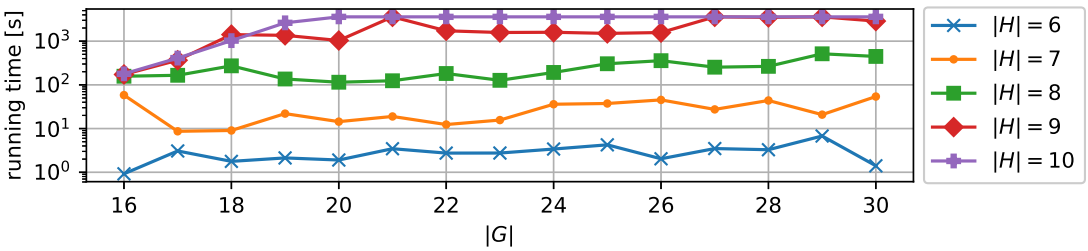


Figure 5.8: Median running times of SAT_{sym} for instance set G2.

time improvements by the SAT model compared to the MILP model are crucial to check for all snarks with up to 32 vertices if they contain a planar contraction, a K_5-minor-free contraction, a SUD-K_5-minor-free contraction, or a CCD-containing contraction of a PPM. From the 1 918 812 tested snarks we found

- 25 248 snarks that do not contain a planar contraction of a PPM,
- 19 130 snarks that do not contain a K_5-minor-free contraction of a PPM,
- 1095 snarks that do not contain a SUD-K_5-minor-free contraction of a PPM,
- 0 snarks that do not contain a contraction of a PPM that has no CCD.

The found snarks can be downloaded from https://www.ac.tuwien.ac.at/klocker-snark-collections/. Up until now it was not known if there exist snarks that do not have a PPM whose contraction leads to planar/K_5-minor-free/SUD-K_5-minor-free graphs. With our implementation we could find many examples of snarks that have those properties. Nevertheless, all tested snarks always had a PPM whose contraction leads to a graph that has a CCD. Therefore, it remains an open question if there exists a snark that does not have a PPM whose contraction leads to a CCD-containing graph.

5.9 Conclusion and Future Work

In this chapter we formulated the new problem ESTM for checking if a transitioned graph contains a sup-$(H, \mathcal{S})$ transition minor, which is a generalization of sup-K_5-minors defined by Fleischner et al. [33]. A complexity analysis of the problem showed that it is $\mathcal{NP}$-complete, even if restricted to simple graphs. Furthermore, it can be solved in polynomial time if the size of H is fixed. We also came up with an equivalent problem EBSRTM, which is used as base problem for our models.

In the next step we formulated a mathematical model, which uses simple mathematical objects such as partial functions and trees together with a set of constraints in logical

form. It does not directly model the intermediate graph, which needs to be a transition minor of the input graph $(G, \mathcal{T})$ and a sup-$(H, \mathcal{S})$ graph, but ensures with its constraints on the two input graphs the existence of such. Since it is not trivial that this model solves the problem EBSRTM we provided a thorough proof of the equivalence in Section 5.2.

From the mathematical model we derived in a more or less straightforward manner a MILP with which we can solve the problem EBSRTM in practice. Furthermore, we also derived a SAT model from the mathematical model. To improve the solving times of both models we developed symmetry breaking constraints that are based on the automorphism groups of both input graphs restricted by the additional structure given through the transition systems.

We tested both models on four different instance sets. Two of them are motivated by the CDC conjecture and are based on contractions of PPM of snarks. The other instances are randomly generated to consider the whole scope of the problem. In our computational study we could verify that the SAT approach outperforms the MILP approach significantly and the symmetry breaking constraints could improve the running times especially for proving infeasibility. Through our tests we were able to find graphs that are contractions of perfect matchings of snarks and contain a K_5-minor but no SUD-K_5-minor. Such graphs were not known before and show that the theorem about the existence of CCDs by Fleischner et al. [33] is indeed stronger than the theorem by Fan and Zhang [24].

Furthermore, using the SAT model in a framework we systematically checked PPM contractions of all snarks with up to 32 vertices and were able to find many snarks that do not have PPMs whose contraction leads to SUD-K_5-minor-free graphs, which answers a previously open problem. We also could verify that all snarks with up to 32 vertices have a PPM whose contraction leads to a graph containing a CCD. Whether or not all snarks have this property is still an open problem. If this is the case it would imply the correctness of the CDC conjecture.

In future work the presented mathematical model may be used to develop other algorithms for this problem, which may be able to solve the problem even faster. One approach could be a CP model that uses non-binary variables to represent the mappings between the two input graphs. If we want to test large graphs, also a heuristic approach for finding sup-$(H, \mathcal{S})$ minors in reasonable time would be interesting, although such an algorithm would not be able to prove that there does not exist such a transition minor.

Furthermore, the framework for finding snarks that do not contain a SUD-K_5-minor-free contraction of a PPM may be improved by adding symmetry breaking during the enumeration of the PPMs. Moreover, a desirable extension would be to effectively search for a PPM in a given snark such that the resulting contracted graph is SUD-K_5-minor-free instead of enumerating over all PPMs. Adding this additional level of searching for a PPM adds a new dimension of complexity, especially since we want to find a PPM for which our model is infeasible.

CHAPTER 6

Conclusions

In this thesis we considered different open problems in graph theory that are concerned with the existence or non-existence of graphs with certain properties. We developed algorithms using techniques from combinatorial optimization to search for such graphs. In some cases we could find intended graphs and therefore solve the open problem and in other cases we verified for all graphs with up to a certain size that no graph with the wanted properties exists. For all considered problems the application of our algorithms gave us new graph theoretic insights but also the developed algorithms themselves present a valuable tool set for similar problems and provide a foundation for researching algorithmic approaches for graph construction problems.

The first problem we considered is concerned with planar uniquely hamiltonian graphs. Bondy and Jackson conjectured that there does not exist a uniquely hamiltonian planar graph with minimum degree three (UHPG3). We showed that to disprove this conjecture it would be enough to find a uniquely hamiltonian planar graph with at most one degree two vertex. A degree two vertex can be interpreted as a fixed edge in the unique hamiltonian cycle, which allows us to search for fixed edge uniquely hamiltonian graphs (FEUHGs) that are planar and have minimum degree three.

In a first approach we reformulated the problem as an optimization problem that given a fixed number of vertices n searches for an FEUHG with n vertices that minimizes the number of crossings and the number of degree two vertices. We proposed a general variable neighborhood search (GVNS) approach to search for such graphs. The algorithm only considers solutions where all crossings appear between chords, which are the edges that are not in the hamiltonian cycle. This leads to a deterministic number of crossings given a bi-partitioning of the chords without directly modeling a drawing in the plane. The algorithm starts with a cycle graph consisting only of a hamiltonian cycle and no chords, which is a feasible solution, and uses neighborhoods to add or remove chords or to change the bi-partitioning of the chords. Whenever we add edges we need to recheck if the resulting graph is still uniquely hamiltonian, which is done with a Lin-Kerninghan

Heuristic (LKH) and with Concorde whenever we find a new global optimum. One neighborhood aims for an optimal bi-partitioning of the chords, which is an $\mathcal{NP}$-hard problem on its own and gets solved with a branch-and-bound procedure. Using the approach we could find FEUHGs with minimum degree three and two crossings for all graph sizes between 10 and 100, but we could not find any planar FEUHGs with minimum degree three. To overcome the drawback of a heuristic it would be interesting in the future to develop an exact approach for solving this problem so that we can prove that there do not exist graphs with fewer crossings. Furthermore, to guide the search for good graphs it would be interesting to develop a measure how unique a hamiltonian cycle is in the sense of how much must be changed so that another hamiltonian cycle exists.

In a second approach we reduced the problem of finding a UHPG3 to finding a planar graph with minimum degree three that contains a stable fixed edge cycle (SFE-cycle). We proposed two exact approaches to check if a graph contains an SFE-cycle, one is based on integer linear program (ILP) and the other on a cycle enumeration using a data structure for storing found cycles. Furthermore, we investigated properties that a minimum counterexample, i.e. a planar graph with minimum degree three that contains an SFE-cycle with a minimum number of vertices and edges, must satisfy. Beside other properties we proved that a minimum counterexample must be triangle-free, which reduces the search space a lot. Comparing the two approaches showed that the ILP-based approach performs better for larger graphs with 19 or more vertices and the cycle enumeration based approach is superior for smaller graphs. Using the cycle enumeration approach we could verify that no minimum counterexample with 24 or fewer vertices exists which implies that Bondy and Jackson's conjecture is true for all graphs with up to 25 vertices. To further increase this lower bound one would have to solve the bottleneck of generating suitable candidate graphs by either providing a fast algorithm to generate triangle-free 3-connected planar graphs or proving even stronger properties for the minimum counterexample leading to an easier construction of candidates.

The second research area we considered are smooth graphs, which are a special class of 4-regular hamiltonian graphs, with small independence numbers. Besides other open questions for subclasses of smooth graphs Sarvanov conjectured that every smooth graph with $n > 11$ vertices has independence number larger than or equal to $2n/7$. In this context we developed a branch-and-bound framework to search for a given $n \in \mathbb{N}$, a ratio $0 < q < 1$, and a set of additional properties $\mathcal{P}$ for a smooth graph with n vertices that satisfies $\mathcal{P}$ and has independence number smaller than qn. One important aspect of the branch-and-bound procedure is computing lower bounds for partial solutions. To this end we developed multiple lower bound criteria based on the theoretic work of Fleischner, Sabidussi, and Sarvanov. Furthermore, a fast greedy algorithm is used to compute independent sets for partial solutions. We applied our approach to four different problems and could verify Sarvanov's conjecture for all smooth graphs with up to 24 vertices. Last but not least, we could find smooth graphs with $n = 20$ and $n = 24$ vertices, where all inner cycles have length four, with independence number smaller than $5n/16$, which answers a previously open question. As we did for uniquely hamiltonian

graphs one approach to decrease the search space for this problem would be to research properties of a minimum smooth graph with independence number less than $2n/7$.

Finally, the third research area we considered are transition minors, an extension of the minor concept to transitioned graphs, which appears in the context of compatible circuit decomposition (CCD) and the cycle double cover (CDC) conjecture. Fleischner et al. proved in a previous work that a transitioned graph contains a CCD if it is sup-undecomposable K_5 (SUD-K_5)-minor-free. We showed that the problem of checking SUD-K_5-minors can be done in polynomial time and generalized the problem replacing the K_5 by a transitioned input graph as second input leading to an $\mathcal{NP}$-hard problem. To solve the generalized problem we developed a mathematical model. The main strength of the model is that it is equivalent to the existence of the wanted intermediate transition minor without modeling the minor itself. One of the main tasks of this research was proving this non-trivial equivalence.

We then used the mathematical model to formulate a mixed integer linear programming (MILP) model and a boolean satisfiability problem (SAT). Furthermore, we proposed symmetry breaking constraints for our models based on the automorphism groups of the two input graphs. Comparing the two approaches showed that the SAT approach outperforms the MILP approach and the symmetry breaking constraints further improve the running times especially for graphs with many symmetries such as the K_5. Finally, using the faster SAT approach embedded into a framework, we were able to classify all snarks with up to 32 vertices with respect to the existence of perfect pseudo-matchings (PPMs) whose contraction lead to planar/K_5-minor-free/SUD-K_5-minor-free/CCD-containing graphs. Within this classification we found snarks that have no PPM whose contraction leads to a SUD-K_5-minor-free graph, whose existence was an open question. Moreover, we could prove that all snarks with up to 32 vertices contain a PPM whose contraction leads to a graph containing a CCD. Furthermore, we found graphs that have a PPM whose contraction is SUD-K_5-minor-free but have no PPM whose contraction is K_5-minor-free, which shows that the concept of SUD-K_5-minor-free graphs is stronger than the concept of K_5-minor-free graphs. To improve the performance of our approaches it would be interesting for future work to include the search for a PPM into the model so that we can avoid enumerating all PPMs of a snark. Another approach would be to design a practically efficient polynomial algorithm that checks for SUD-K_5-minors, which can then be compared to the performance of our SAT approach. Furthermore, the computational results show that the contraction of many PPMs lead to graphs that contain a CCD but are not SUD-K_5-minor-free. This gap might motivate to search for a more general sufficient condition for the existence of CCDs.

All in all we showed in this thesis that combinatorial optimization techniques are appropriate tools for addressing challenging graph construction problems from graph theory. Although there are practical limits, and we cannot prove the nonexistence of certain graphs with algorithmic approaches, algorithms can still be useful for getting more insights and providing counterexamples. Furthermore, we saw that graph theoretic results can help to decrease the size of the search space drastically and therefore are important

to incorporate for well performing practical search algorithms.

Index

$\alpha(G)$, 12
adjacent, 8
articulation point, 10
$\mathrm{Aut}(G, \mathcal{T})$, 135
$\mathrm{Aut}(G)$, 13
automorphism, 13, 135
$\mathrm{Aut}_S(H, \mathcal{S})$, 137

bad-cut-vertex, 102
basic variable neighborhood search, 37
basic-sup-$(H, \mathcal{S})$, 107
basic-sup-$(H, \mathcal{S})$-reduced-minor-free, 108
best improvment, 35
bipartite graph, 8
boolean satisfiability problem, 21
branch-and-bound, 22
branch-and-cut, 30
bridge, 10
bridgeless, 10
BVNS, 37

CDC, 97
CDC conjecture, 97
certificate, 19
certificate-checking algorithm, 19
chords, 45
chromatic index, 12
chromatic number, 12
circuit, 9
clause, 21
claw, 13
closed walk, 9
CNF, 21
co-$\mathcal{NP}$, 20
combinatorial optimization problem, 18
complement of decision problems, 20
complete bipartite graph, 8
complete graph, 8
completely transitioned graph, 101
conditioning, 31
conjunctive normal form, 21
connected graph, 10
connectivity, 10
COP, 18
cost function, 25
crossing, 13
crossing number, 14
cubic graph, 8
curve, 13
cut, 30
cut vertex, 10
cutting plane method, 30
cycle, 9
cycle double cover, 97
cycle vertex, 65
cyclical edge cut, 10
cyclically k-edge connected, 10

$\Delta(G)$, 8
$\delta(G)$, 8
Davis–Putnam–Logemann–Loveland, 31
decision problem, 16
degree, 8
diversification, 33
dominating cycle, 11
dominating FE-cycle, 57
DPLL, 31
drawing, 13

dual bound, 29
dual graph, 14, 15

$E(v)$, 8
$E[X, Y]$, 10
EBSRTM, 108
edge chromatic number, 12
edge connectivity, 10
edge cut, 10
edge deletion, 9
end vertex, 7
end vertices of a path, 9
enumeration, 22
essential edge cut, 10
essentially k-edge connected, 10, 70
ESSI$(q, \mathcal{P})$, 85
ESTM, 104
eulerian graph, 11
eulerian tour, 11
even, 11
exhaustive search, 22
existence of basic sup-reduced-transition minors, 108
existence of sup-transition minors, 104
extreme point, 26

$F(G)$, 14
face, 14
factor, 11
FE-cycle, 42
feasible region, 25
feasible solution, 25
FEUHG, 42
first improvement, 35
fixed edge cycle, 42
fixed edge uniquely hamiltonian graph, 42
fixed large vertex, 65
formulation, 29

$G[Y]$, 9
gap sequence, 92
general variable neighborhood search, 38
globally optimal solution, 18
graph, 7
greedy construction heuristic, 34
GVNS, 38

hamiltonian cycle, 11
hamiltonian cycle problem, 17
hamiltonian graph, 11
homomorphism, 12, 135
hybrid metaheuristic, 39

ILP, 27
incident, 8
independence number, 12
independent set, 12
induced subgraph, 9
inner cycles, 84
inner face, 14
instance of an optimization problem, 17
integer linear program, 27
intensification, 33
isomorphic graph, 13
isomorphism, 13, 135

$\kappa(G)$, 10
k-colorable, 12
k-coloring, 12
k-connected, 10
k-edge-colorable, 12
k-edge-coloring, 12
k-edge-connected, 10
k-factor, 11
k-opt neighborhood structure, 35
k-regular, 8
k-vertex-coloring, 12
k-vertex-connected, 10
Karp reduction, 20

$\lambda(G)$, 10
large neighborhood search, 38
large vertex, 65
lazy constraints, 30
length of walk, 9
line graph, 98
linear program, 24
linear programming relaxation, 28

literal, 21
LNS, 38
local optimum, 26, 36, 37
local search, 35
LP, 24

many-one reduction, 20
matching, 13
maximal FE-cycle, 57
maximization problem, 18
maximum degree, 8
maximum pseudo-matching, 67
metaheuristic, 33
MILP, 27
minimization problem, 18
minimum degree, 8
minor, 102
mixed integer linear program, 27
move, 35
multiple edges, 8

$\nu(G)$, 14
$N(v)$, 8
nearest neighbor heuristic, 34
neighborhood, 35
neighborhood structure, 35
$\mathcal{NP}$, 19
$\mathcal{NP}$-complete, 20
$\mathcal{NP}$-hard, 20
number of faces, 15

objective function, 25
optimization problem, 17
$\mathrm{orb}(v)$, 136
orbit, 136
$\mathrm{orb}_S(v)$, 137
order, 8
outer face, 14
outer vertex, 65

$\mathcal{P}$, 18
$\mathcal{P}_2(X)$, 7
path, 9
perfect pseudo-matching, 13
planar embedding, 14
planar graph, 14
plane graph, 14
polyhedron, 25
polynomial reduction, 20
polynomial transformation, 20
PPM, 13
pseudo-matching, 13

random improvement, 35
really large vertex, 65
reduced linear program (LP), 25
reduced transition minor, 102
reduced variable neighborhood search, 37
region, 14
relaxation, 28
resolvent, 31
resolving, 31
RVNS, 37

SAT, 21
satisfiable, 21
satisfied clause, 21
search problem, 17
search strategy, 23
separating subgraphs, 101
separation problem, 30
separator, 101
SFE-cycle, 57
SFE-graph, 57
shaking, 37
simple graph, 8
Simplex method, 25
small vertex, 65
smooth graph, 84
snark, 97
spanning, 11
stabilizer, 137
stabilizer orbits, 137
stable cycle, 78
stable fixed edge cycle, 57
stable fixed edge graph, 57
standard form, 26
subgraph, 9
SUD-K_5, 103

SUD-K_5-minor-free graph, 103
sup-$(H, \mathcal{S})$, 103
sup-$(H, \mathcal{S})$-minor-free, 103
sup-undecomposable K_5, 103
symmetry breaking, 22

t-separator, 101
total enumeration, 22
tour, 11
trail, 9
transition minor, 102
transition system, 101
transitioned graph, 101
transitions, 101
traveling salesperson problem, 18
truth assignment, 21
TSP, 18

UD-K_5, 103
UHPG3, 44
undecomposable K_5, 103
uniquely hamiltonian, 41
uniquely hamiltonian fixed edge cycle, 42
uniquely hamiltonian planar graph with minimum degree three, 44
unit resolution, 31

valid inequality, 30
variable neighborhood descent, 36
vertex cut, 10
vertex deletion, 9
vertex-connectivity, 10
very large scale neighborhood search, 38
very small vertex, 67
VLSN, 38
VND, 36

walk, 9

$\chi(G)$, 12
$\chi'(G)$, 12

Acronyms

BVNS basic variable neighborhood search.

CCD compatible circuit decomposition.

CDC cycle double cover.

CDCL conflict driven clause learning.

CNF conjunctive normal form.

COP combinatorial optimization problem.

CP constraint programming.

DP dynamic programming.

DPLL Davis–Putnam–Logemann–Loveland.

EBSRTM existence of basic sup-reduced-transition minors.

ESTM existence of sup-transition minors.

FE-cycle fixed edge cycle.

FEUHG fixed edge uniquely hamiltonian graph.

GVNS general variable neighborhood search.

ILP integer linear program.

LKH Lin-Kerninghan Heuristic.

LNS large neighborhood search.

LP linear program.

MAX-SAT maximum satisfiability problem.

MILP mixed integer linear program.

MTZ Miller-Tucker-Zemlin.

PPM perfect pseudo-matching.

RVNS reduced variable neighborhood search.

SAT boolean satisfiability problem.

SFE-cycle stable fixed edge cycle.

SFE-graph stable fixed edge graph.

SUD-K_5 sup-undecomposable K_5.

TSP traveling salesperson problem.

UD-K_5 undecomposable K_5.

UHPG3 uniquely hamiltonian planar graph with minimum degree three.

VLSN very large scale neighborhood search.

VND variable neighborhood descent.

Bibliography

[1] S. Abbasi and A. Jamshed. A Degree Constraint for Uniquely Hamiltonian Graphs. *Graphs and Combinatorics*, 22(4):433–442, 2006.

[2] F. A. Aloul, A. Ramani, I. L. Markov, and K. A. Sakallah. Solving Difficult SAT Instances in the Presence of Symmetry. In *Proceedings of the 39th Annual Design Automation Conference*, DAC '02, pages 731–736, New York, NY, USA, 2002. ACM.

[3] D. L. Applegate, R. E. Bixby, V. Chvatal, and W. J. Cook. *The traveling salesman problem: a computational study*. Princeton University Press, 2006.

[4] D. Bertsimas and J. N. Tsitsiklis. *Introduction to linear optimization*, volume 6. Athena Scientific Belmont, MA, 1997.

[5] S. Bevc and I. Savnik. Using tries for subset and superset queries. In *Proceedings of the ITI 2009*, pages 147–152, 2009.

[6] A. Biere, M. Heule, and H. van Maaren. *Handbook of satisfiability*, volume 185. IOS press, 2009.

[7] C. Blum and G. R. Raidl. *Hybrid Metaheuristics: Powerful Tools for Optimization*. Artificial Intelligence: Foundations, Theory, and Algorithms. Springer International Publishing, 2016.

[8] C. Blum, J. Puchinger, G. R. Raidl, and A. Roli. Hybrid metaheuristics in combinatorial optimization: A survey. *Applied Soft Computing*, 11(6):4135–4151, 2011.

[9] J. A. Bondy. Basic graph theory: paths and circuits. *Handbook of combinatorics*, 1: 3–110, 1995.

[10] J. A. Bondy and B. Jackson. Vertices of Small Degree in Uniquely Hamiltonian Graphs. *Journal of Combinatorial Theory, Series B*, 74(2):265–275, 1998.

[11] J. A. Bondy and M. R. Murty. *Graph Theory*. Graduate Texts in Mathematics. Springer-Verlag, London, 2008.

[12] G. Brinkmann, B. D. McKay, and others. Fast generation of planar graphs. *MATCH Communications in Mathematical and in Computer Chemistry*, 58(2):323–357, 2007.

[13] G. Brinkmann, K. Coolsaet, J. Goedgebeur, and H. Mélot. House of Graphs: a Database of Interesting Graphs. *Discrete Applied Mathematics*, 161:311–314, 2013. Available at http://hog.grinvin.org.

[14] G. Brinkmann, J. Goedgebeur, J. Hägglund, and K. Markström. Generation and properties of snarks. *Journal of Combinatorial Theory, Series B*, 103(4):468–488, 2013.

[15] R. L. Brooks. On colouring the nodes of a network. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 37, pages 194–197, 1941.

[16] G. Caporossi and P. Hansen. Variable neighborhood search for extremal graphs: 1 The AutoGraphiX system. *Discrete Mathematics*, 212(1–2):29–44, 2000.

[17] M. Charikar, P. Indyk, and R. Panigrahy. New algorithms for subset query, partial match, orthogonal range searching, and related problems. In *Proc. of ICALP*, volume 2380 of *LNCS*, pages 451–462. Springer, 2002.

[18] G. Chartrand and P. Zhang. *A first course in graph theory*. Courier Corporation, 2013.

[19] S. A. Cook. The Complexity of Theorem-proving Procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC '71, pages 151–158, Shaker Heights, Ohio, USA, 1971. ACM.

[20] S. B. Cooper. *Computability theory*. Chapman and Hall/CRC, 2017.

[21] G. Dantzig, R. Fulkerson, and S. Johnson. Solution of a Large-Scale Traveling-Salesman Problem. *Journal of the Operations Research Society of America*, 2(4): 393–410, 1954.

[22] R. Diestel. *Graph Theory*. Graduate Texts in Mathematics. Springer-Verlag, Berlin Heidelberg, 5th edition, 2017.

[23] R. Entringer and H. Swart. Spanning cycles of nearly cubic graphs. *Journal of Combinatorial Theory, Series B*, 29(3):303–309, 1980.

[24] G. Fan and C.-Q. Zhang. Circuit Decompositions of Eulerian Graphs. *Journal of Combinatorial Theory, Series B*, 78(1):1–23, 2000.

[25] H. Fleischner. Eulersche Linien und Kreisüberdeckungen, die vorgegebene Durchgänge in den Kanten vermeiden. *Journal of Combinatorial Theory, Series B*, 29(2):145–167, 1980.

[26] H. Fleischner. Uniqueness of maximal dominating cycles in 3-regular graphs and of hamiltonian cycles in 4-regular graphs. *Journal of Graph Theory*, 18(5):449–459, 1994.

[27] H. Fleischner. Uniquely hamiltonian graphs of minimum degree 4. *Journal of Graph Theory*, 75(2):167–177, 2014.

[28] H. Fleischner and B. Jackson. A Note Concerning some Conjectures on Cyclically 4–Edge Connected 3–Regular Graphs. In *Annals of Discrete Mathematics*, volume 41 of *Graph Theory in Memory of G.A. Dirac*, pages 171–177. Elsevier, 1988.

[29] H. Fleischner and V. I. Sarvanov. Small maximum independent sets in Hamiltonian four-regular graphs. *Reports of the National Academy of Sciences of Belarus*, 57(1): 10, 2013.

[30] H. Fleischner and M. Stiebitz. A solution to a colouring problem of P. Erdős. *Discrete Mathematics*, 101(1–3):39–48, 1992.

[31] H. Fleischner, G. Sabidussi, and V. I. Sarvanov. Maximum independent sets in 3-and 4-regular Hamiltonian graphs. *Discrete Math.*, 310(20):2742–2749, 2010.

[32] H. Fleischner, B. Bagheri Gh., and B. Klocker. Perfect pseudo-matchings in cubic graphs. Technical report, Algorithms and Complexity Group, TU Wien, 2019. Submitted to ARS COMBINATORIA.

[33] H. Fleischner, B. Bagheri Gh., C.-Q. Zhang, and Z. Zhang. Cycle covers (III) – Compatible circuit decomposition and K5-transition minor. *Journal of Combinatorial Theory, Series B*, 137:25–54, 2019.

[34] M. Garey and D. Johnson. Crossing Number is NP-Complete. *SIAM Journal on Algebraic Discrete Methods*, 4(3):312–316, 1983.

[35] M. Gendreau and J.-Y. Potvin. *Handbook of metaheuristics*, volume 2. Springer, 2010.

[36] J. Goedgebeur, B. Meersman, and C. T. Zamfirescu. Graphs with few hamiltonian cycles. *Mathematics of Computation*, 2019. To appear.

[37] R. J. Gould. Advances on the Hamiltonian problem–a survey. *Graphs and Combinatorics*, 19(1):7–52, 2003.

[38] J. R. Griggs. Lower bounds on the independence number in terms of the degrees. *Journal of Combinatorial Theory, Series B*, 34(1):22–39, 1983.

[39] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*, volume 32. Prentice-Hall, 1982.

[40] M. Halldórsson and J. Radhakrishnan. Greed is good: Approximating independent sets in sparse and bounded-degree graphs. *Algorithmica*, 18(1):145–163, 1997.

[41] P. Hansen and N. Mladenović. An introduction to variable neighborhood search. In *Metaheuristics, Advances and Trends in Local Search Paradigms for Optimization*, pages 433–458. Kluwer, 1999.

[42] P. Hansen and N. Mladenović. Variable Neighborhood Search. In *Handbook of Metaheuristics*, International Series in Operations Research & Management Science, pages 145–184. Springer US, Boston, MA, 2003.

[43] S. Helmer, R. Aly, T. Neumann, and G. Moerkotte. Indexing set-valued attributes with a multi-level extendible hashing scheme. In *Database and Expert Systems Applications*, pages 98–108. Springer, 2007.

[44] K. Helsgaun. Effective implementation of the Lin-Kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126(1):106–130, 2000.

[45] F. Jaeger. A survey of the cycle double cover conjecture. In *Annals of Discrete Mathematics (27): Cycles in Graphs*, volume 115 of *North-Holland Mathematics Studies*, pages 1–12. North-Holland, 1985.

[46] M. Janota, R. Grigore, and V. Manquinho. On the Quest for an Acyclic Graph. *arXiv:1708.01745 [cs]*, 2017.

[47] T. Januschowski and M. E. Pfetsch. Branch-Cut-and-Propagate for the Maximum k-Colorable Subgraph Problem with Symmetry. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 6697 of *LNCS*, pages 99–116. Springer Berlin Heidelberg, 2011.

[48] T. S. Jayram, S. Khot, R. Kumar, and Y. Rabani. Cell-probe Lower Bounds for the Partial Match Problem. In *Proceedings of the Thirty-fifth Annual ACM Symposium on Theory of Computing*, STOC '03, pages 667–672, New York, USA, 2003. ACM.

[49] K. F. Jones. Independence in graphs with maximum degree four. *Journal of Combinatorial Theory, Series B*, 37(3):254–269, 1984.

[50] B. Klocker. Heuristic approaches for finding uniquely hamiltonian graphs of minimum degree three with small crossing numbers. Austrian Workshop on Metaheuristics 11, Graz, Austria, 2016.

[51] B. Klocker. Searching uniquely hamiltonian planar graphs with minimum degree three. Graph Theory Workshop on How to Span a Graph, Bucharest, Romania, 2018.

[52] B. Klocker and G. R. Raidl. Solving a weighted set covering problem for improving algorithms for cutting stock problems with setup costs by solution merging. In *Computer Aided Systems Theory – EUROCAST 2017, Part I*, volume 10671 of *LNCS*, pages 355–363, Gran Canaria, Spain, 2018. Springer.

[53] B. Klocker, H. Fleischner, and G. R. Raidl. Finding uniquely hamiltonian graphs of minimum degree three with small crossing numbers. In *Hybrid Metaheuristics: 10th International Workshop, HM 2016*, volume 9668 of *LNCS*, pages 1–16. Springer, 2016.

[54] B. Klocker, H. Fleischner, and G. R. Raidl. A Model for Finding Transition-Minors. Technical Report AC-TR-18-009, Algorithms and Complexity Group, TU Wien, 2018. Submitted to Discrete Applied Mathematics.

[55] B. Klocker, H. Fleischner, and G. R. Raidl. Finding smooth graphs with small independence numbers. In *MOD 2017: Machine Learning, Optimization, and Big Data – Third International Conference*, volume 10710 of *LNCS*, pages 527–539. Springer, 2018.

[56] B. Klocker, H. Fleischner, and G. Raidl. A SAT Approach for Finding Sup-Transition-Minors. In *Learning and Intelligent Optimization*, LNCS. Springer, 2019. To appear.

[57] B. Klocker, H. Fleischner, and G. R. Raidl. A lower bound for the smallest uniquely hamiltonian planar graph with minimum degree three. Technical Report AC-TR-19-007, Algorithms and Complexity Group, TU Wien, 2019. Submitted to Applied Mathematics and Computation.

[58] B. Korte and J. Vygen. *Combinatorial optimization: Theory and algorithms*. Springer, Berlin, Heidelberg, 3 edition, 2008.

[59] F. R. Leder. Uniquely hamiltonian graphs. Master's thesis, TU Wien, 2012.

[60] E. M. Luks. Isomorphism of graphs of bounded valence can be tested in polynomial time. *Journal of Computer and System Sciences*, 25(1):42–65, 1982.

[61] B. D. McKay and A. Piperno. Practical graph isomorphism, II. *Journal of Symbolic Computation*, 60:94 – 112, 2014.

[62] C. E. Miller, A. W. Tucker, and R. A. Zemlin. Integer programming formulation of traveling salesman problems. *Journal of the ACM (JACM)*, 7(4):326–329, 1960.

[63] D. Pisinger and S. Ropke. Large Neighborhood Search. In *Handbook of Metaheuristics*, International Series in Operations Research & Management Science, pages 399–419. Springer US, Boston, MA, 2010.

[64] G. R. Raidl, J. Puchinger, and C. Blum. Metaheuristic Hybrids. In *Handbook of Metaheuristics*, International Series in Operations Research & Management Science, pages 469–496. Springer US, Boston, MA, 2010.

[65] B. Reed and Z. Li. Optimization and Recognition for K5-minor Free Graphs in Linear Time. In *LATIN 2008: Theoretical Informatics*, volume 4957, pages 206–215. Springer Berlin Heidelberg, 2008.

[66] G. Reinelt. *The traveling salesman: computational solutions for TSP applications*. Springer-Verlag, 1994.

[67] N. Robertson and P. D. Seymour. Graph minors. XIII. The disjoint paths problem. *Journal of combinatorial theory, Series B*, 63(1):65–110, 1995.

[68] K. H. Rosen. *Discrete mathematics and its applications*. McGraw-Hill, 2019.

[69] V. I. Sarvanov. Institute of Mathematics at the National Academy of Sciences of Belarus. Personal communication, 2016.

[70] A. Schrijver. *Theory of linear and integer programming*. John Wiley & Sons, 1998.

[71] P. D. Seymour. Sums of circuits. *Graph theory and related topics*, 1:341–355, 1979.

[72] I. E. Shanthi, Y. Izaaz, and R. Nadarajan. On the SD-tree Construction for Optimal Signature Operations. In *Proceedings of the 1st Bangalore Annual Compute Conference*, COMPUTE '08, pages 14:1–14:8, New York, NY, USA, 2008. ACM.

[73] J. B. Shearer. A note on the independence number of triangle-free graphs. *Discrete Mathematics*, 46(1):83–87, 1983.

[74] C. C. Sims. Computational methods in the study of permutation groups. In *Computational Problems in Abstract Algebra*, pages 169–183. Pergamon, 1970.

[75] G. Szekeres. Polyhedral decompositions of cubic graphs. *Bulletin of the Australian Mathematical Society*, 8(03):367, 1973.

[76] K. Sörensen and F. W. Glover. Metaheuristics. *Encyclopedia of operations research and management science*, pages 960–970, 2013.

[77] A. G. Thomason. Hamiltonian cycles and uniquely edge colourable graphs. *Advances in graph theory*, 3, 1978.

[78] G. S. Tseitin. On the Complexity of Derivation in Propositional Calculus. In *Automation of Reasoning: 2: Classical Papers on Computational Logic 1967–1970*, Symbolic Computation, pages 466–483. Springer, Berlin, Heidelberg, 1983.

[79] W. T. Tutte. On hamiltonian circuits. *Journal of the London Mathematical Society*, s1-21(2):98–101, 1946.

[80] D. B. West. *Introduction to graph theory*, volume 2. Prentice hall Upper Saddle River, NJ, 1996.

[81] L. A. Wolsey. *Integer programming*. Wiley, 1998.

Curriculum Vitae

Personal

Name	Benedikt Klocker
Address	Neustiftgasse 31/27, 1070 Vienna, Austria
E-Mail	benedikt.klocker@yahoo.com
D.O.B.	February 28, 1990
Birthplace	Lustenau
Nationality	Austria

Education

since 2015 **PhD in Computer Science**, *TU Wien*.
Thesis: "Combinatorial Optimization Approaches for Graph Construction Problems"
Supervisor: Günther Raidl

2012–2019 **Master of Science**, *TU Wien*.
Master program: Technical Mathematics
Thesis: "Solving a Weighted Set Covering Problem for Improving Algorithms for Cutting Stock Problems with Setup Costs by Solution Merging"
Supervisor: Günther Raidl

2012–2015 **Master of Science**, *TU Wien*.
Master program: Computational Intelligence
Thesis: "Optimization Approaches for Recreational Bicycle Tour Planning"
Supervisor: Günther Raidl

2014 **Exchange Semester**, *University of Illinois*, Urbana–Champaign, USA.
As part of the master program Technical Mathematics

2009–2012 **Bachelor of Science**, *TU Wien*.
Bachelor program: Mathematics in Computer Science
Thesis: "Abstrakte harmonische Analysis auf lokalkompakten abelschen Gruppen"
Supervisor: Martin Blümlinger

2004–2008 **Secondary Education Second Stage**, *BORG Dornbirn Schoren*.

2000–2004 **Secondary Education First Stage**, *Bundesgymnasium Dornbirn*.

Work Experience

2016–2019 **Lecturer in Heuristic Optimization Techniques**, *TU Wien*.

2015–2019 **FWF-Project Assistant**, *TU Wien*.
Project Assistant for the project "Cycles on Graphs and Properties of Graphs with Special Cycle Structure" (FWF P27615) lead by Herbert Fleischner
2016–2019: Project assistant for an additional project in the field of packing and cutting in cooperation with Lodestar GmbH

2012 – 2015 **Tutor in Algorithms and Datastructures**, *TU Wien*.

2014 **Tutor in Functional Programming**, *TU Wien*.

2012 – 2014 **Tutor in Algebra and Discrete Mathematics**, *TU Wien*.

2012 – 2013 **Tutor in Introduction to Programming**, *TU Wien*.

2012, 2013 **Summer Intern**, *V-Research GmbH*, Dornbirn.

2008-2009 **Compulsory Civilian Service**, *Rotes Kreuz - Landesverband Vorarlberg*, Bregenz.

2008 **Summer Intern**, *OMICRON electronics GmbH*, Klaus.

Publications

2019 Fleischner, Herbert, Behrooz Bagheri Gh., and Benedikt Klocker (2019). *Perfect Pseudo-Matchings in cubic graphs*. Tech. rep. Submitted to ARS COMBINATORIA. Algorithms and Complexity Group, TU Wien.

- Klocker, B., H. Fleischner, and G. R. Raidl (2019). *A Lower Bound for the Smallest Uniquely Hamiltonian Planar Graph with Minimum Degree Three*. Tech. rep. AC-TR-19-007. Submitted to Applied Mathematics and Computation. Algorithms and Complexity Group, TU Wien.
- Klocker, Benedikt, Herbert Fleischner, and Günther Raidl (2019). "A SAT Approach for Finding Sup-Transition-Minors". In: *Learning and Intelligent Optimization*. LNCS. To appear. Springer.

2018 Klocker, Benedikt, Herbert Fleischner, and Günther R. Raidl (2018a). *A Model for Finding Transition-Minors*. Tech. rep. AC-TR-18-009. Submitted to Discrete Applied Mathematics. Algorithms and Complexity Group, TU Wien.

- Klocker, Benedikt, Herbert Fleischner, and Günther R. Raidl (2018b). "Finding Smooth Graphs with Small Independence Numbers". In: *MOD 2017: Machine Learning, Optimization, and Big Data – Third International Conference*. Vol. 10710. LNCS. Springer, pp. 527–539.
- Klocker, Benedikt and Günther R Raidl (2018). "Solving a Weighted Set Covering Problem for Improving Algorithms for Cutting Stock Problems with Setup Costs by Solution Merging". In: *Computer Aided Systems Theory – EUROCAST 2017, Part I*. Vol. 10671. LNCS. Gran Canaria, Spain: Springer, pp. 355–363.

2016 Klocker, B., H. Fleischner, and G. R. Raidl (2016). "Finding Uniquely Hamiltonian Graphs of Minimum Degree Three with Small Crossing Numbers". In: *Hybrid Metaheuristics: 10th International Workshop, HM 2016*. Vol. 9668. LNCS. Springer, pp. 1–16.