Branch-and-cut algorithms for graph problems

Branch-and-cut algorithms for graph problems

Birol Yüceoğlu

This book was typeset by the author using LaTeX.

ISBN: 978-605-9092-11-1
Published by Dahi Yayıncılık Eğitim Basım Ticaret (Certificate no: 26015)
Ramazan Güven
Esatpaşa Mah. Şehit Cahar Dudayev Cad. No:159/6-4 Ataşehir, İstanbul, Turkey
Printed in Ege Reklam Basım Sanatları San. Tic. Ltd. Şti.
(Certificate No: 12468)
Esatpaşa Mah. Ziyapaşa Cad. No:4 Ataşehir, İstanbul, Turkey

Branch-and-cut algorithms for graph problems

PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Universiteit Maastricht,
op gezag van Rector Magnificus,
Prof. dr. L.L.G. Soete,
volgens het besluit van het College van Decanen,
in het openbaar te verdedigen
op donderdag 26 februari 2015, om 10:00 uur

door

Birol Yüceoğlu

Promotor:

Prof. dr. ir. C.P.M. van Hoesel

Co-Promotor:

Dr. A. Grigoriev

Beoordelingscommissie:

Prof. dr. R.J. Müller (voorzitter)

Dr. A. Berger

Dr. J.B.G. Frenk (Sabancı University)

Prof. dr. A.P.M. Wagelmans (Erasmus University Rotterdam)

Dit onderzoek werd financieel mogelijk gemaakt door de Graduate School of Business and Economics (GSBE).

To my family...

Acknowledgements

Reaching the end of the line is a nice feeling in case of a Ph.D. but it also means it is the time to thank people, which is always difficult. In my time in Maastricht, I felt it was a second home to me and being so far from it still feels strange. This acknowledgment is also a proper good bye to my life in Maastricht.

I would like to thank my colleagues and friends from Maastricht University. Especially Karin van der Boorn, Hayde Hallmans, who were always there when I had a question and my officemates during my stay, Elnaz Bajoori, Jiwoong Lee, Hans Ensinck, Greg Czapó, and Martijn van Brink.

I would like to thank my promoter Stan van Hoesel for showing me how to do things. Apart from his professional support, I also would like to thank him for his personal support and for always advising me to do what I enjoy. In the stressful moments, his help was the difference between resigning and carrying on. I would also like to thank Rudolf Müller, head of reading committee, for his help and sympathy. I also thank the other members of the reading committee, André Berger, Hans Frenk, and Albert Wagelmans for their invaluable comments. I am especially grateful to André Berger for his comments and for helping me when I had a question. I thank Alexander Grigoriev for helping me with some results on Chapter 5 of this thesis.

I had the chance to meet great people during my stay in Maastricht. Çiğdem Akbulut, İrem Bozbay, Hande Erkut, Seher Fazlıoğlu, Mehmet İsmail, Hande Karabıyık, Bahar Kaynar, Emel Öztürk, Murat and Demet Öztürk made my stay in the Netherlands very special with their presence.

I thank Laura Brouwers for a great friendship and for all the wonderful dinners we had together.

Duygu Taş made my life in the Netherlands much better with her presence. I miss our times in Maastricht and Eindhoven but I know better times awaits us in İstanbul.

Umut Deniz Özuğurel was with me in every step of this thesis. Our similar experiences brought me strength in my difficult times.

I thank Özge Gökdemir and Devrim Dumludağ for the joy they brought to my life and all the things we did together. I am glad to have had the chance to meet them.

When I was in Maastricht my family grew all the time. Erkan, Ceren and little Ece Yönder have been a family to me and I cannot thank them enough for accepting me in their lives. I will never forget wonderful times we had together in Maastricht.

Ingrid Rohde and Burak Can have been a family for me in Maastricht, a family that grew with little Noah İdris. I thank them, Kirsten Rohde and their family for letting me be a part of them. Cooking chicken, killing zombies, our coffee and lunch breaks I still miss.

I would also like to thank Seda İrem Çakırca, who shared with me the last and the most important steps of this thesis. The words are not enough to describe her support, her understanding and the way she could make me feel during those times. In fact, the words are not enough to describe anything related to her.

Finally I would like to thank my mother, my father, my sister and my grandmothers. Their unconditional support throughout my life encouraged me to be myself even though it is hard for them to understand my choices. Without such a great family I would never be able to to go for such an adventure. Without such a great family I would never be able to finish this adventure. Without such a great family nothing would not taste this sweet. For this reason, this thesis is dedicated to them.

Birol Yüceoğlu
İstanbul, January 2015

Contents

Chapter 1

Introduction

Operations research uses mathematical techniques to help with decision making. These mathematical tools include mathematical modeling, algorithmic tools, simulation, statistics, and decision analysis. An element that is shared by all these tools is that they all model real life processes. Even though these models may not completely reflect reality and complexity of real life decision making, they have proved to be useful in different fields. As a result of that, operations research is commonly used in different sectors, such as airlines, railways, manufacturing, inventory management, transportation and logistics, facility location and layout, and telecommunications. In airlines and railways, problems such as crew scheduling and fleet assignment are extensively studied. Scheduling of jobs, managing the inventory, and deciding on allocation of resources are among the problems that have an important place in manufacturing. In transportation and logistics, the vehicle routing problem and its variants are widely studied. In facility location and layout problems, the positioning of the production facilities in relation to other facilities (such as customers) and optimal layouts for a better flow of materials are considered. In telecommunications, assignment of capacities, routing of the network traffic, and survivability of the networks are popular topics.

Problems in the aforementioned fields can be formulated and solved using special representations, such as graphs, and mathematical tools, such as linear and integer programming. The first problem we study is minimum triangulation of graphs, which is used in Gaussian matrix elimination, probabilistic networks, and computer vision. The second problem is graph coloring, which is frequently used in scheduling, timetabling, and telecommunication networks, in particular frequency

assignment. The third problem considers determining safe dike heights in the Netherlands by balancing the cost of safety against the benefit of security from the floods. The three problems discussed in this thesis are combinatorial optimization problems. Furthermore, they all have an underlying graph structure and they are solved using the same solution method, branch-and-cut. For this reason, we start by introducing these three fields.

1.1 Combinatorial optimization

Combinatorial optimization deals with searching for the best solution to a problem among a finite (or countably infinite) set of solutions. The problem defining a set of solutions has a short description, such as an (integer) linear programming formulation or a graph representation, but the number of solutions in the set can be exponential in the size of the description. For that reason, efficient methods to search for the optimum solution are used instead of exhaustive search techniques.

To illustrate the terms we use in the sequel, we consider two problems: the *shortest path problem (SPP)* and the *traveling salesman problem (TSP)*. Given a set of cities, whose distances to each other are known, SPP aims at finding the shortest path between two cities and TSP aims at finding the shortest route that visits each city. Even though, describing these problems takes only a sentence, the set of solutions (which corresponds to all paths between two cities and all possible tours visiting each city) contains an exponential number of elements.

For SPP, we can enumerate all paths with fixed starting city and fixed ending city. If the number of intermediate cities is k then there are $\frac{(n-2)!}{k!}$ possible paths. In total, we thus have $\sum_{k=0}^{n-2} \frac{(n-2)!}{k!}$ possibilities which is larger than 2^{n-2}. Similarly, in a TSP with n cities, there are $\frac{(n-1)!}{2}$ possible solutions. We select a city where the tour starts, which leaves us with $n-1$ possibilities for the second city, and $n-2$ possibilities for the third, and so on. In case, the distances are symmetric this corresponds to $\frac{(n-1)!}{2}$ possible solutions (each tour can be traversed in two directions). In both cases there is an exponential number of feasible solutions. Therefore, straightforward enumeration works only for very small numbers of cities. Perhaps surprisingly, we know efficient (fast) methods to solve the SPP, but we do not know whether there exist efficient methods to solve the TSP.

1.2 Graphs

A graph is a mathematical representation of a set of objects V, and connections between them, E. The objects are called vertices and the connections are represented by links. If the connection between two objects i, j is symmetric it is called an edge and denoted by $\{i, j\}$. An edge that has an ordering is called an arc and is denoted by (i, j). In a graph where the vertices are the cities in a country, the euclidean distances among cities can be represented by edges. However, if the distances of road segments between two cities differ in opposite directions, the graph has to be directed and the relationship is represented by arcs.

We illustrate the modelling power of graphs by using the Knight's Move Puzzle. In a 3×3 chessboard we place 2 black and 2 white knights as shown in Figure 1.1. We want to switch the location of the black and white knights by making a minimum number of moves. The problem on a chessboard may not look trivial to solve. In Figure 1.2(a), the problem is represented on a graph. The nodes of the graph are the squares of the chessboard and the edges correspond to the moves allowed for a knight on that square. Even in that figure, the problem does not look trivial to solve. By redrawing the graph and deleting the node $2b$, we obtain the graph in Figure 1.2(b). It is immediately clear in this picture that, the puzzle requires 16 moves. All knights have to make 4 clockwise (or anticlockwise) moves to obtain the configuration with the desired property, which in total makes 16 moves. This example shows that with an appropriate representation it is possible to obtain easy solutions for problems that look difficult.

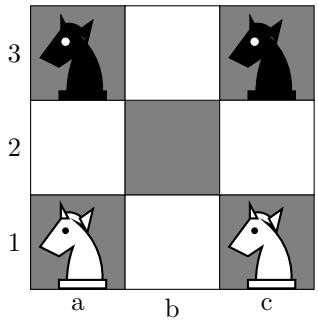
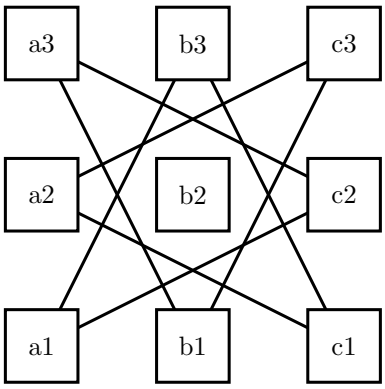
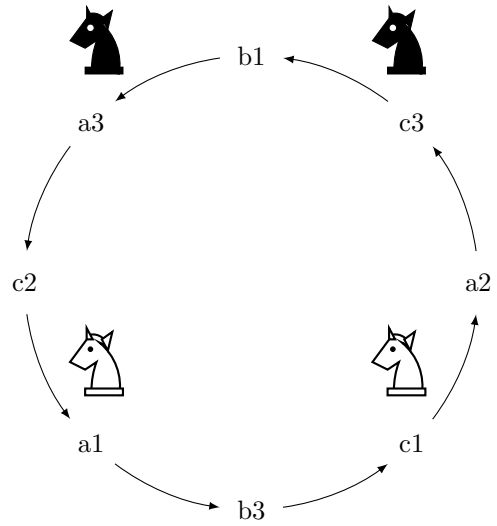


Figure 1.1: Knight's Move Puzzle on a 3×3 chessboard.

The Icosian game is the first application of the TSP and can also be used to illustrate graphs. It was invented in 1857 by William Rowan Hamilton. The aim



(a) Representation of the chessboard with a graph. Nodes are the squares of the chessboard and edges correspond to the moves the knights can make.



(b) A rearrangement of graph in Figure 1.2(a) with the possible movements of knights.

Figure 1.2: Graph representation of the Knight's Move Puzzle.

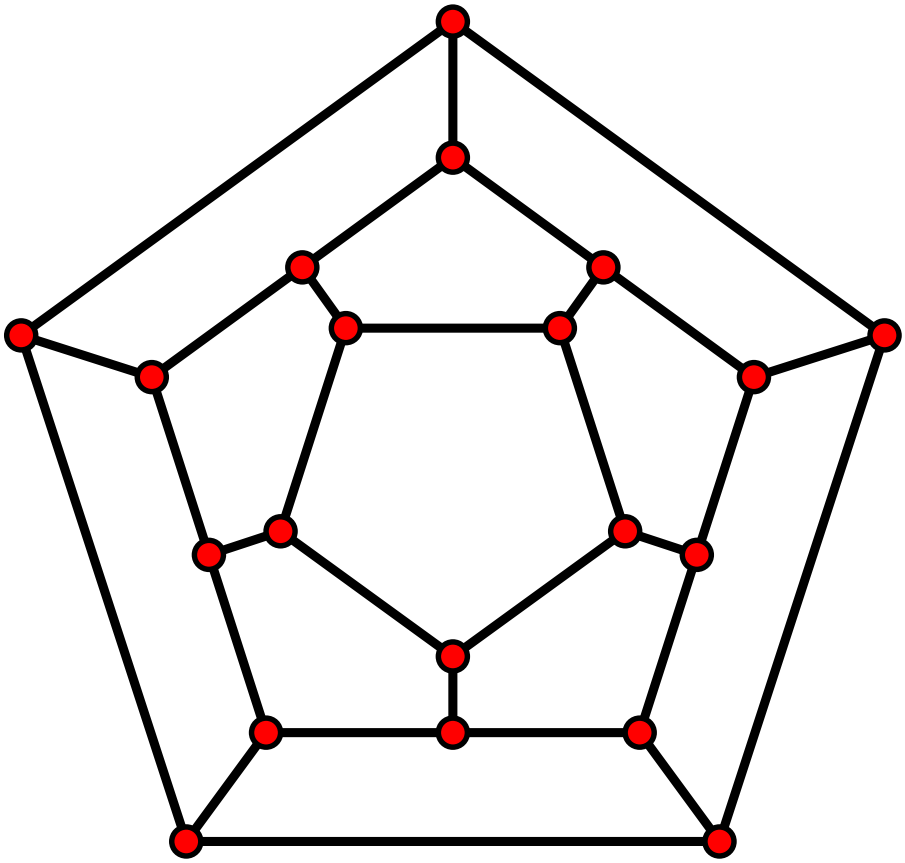


Figure 1.3: Two dimensional projection of a dodecahedron. All the edges have the same length.

of the game is to create a cycle in a dodecahedron such that each vertex is visited exactly once. A dodecahedron is a three dimensional object with twelve regular pentagonal faces. The solution to the game is a Hamiltonian cycle or a TSP tour. Even though a dodecahedron is a three dimensional object, it can be projected on the two dimensional plane as shown in Figure 1.3. However, the game was easy to solve and as a result of that a commercial failure. TSP, on the other hand, continues to spur significant interest in various fields.

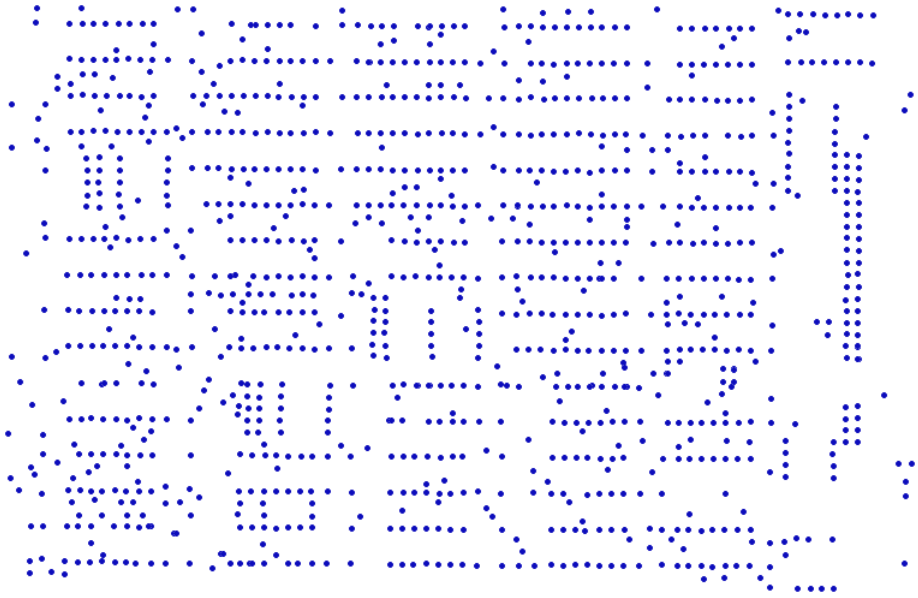
Even though the Knight's Move Problem or the Icosian game are interesting in order to see the usefulness of graph theory, they are not interesting real-life problems. In Figure 1.4(a), the data for a real-life circuit board problem is shown. The dots in the figure are points that have to be drilled in a circuit board. The

problem of minimizing the distance covered by the drill corresponds to the TSP problem and in this instance there are 1173 nodes to be visited. In Figure 1.4(b) the optimal solution for the problem is shown. The printed circuit board design problem is studied in Ancău (2008), where on a problem with 2096 nodes there is a 3 fold improvement over the conventional methods of drilling horizontally or vertically. This is a relatively small problem instance compared to the benchmark. In 2001, the TSP instance with 15112 German cities was solved optimally. In 2004, an instance with 24987 cities in Sweden was solved. The biggest instance solved optimally is an instance with 85900 locations, Applegate et al. (2009). The problem is a VLSI application that dates from the 1980s from Bell Laboratories.

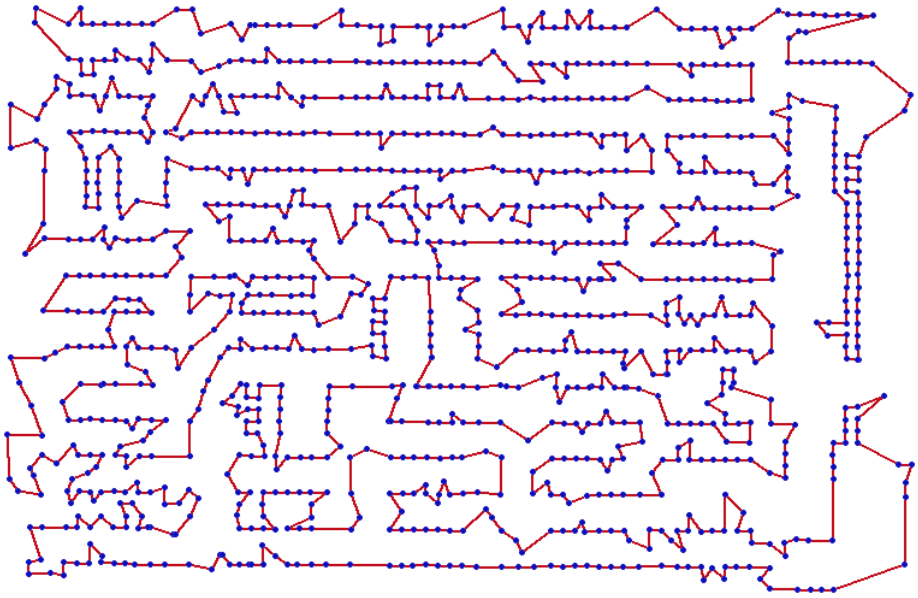
1.3 Linear and integer programming

Linear Programming (LP) is one of the strongest tools in combinatorial optimization for finding the optimal values for decision variables, that maximize or minimize a linear objective function, by satisfying a set of constraints given by linear equations. Linear programming was developed by Kantorovich in 1939. The problem he had at hand was to achieve the highest output (profit) by optimally allocating resources (material, labor, and equipment). In his work, Kantorovich (1960) models several problems as linear programs and lays foundations of a solution method for these problems. Even though this rudimentary solution method was developed in 1939, it was kept secret during the war. In 1947, Dantzig (1951) developed the simplex algorithm which was able to solve linear programming models guaranteed to optimality. The simplex method consists of moving from one feasible solution to another with a better objective function value. This way the optimal solution can be reached in a finite number of such steps. A linear programming problem is not a combinatorial optimization problem as the variables in the problem are continuous and the set of feasible solutions is not finite. However, there is "always" an optimal solution that is a corner point of the polyhedron defined by the linear constraints. Linear programming problems can be solved efficiently, even though the simplex algorithm is not efficient: the ellipsoid method (Khachiyan (1980), and interior point methods (Karmarkar (1984)) are efficient.

In linear programming, the variables can take fractional values. Even though it is a strong tool in operations research, there are many problems where the variables are naturally restricted to integer values. In a decision problem for an airline company it does not make sense to buy half an airplane, or assign half a



(a) The points that need to be drilled in a circuit board.



(b) The optimal TSP tour for the problem in Figure 1.4(a).

Figure 1.4: A circuit drilling instance with 1173 holes to be drilled.

journey to a certain pilot. Linear programming problems with integer variables are called integer (linear) programming problems.

The first work on integer programming was done in the 1950s, by Dantzig et al. (1954), Dantzig (1957), and Markowitz and Manne (1957). They use problem specific information to do two things: generate cuts, i.e., new linear constraints that tighten the formulation, and branches, i.e., splitting the set of feasible solutions into parts that are separately considered. Even though they do not present an automated algorithm both ideas were very important in the development of solution methods for integer programs.

Gomory (1958) proposed a generic procedure to develop tightened constraints: the cutting-plane method. Gomory developed an automated procedure for solving integer programming problems by generating these cutting-planes and proved that the procedure was finite. Land and Doig (1960) first presented the branch-and-bound algorithm. In this algorithm the feasible region was split into two (or more) parts such that an optimal solution is present in at least one of the parts. Each of the parts (the subproblems) is examined separately. Repeated application of the splitting method finally results in easily solvable subproblems.

Both algorithms rely on solving linear programming problems by ignoring integrality constraints. In the cutting-plane method, developed by Gomory (1958), new linear inequalities are generated from the fractional solution by a rounding procedure. A cut is a linear constraint that does not separate any of the feasible (integer) solutions to the problem but cuts off fractional (optimal) solutions to the linear programming relaxation. Gomory's cutting-plane method is guaranteed to find the optimal solution in a finite number of steps. However, the number of steps required to reach the optimal solution can be very large. Dantzig et al. (1954) solves a TSP problem with 49 cities using cuts that were generated from problem specific information. This paper is especially important as it still forms the basis of most exact TSP algorithms, and many other exact algorithms for hard combinatorial optimization problems.

It is possible to combine the two methods to the branch-and-cut algorithm, which has shown to be a very powerful method to solve integer linear programs. This solution method is used throughout this thesis.

1.4 Problems studied in this thesis

In this section, we shortly describe the problems we discuss in this thesis. The problems share three characteristics: first, they have some applications in real life; second, they are all modeled using graphs; third, they are all solved using branch-and-cut in this thesis.

1.4.1 Minimum triangulation of graphs

A chordal graph is a graph where the largest cycle without a chord (an edge joining vertices non-adjacent on the cycle) has size 3. Consequently, any cycle of size greater than 3 has a chord. A triangulation (chordalization) of a graph is the addition of edges to the graph, such that the new graph obtained is chordal. The minimum triangulation problem is to find the smallest number of additional edges to make the graph chordal. It has applications in Gaussian elimination of sparse positive definite matrices, database management, knowledge based systems, and Bayesian networks and artificial intelligence. Furthermore, chordal graphs admit polynomial time algorithms for computationally hard problems such as graph coloring, and maximum clique. The concept of treewidth is related to chordal graphs, as the width of a tree decomposition, which corresponds to a chordalization of the graph, is the size of the largest clique minus one. The treewidth of a graph is the minimum width over all possible tree decompositions. In graphs with bounded treewidth, problems such as TSP can be solved in polynomial time. For this reason, minimum triangulation and treewidth have been extensively studied even though they are computationally complex. In this thesis, we consider the minimum triangulation problem. In Figure 1.5, a minimum triangulation for a 3 by 3 grid graph is shown.

To solve the minimum triangulation problem, we use a branch-and-cut algorithm. We formulate the problem by using the property that a chordal graph has a perfect elimination ordering and present various classes of valid inequalities. To the best of our knowledge, it is the first branch-and-cut algorithm for the problem. In addition to that, it is the first time a lower bound is provided for the problem. The branch-and-cut algorithm performs especially well on dense graphs.

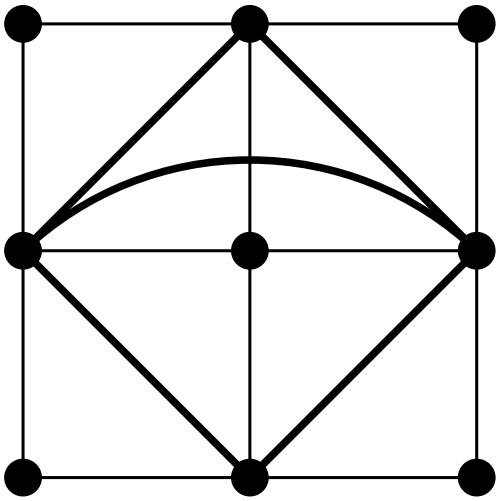


Figure 1.5: Triangulation of a 3 by 3 grid graph. Thick edges are added in order to triangulate the graph.

1.4.2 Graph coloring

The (vertex) graph coloring problem corresponds to finding an assignment of colors to the vertices of a graph such that the end points of an edge have different colors. The problem is computationally complex in general graphs, even for a fixed number of colors $k \geq 3$, Garey and Johnson (1979). The problem has applications in scheduling, timetabling, bandwidth allocation, and sequencing. In these applications a color class (vertices having the same color) corresponds to the activities that can be carried out together without any restriction. If two activities are connected then they are mutually exclusive, such as two courses that cannot be scheduled at the same time slot because they have to be taken by the same students.

The graph coloring problem originated from the need to color geographical maps with a minimum number of colors. Francis Guthrie, in 1852, conjectured that the problem of coloring a map can be solved using four colors, Kubale (2004). The conjecture remained open for over a century. In 1975, Martin Gardner incorrectly claimed that the map illustrated in Figure 1.6 requires five colors, which later turned out to be four-colorable.

The conjecture was proved in 1976, Appel and Haken (1977). The proof of Appel and Haken was one of the first proofs that relied on computer power. Coloring a map corresponds to coloring the vertices of a planar graph, which corresponds to the class of graphs that can be drawn in a plane in such a way that no edges

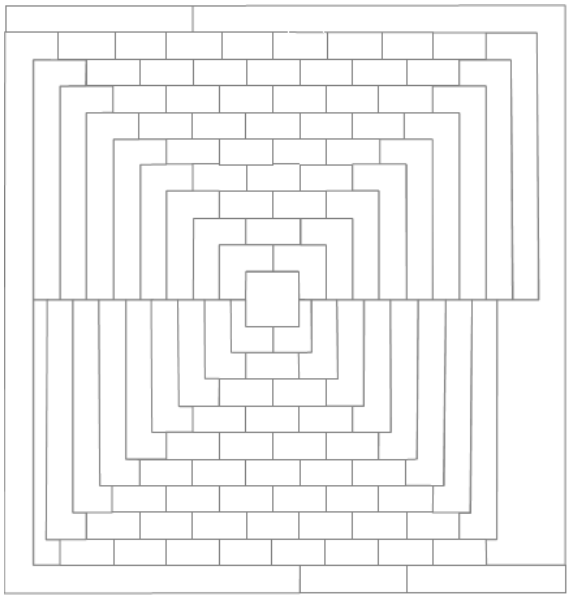


Figure 1.6: Gardner's map.

cross each other apart from the end points. The coloring problem is polynomially solvable on planar graphs.

The branch-and-cut algorithm has been extensively used to solve the graph coloring problem, with different formulations. In our problem, we use the ILP formulation introduced by Campêlo et al. (2008). The advantage of the formulation is that it reduces symmetry in the model: colors are no longer exchangeable. We present various classes of valid inequalities, some of which are also presented by Campêlo et al. (2008). For most classes of valid inequalities we develop separation routines. We also modify the model to reduce the number of variables and present our computational study.

1.4.3 Safe dike heights in the Netherlands

Determining safe dike heights in the Netherlands became a problem after the flood of 1953. The first work about dike heights was carried out by van Dantzig in a paper that is considered to be the start of Operations Research in the Netherlands. Recent floods in Germany in 2013 resulted in a loss of €12 billion, whereas Hurricane Katrina in New Orleans in 2005 caused damage of $108 billion and claimed the lives of nearly 2000 people. With the change in climate (global warming and

increasing sea levels) protection against floods becomes more important than ever. The cost-benefit analysis (CBA), we are using balances social costs of heightening the dikes against the social benefits of security (avoiding material and immaterial damage).

In our problem, we consider a scenario where dikes can be heightened individually. The difference with models in the literature is the structure of the expected damage costs. In the literature, the damage costs are calculated by considering the failure of individual dikes. In our problem, due to the structure of the dikes in the Netherlands the damage costs in case of failure depend on multiple dikes that protect an area. We present the formulation for the problem given in Zwaneveld and Verweij (2014), and add valid inequalities for the problem. Finally, we present our computational study based on branch-and-cut.

Chapter 2

Preliminaries

In this chapter, we define the basic concepts and notation used throughout this thesis. They are from three different fields: computational complexity, graph theory, and integer programming.

2.1 Computational complexity

In Chapter 1, we mentioned that SPP and TSP have an exponential number of solutions. However, solving those problems to optimality requires considerably different effort. The reason behind this is the fact that the two problems belong to different classes in complexity theory.

SPP belongs to the class of problems called P. P is the class of decision problems that can be solved by a deterministic algorithm in time polynomial in the input size. For SPP the shortest path between two cities is found in a number of steps that is polynomial in the number of cities, Dijkstra (1959). The class P is contained in the complexity class NP. NP contains all decision problems that can be solved by a non-deterministic algorithm in time polynomially bounded on the input size. There are problems in NP that contain any other problem in NP as a special case: any problem in NP can be transformed polynomially to such a problem. The latter problem is called NP-complete. In other words, the existence of a polynomial time algorithm for an NP-complete problem would imply polynomial time algorithms for all other problems in NP. Thus, $P = NP$ is an open question, even though the consensus is that $P \neq NP$. The optimization version of an NP-complete problem is called NP-hard. The Directed Hamiltonian Cycle

Problem (finding a tour visiting every city exactly once) is an NP-complete problem. Its optimization version is TSP, which is therefore at least as difficult as any NP-complete problem. Unless $P = NP$, there will not be a fast (polynomially bounded) algorithm to solve TSP. However, many problems that are NP-hard in general admit polynomial algorithms for special cases. The graph coloring problem on planar graphs can be solved in polynomial time even though the problem in general graphs is NP-hard. For more information about computational complexity, we refer the interested reader to Garey and Johnson (1979)

In this thesis, we are going to focus on two NP-hard problems: minimum triangulation and graph coloring. The other problem we focus on is determining the optimal dike heights in the Netherlands, which can be solved in polynomial time for a fixed number of dikes. Its complexity for a variable number of dikes is still unknown.

2.2 Graph theory

A graph $G = (V, E)$ consists of a set V of vertices and a set E of edges. The edges are unordered pairs of vertices $\{i, j\}$, where $i, j \in V$. Vertices i and j are called the end vertices of edge $\{i, j\}$. If there is an ordering between the end vertices of an edge it is called an arc and is denoted (i, j), with i preceding j. A graph that only consists of edges is called undirected, and a graph with arcs only is called directed. An edge $\{i, j\}$ is used to denote a relation between vertices i and j. That relation can represent different things, like a connection or a conflict between the vertices. Two vertices that are the end vertices of an edge are called adjacent. The neighborhood of a vertex i, denoted $N(i)$, is the set of vertices that are adjacent to i. For a given ordering of G, we call $N^-(i) = \{1, 2, ..., i-1\} \cap N(i)$ the in-neighborhood of i and $N^+(i) = \{i+1, i+2, ..., n\} \cap N(i)$ the out-neighborhood of i. If the set of edges does not contain self-loops and multiple edges between two vertices, then G is called simple. The focus in this thesis will be on undirected, simple graphs.

Graph $H = (W, F)$ is a subgraph of G if $W \subseteq V$ and $F \subseteq E$, and then G is called a supergraph of H. We call $H = (W, F)$ an induced subgraph of G if $W \subseteq V$ and $\{u, v\} \in F$ if and only if $\{u, v\} \in E$. $\bar{G} = (V, \bar{E})$ is called the complement of G if $\bar{E}$ consists of $\{i, j\} \notin E$.

A path in a graph is a sequence of vertices, $\{i, j, \ldots, k\}$, such that each vertex is adjacent to its neighbors in the sequence. A cycle is a path that starts and ends

at the same vertex, such as $\{i, j, \ldots, k, i\}$. A path is called simple if no vertex is visited twice, a cycle is called simple if no vertex apart from the first and last vertex is visited twice. A chord in a cycle is an edge joining two nonconsecutive vertices of the cycle. A graph is called chordal if every induced cycle of size at least four has a chord. Alternatively, we can say that in a chordal graph the biggest chordless cycle has size 3.

A clique in a graph is a set of vertices S such that all vertices are pairwise adjacent. A clique is maximal if it cannot be extended by addition of another vertex. A clique is maximum if there is no other clique larger in size. The size of a maximum clique in G is called the clique number of G and denoted by $\omega(G)$. An independent set I is a set of vertices such that all vertices are pairwise non-adjacent. Maximal and maximum independent sets are defined in the same fashion. The size of a maximum independent set in G is called the stability number and denoted by $\alpha(G)$. A clique in G corresponds to an independent set in $\bar{G}$ and vice versa. A (vertex) coloring of a graph is an assignment of colors to its vertices such that the end vertices of an edge do not have the same color. The smallest number of colors needed to color the vertices of G is called the chromatic number of G and denoted $\chi(G)$. The vertices having the same color form an independent set.

An ordering of the vertices is a bijection $\sigma : V \to \{1, 2, \ldots, n\}$ where σ_i represents the position in the ordering of vertex i. The ordering σ implies a direction of the edges of G. If the vertices i and j are connected in G and $\sigma_i < \sigma_j$, then in the associated digraph $\vec{G} = (V, A)$, the arc (i, j) is present. Note that $\vec{G}$ does not contain any directed cycles. A perfect elimination ordering (peo) is a partial elimination of the vertices of G such that, for each vertex i, the neighbors of i that are later in the ordering form a clique. In the minimum triangulation problem peos can be used to create triangulated supergraphs of G as follows: one selects sequentially a vertex i from G, turning the (current) neighborhood of i into a clique, and removing i from the graph. The order in which the vertices are removed from G is called the elimination order (eo). If no edges have to be added, i.e., if all the sets of neighbors encountered in the process of elimination are cliques, then the elimination order is called perfect (peo). The following theorem allows us to use (perfect) elimination orderings for the minimum chordalization problem:

Theorem 2.1. *A graph is chordal if and only if it has a perfect elimination ordering, Fulkerson and Gross (1965).*

Chordalization of graphs and perfect elimination ordering are also related to tree decomposition.

Definition 2.1. A tree decomposition of a graph $G = (V, E)$ is a pair $(\{X_\alpha | \alpha \in A\}, T = (A, B))$ with $\{X_\alpha | \alpha \in A\}$ a subset of vertices of G, and $T = (A, B)$ a tree, such that

- $\cup_{\alpha \in A} X_\alpha = V$,
- for all edges $\{i, j\} \in E$ there is an $\alpha \in A$ with $i, j \in X_\alpha$,
- for all $\alpha_1, \alpha_2, \alpha_3 \in A$: if α_2 is on the path from α_1 to α_3 in T, then $X_{\alpha_1} \cap X_{\alpha_3} \subseteq X_{\alpha_2}$.

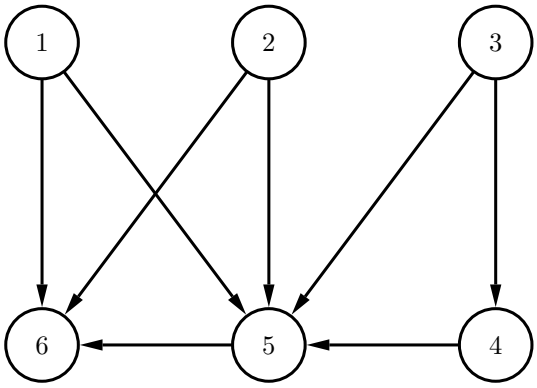
The width of a tree decomposition is $\max_{\alpha \in A} |X_\alpha| - 1$. The treewidth of G corresponds to the minimum width over all tree decompositions of G.

Each tree decomposition of a graph can be transformed into a chordalization and vice versa. To transform a tree decomposition into a chordalization, add the missing edges in all bags $X_i, i \in I$ transforming them into cliques. To transform a chordalization into a tree decomposition, we take a peo of the chordal graph. Starting from the end of the ordering, we create bags, which correspond to the subsets X_i of a tree decomposition. We make a first bag using a maximal clique at the end of the ordering. At each iteration we consider a new vertex i. Using i and its out-neighbors $N^+(i)$, we create a new bag and connect it to the bag associated with the first vertex in $N^+(i)$. By this construction, all the vertices of G appear at least in one bag. If $\{i, j\} \in E$ with $i < j$, then they appear together in the bag associated with i. As we progress by using cliques in the peo, a subtree in the tree decomposition corresponding to a vertex i is connected. An example is shown in Figure 2.1. In the figure, bags $(2, 5, 6)$ and $(1, 5, 6)$ are connected to the bag $(5, 6)$ as 5 is the first node in $N^+(1)$ and $N^+(2)$.

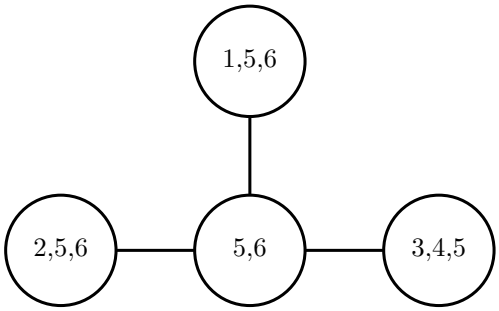
In the tree decomposition and corresponding chordalization of a graph, the size of a largest clique minus one is equal to the width. The treewidth of the graph G is the minimum width among all possible tree decompositions of G, and thus the chordalization with the smallest maximum cliquesize minus 1. Many NP-hard problems, such as independent set, Traveling salesman and Steiner tree, are solved in polynomial time when they are restricted to graphs with bounded treewidth.

2.3 Linear and integer programming

Linear programming is a mathematical tool for finding the best feasible solution in a problem, if the problem is modeled by a linear objective function, and by



(a) A peo with the ordering $(1, 2, 3, 4, 5, 6)$.



(b) Tree decomposition corresponding to the peo in Figure 2.1(a). The width of the decomposition is 2.

Figure 2.1: A graph with 6 vertices and a possible tree decomposition.

linear constraints. The decisions that have to be made are represented by the decision variables of the model. A feasible solution to an LP problem satisfies all the constraints in the model. An optimal solution is a solution that has the best objective function value (such as the maximum profit or minimum cost) among all feasible solutions. All LP models can be expressed in canonical form

$$\text{maximize } \mathbf{c}^{\mathrm{T}}\mathbf{x}, \tag{2.1}$$

$$\text{subject to: } A\mathbf{x} \leq \mathbf{b}, \tag{2.2}$$

$$\mathbf{x} \geq \mathbf{0}. \tag{2.3}$$

In the canonical form $\mathbf{x}$ is the vector of decision variables, $\mathbf{c}$ is the vector of objective function coefficients. A is the matrix of constraint coefficients and $\mathbf{b}$ is the vector of the right hand sides of the constraints.

For a problem with n variables, a polyhedron P is the set of points in $\mathbb{R}^n$ that satisfy a finite set of linear constraints, defining half-spaces in $\mathbb{R}^n$, i.e.

$$P = \{x \in \mathbb{R}^n | Ax \leq b\}.$$

Alternatively a polyhedron can be expressed as a convex combination of points of a finite set $X = \{x^k | k = 1, \ldots, K\}$ and a nonnegative combination of points in a finite set of rays $Y = \{y^l | l = 1, \ldots, L\}$. In this notation, we have

$$P = conv(X) + cone(Y),$$

with

$$conv(X) = \{\sum_{k=1}^{K} \alpha_k x^k | \sum_{k=1}^{K} \alpha_k = 1; \alpha_k \geq 0 \ \forall k \in 1, \ldots, K\},$$

and

$$cone(Y) = \{\sum_{l=1}^{L} \beta_l y^l | \beta_l \geq 0 (l = 1, \ldots, L)\}.$$

The representation theorem of Farkas, Minkowski, and Weyl states that both definitions are equivalent and a description in one form implies the existence of the description in the other form.

A polyhedron that is bounded, i.e. $cone(Y) = \emptyset$, is called a polytope. In the subsequent chapters of this thesis, we focus on polytopes in the n-dimensional cube $\mathbb{B} = \{x \in \mathbb{R}^n | 0 \leq x \leq 1\}$.

A polyhedron P is convex, i.e. $\forall x, y \in P$, $\lambda x + (1 - \lambda)y \in P, 0 \leq \lambda \leq 1$, as it is the intersection of convex halfspaces $\{Ax \leq b\}$, defined by the linear inequalities. As a result of the convexity, and the linearity of the objective, there always exists an optimal solution to an LP which is a corner point of the polyhedron.

An example model with two decision variables is given below:

$$\text{maximize } 2x_1 + x_2, \tag{2.4}$$

$$\text{subject to: } 2x_1 - x_2 \leq 4, \tag{2.5}$$

$$2x_1 + 3x_2 \leq 13, \tag{2.6}$$

$$x_1, x_2 \geq 0. \tag{2.7}$$

The feasible region defined by the constraints (2.5)-(2.7) is the area in Figure 2.2 that lies inside the solid lines. In the example, the optimal solution has the coordinates $(3.125, 2.25)$. The simplex algorithm by Dantzig (1951) starts at a corner point of the polytope and moves to another corner point with a better (or not worse) objective function value. Assuming we start at $(0, 0)$, the simplex algorithm would move to $(0, 4.33)$ and then find the optimal at the point $(3.125, 2.25)$. Though the simplex method is very effective in practice, it is not an efficient algorithm. Nevertheless, LP is polynomially solvable: the ellipsoid method (Khachiyan (1980)), and interior point methods (Karmarkar (1984)) are efficient.

For an integer programming problem P, we define the set of integer feasible points by X, and the relaxation of the problem where variables can take continuous values by P^{LP}. The smallest convex set containing all integer feasible points is called the convex hull. It is denoted by $conv(X)$. Since $conv(X)$ is contained in the polyhedron defined by P^{LP}, this polyhedron can be used to bound the objective function value of the integer programming problem. Furthermore, if $conv(X)$ is known explicitly, the integer programming problem can be solved using the simplex algorithm, as the corner points of $conv(X)$ are integer. In addition to that, branch-and-bound and Gomory's cutting plane method rely on solving the LP relaxation of integer programming problems.

In branch-and-bound, developed by Land and Doig (1960), the relaxation of an integer programming problem is solved without integrality restrictions on the variables. If the solution is not integral, a fractional variable x_i is chosen, and two

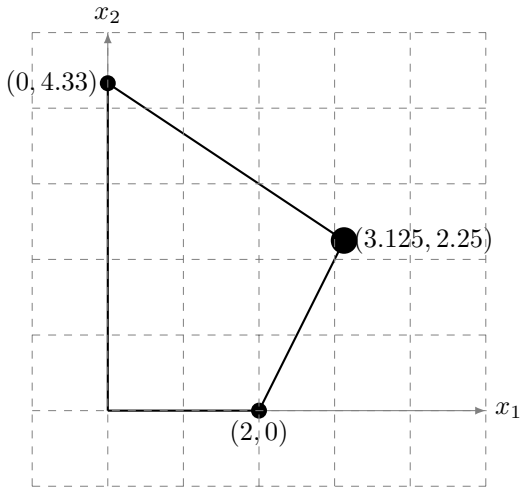


Figure 2.2: The feasible region defined by the formulation (2.5)-(2.7) lies inside the solid lines.

subproblems are created in such a way that the fractional value v_i of the variable is forbidden. In one subproblem x_i is bounded from above by $\lfloor v_i \rfloor$, i.e. $x_i \leq \lfloor v_i \rfloor$, and in the other subproblem it is bounded from below by $\lceil v_i \rceil$, i.e. $x_i \geq \lceil v_i \rceil$. In Figure 2.3(a), an iteration of the branch-and-bound algorithm is shown. In the relaxation given by model (2.5)-(2.7), the optimal solution is obtained at point $(3.125, 2.25)$ with an objective function value of 8.5. However, the solution is not integer. We select a fractional variable and create two subproblems. Assuming we select $x_2 = 2.25$, we create two subproblems by adding the constraint $x_2 \leq 2$ for one subproblem and $x_2 \geq 3$ for the other subproblem. By this method we forbid the current fractional value obtained by the variable without eliminating any of the feasible solutions of the integer programming problem. The feasible regions of the subproblems are shown in Figure 2.3(a). Integrality of the solutions is not guaranteed in the subproblems. If the solution is not integer, we use the same scheme to generate more subproblems. In the end the method boils down to implicitly enumerating all feasible solutions for the problem. However, upper and lower bounds are employed to eliminate some parts of the branch-and-bound tree.

Gomory's cutting-plane method also uses the solution to the linear programming relaxation. The method relies on the following observation: for a vector $u \geq 0$, the following inequality holds because of the integrality (and nonnegativ-

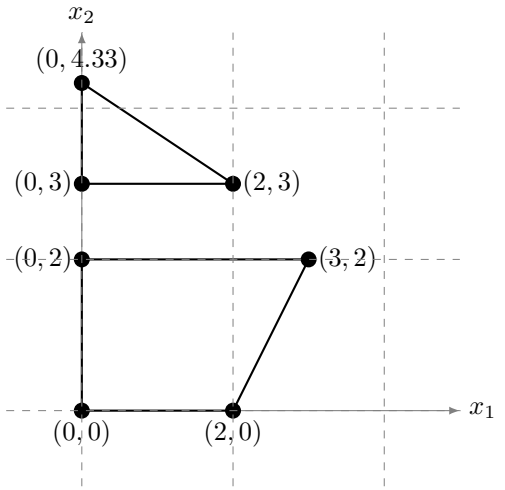
ity) of the variables:

$$\lfloor uA \rfloor x \leq \lfloor ub \rfloor .$$

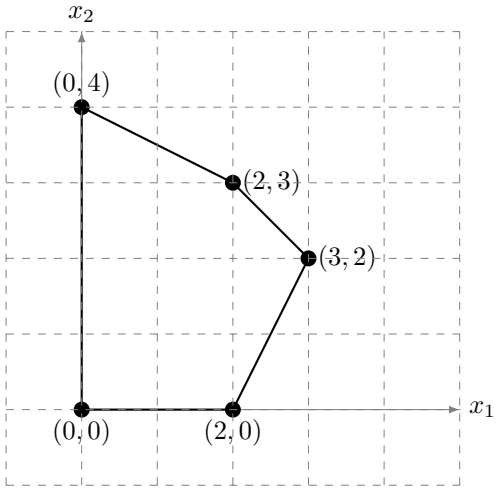
Gomory also presents an algorithm for adding new inequalities to the model and proves the procedure ends after a finite number of steps. However, the number of steps is not polynomially bounded. Furthermore, the inequalities added contain a lot of non-zero elements, which makes the method prone to numerical instabilities. For this reason, problem specific cuts are frequently studied, since they were introduced by Dantzig et al. (1954) for the traveling salesman problem. Integrating problem specific cuts with branch-and-bound led to the branch-and-cut algorithm. In this method branches are created like in the branch-and-bound algorithm, but the relaxation is strengthened with the addition of valid inequalities. The convex hull of all integer feasible points for model (2.5)-(2.7) is shown in Figure 2.3(b). Applying the simplex algorithm to the convex hull gives the optimal integer solution. However, obtaining the convex hull, even for a small size problem is generally not very practical due to the enormous amount of valid inequalities to be added.

Even though obtaining the complete convex hull for an integer programming problem is difficult, obtaining some inequalities that are necessary for the description of the polytope is quite possible. Next, we describe the properties that necessary inequalities have as well as some other concepts from polyhedral theory that we need in this thesis.

A characteristic vector $\mathbf{x}$ corresponding to a solution is a vector of dimension n for which component x_i takes the value of the corresponding decision variable. A characteristic vector corresponds to a point in the polytope for which the coordinates are the values that the decision variables take in that particular solution. In our proofs for the dimension of the polytopes, we are going to use difference vectors on the polytope. A difference vector on a polytope corresponds to the difference of two characteristic vectors in the polytope, i.e. two solutions that are feasible for the problem. Similarly, in order to prove that valid inequalities are facet-defining, we use difference vectors on the face (vectors obtained by subtracting two characteristic vectors that satisfy the inequality at equality). As we are using the difference of two characteristic vectors, we need to find the necessary number of linearly independent vectors in our proofs. In the rest of this thesis, we use the term vector instead of the terms difference vector on the polytope and difference vector on the face. Furthermore, the vectors that we use are always linearly independent and we do not use the term in our proofs. We explicitly state



(a) Creating two subproblems where $x_2 \leq 2$ and $x_2 \geq 3$.



(b) Convex hull of all integer feasible solutions to the problem (2.5)-(2.7).

linear dependency if we encounter such a case. Because of the number of zero components in the vectors we use a shorter notation. We use $(x_i, x_j : 1; x_k : -1)$ to denote a vector where $x_i = x_j = 1$, $x_k = -1$, and all the other components are zero.

The set of linear combinations of a set of vectors $x^1, \ldots, x^K \subset \mathbb{R}^n$ is a linear space $LS = \{\sum_{k=1}^{K} \alpha_k x^k | \alpha \in \mathbb{R}^K\}$. If the set of vectors $x^1, \ldots, x^K$ is minimal, i.e. none of the vectors is a linear combination of others, they are linearly independent. If $x^1, \ldots, x^K$ are linearly independent, the linear space defined by these vectors has dimension K. The set of affine combinations of a set of vectors $x^0, x^1, \ldots, x^K$ is called an affine space $AS = \{\sum_{k=0}^{K} \alpha_k x^k | \alpha \in \mathbb{R}^{K+1}; \sum_{k=0}^{K} \alpha_k = 1\}$. If the set of vectors $x^0, \ldots, x^K$ is minimal, i.e. none of the vectors is an affine combination of others, they are affinely independent. If the points $x^0, x^1, \ldots, x^K$ are affinely independent, then the affine space defined by these points has dimension K. Next, we explain the relation between affine and linear independence, and we relate these concepts to polytopes.

Theorem 2.2. *(Nemhauser and Wolsey, 1988) Let $x^0, \ldots, x^K \in \mathbb{R}^n$. The following statements are equivalent*

- *$x^0, \ldots, x^K$ are affinely independent;*
- *$x^1 - x^0, \ldots, x^K - x^0$ are linearly independent;*
- *$\binom{x^0}{1}, \ldots, \binom{x^K}{1}$ are linearly independent.*

To explain the dimension of a polytope, we use the following theorem.

Theorem 2.3. *If the affine space $AS = \{x \in \mathbb{R}^n | Ax = b\} \neq \emptyset$ then*

- *$r(A) = r(A|b)$, where $r(A)$ is the rank of the matrix A;*
- *The maximum number of affinely independent solutions to the system $Ax = b$ is $n + 1 - r(A)$.*

Consider the polytope $P = \{x \in \mathbb{R}^n | Ax \leq b\}$, and denote the set of inequalities that are satisfied at equality in P by $\{A^= | b^=\}$. For a nonempty polytope P we have: $dim(P) + r(A^= | b^=) = n$. The dimension of P corresponds to the maximum number of linearly independent vector of differences in P.

An inequality $\pi x \leq \pi_0$ is valid if P lies in the halfspace defined by the inequality. A valid inequality represents a face $F \subseteq P$, where F is the subset of P that satisfies the inequality at equality, i.e. $F = \{x \in P | \pi x = \pi_0\}$. A face F is proper if it is

not empty, i.e., $F \neq \emptyset$, and if it is a proper subset of P, i.e., $F \neq P$. A face is called a facet if $dim(F) = dim(P) - 1$. A facet is a maximal proper face. Facets are important, since the inequalities defining them are exactly (up to uniqueness) necessary to describe the polytope P and any inequality that is not facet-defining is irrelevant to the description of the polytope.

In order to illustrate the concepts explained in this chapter, we use the independent set polytope of a clique of size 4. Decision variables for the problem are defined as follows:

$$x_i = \begin{cases} 1, & \text{if vertex } i \text{ is included in the independent set,} \\ 0, & \text{otherwise.} \end{cases}$$

The independent set problem on a clique of size 4 is modeled as follows:

$$\text{maximize} \sum_{i \in V} x_i, \tag{2.8}$$

$$\text{subject to: } x_i + x_j \leq 1, \qquad \forall i, j \in V, i \neq j \tag{2.9}$$

$$x_i \geq 0, \qquad \forall i \in V. \tag{2.10}$$

We let $\mathbf{x^i}$ be the characteristic vector of the integer feasible solution that includes vertex i in the independent set, with $\mathbf{x^0}$ denoting an empty set. The characteristic vectors are shown in Table 2.1.

	$\mathbf{x^0}$	$\mathbf{x^1}$	$\mathbf{x^2}$	$\mathbf{x^3}$	$\mathbf{x^4}$
x_1	0	1	0	0	0
x_2	0	0	1	0	0
x_3	0	0	0	1	0
x_4	0	0	0	0	1

Table 2.1: Characteristic vectors corresponding to integer feasible solutions of the independent set problem.

We can also observe that the difference vectors $\mathbf{x^1} - \mathbf{x^0}, \ldots, \mathbf{x^4} - \mathbf{x^0}$ are linearly independent, showing that the polytope is 4 dimensional. Consider the inequality

$$x_1 + x_2 + x_3 \leq 1.$$

The inequality is valid as only one of the clique vertices is allowed in an independent set. However, the inequality is not facet-defining. There are only three

integer feasible solutions that satisfy the inequality at equality. Vectors $\mathbf{x^2} - \mathbf{x^1}$ and $\mathbf{x^3} - \mathbf{x^1}$ are linearly independent and the face given by the inequality has dimension 2. The inequality

$$x_1 + x_2 + x_3 + x_4 \leq 1$$

is facet-defining as it has dimension 3. It is easy to see that the inequality is obtained by adding a new variable to the *lhs* of the previous inequality. This process is called lifting.

As it is easier to prove linear independence we use a modified version of the second statement in Theorem 2.2. Instead of proving linear independence of vectors $x_1 - x_0, x_2 - x_0, \ldots, x_n - x_0$ by using a fixed solution x_0, we try to obtain n linearly independent vectors that correspond to the difference vector of any two solutions on the face.

Chapter 3

Minimum triangulation of graphs

Finding a minimal triangulation (also called chordalization and fill-in) of graphs has applications in many different areas. In the Gaussian elimination of sparse positive definite matrices, finding a minimal triangulation corresponds to applying a Gaussian elimination creating a minimum number of nonzero entries, Rose (1970, 1973). Bertelè and Brioschi (1971) use the triangulation problem for the Gaussian elimination of sparse matrices and in non-serial dynamic programming. Minimum triangulation also has applications in database management (Tarjan and Yannakakis, 1984; Beeri et al., 1983), knowledge based systems (Lauritzen and Spiegelhalter, 1988), Bayesian networks, and artificial intelligence. For a detailed overview on usage of triangulation in artificial intelligence, we refer the reader to Chung and Mumford (1994) and for a detailed review of the minimal triangulation problem, we refer the interested reader to Heggernes (2006).

Chordal graphs admit efficient algorithms for problems that are NP-hard on general graphs, such as the maximum clique problem and the graph coloring problem, Gavril (1972); Hell et al. (2004).

Yannakakis (1981) proves that the minimum triangulation problem is NP-hard. The problem is NP-hard even on bipartite and square graphs, but admits a linear-time algorithm for outerplanar graphs, Yuan and Zhang (1996). Even though the problem is NP-hard, it has been extensively studied from a computational point of view because of its applications in various fields. The research on triangulations of graphs follows mainly two streams: elimination orderings and minimal separators.

Bouchitté and Todinca (2001) focus on the idea of potential maximal cliques, which correspond to a set of vertices that induce a clique in some minimal triangulation of the graph, and prove that potential maximal cliques of a graph are enough to find the treewidth and a minimum triangulation of a graph. They present an algorithm that is polynomial in the number of vertices and the number of potential maximal cliques. This approach unifies the findings of different papers on special classes of graphs. If a graph class has a polynomially bounded number of potential maximal cliques, then the treewidth and minimum triangulation problem can be solved in polynomial time. Using this idea and the algorithm to list all potential maximal cliques by Bouchitté and Todinca (2002), Fomin et al. (2004) come up with an exact algorithm that computes the treewidth and minimum triangulation in $\mathcal{O}^*(1.9601^n)$ time using minimal separators. By improving the bound for the algorithm to find the potential maximal cliques, Villanger (2006) manages to improve the time bound to $\mathcal{O}^*(1.8899^n)$

We focus on perfect elimination orderings (peos) of the vertices to find a minimum triangulation of a graph. A peo is an ordering of the vertices of a graph such that, for each vertex i, the neighbors of i that are later in the ordering form a clique. A graph is chordal if and only if it has a peo, Fulkerson and Gross (1965). We use this property to formulate the minimum triangulation problem as an integer linear program (ILP). We solve the ILP using branch-and-cut.

Rest of this chapter is organized as follows: In Section 3.1, we present an ILP formulation based on peos. In Section 3.2, we present three classes of valid inequalities, in addition to the model inequalities. Finally, we present the setting for our computational study in Section 3.3 and present our result on benchmark graphs.

3.1 Mathematical model

Let $G = (V, E)$ be an undirected, connected, and simple graph with vertices V and edges E. We define the neighborhood of vertex i, the vertices connected to i in G, as $N(i)$. An ordering of the vertices is a bijection $\sigma : V \to \{1, 2, \ldots, n\}$ where σ_i represents the position in the ordering of vertex i. The ordering σ implies a direction of the edges of G as follows. If the vertices i and j are connected in G and $\sigma_i < \sigma_j$, then in the associated digraph $\vec{G} = (V, A)$, the arc (i, j) is present. Note that $\vec{G}$ does not contain any directed cycles. The neighbors of i that are later than i in the ordering are called the out-neighborhood of i, denoted by $N^+(i)$.

Graph $H = (W, F)$ is a subgraph of G if $W \subseteq V$ and $F \subseteq E$. Then G is called a supergraph of H. We call $H = (W, F)$ an induced subgraph of G if $W \subseteq V$ and for any $u, v \in W$, $\{u, v\} \in F$ if and only if $\{u, v\} \in E$.

In the minimum triangulation problem we look for a supergraph of G that is triangulated and has a minimum number of extra edges over G. Elimination orderings can be used to create triangulated supergraphs of G as follows: one selects sequentially, according to the elimination order, a vertex i from G, turning the out-neighborhood of i into a clique. At the end of the process the graph is chordal and the elimination order is called perfect.

We use this procedure in the other direction: we direct the edges of a supergraph H of G, in such a way that the constructed digraph $\vec{H}$ is acyclic, and the out-neighborhood of every vertex i (head-nodes of arcs leaving i) is a clique in H. In that case H is triangulated, since the partial order can easily be extended to a perfect elimination order. Note that every acyclic digraph allows for a topological ordering (a complete ordering of the vertices such that for each arc (i, j) the tail i precedes the head j). The topological ordering is a peo, as the condition on the out-neighbors is satisfied. We will use this in the ILP model for the minimum triangulation problem as follows. We describe all acyclic digraphs $\vec{H}$ by means of the selected arcs, such that the underlying undirected graph H is a chordal supergraph of G. These are the feasible solutions of the ILP. The objective returns the number of edges that we add to G to obtain H. Note that, in particular, the complete graph on $|V|$ vertices is a supergraph of G. For this graph, every ordering of the vertices is perfect.

The ILP formulation for the problem is derived from Feremans et al. (2002). For all vertices $i, j \in V$ with $i \neq j$ we define the following variable:

$$x_{ij} = \begin{cases} 1, & \text{if the arc } (i, j) \text{ exists in the digraph } \vec{H}, \\ 0, & \text{otherwise.} \end{cases}$$

We define $X_{ij} = x_{ij} + x_{ji}$ $(i, j \in V, i \neq j)$. X_{ij} variables are not used in the formulation, they are only present for the ease of notation. The ILP formulation of the minimum triangulation problem is given as follows:

$$\min \sum_{\substack{i,j \in V \\ \{i,j\} \notin E}} X_{ij}, \tag{3.1}$$

$$\text{subject to: } X_{ij} = 1, \qquad \forall i, j \in V, \{i, j\} \in E, \tag{3.2}$$

$$X_{ij} \leq 1, \qquad \forall i,j \in V, \{i,j\} \notin E, \tag{3.3}$$

$$x_{ij} + x_{ik} \leq 1 + X_{jk}, \qquad \forall i,j,k \in V, \{j,k\} \notin E, \tag{3.4}$$

$$\sum_{(i,j)\in A(D)} x_{ij} \leq |A(D)| - 1, \qquad \forall D \in \mathbb{D}, \tag{3.5}$$

$$x_{ij} \in \{0,1\} \qquad \forall i,j \in V, i \neq j. \tag{3.6}$$

The objective function (3.1) minimizes the number of fill-in edges. Constraints (3.2) make sure that every edge in G is directed in one direction. If an edge is not present in G, then it may or may not be directed, but it cannot be directed in both directions, according to constraints (3.3). If a vertex i precedes vertices j and k, constraints (3.4) force an ordering between the vertices j and k, thus making the out-neighborhood of i a complete digraph. We consider only acyclic digraphs. A cycle D is an ordered set of nodes in G with coinciding first and last vertex. The set of arcs $A(D)$, connecting consecutive nodes in D, cannot be selected all, since the directed supergraph of G must be acyclic. Constraints (3.5) make sure that at most $|A(D)| - 1$ arcs of any cycle D are directed in the order of the cycle. We use $\mathbb{D}$ to denote the set of all possible cycles in the complete supergraph of G. Finally, the variables are restricted to be binary, (3.6).

A feasible solution to the model (3.1)-(3.6) corresponds to a valid peo of G. The corresponding ordering is clearly acyclic (constraints (3.5)) and the out-neighborhood of a vertex form a clique (constraints (3.4)). A solution to the model can be transformed to a peo of G easily. We select a vertex j that does not have a predecessor ($\sum_{i\in V\setminus\{j\}} x_{ij} = 0$). There should always be a vertex without a predecessor because the formulation does not allow for a cycle. We place vertex j in the first position of the peo and remove it from the graph. We repeat the process for the other positions in the ordering until the graph is empty. IF there are multiple vertices without precedessors, we can select any vertex for the next position of the ordering.

Treewidth

Triangulation of graphs is closely related to the treewidth of graphs.

Definition 3.1. A tree decomposition of a graph $G = (V, E)$ is a pair $(\{X_\alpha | \alpha \in A\}, T = (A, B))$ with $\{X_\alpha | \alpha \in A\}$ a subset of vertices of G, and $T = (A, B)$ a tree, such that

- $\cup_{\alpha\in A} X_\alpha = V$,

- for all edges $\{i, j\} \in E$ there is an $\alpha \in A$ with $i, j \in X_\alpha$,
- for all $\alpha_1, \alpha_2, \alpha_3 \in A$: if α_2 is on the path from α_1 to α_3 in T, then $X_{\alpha_1} \cap X_{\alpha_3} \subseteq X_{\alpha_2}$.

The width of a tree decomposition is $\max_{\alpha \in A \in} |X_\alpha| - 1$. The treewidth of G corresponds to the minimum width over all tree decompositions of G.

Each tree decomposition of a graph corresponds to a unique triangulation. The width of a tree decomposition is the size of a largest clique minus one in the triangulation. Therefore, we can use the description of the feasible solutions of our formulation, extending it with an extra variable representing the treewidth tw, the formulation can be used to solve the minimum treewidth problem:

$$\min\ tw, \tag{3.7}$$

$$\text{subject to: } tw \geq \sum_{j \neq i \in V} x_{ij}, \qquad \forall i \in V, \tag{3.8}$$

$$(3.2) - (3.6),$$

$$tw \in \mathbb{Z}^+. \tag{3.9}$$

The rhs of the constraints (3.8) correspond to the size of the out-neighborhood of each vertex in $\vec{H}$. The maximum size represents the width of the corresponding tree decomposition. However, using the formulation to solve the treewidth problem is not practical. Even for a complete graph (which has treewidth $n - 1$), the LP relaxation yields an optimal solution where $x_{ij} = x_{ji} = 0.5$ and $tw = \frac{n-1}{2}$. Besides, the structure of the polytope associated with the treewidth problem is complicated, as most of the facets also involve variable tw, and these facets are difficult to find.

Various NP-hard problems, such as the independent set problem, the Hamiltonian cycle problem, and the Steiner tree problem, can be solved in polynomial time, when they are restricted to graphs with bounded treewidth. For more information on the use of tree decomposition for solving various problems, we refer the reader to Bodlaender (1993) and references therein.

The polytope of the triangulation problem

We denote, for a graph G, the polytope $P(G)$ as the convex hull of the feasible solutions defined by the constraints (3.2)-(3.6). The formulation has $n(n-1)$

variables, for each pair of distinct vertices. Because of the directed cycle constraints there is an exponential number of inequalities. Before looking at the valid inequalities for the problem we first look at the dimension of the polytope.

Theorem 3.1. *The polytope $P(G)$ has dimension $n(n-1)-m$.*

Proof.

- Upper bound. The formulation (3.2)-(3.6) has $n(n-1)$ variables. Furthermore, there are m linearly independent equations in (3.2), one for each edge in the graph.

- Lower bound: To obtain the lower bound on the dimension of the polytope we use the following equality:

$$\begin{aligned} n(n-1)-m &= 2\binom{n}{2} - 2m + m \\ &= 2\left[\binom{n}{2} - m)\right] + m \\ &= \sum_{\{i,j\}\notin E} 2 + \sum_{\{i,j\}\in E} 1. \end{aligned}$$

  The equality shows that for each $\{i,j\} \notin E$ we must find two vectors in the polytope, and for each $\{i,j\} \in E$ we must find one vector in the polytope, all linearly independent. Note that any complete ordering of the vertices of G induces a feasible solution to the model (3.2)-(3.6). In a solution induced by its complete ordering we set $x_{ij} = 1$ if i is before j in the ordering, for all pairs of vertices. The solution we obtain is acyclic and the vertices in the out-neighborhood of any vertex i form a complete digraph.

  Let i and j be two arbitrary vertices.

  First, consider the solution obtained by a complete graph (clearly a supergraph of G) and an ordering with i in first position, and j in second position, and the rest of the vertices in any ordering. Second, consider the solution obtained with the previous ordering, but placing j in first position, and i in second position. The vector obtained by subtracting the second solution from the first has coefficient $+1$ for variable x_{ij} and coefficient -1 for variable x_{ji}. We describe this vector by its nonzero components as follows: $(x_{ij} : 1; x_{ji} : -1)$. Note that this is sufficient when $\{i,j\} \in E$.

  If $\{i,j\} \notin E$, then we have to construct another vector. We use as a first solution, the first one above, i.e., the solution obtained by a complete or-

dering with i in first position, and j in second position, and the rest of the vertices in any ordering. As a second solution, we take the partial ordering, where i and j are not ordered, but precede all the vertices in $V \setminus \{i, j\}$. In this solution, we have $x_{ij} = 0$. The solution is feasible as constraints (3.3) allow $X_{ij} = 0$, when the corresponding edge is not present in G, and there is no other vertex k which precedes both i and j. It is important to note that the out-neighborhoods of all vertices form a complete, acyclic digraph. The vertices in $V \setminus \{i, j\}$ satisfy the property, since $\{i, j\}$ is the only edge missing. Moreover, i and j are not included in the out-neighborhood of each other. The vector we thus obtain is $(x_{ij} : 1)$. Note that it has exactly one nonzero coefficient. Obviously, all created vectors are linearly independent.

□

In the rest of this chapter, we use the dimension of the polytope in order to prove that the valid inequalities in Section 3.2 are facet-defining in their respective polytopes.

Removing an arc from a chordal digraph without destroying the chordality property is only possible if the out-neighborhoods of all vertices remain complete digraphs after removing the arc. In a chordal digraph, it is possible to remove an arc (if the underlying edge is not present in the graph), in case there is no vertex preceding both the tail and the head vertices of the arc. As a special case, it is possible remove any arc from the first vertex of a perfect elimination ordering of the graph as we did in the second part of Theorem 3.1.

Before exploring the facet-defining inequalities of the formulation, we discuss a zero-lifting theorem. The theorem shows that under a mild condition, facet-defining inequalities for polytopes related to subgraphs of G are also facet-defining for $P(G)$.

Theorem 3.2. Zero-lifting theorem. *Consider an induced subgraph H of G and suppose we are given a facet $\pi x \leq \pi_0$ for $P(H)$. If there is a complete ordering of the vertices of H satisfying $\{x : \pi x = \pi_0\} \cap P(H)$, then $\pi x \leq \pi_0$ is also facet-defining for $P(G)$.*

Proof. We have to prove that the dimension of the face of $\pi x = \pi_0$ on $P(G)$ is $n(n-1) - m - 1$. From now on, the solutions satisfying $\{x : \pi x = \pi_0\} \cap P(H)$ are called tight solutions. We consider two cases: first, we reuse the vectors found for $P(H)$, by extending them to vectors for $P(G)$. Second, we consider vertex pairs i, j for which at least one vertex is in $V(G \setminus H)$.

1. Reusability of vectors between $i, j \in V(H)$. We extend tight solutions for the inequality by adding arcs from every vertex in $V(H)$ to every vertex in $V(G \setminus H)$. Furthermore, we order the vertices in $V(G \setminus H)$ in any order and by directing every edge according to that order. This places the vertices in $V(H)$ before the vertices in $V(G \setminus H)$ and creates all the edges between vertices in $V(H)$ and $V(G \setminus H)$. This creates all vectors that we need from $P(H)$, namely $n_H(n_H - 1) - m_H - 1$. In this notation n_H and m_H denote, respectively, the number of vertices and the number of edges in the subgraph H.

2. Vectors between $i \in V(G \setminus H)$ and $j \in V(G)$. Here, we create all vectors for the vertex pairs with at least one vertex not in $V(H)$. If the pair is an edge in G we create one vector, otherwise we create two. Consider, the solution in H that is complete and satisfies the constraint at equality. The ordering of the vertices in this solution is maintained for all solutions in the sequel, i.e., the extensions to all vertices of $V(G)$, for a complete graph. Note that we may position the vertices of $V(G \setminus H)$ in any ordering, and before, between, and behind the vertices in $V(H)$.

   - Place i directly before and directly behind j in the ordering. This way we obtain vector $(x_{ij} : 1; x_{ji} : -1)$
   - If $\{i, j\} \notin E$, then place i before all the other vertices in $V(G)$. A first solution is obtained by the complete digraph; a second solution by removing the arc (i, j). Note that removing the arc (i, j) from the solution still keeps the out-neighborhood of all vertices complete digraphs (see above). Thus, we obtain vector $(x_{ij} : 1)$.

□

Example 1. Consider the model inequality, which is facet-defining when $(2, 3) \notin E$,

$$x_{12} + x_{13} \leq 1 + X_{23}.$$

The inequality admits a solution on a complete digraph at equality, which is presented in Figure 3.1. In order to reuse the vectors we show in Figure 3.4 we place the vertices in $V(G \setminus H)$ in any order and by directing every edge. We place the vertices in $V(H)$ before the vertices in $V(G \setminus H)$ and again direct every edge between the two sets.

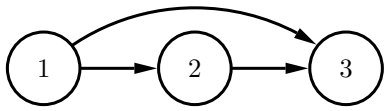
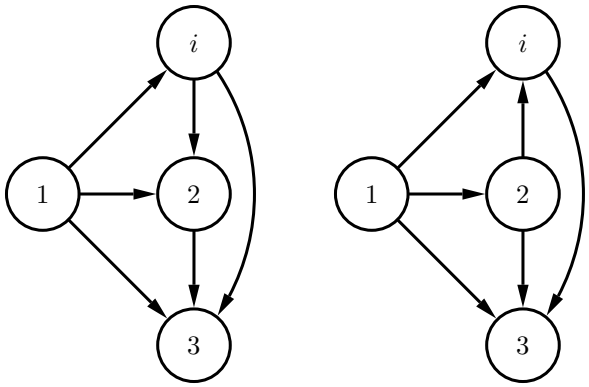


Figure 3.1: The solution that constitutes a complete digraph for the valid inequality.



(a) A solution where vertex i is placed before vertex 2.

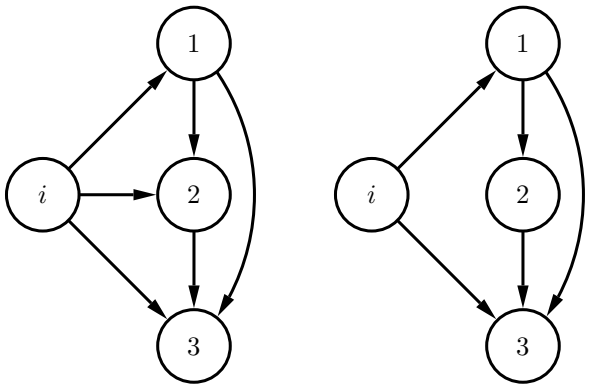
(b) A solution where vertex i is placed after vertex 2.

Figure 3.2: Two solutions used for obtaining vector $(x_{i2} : 1; x_{2i} - 1)$.

In order to obtain the vectors between $i \in V(G \setminus H)$ and $j \in V(H)$ we place the vertex i immediately before and after vertex j. By placing vertex i before and after each vertex $j \in V(H)$, we obtain vectors $(x_{ij} : 1; x_{ji} : -1)$. An example is given in Figure 3.2. The other vertices in $V(G \setminus H \cup \{i\})$ are placed at the end by directing every edge.

This method only gives one vector, which is not enough if $\{i, j\} \notin E$. To obtain a second vector, we place i at the beginning of the ordering by directing every edge from i to vertices in $V(H)$. Then, we remove the arc x_{ij} if it is not present in the original graph. An illustration is given in Figure 3.3.

To obtain vectors among vertices in $V(G \setminus H)$ we use a similar scheme. Consider vertices i, j. We place them at the beginning of the complete digraph with i preceding j in one solution and j preceding i in the other. This way we obtain



(a) A solution where vertex i is placed before the complete digraph.

(b) A solution where vertex i is placed before the complete digraph but with x_{i2} removed.

Figure 3.3: Two solutions used for obtaining vector $(x_{i2} : 1)$ when $\{i, 2\} \notin E$.

vector $(x_{ij} : 1; x_{ji} : -1)$. To obtain vector $(x_{ij} : 1)$ in case $\{i, j\} \notin E$, we place i before the complete digraph and remove the arc (i, j) from the solution. It is possible as i is the first vertex of the ordering.

3.2 Valid inequalities

In this section, we examine the inequalities used in our algorithm, explain the conditions under which they are facet-defining, and present the separation algorithms for each class. Simpliciality and directed cycle inequalities are part of the ILP model. In addition to them, we also present new classes of valid inequalities, namely cyclechord, r-clique, and r-cycle inequalities. Throughout this section, we use $G = (V, E)$ as the graph to be triangulated.

3.2.1 Simpliciality inequalities

Theorem 3.3. *The inequality,*

$$x_{ij} + x_{ik} \leq 1 + X_{jk}. \tag{3.10}$$

is a valid inequality. It is facet-defining for an induced subgraph of three vertices if and only if $\{j,k\} \notin E$.

Proof. Validity. The inequality can only be violated when $x_{ij} = x_{ik} = 1$. However, then $j,k \in N^+(i)$, and thus they should be connected which implies $x_{jk} = 1$ or $x_{kj} = 1$.

Facet-defining. To prove the inequality is facet-defining we enumerate all the tight solutions in Figure 3.4. □

The simpliciality inequalities can be zero-lifted, since they allow for a complete digraph as feasible solution. See Figure 3.4(c).

Note that when $\{j,k\} \in E$, then $X_{jk} = 1$, and thus $x_{ij} + x_{ik} \leq 2$. The inequality is dominated by $x_{ij} \leq 1$ and $x_{ik} \leq 1$ and the simpliciality inequality is not facet-defining.

3.2.2 Directed cycle inequalities

Theorem 3.4. *Consider a directed cycle* D *in* $\vec{H}$ *with* $A(D)$ *as the set of arcs of the cycle. The inequality*

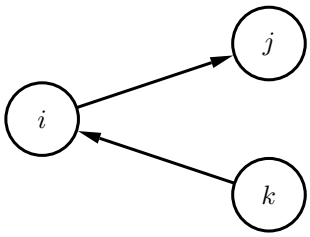
$$\sum_{(i,j)\in A(D)} x_{ij} \leq |A(D)| - 1, \tag{3.11}$$

is valid. It is facet-defining if and only if none of the chords of the cycle is present in E.

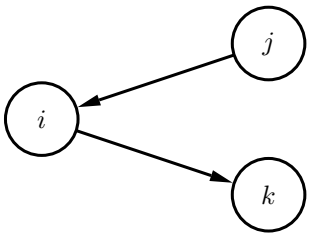
Proof. Validity. The inequality can only be violated when *lhs* is equal to $|A(D)|$. However any feasible digraph corresponding to a partial ordering is acyclic, therefore at least one arc should not be selected.

Facet-defining. To prove that the inequality is facet-defining we consider a cycle D, with vertices $(1,2,\ldots,n,1)$, such the none of the chords is present in E. We obtain the necessary vectors in the following way.

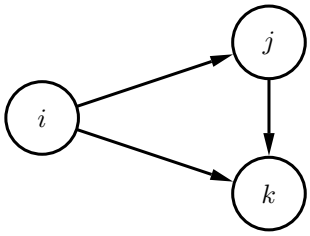
- Order the vertices in the order $(i,i+1,\ldots,i-1)$ by adding all the arcs. By deleting arcs $(i,i+2),(i,i+3),\ldots,(i,i-2)$ one by one, we obtain $n-3$ vectors: $(x_{i,i+2}:1),(x_{i,i+3}:1),\ldots,(x_{i,i-2}:1)$. By placing different vertices at the beginning of the ordering we obtain 2 vectors for each chord of the cycle. In total we obtain $(n-3)n$ vectors and we do not consider the chords in the rest of the proof. An example is presented in Figure 3.5 for vector $(x_{1,n-1}:1)$.



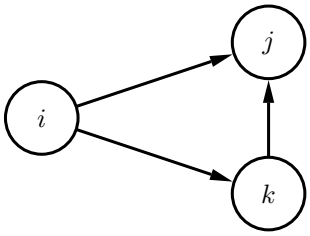
(a) No restriction on the existence of the edges.



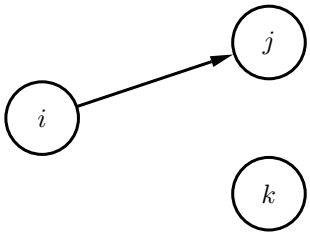
(b) (b)-(a) gives $(x_{ji}, x_{ik} : 1; x_{ij}, x_{ki} : -1)$.



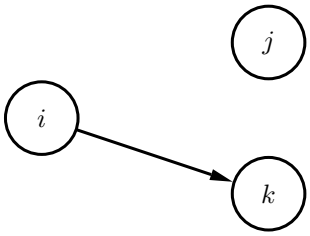
(c) (c)-(b) gives $(x_{ij}, x_{jk} : 1; x_{ji} : -1)$.



(d) (d)-(c) gives $(x_{jk} : 1; x_{kj} : -1)$.

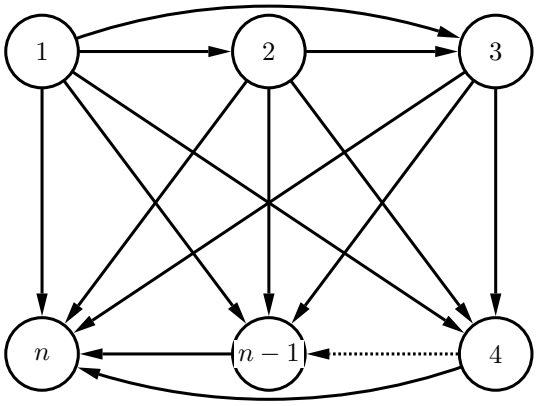


(e) Only valid if $\{i, k\} \notin E$. (a)-(e) gives $(x_{ki} : -1)$.

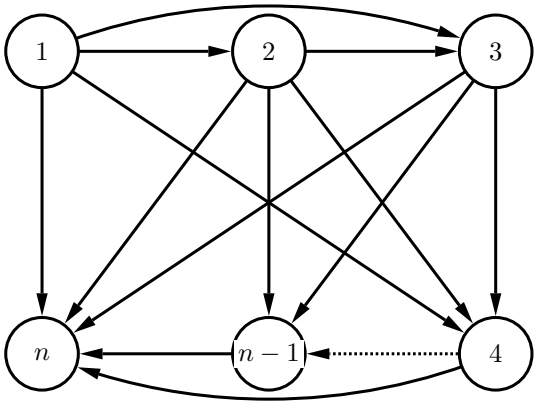


(f) Only valid if $\{i, j\} \notin E$. (b)-(f) gives $(x_{ji} : -1)$.

Figure 3.4: Tight solutions for the simpliciality constraint.



(a) Ordering $(1, 2, \ldots, n)$ with all the possible arcs.



(b) Ordering $(1, 2, \ldots, n)$ without the arc $x_{1,n-1}$.

Figure 3.5: Subtracting two solution vectors gives vector $(x_{1,n-1} : 1)$.

- We consider two solutions with the following orderings: $(1, 2, \ldots, n)$ and $(2, 3, \ldots, n, 1)$. By subtracting the first solution from the second one we obtain vector $(x_{12}, x_{1n} : 1; x_{21}, x_{n1} : -1)$. By considering the ordering $(3, 4, \ldots, n, 1, 2)$ we obtain vector $(x_{23}, x_{1n} : 1; x_{32}, x_{n1} : -1)$ and by considering the ordering $(n, 1, \ldots, n-2, n-1)$ we obtain vector $(x_{n-1,n}, x_{1n} : 1; x_{n,n-1}, x_{n1} : -1)$, which in total gives us $n-1$ vectors. An example is shown in Figure 3.6.

- For the arcs of the cycle for which the underlying edges are not present in E we can find a second vector in the following way. Considering the ordering $(i, i+1, \ldots, i-1)$ and that $\{i, i-1\} \notin E$, we can create a partial ordering where the arc $(i, i-1)$ is omitted as shown in Figure 3.7 to obtain vector $(x_{1n} : 1)$. By changing the ordering we can also obtain vectors $(x_{21} : 1), (x_{32} : 1), \ldots, (x_{n,n-1} : 1)$ providing the related edge is not in the original graph. By this operation, we obtain $n - m$ vectors.

In total we obtain $(n-3)n + n - 1 + n - m = n(n-1) - m - 1$ vectors. □

When the chord $\{i, j\}$ is present, the corresponding cycle inequality can be obtained by adding up the following constraints as illustrated in Figure 3.8:

$$\sum_{a \in A(D_1)} x_a \leq |A(D_1)| - 1,$$
$$\sum_{a \in A(D_2)} x_a \leq |A(D_2)| - 1,$$
$$-X_{ij} = -1.$$

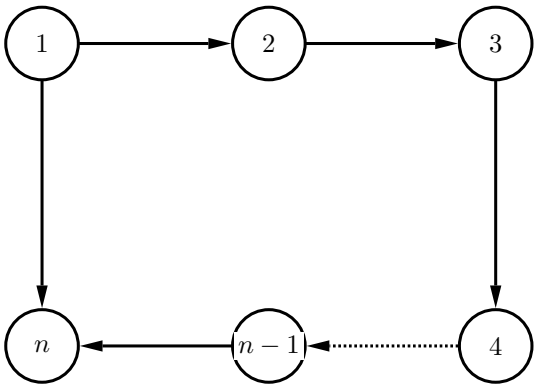
Note that $|A(D)| = |A(D_1)| + |A(D_2)| - 2$.

The directed cycle inequalities allow for zero-lifting, since there exists a complete digraph that satisfies the inequality at equality as shown in Figure 3.5(a).

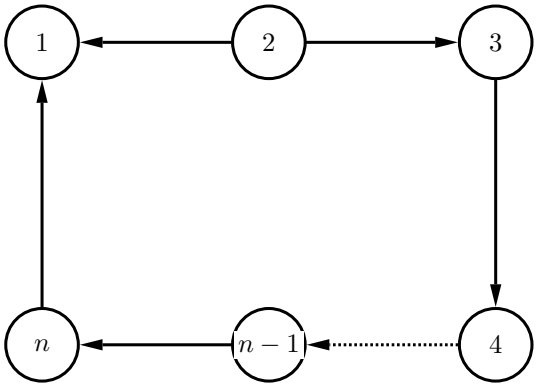
Separation routine for the directed cycle inequalities

The cycle inequalities can be exponential in number so they are not added at the beginning of the branch-and-cut algorithm but instead they are separated, in polynomial time, when we have a fractional or integer solution. The inequality is violated when

$$\sum_{(i,j) \in A(D)} x_{ij} > |A(D)| - 1.$$

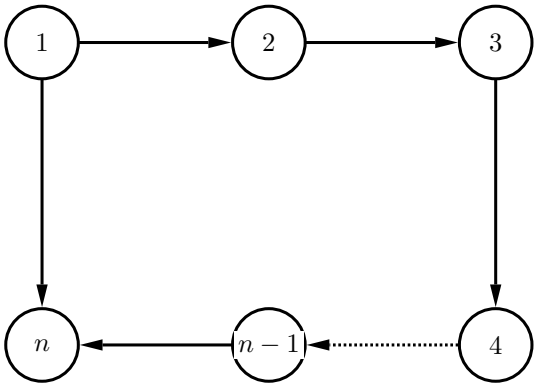


(a) Ordering $(1, 2, \ldots, n)$.

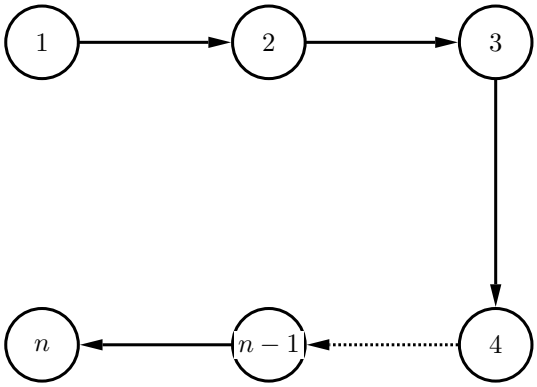


(b) Ordering $(2, 3, \ldots, 1)$.

Figure 3.6: Subtracting two solution vectors gives us vector $(x_{12}, x_{1n} : 1; x_{21}, x_{n1} : -1)$. The chords are not shown in the solutions.



(a) Ordering $(1, 2, \ldots, n)$ with arc$(1, n)$.



(b) Ordering $(1, 2, \ldots, n)$ without arc $(1, n)$.

Figure 3.7: Subtracting two solutions gives us vector $(x_{1n} : 1)$ providing $\{1, n\} \notin E$.

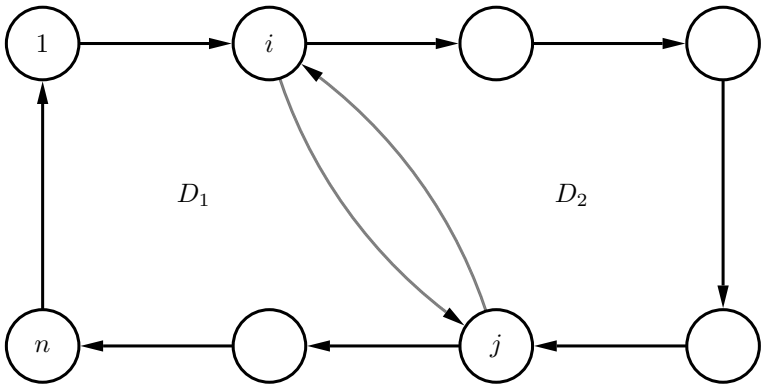


Figure 3.8: Two cycle inequalities for cycles D_1, D_2 and the existence of edge $\{i, j\}$ imply the cycle inequality for the big cycle.

By subtracting $|A(D)|$ from both sides and multiplying the inequality by -1 we reach

$$\sum_{(i,j)\in A(D)} (1 - x_{ij}) < 1.$$

To find violated inequalities, we create a directed graph using the solution to the problem and solve a shortest path problem on this graph. For each vertex $i \in V$, we create two copies i, i' and connect only i to i'. We use $d(i, j)$ to denote the distance between vertices i and j. In the shortest path graph, we set $d(i, i') = 0$ and for $x_{ij} > 0$ we assign $d(i', j) = 1 - x_{ij}$. If the shortest distance from i' to i is less than 1, there is a violated inequality. The shortest path will be $(i', j, j', k, k', \ldots, l, l', i)$. The distance for the shortest path corresponds to $(1 - x_{ij}) + 0 + (1 - x_{jk}) + 0 + \ldots + (1 - x_{li}) + 0$. The cycle we obtain in $\vec{H}$ is $(i, j, k, \ldots, l)$. At each iteration, when a violated directed cycle is found, we also add the reverse cycle to the model. Furthermore when a violated inequality is found we set $d(i, i') = \infty$ in order to avoid finding the same cycle later on. The separation routine only finds the most violated cycle passing through a vertex but it is called until all the directed cycle inequalities are satisfied.

The shortest path graph has $2n$ vertices and by using Dijkstra's shortest path algorithm we can find the shortest path in $\mathcal{O}^*(n^3)$ time.

3.2.3 Additional valid inequalities

Simpliciality and directed cycle inequalities are part of the ILP formulation. Now, we present three additional classes of valid inequalities.

Cyclechord inequalities

Theorem 3.5. *A cycle C in the supergraph H is a sequence of vertices $V(C) = (1, 2, \ldots, n)$ that starts and ends at the same vertex. We denote the underlying edges of the cycle in H by $E(C) = \{\{1, 2\}, \{2, 3\}, \ldots, \{n, 1\}\}$. The inequality*

$$\sum_{\{i,j\} \in E(C)} X_{ij} \leq |V(C)| - 1 + \sum_{i,j \in V(C), \{i,j\} \notin E(C)} \frac{X_{ij}}{|V(C)| - 3}, \tag{3.12}$$

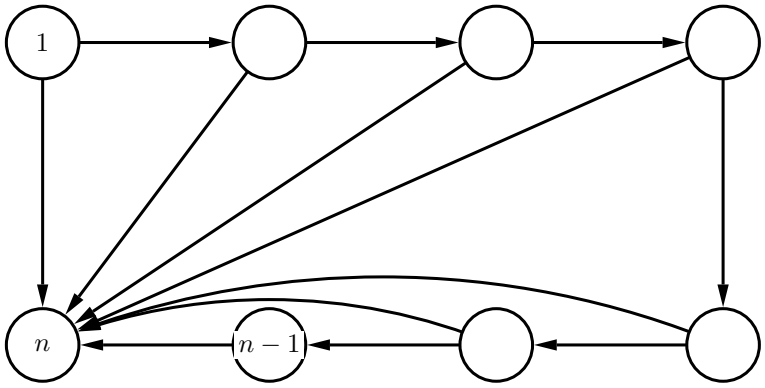
is valid. It is facet-defining in the induced subgraph C if and only if none of the chords are present in $E(C)$.

Proof. Validity. The inequality can only be violated if $\sum_{\{i,j\} \in E(C)} X_{ij} = |V(C)|$, which implies the existence of a cycle in the supergraph H. A cycle must be triangulated by using at least $|V(C)| - 3$ chords.

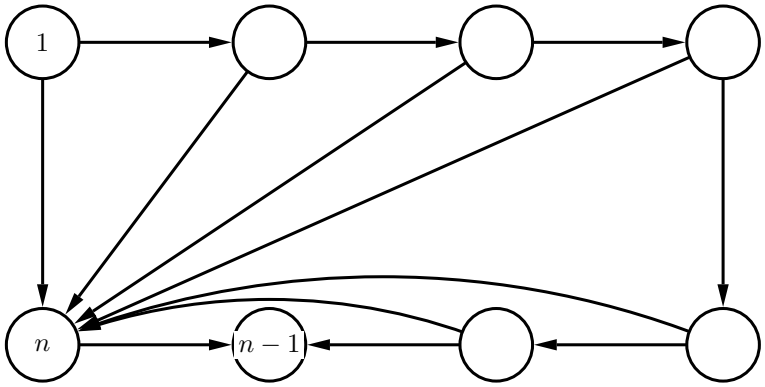
Facet-defining. Note that in the inequality we can eliminate any vertex by connecting its neighbors in the graph. By that logic, we can isolate any four vertices and place them at the end of the ordering. To prove that the inequality is facet-defining in C we consider the following vectors:

- Order the vertices in the order $(1, 2, \ldots, n-1, n)$ by using $|V(C)| - 3$ chords. As vertices n and $n-1$ are at the end of the ordering we can change their order as shown in Figure 3.9 to obtain vector $(x_{n-1,n} : 1; x_{n,n-1} : -1)$. By using different orderings we can obtain in total n vectors in the form of $(x_{12} : 1; x_{21} : -1), (x_{23} : 1; x_{32} : -1), \ldots, (x_{n1} : 1; x_{1n} : -1)$.

- In this part of the proof we create subgraphs of four vertices by selecting three consecutive vertices and another vertex of the cycle such as $\{i, i+1, i+2, j\}$. We then create one solution where $x_{i,i+2} = 1$ and a second solution where $x_{i+1,j} = 1$. For example, we can select vertices $\{1, 2, 3, 4\}$ to obtain vector $(x_{13} : 1; x_{24} : -1)$ or we can select $\{1, 2, 3, 5\}$ to obtain vector $(x_{13} : 1; x_{25} : -1)$ as shown in Figure 3.10. In the solution we have to reverse the arc $(i, i+1)$ but we already obtain the vector associated with this change in the first part. By doing the same thing for other consecutive triplets we obtain



(a) Ordering $(1, 2, \ldots, n-1, n)$.



(b) Ordering $(1, 2, \ldots, n, n-1)$

Figure 3.9: Two solutions used to obtain vector $(x_{n-1,n} : 1; x_{n,n-1} : -1)$.

$n(n-3)$ vectors. The vectors obtained by consecutive quartets of vertices, $(x_{13}:1; x_{24}:-1), (x_{24}:1; x_{35}:-1), \ldots, (x_{n2}:1; x_{13}:-1)$, add up to the zero vector and therefore they are linearly dependent. To obtain linearly independent vectors we have to omit one of them. The other vectors $(x_{13}:1; x_{25}:-1), \ldots, (x_{13}:1; x_{2n}:-1), (x_{24}:1; x_{36}:-1), \ldots, (x_{n-1,1}:1; x_{n,3}:-1)$ are linearly independent as the variables with a negative coefficient appear only once in the vectors. In total we obtain $n(n-3)-1$ vectors concerning the chords of the cycle.

- Order the vertices in the order $(1, 2, \ldots, n)$ by using n and $n-1$ cycle arcs. This way we obtain vector $(x_{1n}, x_{2n}, \ldots, x_{n-2,n} : 1)$ as shown in Figure 3.11. In total we obtain $n-m$ vectors.

  The proof gives $n + n(n-3) - 1 + n - m = n(n-1) - m - 1$ vectors.

□

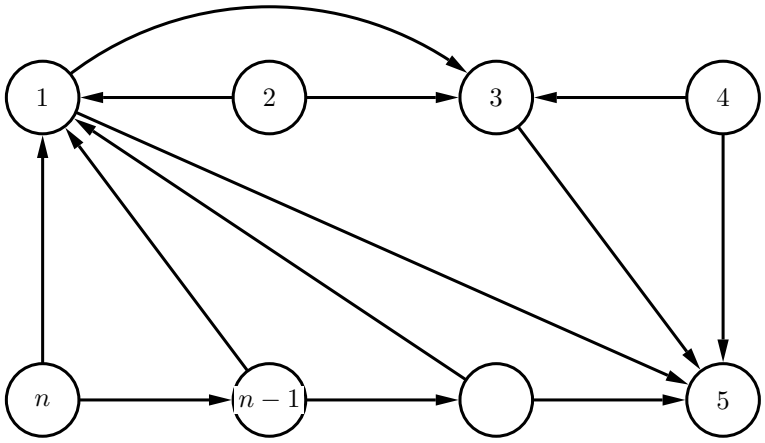
All the tight solutions for the inequality have either $\sum_{\{i,j\}\in E(C)} X_{ij} = |V(C)|-1$ or $\sum_{\{i,j\}\in E(C)} X_{ij} = |V(C)|$. In the latter case the inequality adds the minimum number of necessary chords to the cycle, e.g. the cycle $\{1,2,3,4\}$ only admits one of the edges $\{1,3\}$ and $\{2,4\}$ in the tight solutions for the inequality. When a chord is present in the graph, then the chords that are crossing this chord must be all zero to create a minimum triangulation of the cycle. If in the cycle $\{1,2,3,4,5\}$ the edge $\{1,3\}$ is present then $x_{24} = x_{42} = x_{25} = x_{52} = 0$ in all the tight solutions and the cyclechord inequality for the corresponding graph is dominated the domain inequalities, such as $x_{24} \geq 0$. Cyclechord inequalities do not admit a solution that forms a complete digraph, which means they may not necessarily be zero-lifted. The cyclechord inequality for the cycle $\{1,2,3,4\}$ in the graph shown in Figure 3.12 is not facet-defining. We now examine conditions under which they are facet-defining in $P(G)$.

Theorem 3.6. *Inequality (3.12) is facet-defining in the original graph G if for any pair of vertices $k, l \in V(G \setminus C)$,*

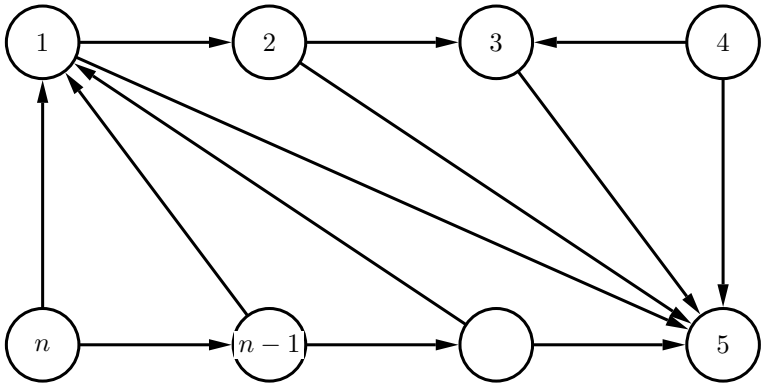
- *either k or l has at most 3 neighbors in the cycle, or*
- *$\{k,l\} \in E$.*

Proof.

- Reusability of the vectors in C as described in the zero-lifting theorem.

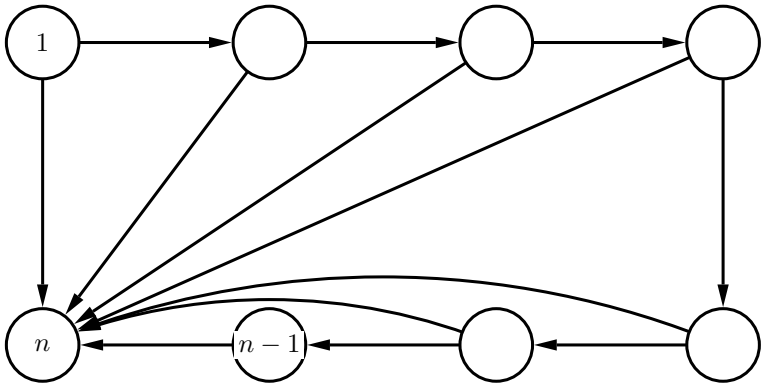


(a) Solution with chord x_{13}.

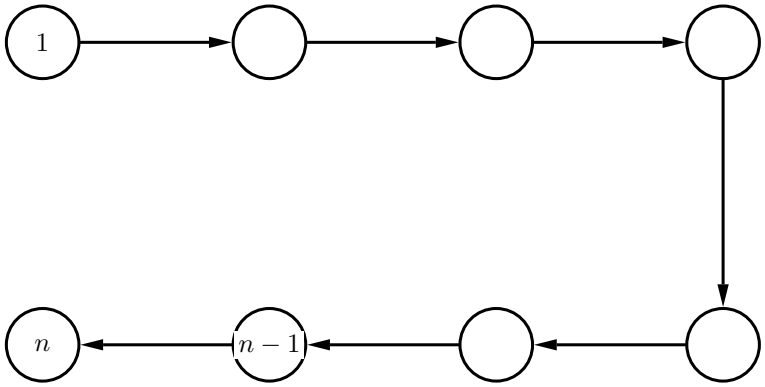


(b) Solution with chord x_{25}.

Figure 3.10: Two solutions used to obtain vector $(x_{13} : 1; x_{25} : -1)$ by placing vertices $\{1, 2, 3, 5\}$ at the end of the ordering. Arc $(1, 2)$ is reversed, but we can ignore it because of the first part of the proof.



(a) Ordering $(1, 2, \ldots, n-1, n)$ with the chords.



(b) Ordering $(1, 2, \ldots, n-1, n)$ without the chords and arc $(n, 1)$.

Figure 3.11: Two solutions used for obtaining vector $(x_{1n}, x_{2n}, \ldots, x_{n-2,n} : 1)$. Note that we do not have to consider the chords of the cycle.

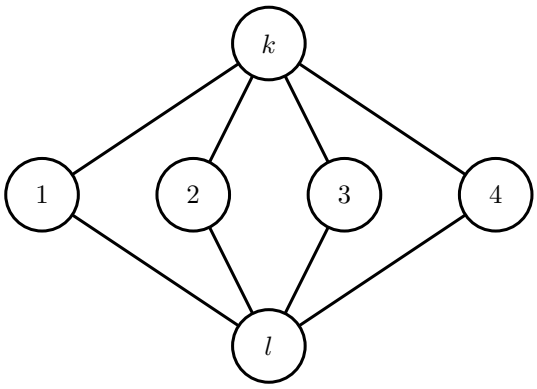


Figure 3.12: Cyclechord inequality for the cycle $\{1, 2, 3, 4\}$ is not facet-defining in this graph.

- Vectors between $k \in V(G \setminus C)$ and $i \in V(C)$: Consider any ordering for the cycle vertices that is tight for the inequality. By placing vertex k before and after the last vertex of the ordering we obtain vector $(x_{ik} : 1; x_{ki} : -1)$. Any vertex of the cycle can be the last element of the ordering, which means we can create these vectors for every vertex $i \in V(C)$.

  To obtain the unit vector again use any ordering that is tight for the inequality and place vertex k at the end of the ordering by directing every edge from $V(C)$ to k. By deleting the edge between the first vertex in the ordering and k we obtain unit vector $(x_{ik} : 1)$ if the edge is not in the graph. We can choose any vertex in the cycle to be the first vertex of the ordering and therefore create the unit vectors when the edges are not present.

- Vectors between $k, l \in V(G \setminus C)$: Vector $(x_{kl} : 1; x_{lk} : -1)$ is obtained by using orderings (C, k, l) and (C, l, k). The ordering of the vertices in C must be tight for the inequality. Note that it is possible to direct all the arcs from the vertices in C to k and l. If $\{k, l\} \in E$ then this is enough.

  For the second case, we assume $N(k) = \{n-2, n-1, n\}$. We can place any three vertices at the end of the cycle so we do not lose generality. By placing k before vertices in $N(k)$ and l after the vertices of the cycle we obtain a solution where (k, l) can be selected or omitted to obtain unit vector $(x_{kl} : 1)$.

□

We already mentioned the cyclechord inequality is not facet-defining for cycle $\{1,2,3,4\}$ in the graph in Figure 3.12. In order to prove that we consider cyclechord inequalities for the following cycles in the graph: $\{1,2,3,4\}$, $\{1,5,3,6\}$, $\{2,5,4,6\}$.

$$X_{12} + X_{23} + X_{34} + X_{14} \leq 3 + X_{13} + X_{24},$$
$$X_{15} + X_{35} + X_{36} + X_{16} \leq 3 + X_{13} + X_{56},$$
$$X_{25} + X_{45} + X_{46} + X_{26} \leq 3 + X_{24} + X_{56}.$$

Furthermore we know that

$$X_{12} + X_{23} + X_{34} + X_{14} + X_{15} + X_{16} + X_{25} + X_{26} + X_{35} + X_{36} + X_{45} + X_{46} \leq 12.$$

If we add up all inequalities and divide the resulting inequality by 2 we obtain:

$$X_{12} + X_{23} + X_{34} + X_{14} + X_{15} + X_{16} + X_{25} + X_{26} + X_{35} + X_{36} + X_{45} + X_{46} \leq 10 + X_{13} + X_{24} + X_{56}.$$

The inequality is valid by construction and it is facet-defining when the edges on the *rhs* are not present in the graph. When an edge on the *rhs* is present the inequality boils down to the original cyclechord inequality. When $\{5,6\} \notin E$ and $N(5) = N(6) = \{1,2,3,4\}$, we obtain a stronger inequality that dominates the cyclechord inequality for the cycle $\{1,2,3,4\}$,

$$X_{12} + X_{23} + X_{34} + X_{14} \leq 2 + X_{13} + X_{24} + X_{56}.$$

When $N(k) = N(l)$ and $|N(k)| \geq 4$, the solutions that are tight for the original cyclechord inequality have $X_{kl} = 1$. Assume $X_{kl} = 0$, it implies that vertices in $N(k)$ can precede at most one of k or l. So k, l, or both of them should precede the all of the vertices in $N(k)$. This makes the vertices in $N(k)$ a complete digraph, which is not a minimal triangulation of the cycle, and therefore is not a tight solution for the cyclechord inequality.

Theorem 3.7. *The following inequality is facet-defining in $C \cup \{k,l\}$ when $N(k) = N(l)$, $|N(k)| \geq 4$ and $(k,l) \notin E$:*

$$\sum_{\{i,j\}\in E(C)} X_{ij} \leq |V(C)| - 1 + \sum_{i,j\in V(C),\{i,j\}\notin E(C)} \frac{X_{ij}}{|V(C)|-3} - \frac{c(1-X_{kl})}{|V(C)|-3}, \tag{3.13}$$

where $c = \frac{(|N(k)|-2)(|N(k)|-3)}{2}$*, which is the number of additional chords we have to add to the graph in order to have* $X_{kl} = 0$*, or equivalently to have* $N(k)$ *a complete digraph.*

Proof. Validity. When $X_{kl} = 1$ the inequality becomes the regular cyclechord inequality. When $X_{kl} = 0$, we have to add c additional edges to make the vertices in $N(k)$ a complete digraph.

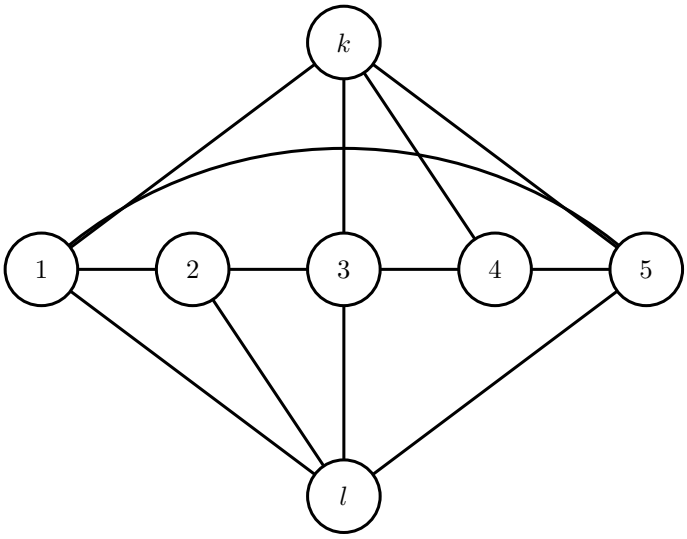
Facet-defining. To prove the inequality is facet-defining we only need to obtain unit vector $(x_{kl} : 1)$. We construct an ordering where k precedes the vertices in $N(k)$ and cycle vertices precede l. It is possible to use or to omit arc (k, l) and we obtain the necessary vector for the proof. □

When the neighbors of k and l are not the same vertices of the cycle, the cyclechord inequality may or may not be facet-defining depending on the edges of the graph. In the graph in Figure 3.13(a) the original cyclechord inequality is facet-defining and by adding chords $\{1, 3\}$ and $\{3, 5\}$ we can triangulate the graph. For the graph in Figure 3.13(b) the cyclechord inequality is valid but is not facet-defining. In all the tight solutions we have $X_{kl} = 1$.

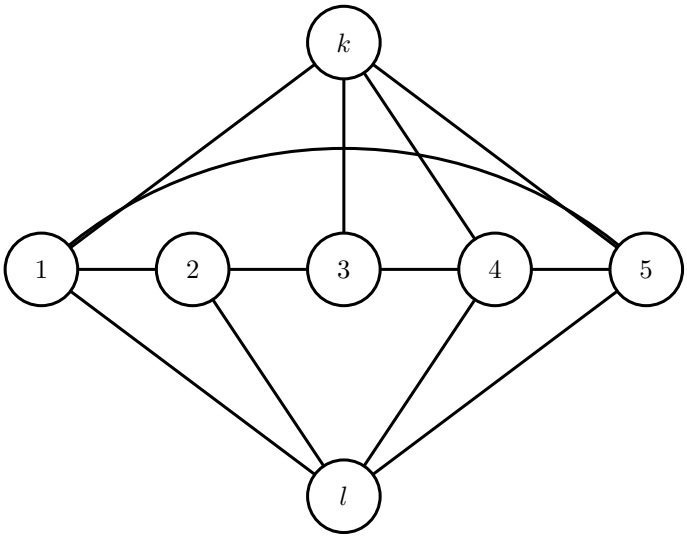
- k or l cannot be the first vertex of the ordering as it would require adding more than the minimum number of chords.
- Vertices 1,4, and 5 cannot be the first vertex of the ordering as they precede both k and l.
- So vertex 2 (vertex 3 is a symmetric case) is the first in the ordering. By eliminating vertex 2, we have to add the edge $\{3, l\}$ to the graph ($\{2, k\}$ for vertex 3). This creates a fourth common neighbor between k and l.

In this case, the following inequality is facet-defining:

$$2(X_{12} + X_{23} + X_{34} + X_{45} + X_{15}) \leq 7 + X_{13} + X_{14} + X_{24} + X_{25} + X_{35} + X_{kl}.$$



(a) The original inequality is facet-defining in this graph.



(b) The inequality is not facet-defining in this graph.

Figure 3.13: Two cases where $|N(k)| = |N(l)| = 4$.

Separation routine for the cyclechord inequalities

In our implementation we focus on cycles of size 4. To separate the violated cyclechord inequalities, we use enumeration. By enumerating all quartets we can find the violated cyclechord inequalities. However, the extended version of the cyclechord inequalities are quite powerful, especially on dense graphs, and for that reason we also want to separate them.

After finding a 4 cycle we check the value of $X_{12}+X_{23}+X_{34}+X_{14}-X_{13}-X_{24}$. If it is greater than 3 the inequality is added to the model. We also check whether there are two additional vertices k, l such that $\{k, l\} \notin E$ and the extended version of the inequality is violated. Note that k, l do not have to be connected to the cycle in the original graph. If we can find two such vertices we check the validity of the extended inequality and add it to the model if necessary. We explain the separation routine in Algorithm 1.

For sparse graphs, such as mycielski or grid graphs, finding small cycles is not enough to obtain a good lower bound. For that reason we enumerate all chordless cycles in the graph and add the corresponding cyclechord inequality to the constraint pool if the graph has at most 40 vertices.

r-clique inequalities

Theorem 3.8. *Let S be a set of n vertices forming an independent set and let r be another vertex. Let s be an arbitrary number such that $2 \leq s < n-1$. Then the inequality*

$$(s-1)\sum_{i\in V(S)} x_{ri} \leq \binom{s}{2} + \sum_{(i,j)\in E(S)} X_{ij}, \tag{3.14}$$

is valid and facet-defining in the graph H with $V(H) = V(S) \cup \{r\}$.

Proof. Validity. Let $T \subset S$ be the set of vertices i with $x_{ri} = 1$. We let t be the size of T, i.e., $t = \sum_{i\in S} x_{ri}$. Since the out-neighborhood of r must be a complete digraph, we must add at least $\binom{t}{2}$ fill-in edges to the rhs. Now $rhs - lhs$ is at least

$$\frac{s(s-1)}{2} + \frac{t(t-1)}{2} - (s-1)t = \frac{1}{2}(s-t)(s-t-1),$$

which is nonnegative for any t between 0 and n. Furthermore, equality can only be true when $t = s-1$ or $t = s$.

```
input  : A fractional solution to LP x
output: A set of violated inequalities
for i, j, k ∈ V|{j, k} ∉ E do
    ineq1 ← X_ij + X_ik − X_jk;
    for l ∈ V \ {i, j, k}|{i, k} ∉ E do
        ineq2 ← ineq1 + X_jl + X_kl − X_ik;
        cycle ← {j, i, k, l};
        if ineq2 > 3 then
            add_cycle (cycle);
        end
        else if ineq2 > 2 then
            for m, n ∈ V\cycle |{m, n} ∉ E do
                ineq3 ← ineq2 + X_jm + X_im + X_km + X_lm + X_jn + X_in +
                X_kn + X_ln − X_nm;
                if ineq3 > 10 then
                    add_extended_cycle (cycle,m, n);
                end
            end
        end
    end
end
```

Algorithm 1: Separation routine for cyclechord inequalities

Facet-defining. If there exists an edge, $\{i, j\} \in E$ for some $i, j \in S$, then the only way to obtain a tight solution of (3.14) is by putting i, j later than r, thus setting $x_{ri} = x_{rj} = 1$. Therefore, the face of the polytope defined by the inequality $x_{ri} \leq 1$ contains the face of the r-clique inequality (3.14).

The digraphs that we create satisfying (3.14) at equality are defined by the position of r in the ordering. We partition S in a set F before r, and a set B after r, where B contains s or $s-1$ vertices. The order will then be (FrB). The arcs are defined as follows: For $i \in F$ we have only one arc (i, r), and the vertices in $r + B$ will have a complete ordering, starting with r.

- Select two arbitrary vertices $i, j \in S$. Let $i, j \in B$. We denote by $B_{ij} = B \backslash \{i, j\}$. The orders $(FrijB_{ij})$ and $(FirjB_{ij})$ give vector $(x_{ij}, x_{ri}, x_{ik} | k \in B_{ij} : 1; x_{ir} - 1)$. In total we obtain $n(n-1)$ vectors. An example is shown in Figure 3.14.

- Select a special vertex say 1, and an arbitrary vertex $j \in S$. Let $1, j \in B$. The orders $(Fr1jB_{1j})$ and $(Frj1B_{1j})$ give vector $(x_{1j} : 1; x_{j1} : -1)$. In total there are $n-1$ vectors of this type. An example is shown in Figure 3.15.

- Consider a vertex $i \in S$ with $\{i, r\} \notin E$. Put i in F. Consider the order $(F_i irB)$. Besides the standard solution, we consider the solution where $(i, r) \notin A$. This gives vector $(x_{ir} : 1)$ if $\{i, r\} \notin E$. In total these are $n - m$ vectors as shown in Figure 3.16.

In total we obtain $n(n-1) + n - 1 + n - m = (n+1)n - m - 1$ vectors for the subgraph $\{r\} \cup S$. □

The inequality (3.14) does not admit a tight solution that corresponds to a complete ordering in $\{r\} \cup S$ and therefore is not necessarily facet-defining in G.

Theorem 3.9. *Inequality (3.14) is valid and facet-defining in the original graph G if for every pair of vertices $k, l \in V(G \setminus S)$:*

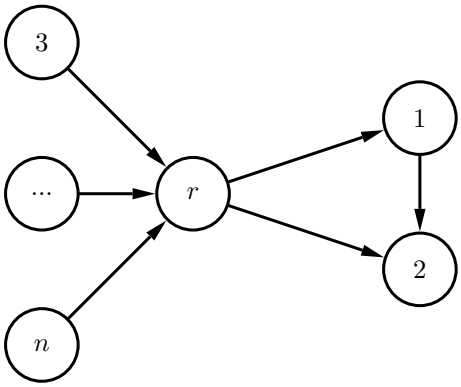
- *either $|N(k) \cap N(l)| \leq s$, where $N(k) = \{v \in V(S) | \{k, v\} \in E\}$,*

- *or $\{k, l\} \in E$.*

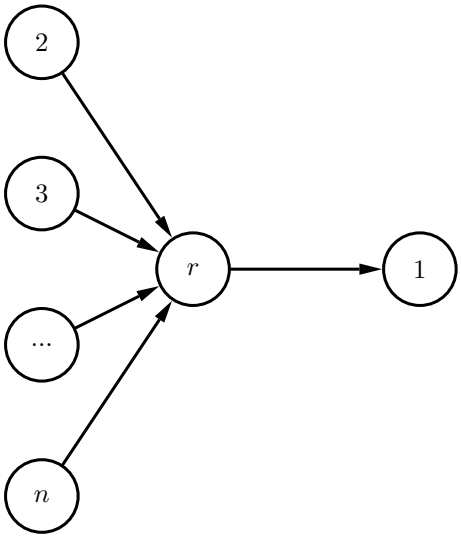
Proof.

- Reusability of the vectors in S as described in the zero-lifting theorem.

- Vectors between $k \in V(G \backslash S)$ and $j \in V(S)$: Place any s vertices in B. Order the vertices in order $\{FrkjB_j\}$. By using the second ordering $(FrjkB_j\}$ we

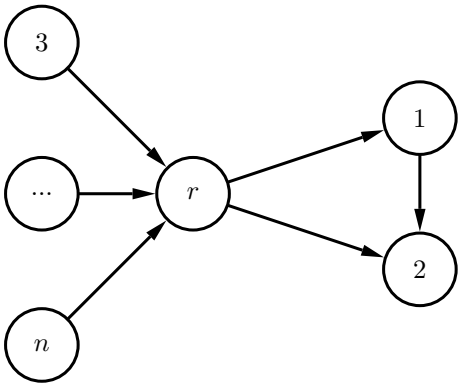


(a) Tight solution with $x_{12} = 1$.

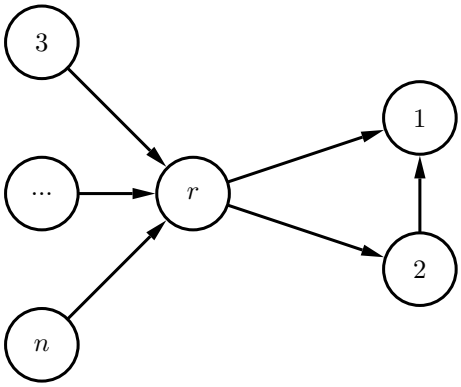


(b) Tight solution with $x_{12} = 0$.

Figure 3.14: Subtracting two solution vectors gives us vector $(x_{12}, x_{r2} : 1; x_{2r} : -1)$.

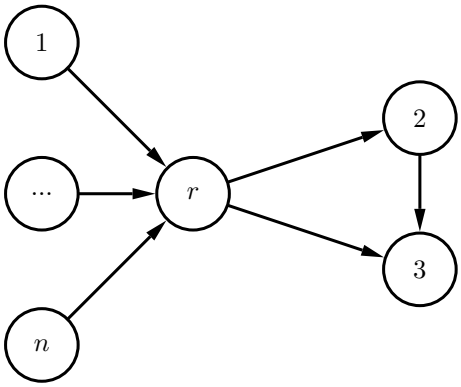


(a) Tight solution with $x_{12} = 1$.

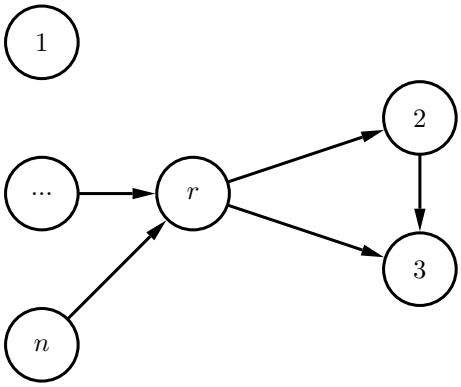


(b) Tight solution with $x_{21} = 1$.

Figure 3.15: Subtracting two solutions gives us vector $(x_{12} : 1; x_{21} : -1)$.



(a) Tight solution with $x_{1r} = 1$.



(b) Tight solution with $x_{1r} = 0$.

Figure 3.16: Subtracting two solution vectors gives us vector $(x_{1r} : 1)$ if $(1, r) \notin E$.

obtain vector $(x_{kj} : 1; x_{jk} : -1)$. By placing k before and after the other vertices of the complete digraph, we obtain other vectors. To obtain the second vector, we follow the same scheme by using the ordering $(FkrB)$, where $j \in B$. We are allowed to omit arc (k, j) to obtain vector $(x_{kj} : 1)$ if the corresponding edge is not present in E.

- Vectors between $k \in V(G \backslash S)$ and $l \in V(G \backslash S)$: Note that in this case we have $r \in V(G \setminus S)$. If $r = k$, we use the orderings $\{FlrB\}$ and $\{FrlB\}$ to obtain vector $(x_{lr} : 1; x_{rl} : -1)$. If $\{l, r\} \notin E$, we set B such that $N(l) \cap N(r) \subseteq B$. In this case vertices in F are connected to at most one of l and r and we can choose to direct or omit arc (l, r) to obtain vector $(x_{lr} : 1)$. When $r \neq k$ and $r \neq l$ it is possible to place them after r and S with any ordering. This is sufficient if $\{k, l\} \in E$. If $\{k, l\} \notin E$ we use the ordering $(FklrB\}$, with $N(k) \cap N(l) \subseteq B$. It is possible to set $x_{kl} = 0$ as vertices in F are connected to at most one of k and l, which gives us vector $(x_{kl} : 1)$.

□

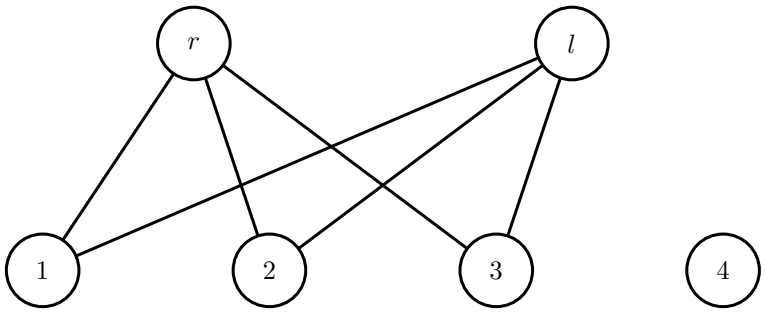
When $|N(k) \cap N(l)| \geq s + 1$ and $\{k, l\} \notin E$, it is impossible to place k or l before all of the vertices in $N(k) \cap N(l)$ as it would be impossible to create a tight solution. So at least one of the vertices in $N(k) \cap N(l)$ must be placed before k and l, which means it is not possible to omit the edge $\{k, l\}$ in the solution. In this case we have to extend the inequality.

Theorem 3.10. *The following inequality is facet-defining in the induced subgraph* $\{S, r, k, l\}$ *when* $|N(k) \cap N(l)| \geq s + 1$ *and* $\{k, l\} \notin E$*:*

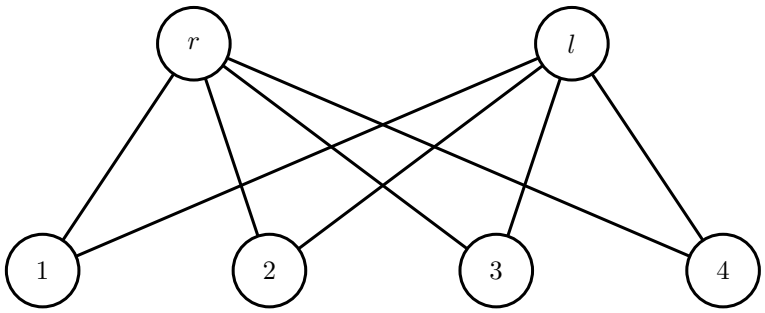
$$(s-1) \sum_{i \in V(S)} x_{ri} \leq \binom{s}{2} + \sum_{(i,j) \in E(S)} X_{ij} - c(1 - X_{kl}), \tag{3.15}$$

where $c = \frac{(|N(k) \cap N(l)| - s + 1)(|N(k) \cap N(l)| - s)}{2}$.

Proof. Validity. When $X_{kl} = 1$ the inequality boils down to the regular r-clique inequality. When $X_{kl} = 0$ then either k or l should precede all the vertices in $N(k) \cap N(l)$, which requires $\binom{|N(k) \cap N(l)|}{2}$ edges to triangulate the subgraph without using the edge $\{k, l\}$. The value c is chosen in a way that yields a tight solution where $N(k) \cap N(l)$ is a complete digraph. □



(a) $N(k) \cap N(l) = \{1, 2, 3\}$.



(b) $N(k) \cap N(l) = \{1, 2, 3, 4\}$.

Figure 3.17: Two graphs where the original r-clique inequality has to be lifted to be a facet.

For the graph in Figure 3.17(a) the r-clique inequality for $s = 2$ can be extended as follows:

$$\sum_{i\in\{1,2,3,4\}} x_{ri} \leq 0 + \sum_{i,j\in\{1,2,3,4\}} X_{ij} + X_{rl}.$$

For the graph shown in Figure 3.17(b) the r-clique inequality for $s = 2$ can be extended as

$$\sum_{i\in\{1,2,3,4\}} x_{ri} \leq -2 + \sum_{i,j\in\{1,2,3,4\}} X_{ij} + 3X_{rl},$$

and for $s = 3$ as

$$2\sum_{i\in\{1,2,3,4\}} x_{ri} \leq 2 + \sum_{i,j\in\{1,2,3,4\}} X_{ij} + X_{rl}.$$

A general form for the lifting the inequality is: c is the difference between the right-hand side and the left-hand side of the original inequality when we force the vertices in $N(k) \cap N(l)$ to be part of a complete digraph.

Separation routine for the r-clique inequalities

To separate r-clique inequalities, we first fix vertex r. We, then, order the vertices in the out-neighborhood of r in non-increasing order of values x_{ri} by omitting the vertices with $x_{ri} = 1$. We try to find $l \geq 3$ vertices, that violate the inequality. To find an independent set of size l we spend $\mathcal{O}^*(n^l)$ time and the whole separation routine takes $\mathcal{O}^*(n^{l+1})$ time for n different values of r.

r-cycle inequalities

Theorem 3.11. *Let $C = (V(C), A(C))$ be a directed cycle and r an additional vertex in $V(G \setminus C)$, then the inequality*

$$\sum_{i\in V(C)} x_{ri} \leq |C| - 1 + \sum_{(i,j)\in A(C)} x_{ij}, \tag{3.16}$$

is valid and facet-defining if C does not have any edges.

Proof. Validity. The inequality can only be violated when $\sum_{i\in V(C)} x_{ri} = |C|$ and $\sum_{(i,j)\in A(C)} x_{ij} = 0$. In this case every edge in the cycle C must be directed as a

result of the simpliciality constraints and at least one cycle edge should be directed in the direction of the cycle because of the model cycle inequalities.

Facet-defining.

- Order the vertices in the order $(r, n-1, \ldots, 2, 1)$ by adding all edges and place n at the beginning, by only setting $x_{nr} = 1$. We obtain vectors $(x_{n2} : 1), (x_{n3} : 1), (x_{n,n-2} : 1), (x_{n,n-1} : 1)$ by directing and omitting corresponding arcs. Note that$\{n, 2\}, \ldots, \{n, n-2\}$ are the chords of the cycle C. By replacing n with the other vertices we obtain $(n-3)n$ vectors for the chords of the cycle and we do not consider them in the remaining parts of the proof. We also use this scheme to obtain vector $(x_{n,n-1} : 1)$, which gives another n linearly independent vectors. Furthermore, if $\{n, r\} \notin E$ we can set $x_{nr} = 0$ to obtain $n - m$ vectors in the form of $(x_{nr} : 1)$. An example is presented in Figure 3.18.

- Order vertices in the order $(r, n, n-1, \ldots, 2, 1)$ by adding all edges. By placing n at the beginning of the ordering by only setting $x_{nr} = 1$, we obtain vector $(x_{rn}, x_{n1}, x_{n,n-1} : 1; x_{nr} : -1)$. We can replace n by any vertex of the cycle to obtain n vectors that are linearly independent among themselves as illustrated in Figure 3.19.

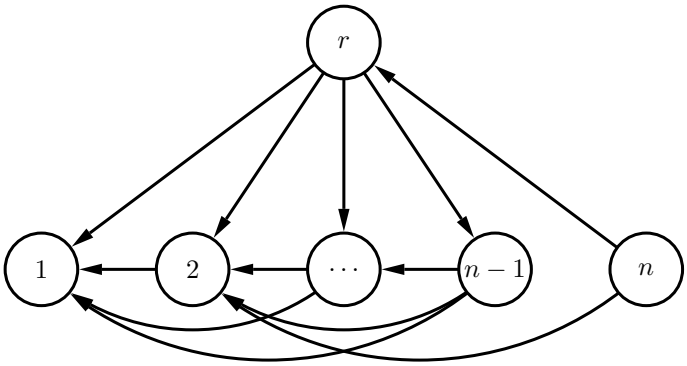
- Order vertices in the order $(r, n-1, \ldots, 2, 1, n)$ by adding all edges. By placing n at the beginning of the ordering by only setting $x_{nr} = 1$, we obtain vector $(x_{rn}, x_{1n}, x_{n-1,n} : 1; x_{nr} : -1)$. We can replace n by any vertex of the cycle to obtain n vectors that are linearly independent among themselves. However, by adding n vectors found in the second part and subtracting the vectors found in this part we obtain zero vector. We do not use one of the vectors we obtained in this part. The scheme is illustrated in Figure 3.20.

  In total we obtain $n(n-3) + n + n - m + n + n - 1 = n(n+1) - m - 1$ vectors for the subgraph $C \cup \{r\}$.

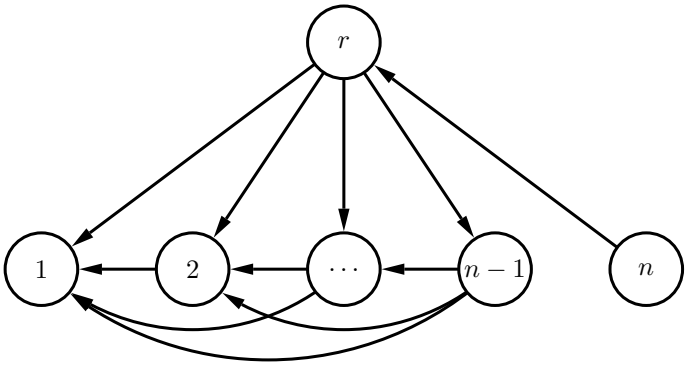
□

The r-cycle inequalities allow for a complete digraph as feasible, tight solution and therefore can be zero-lifted. The inequality (3.16) can be facet-defining even when C contains some edges. Our experiments in PORTA, Christof and Löbel (1997–2000), showed that r-cycle inequalities are facet-defining when

- An edge of the cycle $\{i, i+1\}$ is not present in the graph,

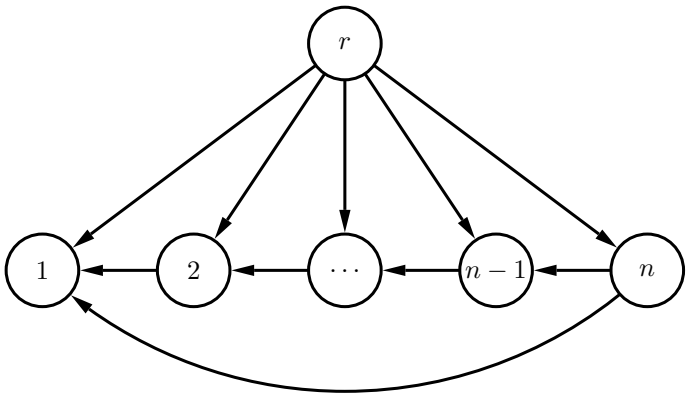


(a) Ordering $(n, r, n-1, \ldots, 2, 1)$ with $x_{n2} = 0$.

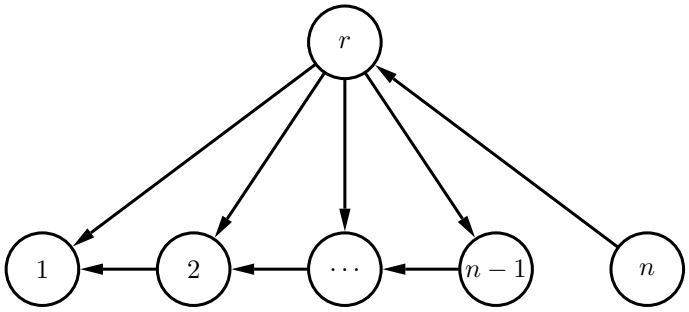


(b) Ordering $(n, r, n-1, \ldots, 2, 1)$ with $x_{n2} = 1$.

Figure 3.18: Subtracting two solution vectors gives vector $(x_{n2} : 1)$.

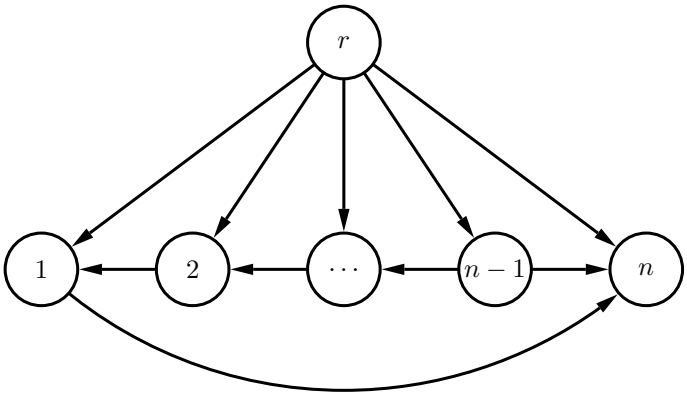


(a) Ordering $(r, n, n-1, \ldots, 2, 1)$ with all edges (chords are not drawn).

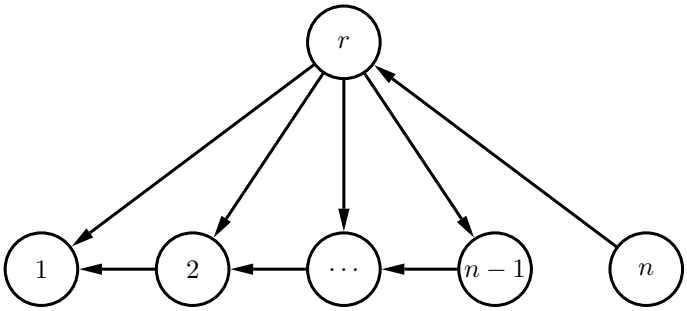


(b) Ordering $(n, r, n-1, \ldots, 2, 1)$ (chords are not drawn).

Figure 3.19: Subtracting two solution vectors gives vector $(x_{rn}, x_{n1}, x_{n,n-1} : 1; x_{nr} : -1)$.



(a) Ordering $(r, n-1, \ldots, 2, 1, n)$ (chords are not drawn).



(b) Ordering $(n, r, n-1, \ldots, 2, 1)$ (chords are not drawn).

Figure 3.20: Subtracting two solution vectors gives vector $(x_{rn}, x_{1n}, x_{n-1,n} : 1; x_{nr} : -1)$.

- $C \cup \{r\}$ does not induce a cycle in G as shown in Figure 3.21.

When a cycle edge $\{i, i+1\} \in E$, in all the tight solutions $x_{ri} = 0$. When $A(C) \cup \{r\}$ induces a cycle, the r-cycle inequality is dominated by the directed cycle inequality for the induced cycle.

Separation routine for the r-cycle inequalities

The r-cycle inequalities can be separated using a similar idea to the directed cycle inequalities. The inequality is violated when:

$$\sum_{i \in V(C)} x_{ri} > |C| - 1 + \sum_{(i,j) \in A(C)} x_{ij},$$

By moving the variables to the *lhs* and subtracting $|C|$ from both sides we obtain:

$$\sum_{i \in V(C)} x_{ri} - \sum_{(i,j) \in A(C)} x_{ij} - |C| > -1,$$

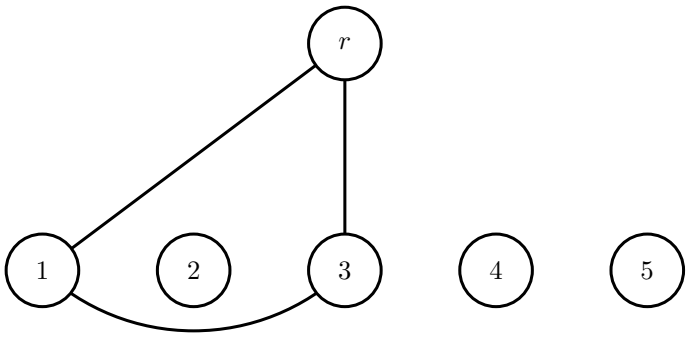
which is equivalent to:

$$\sum_{i \in V(C)} (1 - x_{ri}) + \sum_{(i,j) \in A(C)} x_{ij} < 1.$$

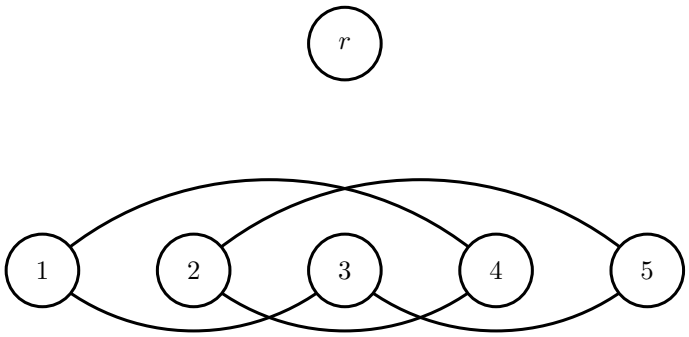
To separate the r-cycle inequalities we first fix a vertex r and solve a shortest path problem in a directed graph we create. For each vertex $i \in V \setminus \{r\}$, we create two copies i, i'. This time, we set $d(i, i') = 1 - x_{ri}$. For $x_{ij} > 0$ we assign $d(i', j) = x_{ij}$. If the shortest distance from i' to i is less than 1, there is a violated inequality. The shortest path will be $\{i', j, j', k, k', \ldots, l, l', i\}$. The distance for the shortest path corresponds to $x_{ij} + (1 - x_{rj}) + x_{jk} + (1 - x_{rk}) + \ldots + x_{li} + (1 - x_{ri})$. The cycle we obtain at the end is $\{i, j, k, \ldots, l\}$.

The shortest path graph has $2n - 2$ vertices and by using Dijkstra's shortest path algorithm for a single vertex r we can find violated inequalities in $\mathcal{O}^*(n^3)$ time. The whole separation routine takes $\mathcal{O}^*(n^4)$ time as there are n candidate vertices for r.

In this section we presented three classes of valid inequalities. The cyclechord inequalities are the most important ones from a computational point of view. Furthermore, the k-fence and the Möbius ladder inequalities (we were only able to check Möbius ladder inequalities on a 6 nodes graph) presented by Grötschel et al. (1985) for the linear ordering polytope are also facet-defining under certain



(a) The r-cycle inequality is dominated by the model inequality for the cycle $\{r, 1, 3\}$.



(b) The r-cycle inequality is dominated by the model inequality for the cycle $\{1, 3, 5, 2, 4\}$.

Figure 3.21: Two graphs where the r-cycle inequality is not facet-defining.

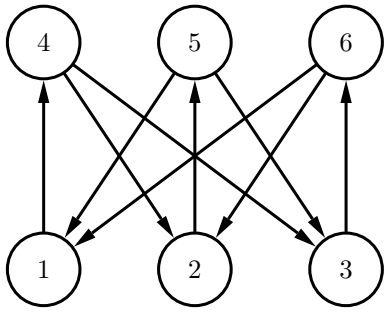


Figure 3.22: 3-fence inequality.

conditions, such as when the graph is empty or complete. For more information on the Möbius ladder inequalities, we refer the interested reader to Müller and Schulz (2002).

For the k-fence inequalities, we consider a subgraph H with two sets of vertices $U = \{u_1, u_2, \ldots, u_k\}$ and $W = \{w_1, w_2, \ldots, w_k\}$ where $k \geq 3$. We consider the arcs from w_i to u_i and the arcs from u_i to w_j such that $i \neq j$, so $A(H) = \cup_{i=1}^{k}\{\{(w_i, u_i)\} \cup \{(u_i, w_j) | i \neq j\}\}$. The k-fence inequality is as follows:

$$\sum_{(i,j)\in A(H)} x_{ij} \leq k^2 - k + 1. \tag{3.17}$$

The arcs (u_i, w_i) are called pales and the arcs (w_j, u_i) are called pickets. An example is presented in Figure 3.22 for $k = 3$ and in Figure 3.23 for $k = 4$. The arcs in these figures correspond to the variables on the lhs.

The inequalities are valid for $P(G)$ for $k = 3$ as they correspond to zero-half cuts obtained by the following directed cycle inequalities (and some domain constraints):

$$x_{14} + x_{42} + x_{25} + x_{51} \leq 3,$$
$$x_{14} + x_{43} + x_{36} + x_{61} \leq 3,$$
$$x_{25} + x_{53} + x_{36} + x_{62} \leq 3.$$

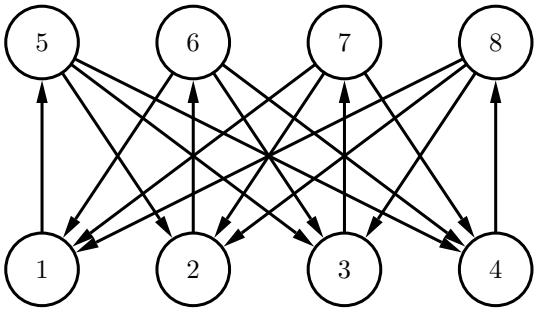


Figure 3.23: 4-fence inequality.

Similarly 4-fence inequality in Figure 3.23 can be obtained as a zero-half cut from 3-fence inequalities (and some domain constraints)

$$x_{15} + x_{26} + x_{37} + x_{52} + x_{53} + x_{61} + x_{63} + x_{71} + x_{72} \leq 7,$$
$$x_{15} + x_{26} + x_{48} + x_{52} + x_{54} + x_{61} + x_{64} + x_{81} + x_{82} \leq 7,$$
$$x_{15} + x_{37} + x_{48} + x_{53} + x_{54} + x_{71} + x_{74} + x_{81} + x_{83} \leq 7,$$
$$x_{26} + x_{37} + x_{48} + x_{63} + x_{64} + x_{72} + x_{74} + x_{82} + x_{83} \leq 7.$$

The Möbius ladder inequalities are defined on the arcs, M, of a digraph of k cycles satisfying certain conditions, Grötschel et al. (1985), such as $k \geq 3$ and k is odd. The inequality has the following form:

$$\sum_{(i,j)\in M} x_{ij} \leq |M| - \frac{k+1}{2}. \tag{3.18}$$

An example is given in Figure 3.24 for a 6 nodes graph. The cycles in the figure are $\{1,3,2\}, \{1,3,4\}, \{1,5,4\}, \{1,5,6\}, \{2,5,6,3\}$. The inequality can be obtained as a zero-half cut from the following directed cycle inequalities (and some domain constraints):

$$x_{21} + x_{13} + x_{32} \leq 2,$$
$$x_{13} + x_{34} + x_{41} \leq 2,$$
$$x_{41} + x_{15} + x_{54} \leq 2,$$
$$x_{15} + x_{56} + x_{61} \leq 2,$$

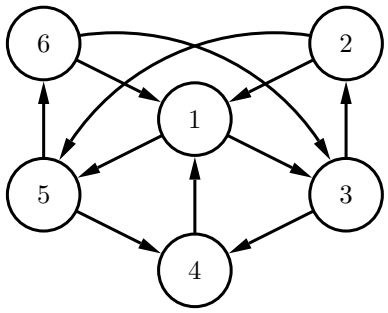


Figure 3.24: Möbius ladder inequality.

$$x_{56} + x_{63} + x_{32} + x_{25} \leq 3.$$

3.3 Computational results

In this section, we present the results of the computational study we carried on a number of benchmark problems. Branch-and-cut is an exhaustive search method that combines branch-and-bound with problem specific cuts. At each node of the branch-and-cut tree, an LP relaxation of the problem is solved, possibly by adding cuts in the hope of obtaining better bounds. If the optimal solution to the LP relaxation is fractional, subproblems are created in such a way that the fractional value of the variable is forbidden. By using upper and lower bounds it is possible to eliminate some parts of the branch-and-cut tree. The nodes that cannot be eliminated remain candidates for further exploration of the search tree.

Even though we present varios classes of inequalities, our tests showed that the cyclechord inequalities (including the extended version) is the most important class of inequalities in the computation. However, in certain types of graphs, such as grid graphs, cyclechord inequalities of size 4 are not enough to obtain a good lower bound. For that reason, we generate all chordless cycles for small graphs (graphs with at most 40 vertices) and add the corresponding cyclechord constraints to a lazy constraint pool. We also add the cyclechord inequalities for cycles of size 4, when the cycle is present in the graph. Furthermore, when there are 2 additional vertices that are connected to all cycle vertices, we also add the lifted inequality. We generate the simpliciality inequalities at the beginning of the algorithm and

add them to a constraint pool. They are added as they become necessary to reduce the number of constraints in the model. We observed that this approach makes the lower bound slightly worse. However, by being able to solve more subproblems we were able to obtain better results.

By using the result of Rose (1973), we find a maximum clique $S \subseteq G$ and put it at the end of the ordering. This means directing the edges inside the clique (in any order), fixing variables $x_{ij} = 0$ for $i \in S, j \in G \setminus S$. If $\{i, j\} \in E$, this also means setting $x_{ji} = 1$.

3.3.1 Branching

In our algorithm we chose a pair of vertices i, j such that $\{i, j\} \notin E$ and such that the value $x_{ij} + x_{ji}$ is closest to 0.5. We, then, create two branches where $x_{ij} + x_{ji} = 1$ and another branch where $x_{ij} + x_{ji} = 0$. The idea behind this is to obtain a chordal graph with the addition of edges in the branch-and-cut tree. The approaches do not guarantee that at each node of the branch-and-cut tree we can create branches. In other words, by themselves the branching schemes do not guarantee finding a feasible solution for the problem. However, if we cannot create branches for a fractional solution then CPLEX creates branches with its own criteria.

The branching scheme performed well, especially on dense graphs. So we chose to use it in our algorithm.

3.3.2 Computational study

We implemented our algorithm in CPLEX 12.2, IBM (June 2014). The computations are run on a desktop computer with 8.00 GB RAM and 2.53 GHz processor speed under Windows 7. We used 4 threads for parallel computation and set a CPU time limit of 3600 seconds. If the optimal is not found in this time limit, we report the gap between the upper and lower bounds. We used default settings in CPLEX, which means that the emphasis was on proving the optimality of a solution with some effort toward finding good feasible solutions.

The results we obtain are shown in tables 3.1, 3.2. In Table 3.1 results for sparser graphs like grid and mycielski graphs are shown. In general, for the graphs with fewer than 40 vertices, we obtain good results as a result of adding cyclechord inequalities for bigger cycles. For bigger grid graphs, the gap is quite large and the table shows that obtaining good lower bounds is the main problem in those

instances. For grid graphs we also explore a bigger part of the branch-and-bound tree, which is shown under the 'Nodes' column. In Table 3.2, results for queen graphs are shown. Even though, we fail to obtain the optimal solution for bigger instances, the gap is smaller than the results in Table 3.1. For denser graphs there is also potential for improvement by obtaining a better upper bound as the branch-and-cut algorithm spends more time on generating cuts than the branch-and-bound tree.

Graph			Cuts		Nodes(1000)		Root		Best		Time	Gap
Instance	$\|V\|$	$\|E\|$	Cplex	User	Exp.	Rem.	LB	UB	LB	UB	(sec.)	(%)
grid3_3	9	12	0	3	0.0	0.0	5.0	5	5.0	5	0.01	
grid3_4	12	17	0	6	0.0	0.0	9.0	9	9.0	9	0.00	
grid3_5	15	22	0	10	0.0	0.0	13.0	13	13.0	13	0.02	
grid3_6	18	27	0	15	0.0	0.0	17.0	17	17.0	17	0.02	
grid3_7	21	32	0	21	0.0	0.0	21.0	21	21.0	21	0.01	
grid3_8	24	37	0	28	0.0	0.0	25.0	25	25.0	25	0.02	
grid3_9	27	42	0	36	0.0	0.0	29.0	29	29.0	29	0.02	
grid3_10	30	47	0	45	0.0	0.0	33.0	33	33.0	33	0.03	
grid4_4	16	24	1	387	1.1	0.0	17.0	18	18.0	18	1.23	
grid4_5	20	31	1	1002	11.3	0.0	24.0	26	25.0	25	18.11	
grid4_6	24	38	0	881	854.5	337.1	30.8	34	32.2	34		5.4
grid4_7	28	45	5	984	444.2	233.2	37.6	42	39.0	41		4.8
grid4_8	32	52	5	1293	216.3	185.9	44.7	52	45.5	52		12.5
grid4_9	36	59	15	1701	130.0	114.0	51.8	60	52.5	58		9.6
grid4_10	40	66	14	2145	85.6	76.0	58.8	66	59.3	66		10.1
grid5_6	30	49	9	1149	200.1	172.9	45.3	53	46.2	53		12.8
grid5_7	35	58	12	1630	102.8	92.2	55.8	65	56.9	65		12.5
grid5_8	40	67	10	2164	59.0	54.4	66.6	77	67.5	77		12.4
grid5_9	45	76	34	2146	393.7	166.6	32.0	90	33.3	90		63.1
grid6_6	36	60	17	1699	78.0	71.6	59.9	77	60.9	77		20.9
grid6_7	42	71	37	2681	385.1	157.2	30.0	94	31.0	94		67.0
grid7_7	49	84	31	2827	334.8	96.0	36.0	125	37.0	125		70.4
myciel3	11	20	0	0	0.0	0.0	10.0	10	10.0	10	0.00	
myciel4	23	71	0	672	0.0	0.0	46.0	46	46.0	46	0.06	
myciel5	47	236	0	21018	11.3	1.1	185.0	197	189.7	197		3.7

Table 3.1: Computational results.

3.4 Conclusion

In this chapter we presented a branch-and-cut algorithm for solving the minimum triangulation problem. We, first presented an ILP formulation based on peos. We then discussed various classes of valid inequalities, in addition to the model inequalities. Finally, we presented the setting for our computational study and our results on benchmark graphs.

In addition to providing a feasible solution to the triangulation problem, the LP relaxation also yields a lower bound, which, to the best of our knowledge, is the only lower bound available so far. For future research, it is possible to integrate

Graph			Cuts		Nodes		Root		Best		Time	Gap
Instance	$\|V\|$	$\|E\|$	Cplex	User	Exp.	Rem.	LB	UB	LB	UB	(sec.)	(%)
anna	138	986	61	1069	161	0	44.3	48	47.0	47	1386.04	
david	87	812	186	2250	2341	1011	55.5	66	59.5	65		8.5
games120	120	1276	441	13623	2	3	496.4	1626	496.4	1626		69.5
huck	74	602	3	260	0	0	5.0	5	5.0	5	2.92	
jean	77	508	8	407	0	0	16.0	16	16.0	16	6.13	
miles250	125	774	261	2500	2079	1686	43.5	61	45.7	61		25.1
miles500	128	2340	134	11611	1	2	196.4	447	196.4	447		56.1
miles750	128	4226	0	55545	0	1	352.1	954	352.1	954		63.1
pathfinder	109	211	14	196	0	0	9.0	9	9.0	9	5.23	
queen3_3	9	28	3	11	0	0	5.0	5	5.0	5	0.00	
queen3_4	12	46	11	42	0	0	12.0	12	12.0	12	0.01	
queen3_5	15	67	4	471	5	0	21.0	23	22.0	22	0.31	
queen3_6	18	91	26	618	11	0	34.4	37	36.0	36	1.03	
queen3_7	21	118	17	1191	26	0	49.5	55	53.0	53	2.17	
queen3_8	24	148	41	1773	61	0	69.3	76	74.0	74	8.49	
queen3_9	27	181	33	2636	104	0	91.0	101	98.0	98	15.77	
queen3_10	30	217	40	3261	550	0	116.3	129	126.0	126	65.91	
queen4_4	16	76	42	296	0	0	26	28	26.0	26	0.19	
queen4_5	20	110	12	1547	34	0	49.5	53	51.0	51	4.54	
queen4_6	24	148	35	2027	208	0	75.3	85	83.0	83	16.54	
queen4_7	28	190	46	2973	531	0	108.0	122	119.0	119	68.22	
queen4_8	32	236	56	4460	3178	0	147.5	166	164.0	164	636.28	
queen4_9	36	286	52	5574	8329	1613	191.5	220	209.8	217		3.3
queen4_10	40	340	80	6717	3044	1572	243.1	285	255.5	278		8.1
queen5_5	25	160	12	4343	331	0	81.2	98	93.0	93	41.03	
queen5_6	30	215	77	3755	719	0	133.7	154	144.0	144	185.81	
queen5_7	35	275	83	5792	5936	2086	188.4	224	203.1	214		5.1
queen5_8	40	340	110	7705	2189	1020	256.4	307	265.8	293		9.3
queen5_9	45	410	145	9359	710	643	334.8	393	339.8	393		13.6
queen5_10	50	485	212	12577	320	321	423.1	501	424.9	501		15.2
queen6_6	36	290	78	6621	1917	907	206.0	244	214.9	232		7.4
queen6_7	42	371	127	8988	782	685	293.0	351	299.2	351		14.8
queen6_8	48	458	210	11635	67	68	398.3	481	400.7	481		16.7
queen6_9	54	551	290	16868	88	89	518.9	622	521.4	622		16.2
queen6_10	60	650	322	16811	1	2	656.7	786	656.7	786		16.5
queen7_7	49	476	188	10882	1	2	423.7	520	423.7	520		18.5
queen7_8	56	588	339	18013	20	21	575.9	710	577.6	710		18.7
queen7_9	63	707	379	19484	0	1	751.8	935	751.8	935		19.6
queen7_10	70	833	499	22456	0	1	948.5	1177	948.5	1177		19.4
queen8_8	64	728	388	18534	0	1	782.1	965	782.1	965		19.0

Table 3.2: Computational results.

branch-and-cut algorithm with minimal separators. Further exploration of cycle-chord inequalities can also be useful as they are the most promising inequalities in our algorithm. Another idea is branching on the order of the vertices, which could be stronger than the current branching scheme.

Chapter 4

Graph coloring

The (vertex) graph coloring problem has its origins from the need to color maps with a minimum number of colors. The problem dates back to 1852, when Francis Guthrie conjectured that a map could be colored using four colors, Kubale (2004). The conjecture was not proved until 1976, Appel and Haken (1977). The problem of coloring a map (which can be transformed into a planar graph) is polynomially solvable, however the problem is NP-hard for general graphs, Karp (1972).

The importance of the problem in different fields led to various algorithms that can be successfully used in practice. Chaitin et al. (1981) model the register allocation problem for compiler optimization as a graph coloring problem and successfully solve real-life problems. Lotfi and Sarin (1986) give a heuristic algorithm to solve the graph coloring problem and their computational study also includes real-life data of an exam scheduling problem. Aardal et al. (2007) study the frequency assignment problem. The problem of assigning a minimum number of frequencies in GSM networks corresponds to the graph coloring problem if the distances between frequencies are one. Burke et al. (2010) use graph coloring to solve a course timetabling problem, where two courses taken by the same student cannot be allocated to the same slot (cannot have the same color).

For a given undirected graph $G = (V, E)$, with V the set of vertices and E the set of edges, a *(vertex) coloring* of a graph is an assignment of colors to vertices such that no two vertices of an edge get the same color. A *k-coloring* of a graph uses k colors. The minimum number of colors necessary to color a graph is called the chromatic number of G and is denoted by $\chi(G)$.

Several exact and heuristic methods have been proposed for the graph coloring problem. Galinier et al. (2013) give a thorough survey of the heuristic methods used in the literature. Exact methods to solve the graph coloring problem can be examined under two categories: special algorithms for the problem and integer programming and its extensions. Byskov (2004) gives an $O(2.4023^n)$ algorithm using $O(2^n)$ space. The algorithm is based on observations by Lawler (1976) and finds all 4-colorable maximal graphs in $O(2.4023^n)$ time and $O(2^n)$ space. For each maximal independent set I, all 4-colorable maximal subgraphs in $G \setminus I$ are found, in order to find all 5-colorable subgraphs. Byskov (2004) extends this idea to find the chromatic number of the graph. Björklund et al. (2009) improve on the result of Byskov (2004) to obtain an $O(2.2461^n)$ time algorithm using the inclusion-exclusion principle.

Branch-and-cut algorithms were also used to attack the graph coloring problem. Coll et al. (2002); Méndez-Díaz and Zabala (2006, 2008) present an integer linear programming (ILP) formulation that consists of binary variables for colors and for assignment of colors to the vertices. Coll et al. (2002) give an ILP formulation and present some valid inequalities. However, the assignment model presented in the paper contains symmetry, i.e., colors can be exchanged. For that reason additional inequalities are added to the model to break the symmetry in Méndez-Díaz and Zabala (2006, 2008). They, then, define various classes of valid inequalities for different substructures such as independent sets, cliques, odd holes, anti-holes, and paths. Furthermore, Palubeckis (2010) presents web and anti-web inequalities for graph coloring.

Another approach to solve the graph coloring problem is column generation and branch-and-price, Mehrotra and Trick (1995). In this approach, columns of the master problem correspond to independent sets. The authors present a heuristic and an exact method in order to solve the pricing subproblem and discuss the branching scheme for the branch-and-price algorithm. Hansen et al. (2009) present two formulations using set covering and set packing. For their branch-and-cut-and-price algorithm, they give some facet-defining inequalities and preprocessing rules.

Burke et al. (2010) use graph coloring to solve a course timetabling problem. By using the idea of *indistinguishable* vertices (which are connected and have the same closed neighborhood) and reversible clique partition, the authors are able to reduce the problem size even on graphs that do not have an underlying timetabling problem.

There are also other formulations used for the graph coloring problem. Williams and Yan (2001) use constraint satisfaction with `all_different` constraints and general integer variables for colors. `all_different` constraints are used when a set of variables must be pairwise different, which can correspond to the colors of the vertices of a clique. Bergman and Hooker (2012) present facets for the constraint satisfaction formulation of the problem.

In our algorithm we are going to use the asymmetric representatives formulation introduced by Campêlo et al. (2008). The representatives formulation of Campêlo et al. (2004), uses the idea that vertices with the same color can represent each other. Even though the formulation has some symmetry, it is less than the original formulation of Méndez-Díaz and Zabala (2006). Campêlo et al. (2008) further reduce symmetry by using an ordering of the vertices and the number of variables.

In Section 4.1, we present our notation and the integer linear programming formulation we use for the rest of the chapter. Section 4.2 focuses on the polytope and valid inequalities we use in the branch-and-cut algorithm. In Section 4.3, we give the details of the branch-and-cut algorithm and present a variant of the original ILP formulation to reduce the number of variables. A computational study is presented in Section 4.4.

4.1 Mathematical model

In this section, we present the notation we use in the rest of the chapter. We give the ILP formulation we use for the problem, and show it is equivalent to the independent set problem of the conflict graph of the variables.

4.1.1 Notation

We assume that the graph $G = (V, E)$ is simple, undirected, and connected with $n = |V|$ vertices and $m = |E|$ edges. Two vertices i, j are adjacent if $\{i, j\} \in E$. $N(i) = \{j \in V \mid \{i, j\} \in E\}$ is called the neighborhood of i. We use an ordering of the vertices to reduce the symmetry. An ordering of G is a mapping $\sigma : V \to V$, where $\sigma(i)$ denotes the position of i in the ordering. In the sequel we will identify each vertex with its position in the ordering, i.e., the vertices are numbered $1, 2, \ldots, n$. For this ordering of G, we call $N^-(i) = \{\{1, 2, \ldots, i-1\} \cap N(i)\}$ the in-neighborhood of i and $N^+(i) = \{\{i+1, i+2, \ldots, n\} \cap N(i)\}$ the out-neighborhood

of i. $\bar{G} = (V, \bar{E})$ denotes the complement of G, where $\bar{E}$ consists of $\{i, j\} \notin E$. Now $\bar{N}(i) = \{j \in V \mid \{i, j\} \in \bar{E}\} \setminus \{i\}$ is called the antineighborhood of i. The in- and out-antineighborhoods of i in $\bar{G}$ are defined similarly as $\bar{N}^+(i), \bar{N}^-(i)$. The closed (anti)neighborhoods, where i is included, are denoted by, respectively, $N[i], N^-[i], N^+[i], \bar{N}[i], \bar{N}^-[i], \bar{N}^+[i]$.

Finally, we call $H = (V_H, E_H)$ an induced subgraph of G if $V_H \subseteq V$ and $\{i, j\} \in E_H$ if and only if $i, j \in V_H$ and $\{i, j\} \in E$. H is called a clique if all vertices in H are pairwise adjacent. H is called a cycle if the vertices form a sequence $(i, j, \ldots, k, i)$, such that each vertex is adjacent to the next one in the sequence and no node is repeated twice apart from the first node which also ends the sequence. A chord in a cycle is an edge joining vertices not adjacent on the cycle. If H is a chordless cycle it is called a hole. An odd hole is a chordless cycle consisting of $2k + 1$ nodes. An independent set is a set of vertices, from which no two vertices are adjacent.

4.1.2 ILP formulation

We use the asymmetric representatives formulation given in Campêlo et al. (2008). Here, some vertices are selected to represent the colors. Vertices that do not represent a color, are identified by the color of one of their predecessors. $\bar{N}^+[i]$, i.e. the out-antineighborhood of i, corresponds to the vertices that can be represented by i (including itself). $\bar{N}^-[i]$, i.e. the in-antineighborhood of i, corresponds to the vertices that can represent i (including itself).

For each vertex j, we define decision variables for the nodes $i \in \bar{N}^-[j]$. It means that $i \leq j$ and $\{i, j\} \notin E$. The decision variables for the problem are defined as:

$$x_{ij} = \begin{cases} 1, & \text{if vertex } i \text{ represents vertex } j, \\ 0, & \text{otherwise.} \end{cases}$$

If $x_{ii} = 1$, the vertex i is called a representative vertex and the vertex i represents a color class. The ILP formulation for the asymmetric representatives formulation is given as follows:

$$\min \sum_{i \in V} x_{ii}, \tag{4.1}$$

$$\text{subject to: } \sum_{i \in \bar{N}^-[j]} x_{ij} = 1, \qquad \forall j \in V, \tag{4.2}$$

$$x_{ij} + x_{ik} \leq x_{ii}, \qquad \forall j,k \in \bar{N}^+(i), \{j,k\} \in E, \tag{4.3}$$

$$x_{ij} \leq x_{ii}, \qquad \forall j \in \bar{N}^+(i), j \text{ singleton}, \tag{4.4}$$

$$x_{ij} \in \{0,1\}, \qquad \forall i \in V, j \in \bar{N}^+[i]. \tag{4.5}$$

The objective function (4.1) minimizes the number of vertices that act as representatives, and therefore the number of colors used. Constraints (4.2) make sure that each vertex j is represented once by itself or by a representative vertex ordered earlier than j. For two vertices $j,k \in \bar{N}^+(i)$ that are connected, constraints (4.3) dictates that vertex i cannot represent both vertices j and k and can represent one of them only if it is a representative vertex. If vertex $j \in \bar{N}^+(i)$ is not connected to another vertex in $\bar{N}^+(i)$, i can only represent j in case it is a representative vertex, constraints (4.4). We can combine and strengthen constraints (4.3) and (4.4) by considering maximal cliques in $\bar{N}^+(i)$. By using maximal cliques, we can replace these inequalities by maximal clique constraints $\sum_{j \in C} x_{ij} \leq x_{ii}$, for every maximal clique $C \subseteq \bar{N}^+(i)$ and for every vertex $i \in V$. A maximal clique in the out-antineighborhood of a vertex i can only have one vertex that is represented by i only if i is a representative vertex. Note that a maximal clique can be a singleton vertex j when j is not connected to any of the vertices in $\bar{N}^+(i)$. Constraints (4.5) express the binary nature of the variables.

A difference of the formulation we are using compared to the formulation used by Campêlo et al. (2008) is constraints (4.2), the representation constraints. In our problem we use equality, whereas in Campêlo et al. (2008) it is possible for a vertex to be represented by more than one vertex. The advantage of the latter is that it allows for application of column generation techniques. In our approach the problem becomes the maximum independent set problem on the conflict graph of the variables as shown in Section 4.1.3. This enables us to use the facets of the independent set polytope.

4.1.3 Equivalence to the independent set problem

The graph coloring formulation (4.1)-(4.5) is equivalent to the maximum independent set problem on a conflict graph of the formulation. The vertices of the conflict graph G_σ are the decision variables of the coloring problem without the variables x_{ii}, i.e, $V_\sigma = \{x_{ij} \mid i \in \bar{N}^-(j)\}$. The edges of the conflict graph are added according to the following criteria:

1. x_{ik} and x_{jk} $(i, j \in \bar{N}^-(k))$ are connected: Only one vertex is allowed to represent vertex k.

2. x_{ij} and x_{jk} $(i \in \bar{N}^-(j), k \in \bar{N}^+(j))$ are connected: j cannot represent k if it is itself represented by i.

3. x_{ij} and x_{ik} $(j, k \in \bar{N}^+(i))$ are connected whenever $\{j, k\} \in E$: i can only represent one of them.

Theorem 4.1. *The graph coloring formulation* (4.1)-(4.5) *is equivalent to the independent set problem defined on the conflict graph* $G_\sigma = (V_\sigma, E_\sigma)$ *of the decision variables.*

There is a one-to-one correspondence between the solutions of the coloring problem (4.1)-(4.5) of a graph G and the solutions of the independent set problem of its conflict graph G_σ. Furthermore $n = \chi(G) + \alpha(G_\sigma)$ and a minimum coloring of G corresponds to a maximum independent set of G_σ.

First, consider a solution of (4.1)-(4.5). The pairs of variables with value 1 (except the representative variables x_{ii} $(i \in V)$ do not satisfy any of the three conditions mentioned above and are therefore not connected in the conflict graph, i.e. they form an independent set.

Second, consider an arbitrary independent set in the conflict graph. Each selected $x_{ij} = 1$ represents the color of j to be the color of its representative i. Due to condition 1, $\sum_{i \in \bar{N}^-(j)} x_{ij}$ is either 0 (j is a representative, and thus $x_{jj} = 1$, or 1 (j is represented by an earlier node, and thus $x_{jj} = 0$. In the first case, $x_{jk} = 0$ for all vertices k after j and maximal clique constraints are satisfied. In the second case $x_{jj} = 1$, and thus, due to condition 3 at most one of the later vertices can be selected to be represented by j. Again, maximal clique constraints are satisfied. Finally, in both cases constraints (4.2) is satisfied.

The vertices of V are either represented by another vertex (in case when $\sum_{i \in \bar{N}^-(j)} x_{ij} = 1$), or are representatives themselves (in case when $\sum_{i \in \bar{N}^-(j)} x_{ij} = 0$). In other words, we have $n = \sum_{i \in V} x_{ii} + \sum_{i,j \in V : i \in \bar{N}^-(j)} x_{ij}$. Therefore, a minimum set of representative vertices corresponds to a maximum size independent set in G_σ.

Showing the equivalence between the two problems allows us to use facet-defining inequalities of the independent set polytope (as discussed by Nemhauser and Sigismondi (1992)) in the subsequent parts of this chapter. Furthermore, it

also enables us to generate valid inequalities that are not limited to variables in the out-antineighborhood of a single vertex.

Structure of neighborhood of nodes in the conflict graph

In the conflict graph for each vertex x_{ij} we have the following neighbors:

- Set 1: Nodes that can represent vertex i. The set contains all the nodes in the set $\bar{N}^-(i)$ and forms a clique.
- Set 2: Nodes that can represent vertex j excluding i. The set contains all the nodes in the set $\bar{N}^-(j) \setminus \{i\}$ and forms a clique.
- Set 3: Nodes that can be represented by vertex j. The set contains the nodes in $\bar{N}^+(j)$. These vertices are connected among themselves when $\{k, l\} \in E$ with $k, l \in \bar{N}^+(j)$.
- Set 4: Nodes that can be represented by vertex i and that are connected to j. The set contains the nodes in $\bar{N}^+(i) \cap N(j)$. These vertices are connected among themselves when $\{k, l\} \in E$ with $k, l \in \bar{N}^+(i) \cap N(j)$.

The vertices in Set 1 are connected to the vertices in Set 4 and similarly vertices in Set 2 are connected to the vertices in Set 3. There are no edges between sets 1 and 2. If the variable x_{ij} is defined, i and j are not connected, therefore same vertex can represent i and j. The vertices that are in Set 4 are connected to j and the vertices in Set 3 are not. For this reason, there are no edges between sets 3 and 4. The structure of the neighborhood is shown in Figure 4.1.

4.2 Valid inequalities

In this section, we discuss the dimension of the polytope and the valid inequalities we use in the branch-and-cut algorithm.

The polytope of the graph coloring problem

The independent set polytope is full dimensional. The zero-vector (which corresponds to an empty independent set) and the set of all unit vectors (which correspond to all possible independent sets of size 1) can be used to prove this. However, the number of variables in our independent set problem depends on the

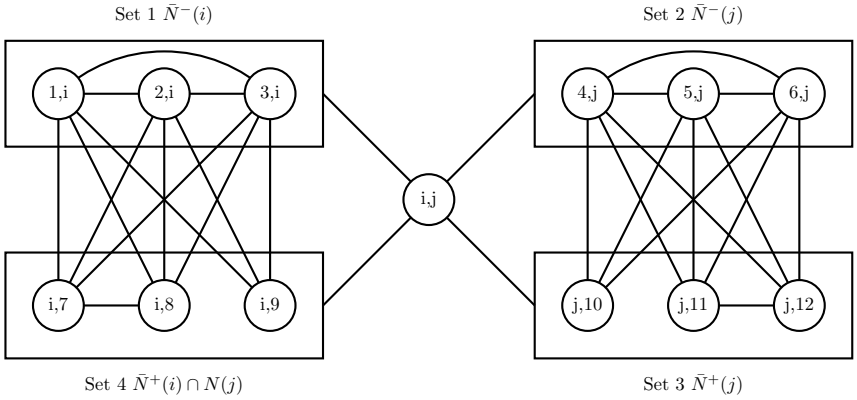


Figure 4.1: Neighborhood of x_{ij}. All the vertices are connected to x_{ij}.

ordering of the vertices. Therefore, we will only assert validity of the forthcoming inequalities. They are, however, also facet-defining under mild conditions.

Previous branch-and-cut algorithms on the graph coloring problem focus on inequalities in small structures in the graph. Like the previous works of Méndez-Díaz and Zabala (2006) and Campêlo et al. (2008), we choose to focus on structures such as cliques, odd holes. Furthermore we develop a new separation routine for rank inequalities in the out-antineighborhood of a representative vertex. In the sequel, we will discuss some standard classes of valid inequalities, and their shape in the specific structure of the conflict graph. Besides, we introduce some new inequalities.

4.2.1 Generalized rank inequalities

Let H be an induced subgraph of $\bar{N}^+(i)$. We define $\alpha(H)$ as the size of the maximum independent set in H. The following inequality is valid:

$$\sum_{j \in H} \frac{x_{ij}}{\alpha(H)} \leq x_{ii}.$$

When $x_{ii} = 0$, the variables in the lhs are also 0. When $x_{ii} = 1$, the set of vertices represented by i has to form an independent set and thus at most $\alpha(H)$ variables can be set to 1, proving the validity of the inequality. We now strengthen the inequality. For any vertex $j \in H$, we denote the size of the maximum independent set including j by $\alpha_j(H)$ and for any independent set $W \subseteq H$ we denote

the size of the maximum independent set containing W by $\alpha_W(H)$. Note that if $j \in W$, then $\alpha_W(H) \leq \alpha_j(H) \leq \alpha(H)$. Now the following inequality is valid:

$$\sum_{j \in H} \frac{x_{ij}}{\alpha_j(H)} \leq x_{ii}.$$

We let $W \subseteq H$ be an independent set represented by i. We define $\alpha = \min_{j \in W} \alpha_j(H)$. By definition we have $\alpha_j(H) \geq \alpha$ for all the vertices in W. Validity of this inequality is shown as follows. If $x_{ii} = 0$, then all the variables on the lhs are also 0. Consider $x_{ii} = 1$ and we obtain

$$\sum_{j \in H} \frac{x_{ij}}{\alpha_j(H)} \leq \sum_{j \in W} \frac{1}{\alpha_j(H)} \leq \sum_{j \in W} \frac{1}{\alpha} \leq 1$$

Other inequalities presented by Campêlo et al. (2008) can be unified as valid inequalities of the independent set polytope. The external facets given by Campêlo et al. (2008) make sure that a vertex can only represent an independent set of maximum size in its out-antineighborhood. The internal facets make sure that only a certain number of vertices in an induced subgraph H can be represented by a clique at the beginning of the ordering of H. For an odd hole of size $2k+1$, only $2k$ vertices can be represented by the clique (or size 2) at the beginning of the odd hole and the remaining vertex must be represented by a vertex that does not belong to the odd hole. The internal facets can also be unified under the independent set polytope because of our choice of using equalities in the representation constraints.

4.2.2 Clique inequalities

The well-known clique inequalities for the independent set polytope have the standard form of:

$$\sum_{j \in C} x_j \leq 1, C \text{ maximal clique.} \tag{4.6}$$

Inequality (4.6) is facet-defining when C is a maximal clique, in G. The inequalities we generate are for a single representative vertex and have the following form, again with C being a maximal clique:

$$\sum_{j \in C} x_{ij} \leq x_{ii}, C \subseteq \bar{N}^+(i). \tag{4.7}$$

By substituting $x_{ii} = 1 - \sum_{k \in \bar{N}^-(i)} x_{ki}$, we obtain the inequality

$$\sum_{j \in C} x_{ij} + \sum_{k \in \bar{N}^-(i)} x_{ki} \leq 1, C \subseteq \bar{N}^+(i).$$

The inequality consists of the variables that can represent i and the vertices in C that can be represented by i (C forms a clique). This means that the variables are connected in the conflict graph G_σ. When C is a maximal clique, the inequality is facet-defining as it induces a maximal clique in G_σ by the construction of the neighborhood.

It is possible to generate all maximal cliques in G using the Bron-Kerbosch algorithm (Bron and Kerbosch, 1973) and to use the ordering of the vertices to generate maximal cliques in $N^+(i)$. This approach, however, is only suitable for small and medium sized graphs. In the literature, clique inequalities are usually heuristically separated with a greedy approach, see Méndez-Díaz and Zabala (2008). Under the greedy approach, a small clique (preferably with a high value for variables) is enlarged by the addition of new vertices. In our algorithm, we separated the clique inequalities using the separation routine in Section 4.2.4.

4.2.3 Odd hole inequalities

Another commonly studied structure in graph coloring and independent set problems is odd holes. A hole in a graph is a cycle that does not contain any chords. An odd hole is a chordless cycle of size $2k + 1$. An even hole can be colored using two colors but an odd hole requires at least three colors as the size of the maximum independent set in an odd hole is k. For that reason, odd holes have an important place in branch-and-cut algorithms for the graph coloring problem. The inequality for odd holes can be written as:

$$\sum_{j \in H} \frac{x_{ij}}{k} \leq x_{ii}, H \subseteq \bar{N}^+(i), H \text{ odd hole of size } 2k + 1. \tag{4.8}$$

To separate odd hole inequalities we use the algorithm presented in Nemhauser and Sigismondi (1992). For an odd hole of size $2k + 1$ and a representative vertex r, the odd hole inequality is violated when

$$(1 - x_{r1} - x_{r2}) + (1 - x_{r2} - x_{r3}) + \ldots + (1 - x_{r,2k+1} - x_{r1}) < 1.$$

To separate the odd hole inequalities we create a bipartite graph $G' = (V \cup V', E')$ with each vertex having two copies in V and V'. If $\{i, j\} \in E$, then we set

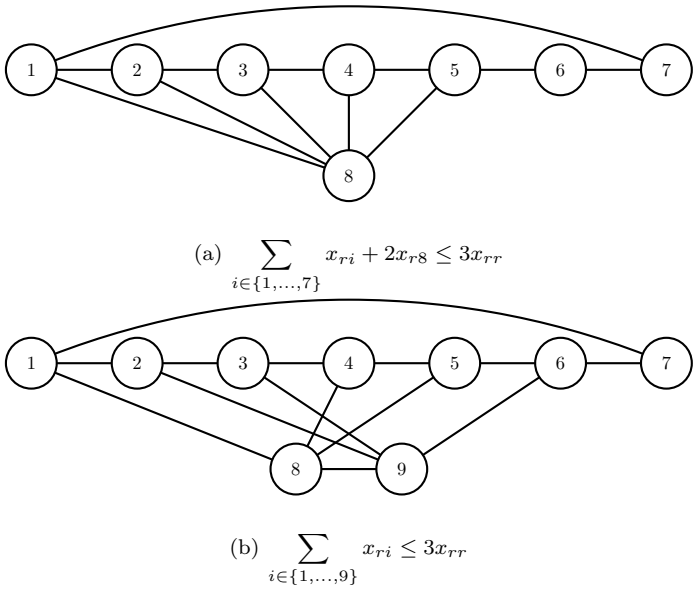


Figure 4.2: In the graphs the inequality $\sum_{i\in\{1,\ldots,7\}} x_{ri} \leq 3x_{rr}$ can be lifted.

$d(i, j') = d(i', j) = 1 - x_{ri} - x_{rj}$. A path from i to i' consists of $2k + 1$ edges and the cycle we obtain yields a violated inequality if the sum of the distances is less than one.

An odd hole inequality can be lifted under certain conditions. If a vertex is connected to at least 3 nodes of the odd hole, we check the cardinality of the maximum independent set among the cycle nodes that are not connected to the external node. If the maximum independent set still has cardinality k the vertex cannot be used for lifting. Otherwise we determine the lifting coefficient and add the inequality to the model. In Figure 4.2 some examples are shown for the lifting of the odd hole inequalities. It is worth noting that the odd hole inequalities are also rank inequalities and the lifting scheme corresponds to finding α_v value for each vertex v that is used in the lifting. Nemhauser and Sigismondi (1992) give an algorithm to find all violated inequalities based on an odd hole by using a lifting tree. In our algorithm we do not generate the whole tree, but we generate 2 levels of it.

4.2.4 Rank inequalities

Clique and odd hole inequalities are so-called rank inequalities. These are inequalities where all variables that appear in its lhs have the same coefficient:

$$\sum_{j \in H} \frac{x_{ij}}{k} \leq x_{ii}, H \subseteq V'. \tag{4.9}$$

Separation routines for rank inequalities

In this section we present an algorithm for the separation of the rank inequalities (4.9). The algorithm attempts to find violated rank inequalities in the outantineighborhood of a single representative vertex i. In the graph we look for vertices i such that $x_{ii} > 0$. We sort the vertices j by nonincreasing value of x_{ij} without considering the vertices for which $x_{ij} = 1$. We select first s vertices and find the cardinality of a maximum independent set. Then, we keep on adding new vertices to the list as long as the independent set size does not increase. We make one pass over the list of vertices.

We run the algorithm for different values of s and we also initiate the algorithm from different vertices in the list as described in Algorithm 2. Note that if the initial independent set size is 1 we end up with a separator for maximal cliques.

```
input  : A fractional solution to LP x, size of initial set s
output: A set of violated valid inequalities
for i ← 1 to n such that x_ii > 0 do
    list ← {};
    for j ← 1 to n such that (i,j) ∉ E, x_ij ≠ 1 do
        list ←list ∪{j};
    end
    list ←sort(list);
    rankcandidate ← {list[1], ..., list[s]};
    indsetsize ← FindMaximumIndependentSet(rankcandidate);
    for j ← s + 1 to n do
        if FindMaximumIndependentSet(rankcandidate ∪list[j]) ==
        indsetsize then
            rankcandidate ← rankcandidate ∪{list[j]};
        end
    end
end
```

Algorithm 2: Separation routine for rank inequalities

It is possible to generalize all the inequalities for the variables in the conflict graph. For a cycle of size 5 with nodes {1,2} at the beginning of the ordering we can use the following constraint:

$$x_{13} + x_{35} + x_{25} + x_{24} + x_{14} \leq 2.$$

The inequality is an odd hole inequality in G_σ and it consists of variables belonging to the out-antineighborhood of vertices $1, 2$, and 3. It is also possible to lift the inequalities using other variables from the conflict graph. We can also generate rank inequalities using G_σ. In that case, we list all the variables with respect to their values in the fractional LP relaxation solution and apply the same routine of adding variables to the list.

However, in our experiments we found out that separating these inequalities requires considerable computational effort as we are dealing with the conflict graph of the variables: the corresponding separation problems are much bigger than the separation problems in the original graph. In the case of rank inequalities we were able to find many violated inequalities. Even though the violations of the inequalities are big, they also involve a lot of variables and they do not contribute to the lower bound. In fact, we obtained worse results for the problem by separating the inequalities in the conflict graph. As a result of this, we use the separation algorithms for a single representative vertex.

4.3 Branch-and-cut

In the Branch & Cut algorithm we consider three elements: the ordering of the vertices, preprocessing, branching, and variable reduction.

4.3.1 Ordering of the vertices

Campêlo et al. (2008) choose to place a maximum clique, S, at the beginning of the ordering and a maximum clique in the remaining graph to the end of the ordering. In the maximum clique at the beginning, each vertex has its own color (the variables x_{ii} are set to 1). This provides a starting lower bound on the chromatic number of the graph. In our algorithm, after placing the maximum clique at the beginning, we place a maximum independent set in the remaining graph at the end of the ordering. Let i be the first vertex of the independent set in the ordering. The entire independent set can be represented by one vertex. If

another vertex in the independent set is representative, we can create a solution where i is representative, without changing the objective function value. This approach eliminates the variables that belong to the vertices in the independent set and thus leads to a slightly smaller problem.

The approach itself can be extended by placing a second independent set before the independent set at the end of the ordering. In this case, the new independent set can have at most one representative vertex. In case there are more representative vertices, we can always alter the coloring in such a way that two independent sets are colored using the two vertices ordered first in the respective independent sets.

4.3.2 Preprocessing

Consider distinct vertices i and j, with $i < j$ and $\bar{N}^+[j] \subset \bar{N}^+[i]$. This means that j and its out-antineighbors can be represented by i. For any optimal coloring with the property $x_{jj} = 1$ and $x_{ii} = 0$, we can create an alternative optimal coloring by letting all vertices that are represented by j, be represented by i, so that $x_{ii} = 1$, $x_{jj} = 0$, Thus, we can replace any solution with $x_{jj} = 1$ and $x_{ii} = 0$ by a solution at least as good with $x_{jj} = 0$ and $x_{ii} = 1$. Thus, we can add the following valid inequality to the model:

$$x_{jj} \leq x_{ii}.$$

We can generalize this idea when $\bar{N}^+[j] \setminus \bar{N}^+[i] \neq \emptyset$. Let $\bar{N}^+[j] \setminus \bar{N}^+[i] = \{v_1, v_2, \ldots, v_k\}$. Now, the following inequality is valid:

$$x_{jj} \leq x_{ii} + x_{j,v_1} + x_{j,v_2} + \ldots + x_{j,v_k}.$$

When at least one of the variables $x_{j,v_1}, x_{j,v_2}, \ldots, x_{j,v_k}$ is equal to 1, then the inequality is automatically satisfied. If $x_{j,v_1} = x_{j,v_2} = \ldots = x_{j,v_k} = 0$, then the inequality reduces to $x_{jj} \leq x_{ii}$ as we obtain a case which is equivalent to $\bar{N}^+[j] \subset \bar{N}^+[i]$.

4.3.3 Branching on variables

For branching we use the idea of Balas and Yu (1986). The idea behind the branching scheme is to set the value of a variable to 1 in each branch of the branch-and-cut tree. In traditional branching we select a variable at each node of

the branch-and-cut tree and set its value to 1 and 0 to create two new subproblems. However setting a variable to 0 does not have a big effect on the lower bound. In our problem we use a more structured approach. We know that for each vertex j we have $\sum_{i \in \bar{N}^-[j]} x_{ij} = 1$. For this reason, at each branch we select a vertex j that is early in the ordering and that is fractionally represented by other vertices. The branching scheme works as follows: At each node, starting from the beginning of the ordering we select a vertex j that is represented by more than one vertex. Then we select the vertex i with the maximum x_{ij} value and set it to 1 in the first branch and to 0 in the second branch. For the first branch, a new vertex has to be selected in the subsequent subproblems as vertex j is now represented by one vertex. For the second branch representation for vertex j can be integer or fractional. In the latter case, j will be selected again and new branches will be created. If the value is integer, we proceed to the next fractional vertex.

4.3.4 Branching on a set of variables

In addition to the branching on a single variable we are going to exploit the structure of the graph G to create stronger branches in our algorithm. We use the following observation: If $N(j) \subseteq N(i)$ and $\{i, j\} \notin E$, then there exists an optimal coloring of G, where i and j have the same color. If i is a representative vertex, then this means j is represented by i. In our algorithm we search for vertices with the same neighborhood and assign them the same color. However, it may be difficult to find two such vertices in general graphs. Now, we are going to extend this observation to the case where there is a single vertex k such that $N(j) \setminus N(i) = \{k\}$. Consider in the branching we let vertex l represent k. If $\{l\} \in N(i)$, then i and j have the same color. When $x_{kl} = 1$, we connect $N(l)$ to the vertex k and if $\{i, l\} \in E$, vertices i and j end up having the same neighborhood. Similarly if during branching we set $x_{jl} = 0$, then i and j end up having the same color.

If k is not represented by i, then i and j end up having the same neighborhood in G and therefore they have the same color in an optimal solution of that branch. In other words if an optimal solution does not satisfy this property, the solution can be altered so that this property is satisfied.

We give an example of that situation in the Mycielski3 graph in Figure 4.3. The nodes are named according to the ordering. In the graph, we have $N(7) \setminus N(2) = \{11\}$. If $x_{2,11} = 0$ vertices 2 and 7 have the same neighborhood ($N(7) \subseteq N(2)$), therefore can safely get the same color, which can be any admissible color. Consider

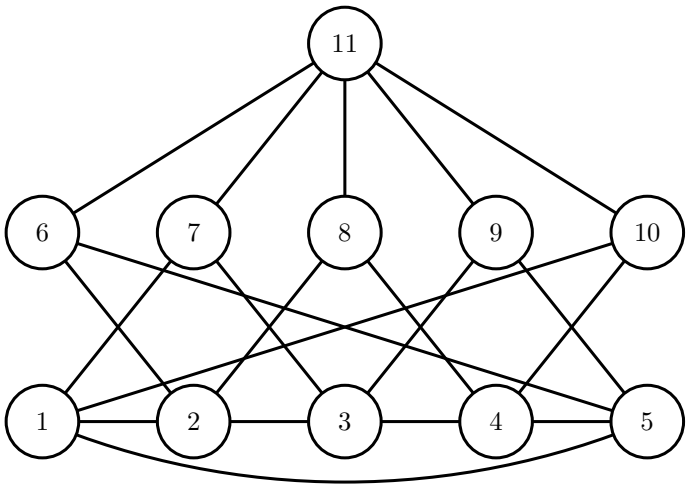


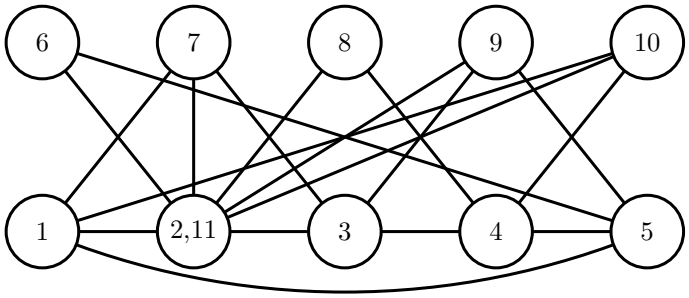
Figure 4.3: Mycielski3 graph. Ordering is the same as the numbering of the vertices.

vertex 11 is represented by vertex 2 ($x_{2,11} = 1$). In that case, $\{2\} \in N(1)$ and $\{2\} \in N(3)$. Therefore, vertices 1 and 6 and vertices 3 and 8 get the same color as shown in Figure 4.4(a). The same happens when $x_{7,11} = 1$ as shown in Figure 4.4(b).

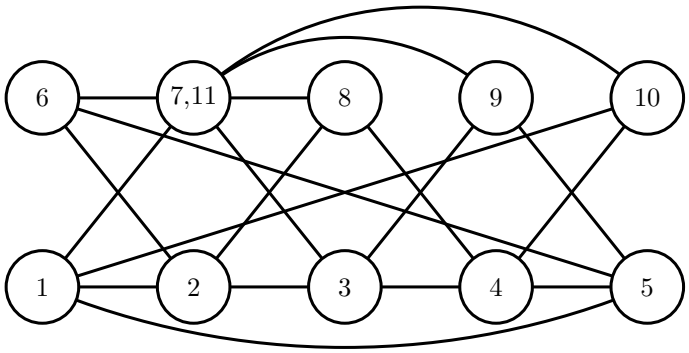
Example 2. In the Mycielski3 graph when we give vertex 11 its own color, it corresponds to removing vertex 11 from the graph as it is not allowed to represent any other vertices due to the ordering. When vertex 11 is assigned its own color $N(1) = N(6), \ldots, N(5) = N(10)$ and we are left with a cycle of size 5, which is 3 colorable.

4.3.5 Reduction of number of variables

If an upper bound is known for the chromatic number of the graph, we can alter the formulation to obtain a problem with fewer variables than the original formulation. We let $\chi'(G)$ be an upper bound on the chromatic number of G and $\omega(G)$ be the size of a maximum clique, say S. Let S be the set of nodes at the beginning of the ordering. Then, we can add $\chi'(G) - \omega(G)$ vertices directly after S. The new vertices, the set W, will only be connected to S in the original graph, which is the clique at the beginning of the ordering. For this reason, they can represent all the vertices in the graph excluding the clique at the beginning of the ordering. With



(a) Case where $x_{2,11} = 1$. As $\{2\} \in N(1)$ and $\{2\} \in N(3)$, vertex 6 has the same color as vertex 1 and vertex 8 has the same color as vertex 3.



(b) Case where $x_{7,11}$. As $\{7\} \in N(1)$ and $\{7\} \in N(3)$, vertex 6 has the same color as vertex 1 and vertex 8 has the same color as vertex 3.

Figure 4.4: Mycielski graph. Ordering is the same as the numbering of the vertices.

this modification, the only potential representative vertices are in $S \cup W$. We can now set $x_{ii} = 0$, $\forall i \in V \setminus S$. Note that we only added vertices and edges to G, and thus the formulation remains equal. However, the newly added vertices have no particular ordering, since they are exchangeable. Therefore we add constraints to prevent this. Furthermore vertices in W do not have to be colored. This gives the following model:

$$\min \sum_{i \in W} x_{ii}, \tag{4.10}$$

$$\text{subject to:} \sum_{i \in \bar{N}^-[j] \cap S \cup W} x_{ij} = 1, \qquad \forall j \in V \setminus S, \tag{4.11}$$

$$x_{ii} \leq 1, \qquad \forall i \in W, \tag{4.12}$$

$$x_{i+1,i+1} \leq x_{ii}, \qquad \forall i \in W, \tag{4.13}$$

$$\sum_{j \in C} x_{ij} \leq x_{ii}, \qquad \forall i \in S \cup W, \forall C \in \bar{N}^+(i), \tag{4.14}$$

$$x_{ij} \in \{0, 1\}, \qquad \forall i \in S \cup W, j \in \bar{N}^+[i]. \tag{4.15}$$

The objective funtion (4.10) minimizes the number of colors used in the model (the only available colors are from vertices in $S \cup W$). The equality (4.11) assigns each vertex in $V \setminus S$ to a representative vertex. The representative vertex can only be from the set $S \cup W$. It is possible for vertices in W to be colored or not (4.12). The new model reduces the number of variables in the problem, but introduces some symmetries. The colors in W are not distinguishable. For this reason we add constraints (4.13) to break symmetries. These constraints make sure that new colors are assigned in an order. Furthermore, if a solution with a smaller number of colors is found, this eliminates more variables. The maximal clique constraints (4.14) remain in the formulation but they are only added for the vertices in $S \cup W$. Finally constraints (4.15) are the domain constraints.

The formulation has a slightly worse lower bound than the model (4.1)-(4.5). In the original formulation if a vertex becomes representative, it is not allowed to represent certain vertices. However, newly added vertices in formulation (4.10)-(4.15) can represent all the vertices in the graph.

4.4 Computational study

The results of the computational study are presented in tables 4.1 and 4.2. We implemented our algorithm in CPLEX 12.2, IBM (June 2014). The computations are run on a desktop computer with 8.00 GB RAM and 2.53 GHz processor speed under Windows 7. We used 4 threads for parallel computation and set a CPU time limit of 3600 seconds. In all the instances, we placed a maximum clique at the beginning of the ordering and a maximum independent set in the remaining graph at the end of the ordering. For graphs with at most 100 nodes, we generated the rank inequalities for $\bar{N}^+(i)$ for each vertex i. If the number of nodes was less than 200, we generated the same inequalities for the vertices of the clique at the beginning of the ordering only. For bigger graphs, it did not make sense to generate the inequalities because of the intense computational effort.

The branch-and-cut algorithm performs especially well when the chromatic number and the clique number of a graph are close. However, certain types of graphs were problematic for the algorithm. In Mycielski graphs, the optimal solution is found quickly but proving optimality was not always possible (Mycielski5). However, separation routines in our algorithm helped us reduce the number of subproblems substantially compared to a standard branch-and-bound algorithm. However in some graphs, such as Queen9_9, the separation routine takes a lot of time. Even though obtaining an optimal solution can be easy using branching, the computation time was spent in the separation routine at the root node, without any improvement to the lower bound, which is already equal to the chromatic number of the graph. In Queen10_10 or bigger instances, we were not able to find the optimal solution. Such graphs may necessitate the use of a branch-and-price algorithm.

In other graphs, such as Queen8_8, mug88, and mug100 we were able to obtain a lower bound slightly better than the maximum clique size in the graph, which was enough to prove optimality. For instances such as school, school1, school1_nsh we run into memory problems.

4.5 Conclusion

In this chapter, we presented a branch-and-cut algorithm for the graph coloring problem. By using the asymmetric representative formulation developed by Campêlo et al. (2008) and various classes of valid inequalities we were able to

Graph			Cuts		Nodes		Best		Time	Gap
Instance	$\|V\|$	$\|E\|$	Cplex	User	Exp.	Rem.	LP	Int	(sec)	(%)
anna	138	986	0	0	0	0	11	11	0.98	
david	87	812	0	0	0	0	11	11	0.52	
homer	556	3258	0	108	0	0	13	13	335.82	
huck	74	602	0	0	0	0	11	11	0.48	
jean	77	508	0	0	0	0	10	10	0.20	
DSJC125_1	125	736	1	16096	0	1	5	7		28.57%
DSJC125_5	125	3891	201	3579	40	41	14.75	22		32.98%
DSJC125_9	125	6961	85	156	119	0	44	44	5.40	
DSJC250_1	250	3218	961	5769	0	1	4.00	26		84.60%
DSJC250_5	250	15668	267	17835	0	1	12.54	64		80.41%
fpsol2i1	269	11654	0	0	0	0	65	65	46.82	
fpsol2i2	363	8691	0	0	0	0	30	30	112.98	
fpsol2i3	363	8688	0	0	0	0	30	30	82.93	
games120	120	1276	0	31	0	0	9	9	13.91	
inithxi1	519	18707	0	0	0	0	54	54	251.29	
inithxi2	558	13979	0	0	0	0	31	31	356.68	
inithxi3	559	13969	0	0	0	0	31	31	295.48	
miles250	125	774	0	0	0	0	8	8	7.30	
miles500	128	2340	0	0	0	0	20	20	13.23	
miles750	128	4226	0	0	0	0	31	31	15.60	
miles1000	128	6432	0	0	0	0	42	42	17.36	
miles1500	128	10396	0	0	0	0	73	73	9.61	
mug88_1	88	146	0	0	0	0	4	4	2.67	
mug88_25	88	146	1	108	0	0	4	4	4.59	
mug100_1	100	166	0	0	0	0	4	4	4.35	
mug100_25	100	166	0	0	0	0	4	4	3.82	
mulsoli1	138	3925	0	0	0	0	49	49	3.23	
mulsoli2	173	3885	19	2902	0	0	31	31	26.50	
mulsoli3	174	3916	38	3966	0	0	31	31	25.29	
mulsoli4	175	3946	0	1195	0	0	31	31	27.44	
mulsoli5	176	3973	0	397	0	0	31	31	23.96	
myciel3	11	20	11	0	0	0	4	4	0.02	
myciel4	23	71	3	73	85	0	5	5	1.06	
myciel5	47	236	7	1226	88844	2411	5	6		16.67%
myciel6	95	755	21	8838	35	36	3.65	7		47.84%
myciel7	191	2360	49	36409	0	1	3.71	8		53.57%
pathfinder	109	211	0	0	0	0	6	6	0.13	
zeroini1	126	4100	0	0	0	0	49	49	0.76	
zeroini2	157	3541	0	0	0	0	30	30	1.40	
zeroini3	157	3540	0	0	0	0	30	30	1.25	

Table 4.1: Computational results.

Graph			Cuts		Nodes		Best		Time	Gap
Instance	$\|V\|$	$\|E\|$	Cplex	User	Exp.	Rem.	LP	Int	(sec)	(%)
queen5_5	25	160	0	0	0	0	5	5	0.02	
queen5_6	30	215	1	78	0	0	7	7	0.14	
queen5_7	35	275	0	0	0	0	7	7	0.08	
queen5_8	40	340	0	33	0	0	8	8	0.23	
queen6_6	36	290	5	188	0	0	7	7	0.75	
queen6_7	42	371	1	301	0	0	7	7	1.54	
queen6_8	48	458	1	531	13	0	8	8	18.69	
queen6_9	54	551	0	305	0	0	9	9	1.40	
queen7_7	49	476	0	0	0	0	7	7	0.31	
queen7_8	56	588	0	1777	0	0	8	8	9.48	
queen7_9	63	707	0	2124	2	0	9	9	12.45	
queen7_10	70	833	0	1559	26	0	10	10	98.81	
queen8_8	64	728	33	637	83	0	9	9	284.11	
queen8_9	72	876	0	16944	3	4	9	11		18.18%
queen8_10	80	1032	0	47889	1	2	10	11		9.09%
queen8_12	96	2736	0	3367	4	0	12	12	169.73	
queen9_9	81	1056	0	10786	1	2	9	12		25.00%
queen9_10	90	1245	0	41524	0	1	10	12		16.67%
queen10_10	100	1470	0	31055	0	1	10	13		23.08%
queen11_11	121	1980	0	28986	0	1	11	15		26.67%
queen12_12	144	5192	0	35091	0	1	12	17		29.41%
queen13_13	169	6656	0	26536	0	1	13	18		27.78%
queen14_14	196	8372	0	25265	0	1	14	21		33.33%

Table 4.2: Computational results for queen graphs.

obtain good results on a number of benchmark problems. It may be possible to extend the branch-and-cut algorithm into a branch-and-cut-and-price as suggested by Campêlo et al. (2008) as branching on a single variable is not always effective. Using local branching may be useful in order to find better incumbent solutions even though the upper bound obtained by the branch-and-cut algorithm is close to the optimal solution. Local branching, Fischetti and Lodi (2003), is used to search the neighborhood of an incumbent solution by limiting the number of changes.

Chapter 5

Safe dike heights in the Netherlands

Protection against increasing sea levels is an important issue around the world. In 2013 floods in Europe, Canada, Asia, and Australia accounted for 47% of the overall financial losses caused by natural disasters. Floods in Germany in 2013 resulted in a loss of €12 billion, whereas Hurricane Katrina in New Orleans in 2005 caused damages of $108 billion and claimed the lives of nearly 2000 people, Munich RE (2013). Protection against floods is a very important issue in the Netherlands as a large part of the country is below the sea level. The risk became apparent when a flood hit the Southwestern part of the Netherlands, Belgium, and United Kingdom in February 1953. The flood claimed more than 2000 lives and caused damage to the farms, buildings and livestock that amounted to 1 billion guilders (€450 million). With changes to the climate and increase in sea levels, new measures against floods become a necessity.

The first work on safe dike heights in the Netherlands was carried out by van Dantzig in 1953 when he was asked to do a cost-benefit analysis (CBA) on the optimal height of dikes as a result of the flood of 1953. In 1956 he published his results in a paper that is considered to be the start of Operations Research in the Netherlands, (van Dantzig, 1956). van Dantzig's CBA balances financial cost of heightening the dikes against the social benefit of security (avoiding material and immaterial damage). In our work, we are also going to use CBA, which is often used (by public and private sector) to analyze the desirability of a project. CBA helps predict whether the cost of a project (an investment decision) should

be incurred compared to the benefits it might bring (profit, social welfare, or security). In our case, the cost is related to the investments made for heightening or maintaining the dikes and the benefit is related to the security provided by the dikes to the people, infrastructure, and businesses that are under risk. Doing nothing for the dikes would reduce the costs related to the construction, but it increases the risk of floods and because of low level of protection the damage will be higher. On the other hand, one might consider the totally opposite approach of heightening the dikes as much as possible. There is another problem with this approach. First of all, even though the protection increases by increased dike heights, the costs of increasing dike height to such a level are also high. In addition to that, van Dantzig states that there is no upper limit for a completely safe dike height (which means complete protection against floods). For this reason, we try and balance the costs of heightening the dikes and the security benefits that will result from the height of the dikes.

In his analysis, which was used until recently, van Dantzig considers linear investment costs, which is a valid assumption for small heightenings. For the damage costs, he considers the exponential flood probability function. With the growth in the population and wealth, new safety standards for different areas had to be considered. Eijgenraam et al. (2010) and Brekelmans et al. (2012) were asked to determine new safety standards and solve the resulting problems. Eijgenraam et al. point out that van Dantzig's solution is not optimal if one also considers the growth in the population and wealth (which corresponds to an increase in potential damage). Furthermore, unlike Eijgenraam et al. (2010), van Dantzig does not consider the timing of dike heightenings, which is a result of the urgency of the situation in 1953. With the increased risk, it is necessary to develop a different CBA for the safe dike height problem.

Before going into the details of the problem, we explain terminology and geographical configuration of the dikes in the Netherlands. A dike segment is part of a dike that is protecting a region. It is possible that several segments protect the same area and in that case they are called a dike ring. In the Netherlands, dike ring areas and smaller dikes lie beneath the Afsluitdijk, which is the most outer dike located in the north. The Afsluitdijk separates the North Sea and the IJsselmeer, an artificial lake. The IJsselmeer itself is separated by the Houtribdijk, another major dike segment. The smaller dike segments behind the two major dikes protect in total 53 areas with a higher safety standard (yearly flood probability) than 1/1000.

In their analysis, Eijgenraam et al. (2010) consider the homogeneous model, where flood probabilities and investment costs are not dependent on dike rings. In the homogeneous model, we assume that all dike segments in a dike ring are kept at the same level. In other words they are heightened and maintained together. Furthermore the risk associated with a failure does not depend on the individual segments of the dike ring. The authors use an exponential distribution for flood probabilities and exponential and quadratic investment functions. They solve their model for each dike ring separately and show that under certain conditions the increase in dike heights is periodic. For other classes of problems they present a dynamic programming approach based on a discretization of dike heights.

In a subsequent paper, Brekelmans et al. (2012), the nonhomogeneous model is discussed. In the nonhomogeneous model, different segments of a dike ring can be treated separately with respect to flooding probabilities and investment costs. Furthermore, the flooding probability of a dike ring depends on its weakest segment. The problem is modeled as a non-linear programming (NLP) model and is solved as a mixed-integer NLP model by employing a discretization of the planning horizon. The authors also discuss the robustness of solutions with respect to different scenarios.

In our problem, we consider the homogeneous case but we use a different approach and cost structure. For a single dike segment that is protecting a single region, there are two cost components for determining the optimal dike height: the construction costs and the expected damage costs. The construction costs depend on the current level of the dike and the increase in height. The expected damage costs are determined using the probability of a flood happening and the damage such a flood might create with a higher dike level resulting in a smaller probability of flood. The geographical configuration of the dikes also determines the structure of the damage costs. Consider a region that is protected by two dike rings, such as the Afsluitdijk protecting other dike segments that are located next to the IJsselmeer. In this case when a single dike breaks, the damage to the region will be limited because of the other dike protecting the region. However, if both of them fail, then the damage will be considerably higher. In the calculation of expected damage costs, we consider all these cases and therefore the expected damage costs will depend on the level of the two dikes. A representation of the dike structure in the Netherlands is shown in Figure 5.1. The Afsluitdijk is the outer dike. To the south of the Afsluitdijk lies a lake with other dikes. Among those dikes, the Houtribdijk has a special place as it is also the parent dike for the region that lies

in the south. So the dikes D,E, and F are protected by the Afsluitdijk and the Houtribdijk and dikes B and C are protected by the Afsluitdijk. As can be seen from the example, a parent dike can have many children and its children can be the parent of other dikes.

In our problem, we use the damage and investment cost functions used in Eijgenraam et al. (2010) and Brekelmans et al. (2012). However, our problem differs from their problem in some respects. Unlike these two papers discretization of the planning horizon and dike heightenings are an integral part of our model, as introduced by Zwaneveld and Verweij (2014). As explained before, the damage costs in our model depend on the level of multiple dikes. Our model is more versatile as the investment or damage costs are simply the coefficients and the solution method does not depend on the cost functions we use. In other words, we do not have to restrict the structures of the cost functions (construction and damage) in the integer programming approach. Furthermore, the integer programming approach allows us to consider different scenarios, such as risks associated with failures in the mechanisms of the dikes, see Zwaneveld and Verweij (2014).

In the subsequent sections of this chapter we first explain the dike structure in the Netherlands and present the integer linear programming formulation for the problem, Section 5.1. Then, we discuss valid inequalities and their separation in Section 5.2. In Section 5.3, we show that the problem is polynomially solvable for a fixed number of dikes. We extend our model to other scenarios in Section 5.4 and present our computational study in Section 5.5.

5.1 Problem definition

CBA of determining safe dike heights in the Netherlands requires balancing two cost components: investment costs for heightening the dikes and expected loss that might occur as a result of a flood. The construction costs have a fixed and a variable component. They depend also on the initial height of the dike and the heightening. It is assumed that the present value of the costs of construction decreases in time due to advances in materials and construction technology and financial reasons. The loss corresponds to the expected damage in case there is a flood in an area. The damage costs include the physical damage to cities, infrastructure, buildings, and fields in a flooded region, reallocation and rescue of people, and the casualties. The construction costs for dikes are independent from the state or possible construction works on other dikes. However, for the expected

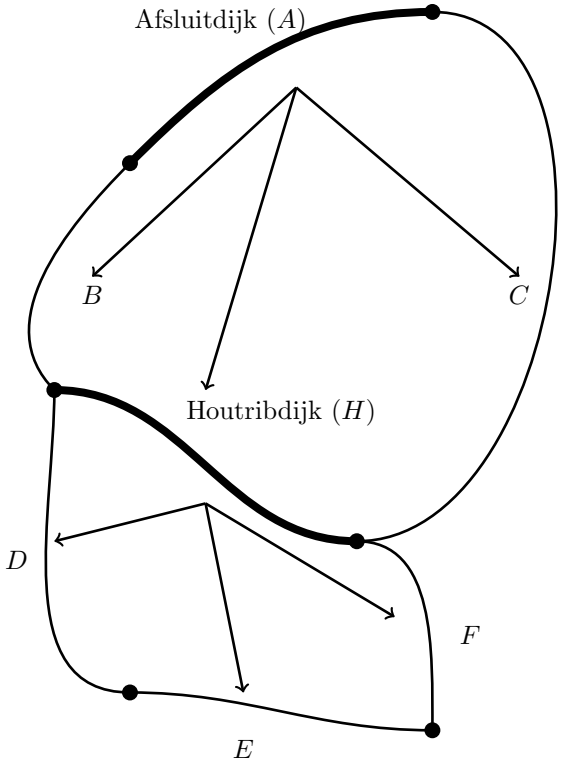


Figure 5.1: Representation of the structure of the dikes in the Netherlands.

damage costs there is a special structure. It is possible for more than one dike to protect the same region, when that dike has a parent. Note that the parent dike does not have to be next to the region. However, the parent dike limits the damage as it can be used in order to manipulate the water levels. In this case, if the parent dike fails, then the child dike has a higher probability of failure. For this reason, the expected damage costs depend on the level of the parent and the child dikes. See Figure 5.1.

Mathematical model

In this section we give the mathematical formulation of the safe dike heights problem, as posed in Zwaneveld and Verweij (2014). We write the model for a two dikes scenario: the A-dike as the parent and the B-dike as the child. The model is straightforwardly extended to a multi-dike scenario. The sets and the variables in our model are defined as follows:

$t \in Y = \{0, 1, \ldots, T\}$	set of time periods.
$i \in I = \{0, 1, \cdots, L_A\}$	set of levels for the A-dike.
$j \in J = \{0, 1, \cdots, L_B\}$	set of levels for the B-dike.
$I^A(t, i_1, i_2)$	cost of heightening the A-dike from level i_1 to level i_2 in period t.
$I^B(t, j_1, j_2)$	cost of heightening the B-dike from level j_1 to level j_2 in period t.
$S(t, i, j)$	expected damage costs in period t with the A-dike at level i and the B-dike at level j.

$$x_A(t, i_1, i_2) = \begin{cases} 1, & \text{if the } A\text{-dike is heightened from level } j_1 \text{ to level } j_2 \text{ in period } t, \\ 0, & \text{otherwise.} \end{cases}$$

$$x_B(t, j_1, j_2) = \begin{cases} 1, & \text{if the } B\text{-dike is heightened from level } j_1 \text{ to level } j_2 \text{ in period } t, \\ 0, & \text{otherwise.} \end{cases}$$

$$p(t, i, j) = \begin{cases} 1, & \text{if the } A\text{-dike is at level } i \text{ and the } B\text{-dike is at level } j \text{ at the end of period } t, \\ 0, & \text{otherwise.} \end{cases}$$

The Integer Linear Programming (ILP) formulation for the safe dike heights problem is given in equations (5.1)-(5.10):

$$\begin{aligned} \min & \sum_{t \in Y} \sum_{i_1 \in I} \sum_{i_2 \in I : i_2 \geq i_1} I^A(t, i_1, i_2) x_A(t, i_1, i_2) + \\ & \sum_{t \in Y} \sum_{j_1 \in J} \sum_{j_2 \in J : j_2 \geq j_1} I^B(t, j_1, j_2) x_B(t, j_1, j_2) + \\ & \sum_{t \in Y} \sum_{i \in I} \sum_{j \in J} S(t, i, j) p(t, i, j), \end{aligned} \tag{5.1}$$

$$\text{s.t.: } x_A(0, 0, 0) = 1, \tag{5.2}$$

$$x_B(0, 0, 0) = 1, \tag{5.3}$$

$$\sum_{i_1:i_1\leq i_2} x_A(t,i_1,i_2) = \sum_{i_3:i_2\leq i_3} x_A(t+1,i_2,i_3), \qquad \forall t \in Y \setminus \{T\}, i_2 \in I, \tag{5.4}$$

$$\sum_{j_1:j_1\leq j_2} x_B(t,j_1,j_2) = \sum_{j_3:j_2\leq j_3} x_B(t+1,j_2,j_3), \qquad \forall t \in Y \setminus \{T\}, j_2 \in J, \tag{5.5}$$

$$\sum_{j\in J} p(t,i,j) = \sum_{i_1:i_1\leq i} x_A(t,i_1,i), \qquad \forall t \in Y \setminus \{0\}, i \in I, \tag{5.6}$$

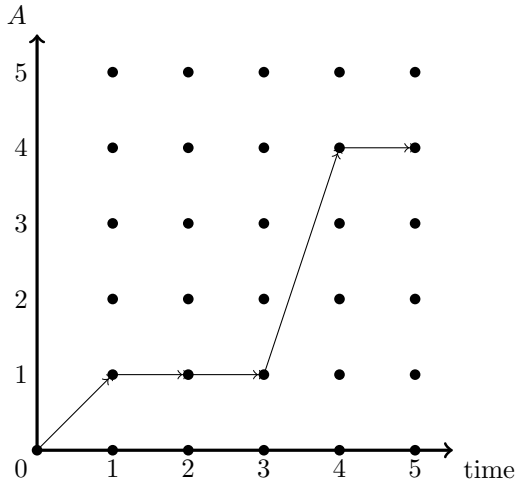
$$\sum_{i\in I} p(t,i,j) = \sum_{j_1:j_1\leq j} x_B(t,j_1,j), \qquad \forall t \in Y \setminus \{0\}, j \in J, \tag{5.7}$$

$$x_A(t,i_1,i_2) \in \{0,1\} \qquad \forall t \in Y, i_1, i_2 \in I, i_2 \geq i_1, \tag{5.8}$$

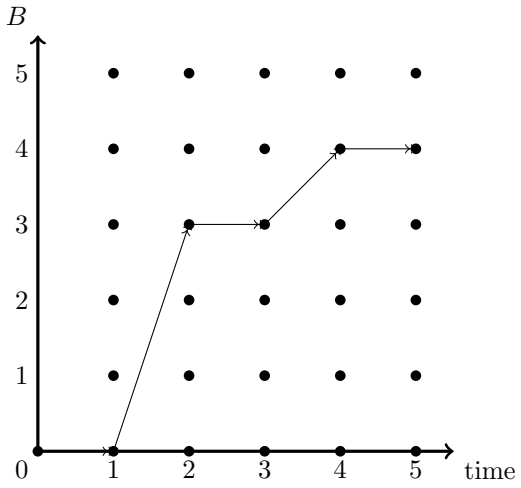
$$x_B(t,j_1,j_2) \in \{0,1\} \qquad \forall t \in Y, j_1, j_2 \in J, j_2 \geq j_1, \tag{5.9}$$

$$p(t,i,j) \in \{0,1\} \qquad \forall t \in Y, i \in I, j \in J. \tag{5.10}$$

The objective function (5.1) contains the heightening costs for the two dikes A and B and the expected damage costs of having a flood which depends on the levels of both dikes. The constraints (5.2)-(5.3) set the level of the dikes at their current level for the beginning of the planning horizon (for time period 0). Constraints (5.4)-(5.5) are the balance constraints for the level of the dikes. If a dike is at level i at the end of time period t, then at the start of time period $t+1$ the starting level of the dike must be i. The underlying network flow structure for these constraints are shown in figures 5.2(a) and 5.2(b). In constraints (5.6)-(5.7) we connect the state of the dikes to the damage variable that depends on the state of the parent and the child dike. If $p(t,i,j) = 1$, then the A-dike should be at level i at the end of the time period t (or $x_A(t,i_1,i) = 1$ for some level $i_1 \leq i$). Similarly, the B-dike must be at level j (or $x_B(t,j_1,j) = 1$ for some level $j_1 \leq j$). The inequalities are further explained in Figure 5.3. Finally, constraints (5.8)-(5.10) are the domain constraints for the variables.



(a) Flow network for the level of the A-dike related to constraints (5.4). At time period 1 A-dike is brought from level 0 to level 1, i.e. $x_A(1, 0, 1) = 1$.



(b) Flow network for the level of the B-dike related to constraints (5.5). At time period 2 B-dike is brought from level 0 to level 3, i.e. $x_B(2, 0, 3) = 1$.

Figure 5.2: Representation of the flow balance constraints (5.4)-(5.5).

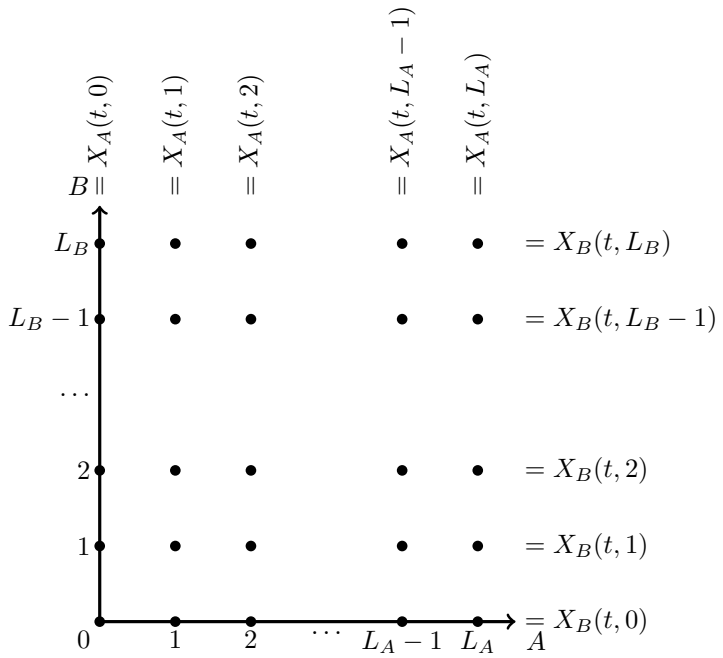


Figure 5.3: Inequalities (5.6)-(5.7) connect the levels of two dikes to the damage variables, shown by using dots. If a damage variable is selected for time period t, then the level of two dikes must be consistent with the selected variable. To facilitate the illustration of the model we use $X_A(t,i) = \sum_{i_1:i_1 \leq i} x_A(t,i_1,i)$ and $X_B(t,j) = \sum_{j_1:j_1 \leq j} x_B(t,j_1,j)$. In other words $X_A(t,i) = 1$ means that the A-dike is at level i at the end of time period t.

The polytope of the safe dike heights problem

We call the convex hull formed by all feasible points to the problem (5.2)-(5.10), $conv(P)$. We know that the A-dike can be heightened from level 0 to level L_A and the B-dike from level 0 to level L_B. There are T time periods from 1 to T, with time period 0 representing the initial time period. Then we have the following result for $conv(P)$:

Theorem 5.1. *The dimension of the polytope for a two dikes problem is* $[(L_A+1)(L_B+1)-1]\,T + \frac{(L_A)(L_A-1)}{2}(T-1) + \frac{(L_B)(L_B-1)}{2}(T-1)$.

Proof. For each time period from 1 to T, the model has $(L_A+1)(L_B+1)$ damage variables. For the first time period each dike has respectively L_A+1 and L_B+1 construction variables in the form of $x_A(1,0,0),\ldots,x_A(1,0,L_A)$. For the time periods 2 to T there are in total $\frac{(L_A+2)(L_A+1)}{2} + \frac{(L_B+2)(L_B+1)}{2}$ construction variables. Considering the variable concerning the initial condition of the dikes ($x_A(0,0,0)$) the problem has $[(L_A+1)(L_B+1)]\,T + \frac{(L_A+2)(L_A+1)+(L_B+2)(L_B+1)}{2}(T-1) + L_A + L_B + 4$ variables.

- Lower bound: In order to find a lower bound for the dimension of the polytope we consider the cases where dikes are only allowed to be heightened at most once in the planning horizon and are kept at that level. Furthermore, any heightening of the dikes happens at the same time period t. Note that the solution where the dikes are at level 0 throughout the planning horizon is feasible for the problem, denoted by $\mathbf{x^0}$. We consider a solution where the A-dike is heightened from level 0 to level 1, i.e. $x_A(T,0,1)=1$. We subtract $\mathbf{x^0}$ from this solution to obtain difference vector $(x_A(T,0,1), p(T,1,0) : 1; x_A(T,0,0), p(T,0,0) : -1)$. In this part of the proof p variables are enough to prove linear independence. It is possible to obtain $(L_A+1)(L_B+1)-1$ linearly independent vectors by making each of the p variables in time period T (except for $p(T,0,0)$) equal to one. By doing the same thing for time periods $T-1, T-2,\ldots,1$ we obtain $[(L_A+1)(L_B+1)-1]\,T$ linearly independent vectors.

  So far we only considered heightening the dikes at most once. This means that the variables $x_A(t,i_1,i_2)$, for which $i_1 \neq 0$ and $i_2 > i_1$, were all equal to zero in the solutions we considered. The same applies for the same type of variables related to the B-dike. Consider a solution, where the B-dike is at level 0 and the A-dike is at level $i_1 \neq 0$ at the beginning of the time

period T. By setting $x_A(T, i_1, i_2) = 1$, where $i_2 > i_1$ we obtain a solution where $x_A(T, i_1, i_2) = 1$. The vector we obtain is $(x_A(T, i_1, i_2), p(T, i_2, 0) : 1; x_A(T, i_1, i_1), p(T, i_1, 0) : -1)$. It is linearly independent from the previous vectors, as we set $x_A(T, i_1, i_2) = 0$ previously by only using solutions with a single heightening. By considering solutions where we use a single variable of this type we obtain $\frac{(L_A)(L_A-1)}{2}$ and $\frac{(L_B)(L_B-1)}{2}$ linearly independent vectors for $T-1$ time periods for dikes A and B. Note that in the first time period this type of variables do not exist.

- Upper bound: The number of linearly independent equalities in the formulation is $(L_A+1)(T-1)+2+(L_B+1)(T-1)+2+(L_A+L_B+1)T$. The formulation contains flow balance constraints for 2 dikes, which are obviously linearly independent between themselves. The network in our formulation corresponds to the following construction: For the beginning of each year we create the nodes for levels and for the last year we add a sink node instead of the levels and connect it to the previous year (possibly with parallel arcs). The existence of variable $x_A(0, 0, 0)$ implies the existence of a source node that is only connected to level 0 at the beginning of time period 0. For the A-dike, there are $(L_A+1)(T-1)+3$ nodes in the network which corresponds to $(L_A+1)(T-1)+2$ linearly independent inequalities, which is equal to the number of nodes in the network minus one, Papadimitriou (1994). Note that the networks shown in figures 5.2(a) and 5.2(b) are modified slightly by adding a source node before the time period 0 and a sink node after time period $T-1$. Furthermore we remove the levels for time period T. In addition to that there are $L_A + L_B + 1$ linearly independent equalities for the constraints that connect the damage variables with the construction variables, as shown in Figure 5.3. We can use $L_A + 1$ constraints for the A-dike and L_B constraints for the B-dike, which corresponds to $(L_A + L_B + 1)T$ linearly independent equalities, as shown in Figure 5.3. In total the lower bound for the number of linearly independent inequalities is $(L_A+1)(T-1)+2+(L_B+1)(T-1)+2+(L_A+L_B+1)T$ linearly independent equality constraints.

□

5.2 Valid inequalities

We now discuss the valid inequalities of the two dikes problem (with dikes A and B). We restrict ourselves to one type of inequalities that involve variables from two consecutive time periods t, $t+1$. Furthermore, the inequalities we focus on have only damage variables for time period t in the lhs. For a set of damage variables on the lhs we want to be able to cover all possibilities in the time period $t+1$. As it is not allowed to decrease the height of a dike, this corresponds to being able to reach all the states in time period $t+1$, where the A-dike is at least at level i and the B-dike is at least at level j for each variable $p(t,i,j)$ on the lhs. The inequalities have the following form:

$$\sum p(t,i,j) \leq \sum p(t+1,i,j) + \sum x_A(t+1,i,i_1) + \sum x_B(t+1,j,j_1).$$

For each variable $p(t,i,j)$ in the lhs, we must cover all the possibilities in time period $t+1$ either with construction variables x_A, x_B or damage variables p in order to make the inequalities valid. In the inequalities the sum of the damage variable for the time period t in the lhs can be at most equal to 1. Every possible state in the time period $t+1$ should be reachable by damage or construction variables. An example of a representation of such an inequality is given in Figure 5.4. In the figure the damage variables are represented with dots and construction and maintenance variables are represented by arrows. In the lhs we only use one variable, $p(t,2,2)$. The set of variables in the rhs, shown in Figure 5.4(b) for time period $t+1$, yields the following valid inequality.

$$p(t,2,2) \leq \sum_{j\in\{2,3,4\}} p(t+1,2,j) + \sum_{j\in\{2,3,4\}} p(t+1,4,j) + x_A(t+1,2,3)$$

In the rhs we do not include the damage variables $p(t+1,3,2)$, $p(t+1,3,3)$, and $p(t+1,3,4)$. However, it is possible to reach these states if $x_A(t+1,2,3)=1$, making the inequality valid.

The first class of valid inequalities we observe are the valid inequalities containing only damage variables. In this case, to be facet-defining the damage variables in the rhs must also appear in the lhs. Otherwise the damage variables that are not in the lhs are 0 for the tight solutions and the inequality is dominated by the

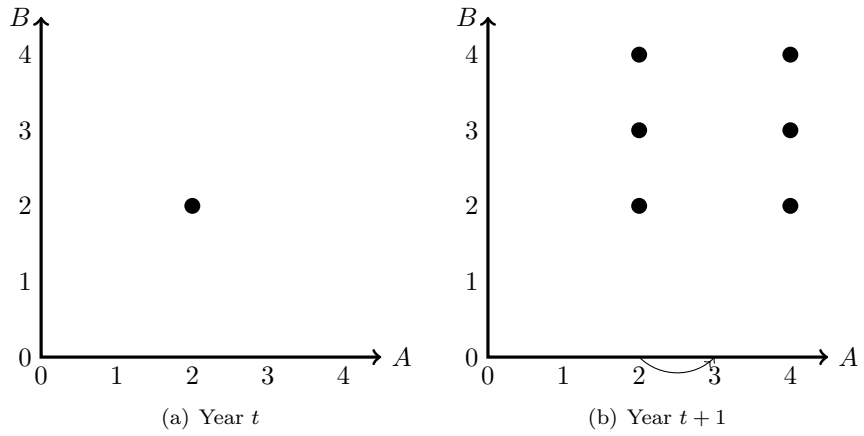


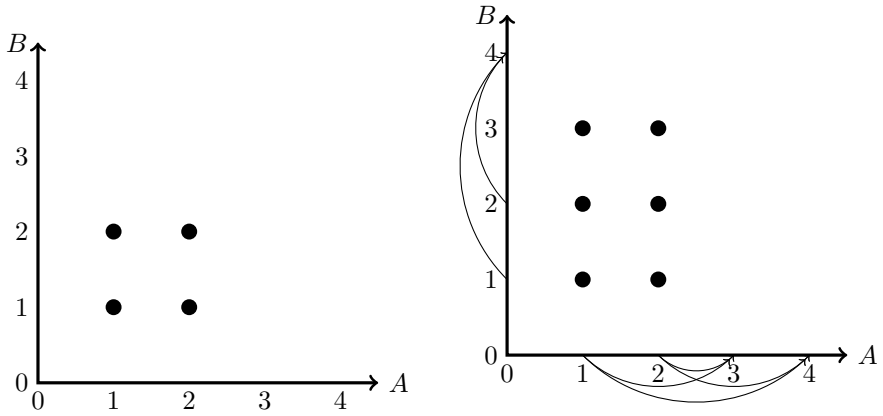
Figure 5.4: A valid inequality.

domain constraints. If a variable in the lhs is not included in the rhs the inequality is not valid. Furthermore, in the case where all the damage variables are included for the top level of one dike, we have to include all the damage variables for the top level of the other dike. Consider the inequality

$$\sum_{i \in I} p(t, i, L_B) \leq \sum_{i \in I} p(t+1, i, L_B).$$

In this inequality, for every tight solution, we restrict the set of variables $x_B(t+1, 0, L_B), \ldots, x_B(t+1, L_B-1, L_B)$ to take value 0. If one of the mentioned variables has value 1, then the $rhs = 1$, but $lhs = 0$ and the solution is not tight.

Now, we discuss valid inequalities that also contain construction variables on the rhs. First, we put some more restrictions on the inequalities we observe. The first restriction concerns the damage variables on the lhs of the inequality. Consider for a level i of the A-dike we use the damage variables associated with levels $\{j_1, j_2, \ldots, j_n\}$ for the B-dike. Then, for any other level i' of the A-dike we either use the same set of levels for the B-dike or we do not use any. Alternatively, we can say that the variables in the lhs are the Cartesian product of two sets of levels for the dikes. We use the same structure for the damage variables on the rhs. Furthermore, we consider the cases where the damage variables on the lhs are a subset of the damage variables on the rhs. In order to focus on facet-defining inequalities, we only use valid inequalities, where each state in rhs are covered only once, i.e. a state on the rhs can only be reached by a damage



(a) Year t (b) Year $t+1$

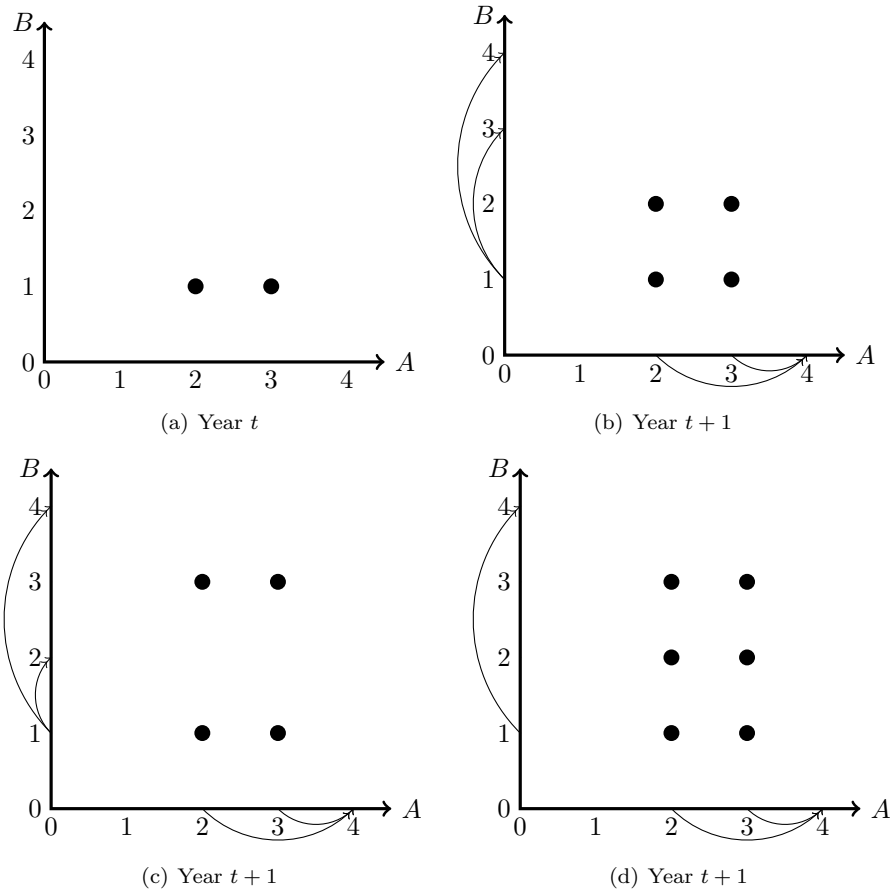
Figure 5.5: The inequalities are facet defining.

variable or a set of construction variables, but not both. If a state is covered by both type of variables, then the corresponding damage variable is 0 in all the tight solutions and the domain constraints dominate the valid inequality. Consider that the valid inequality in Figure 5.4 also contains variable $p(t+1,3,4)$, which can already be reached using variable $x_A(t+1,2,3)$. In this case in the tight solutions $p(t+1,3,4)$ is restricted to value 0. If $p(t+1,3,4)=1$ then the $lhs=1$, which implies $p(t,2,2)=1$. In that case, we also use variable $x_A(t+1,2,3)$ and any solution with $p(t+1,3,4)=1$ is not tight for the inequality.

Another observation we made restricts the use of the damage variables in the rhs for the maximum level of a dike, unless they are already in the lhs. The valid inequality in Figure 5.4 is not facet-defining. Consider a tight solution, where $p(t+1,4,2)=1$, which implies $p(t,2,2)=1$. So, in the tight solutions $p(t,4,2)=0$. For the levels that are not equal to L_A or L_B, we can use damage or construction variables. Examples of facet-defining inequalities are given in figures 5.5 and 5.6. In Figure 5.6, we give an example of four facet defining inequalities with the same lhs.

Separation of valid inequalities

The formulation (5.2)-(5.10) has a flow network structure for the construction of the dikes. The damage variables of the problem should also follow the flow in the network structure. A fractional solution does not belong to $conv(P)$ if the



(a) Year t

(b) Year $t+1$

(c) Year $t+1$

(d) Year $t+1$

Figure 5.6: Four facet defining inequalities with the same lhs.

construction variables and damage variables are not coupled. To separate the violated inequalities we use the following scheme:

We look for the first time period where the solution is fractional. For each fractional damage variable $p(t,i,j)$ we check whether the construction and damage variables are consistent in the time period $t+1$. In this context, the variables are consistent if the damage variables for the time period t and the construction variables for the time period $t+1$ lead to the correct damage variables of the time period $t+1$. If the variable $p(t,i,j)$ has a fractional value and $x_A(t+1,i,i_1) > 0$ and $x_B(t+1,j,j_1) > 0$, then in the time period $t+1$ the variable $p(t,i,j)$ should be followed by the variable $p(t+1,i_1,j_1)$.

To separate the violated inequalities, for each variable $p(t,i,j)$ with a fractional value, we check the value of the corresponding construction variables in the time period $t+1$. If $x_A(t+1,i,i_1) > 0$ for some $i_1 \in \{i, i+1, \ldots, L_A\}$ and $x_B(t+1,j,j_1) > 0$ for some $j_1 \in \{j, j+1, \ldots, L_B\}$, then in the next period the damage variable $p(t+1,i_1,j_1)$ must be chosen. If this is not the case we check whether the following inequality is violated:

$$p(t,i,j) \leq p(t+1,i,j) + \sum_{i_1 \in I: i_1 > i} x_A(t+1,i,i_1) + \sum_{j_1 \in J: j_1 > j} x_B(t+1,j,j_1).$$

If the inequality is violated, we add it to the model and solve the problem again. If the inequality is not violated we try to replace construction variables with damage variables as follows:

$$\begin{aligned} p(t,i,j) \leq & p(t+1,i,j) + p(t+1,i+1,j) \\ & + \sum_{i_1 \in I: i_1 > i+1} x_A(t+1,i,i_1) + \sum_{j_1 \in J: j_1 > j} x_B(t+1,j,j_1). \end{aligned}$$

To explain the idea we use the example in Table 5.1 with two dikes and three levels for each dike.

In the time period t the variables $p(t,0,1) = p(t,1,0) = 0.5$. In the time period $t+1$ the variable $x_B(t+1,0,2) = 0.5$ and the variable is associated with the damage variable $p(t,1,0)$ from the previous time period. It means that the damage variable $p(t,1,0)$ should be followed by the variable $p(t+1,1,2)$. The following inequality is violated:

$$p(t,0,1) \leq p(t+1,0,1) + \sum_{i \in \{1,2,3\}} x_A(t+1,0,i) + \sum_{j \in \{2,3\}} x_B(t+1,1,j).$$

$$
\begin{array}{l}
p(t,0,1) = 0.5 \\
p(t,1,0) = 0.5 \\
x_A(t+1,0,0) = 0.5 \\
x_A(t+1,1,1) = 0.5 \\
x_B(t+1,0,2) = 0.5 \\
x_B(t+1,1,1) = 0.5 \\
p(t+1,0,2) = 0.5 \\
p(t+1,1,1) = 0.5
\end{array}
$$

Table 5.1: Fractional solution that does not belong to $conv(P)$.

The inequalities with a single damage variable in the lhs were enough to solve the instances optimally.

5.3 Computational complexity for fixed number of dikes

If the damage costs for the dikes are independent from the state of the parent dikes (if any) the problem of determining optimal dike heights can be formulated and solved for each dike separately. In this case the problem corresponds to a shortest path problem in a network where the nodes are the heights of dikes in different time windows and the arcs correspond to possible transitions from one state to another in the next time period. The cost of the arcs are the damage and construction costs corresponding to the transition from the state of the tail node to the state of the head node. There are only arcs between nodes belonging to consecutive time periods and there is no arc from a certain height to a lower height in the next time period. In a problem with T time periods and L_A levels for the dike, the network for the minimum flow problem has $(L_A + 1)T + 2$ nodes.

In addition to that, for a fixed number of dikes the problem can be solved in polynomial time. For a 2 dikes problem (with any kind of parent-child relation tree), we construct a network to solve the problem in polynomial time. In the network, for each time period in the planning horizon we create $(L_A + 1)(L_B + 1)$ nodes corresponding to the states of the dikes A and B. A node $N_1 = (t, i_1, j_1)$ corresponds to the state where the first dike is at level i_1, the second dike is at level j_1 at the end of the time period t. If at the next time period, we reach the state $N_2 = (t + 1, i_2, j_2)$, it means that the A-dike is brought from level i_1 to level i_2, the B-dike is brought from level j_1 to level j_2. This incurs the construction

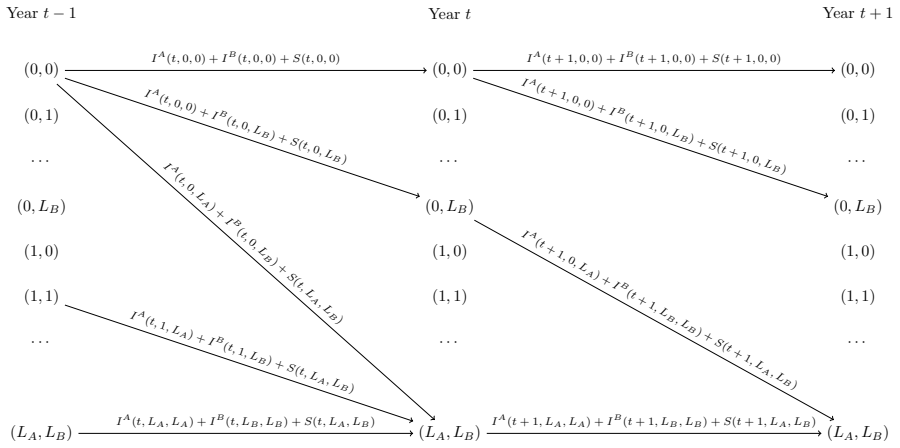


Figure 5.7: The dynamic programming approach for a 2 dike problem.

costs for the dikes denoted by $I^A(t+1,i_1,i_2), I^B(t+1,j_1,j_2)$. Furthermore, we know that in time period $t+1$, the expected damage cost is $S(t+1,i_2,j_2)$. So the arc between N_1 and N_2 has the mentioned construction and damage costs. As we can only increase or maintain dike heights, we do not have arcs between the states where a certain dike height decreases in time.

By using this cost structure and adding a source (representing initial conditions in time period 0 where all dikes are at level 0) and a sink node we obtain a shortest path problem that determines the optimal policy for determining dike heights with minimum cost. An example for the shortest path graph is given in Figure 5.7. The whole network has $(L_A+1)(L_B+1)T+2$ nodes. In our study, we use dynamic programming to solve the resulting shortest path problem. At time period t we note $c(t,i,j)$ as the minimum cost of being at height i for the A-dike and height j for the B-dike. A representation of the network is given in Figure 5.7. From a dynamic programming point of view we have

$$c(t+1,i_2,j_2) = \min_{\substack{i_1:i_1\leq i_2\\ j_1:j_1\leq j_2}} \{c(t,i_1,j_1) + I^A(t+1,i_1,i_2) + I^B(t+1,j_1,j_2) + S(t+1,i_2,j_2)\}.$$

It is easy to extend the idea for problems with more than two dikes. For each additional dike, the nodes of the network have an extra index. So, for each time

period we have $(L_A + 1)(L_B + 1) \dots (L_Z + 1)$ nodes. The arc costs contain the construction costs for all dikes and the damage costs, independent on its structure. For this reason, even though the problem is polynomially solvable for a fixed number of dikes, the number of states for the dynamic program increases a lot with the addition of new levels and dikes. Even for a moderate number of dikes and levels the dynamic programming problem cannot be practically solved. A single time period in the planning horizon for the instances 3 and 4 has 4.04×10^{12} states, whereas the ILP formulation has 123000 variables.

However, the dynamic programming approach also has an advantage. The performance of the dynamic programming formulation does not depend on the structure of the damage costs. In the ILP formulation, when the structure for the damage costs is different, we have to add new variables and constraints to the problem. In dynamic programming, all these costs are associated with the transitions from a state to another and the computation time is not drastically affected by the changes to the model.

5.4 Extensions to the ILP formulation

ILP formulation (5.1)-(5.10) allows dikes to be heightened on consecutive time periods. However, by heightening a dike we incur fixed and variable costs. For that reason it makes sense to heighten a dike once instead of doing it twice in short intervals. Therefore, it is logical to put a time limit between two consecutive construction works on a dike. For this purpose we define the following problem parameters:

- t_A the minimum amount of time between two consecutive constructions on A-dike,
- t_B the minimum amount of time between two consecutive constructions on B-dike.

The following model extension limits the time between two consecutive construction works:

$$\begin{aligned} &\min(5.1), \\ \text{subject to: } &(5.2) - (5.10), \\ &\sum_{\substack{t_1 : t \le t_1 \le t + t_A \\ i_1 < i_2}} x_A(t_1, i_1, i_2) \le 1, \qquad \forall t \in Y - \{0, y\}, \end{aligned} \tag{5.11}$$

$$\sum_{\substack{t_1: t \le t_1 \le t+t_B \\ j_1 < j_2}} x_B(t_1, j_1, j_2) \le 1, \qquad \forall t \in Y - \{0, y\}, \tag{5.12}$$

The minimum time between two consecutive constructions on dikes A and B are covered with constraints (5.11) and (5.12). In the constraint (5.11) if $x_A(t, i_1, i_2) = 1$, where $i_1 < i_2$, then until time $t + t_A$ it is not possible to do another construction on the same dike.

5.5 Computational study

In our study we considered 4 scenarios. The first two instances are essentially the same with a finer and a more crude discretization in terms of dike levels and time periods. The instances have one parent dike and five children dikes. The third and fourth instances correspond to two different scenarios concerning the gates and the pumps on the dikes. The instances have one parent dike and 10 children dikes.

In the previous sections of this chapter, we only considered the scenario with two dikes. However, extending our model to multiple dikes is straightforward. In a multiple children scenario, each dike has its own set of flow balance constraints with respect to its level. For the damage costs we create damage variables for each child dike. With A-dike being the parent dike and B, C, D, E, F-dikes being children dikes we create a set of damage variables for relation between A and B, A and C, ..., and A and F together with the required constraints.

We use CPLEX 12.2, IBM (June 2014) for the computational study on a desktop computer with 8.00 GB RAM and 2.53 GHz processor speed under Windows 7.

In Table 5.2 the constraints on the minimum time between two consecutive constructions are dropped and the solution to model (5.1)-(5.10) is given. In Table 5.3, we add the minimum time constraint between two consecutive constructions with a time limit of 10 years for the parent dike and 5 years for the children. In all of the instances, the optimal solution is found in a reasonable time at the root node. There were two instances where the optimal LP solution was not integer. However, even those instances were solved at the root node with various types of cuts. In the problems, the constraint about the minimum time between two consecutive constructions is not violated. This may be due to the high fixed cost

Dikes	Levels	Periods	Columns	Rows	LP-OPT	ILP-OPT	Time	Cuts
6	19	101	349608	36323	1074.394	1074.401	63.40	8
6	10	56	54183	10484	1110.921	1110.921	1.95	0
11	14	37	122989	17174	2680.676	2680.676	6.52	0
11	14	37	122989	17174	1672.641	1672.644	10.25	45

Table 5.2: Computational results without consecutive construction constraints.

Dikes	Levels	Periods	Columns	Rows	LP-OPT	ILP-OPT	Time	Cuts
6	19	101	349608	37023	1074.394	1074.401	119.73	8
6	10	56	54183	10869	1086.908	1086.908	8.19	0
11	14	37	122989	17606	2652.249	2652.249	12.67	0
11	14	37	122989	17606	1646.092	1646.093	15.41	45

Table 5.3: Computational results with consecutive construction constraints.

of constructions. For this reason it seems logical to add these constraints when they are necessary to obtain faster computation times.

By analyzing the optimal solutions to instances, we noticed that children dikes are heightened before the parent dikes. Furthermore, they are heightened in smaller steps. Usually children dikes are heightened around year 2040. The extra protection provided by heightening children dikes allows us to postpone construction on parent dikes. These dikes are heightened around year 2090. For children dikes heightenings seem to have a period of 40-50 years initially and later on (around year 2200) they are heightened more frequently. Parent dikes are heightened less frequently, with a period of 70 years between two heightenings. The results obtained by different discretization schemes seems to be consistent. In the crude discretization, dikes are heightened slightly later but they are heightened to a higher level.

Comparison with flow formulation

We also compared the dynamic programming and the integer programming methods in terms of the computation time for two and three dikes scenarios. Table 5.4 shows the results for two dikes problems and Table 5.5 shows the results for three dikes scenarios. In problems with two dikes, the dynamic programming approach is faster than the integer programming approach. However, when we add one more dike to the problem, the integer programming gives better results. We also ran the dynamic programming algorithm for a four dikes scenario and could only find the optimal solution after one hour, which justifies the usage of the integer

programming approach as we managed to solve the same problem instance with all 6 dikes in just over a minute as shown in Table 5.2.

Parent	Child	Optimal	Time(DP)	Time(ILP)
Afsluitdijk	Houtribdijk	49.512	0.08	4.40
Afsluitdijk	nfl	398.602	0.08	4.39
Afsluitdijk	nop	307.633	0.08	4.14
Afsluitdijk	wfr	279.032	0.06	4.50
Afsluitdijk	wie	95.007	0.08	2.88
Afsluitdijk	zwf	130.726	0.08	8.30

Table 5.4: Solutions and computation times for two dikes scenarios.

Parent	Child 1	Child 2	Optimal	Time(DP)	Time(ILP)
Afsluitdijk	Houtribdijk	nfl	419.374	17.83	13.38
Afsluitdijk	Houtribdijk	nop	328.835	17.95	10.39
Afsluitdijk	Houtribdijk	wfr	302.595	17.94	13.63
Afsluitdijk	Houtribdijk	wie	115.027	17.84	10.64
Afsluitdijk	Houtribdijk	zwf	153.188	18.09	15.05
Afsluitdijk	nfl	nop	668.913	18.22	10.44
Afsluitdijk	nfl	wfr	638.078	18.33	12.86
Afsluitdijk	nfl	wie	463.822	18.31	10.09
Afsluitdijk	nfl	zwf	492.256	18.22	11.66
Afsluitdijk	nop	wfr	546.406	18.30	11.88
Afsluitdijk	nop	wie	372.850	18.55	15.70
Afsluitdijk	nop	zwf	400.588	22.67	10.81
Afsluitdijk	wfr	wie	344.241	23.02	7.16
Afsluitdijk	wfr	zwf	369.069	18.34	17.91
Afsluitdijk	wie	zwf	195.935	18.14	12.97

Table 5.5: Solutions and computation times for three dikes scenarios.

5.6 Conclusion

In this chapter, we presented a Cost-Benefit Analysis of determining safe dike heights in the Netherlands using an ILP approach. The main benefit of the ILP approach is that it does not require a specific cost function in order to solve the problem. Furthermore, different scenarios and cost structures can be modeled easily. The method performs quite well and we were able to find the optimal solution in all the instances in just over a minute. The strength of the formulation suggests that ILP is a powerful tool even for bigger instances. The valid inequalities

we discuss are a starting point for further research for problems with different cost structures.

Bibliography

K. Aardal, S.P.M. van Hoesel, A.M.C.A. Koster, C. Mannino, and A. Sassano. Models and solution techniques for frequency assignment problems. *Annals of Operations Research*, 153(1):79–129, 2007.

M. Ancău. The optimization of printed circuit board manufacturing by improving the drilling process productivity. *Computers & Industrial Engineering*, 55(2): 279 – 294, 2008.

K. Appel and W. Haken. The solution of the four-color-map problem. *Scientific American*, 237:108–121, October 1977.

D.L. Applegate, R.E. Bixby, V. Chvátal, W. Cook, D.G. Espinoza, M. Goycoolea, and K. Helsgaun. Certification of an optimal TSP tour through 85,900 cities. *Operations Research Letters*, 37(1):11 – 15, 2009.

E. Balas and C.S. Yu. Finding a maximum clique in an arbitrary graph. *SIAM Journal on Computing*, 15(4):1054–1068, 1986.

C. Beeri, R. Fagin, D. Maier, and M. Yannakakis. On the desirability of acyclic database schemes. *Journal of the ACM*, 30(3):479–513, July 1983.

D. Bergman and J.N. Hooker. Graph coloring facets from all-different systems. In N. Beldiceanu, N. Jussien, and E. Pinson, editors, *Integration of AI and OR Techniques in Contraint Programming for Combinatorial Optimzation Problems*, volume 7298 of *Lecture Notes in Computer Science*, pages 50–65. Springer Berlin Heidelberg, 2012.

U. Bertelè and F. Brioschi. On the theory of the elimination process. *Journal of Mathematical Analysis and Applications*, 35(1):48 – 57, 1971.

A. Björklund, T. Husfeldt, and M. Koivisto. Set partitioning via inclusion-exclusion. *SIAM Journal on Computing*, 39(2):546–563, 2009.

H.L. Bodlaender. A tourist guide through treewidth. *Acta Cybernetica*, 11:1–23, 1993.

V. Bouchitté and I. Todinca. Treewidth and minimum fill-in: Grouping the minimal separators. *SIAM Journal on Computing*, 31(1):212–232, 2001.

V. Bouchitté and I. Todinca. Listing all potential maximal cliques of a graph. *Theoretical Computer Science*, 276(1 - 2):17 – 32, 2002.

R. Brekelmans, D. den Hertog, K. Roos, and C. Eijgenraam. Safe dike heights at minimal costs: The nonhomogeneous case. *Operations Research*, 60(6):1342–1355, 2012.

C. Bron and J. Kerbosch. Algorithm 457: finding all cliques of an undirected graph. *Communications of the ACM*, 16(9):575–577, September 1973.

E.K. Burke, J. Marecek, A.J. Parkes, and H. Rudová. A supernodal formulation of vertex colouring withapplications in course timetabling. *Annals of Operations Research*, 179:105–130, 2010.

J. Byskov. Enumerating maximal independent sets with applications to graph colouring. *Operations Research Letters*, 32(6):547–556, November 2004.

M. Campêlo, R.C. Corrêa, and Y. Frota. Cliques, holes and the vertex coloring polytope. *Information Processing Letters*, 89(4):159 – 164, 2004.

M. Campêlo, V.A. Campos, and R.C. Corrêa. On the asymmetric representatives formulation for the vertex coloring problem. *Discrete Applied Mathematics*, 156 (7):1097 – 1111, 2008.

G.J. Chaitin, M.A. Auslander, A.K. Chandra, J. Cocke, M.E. Hopkins, and P.W. Markstein. Register allocation via coloring. *Computer Languages*, 6(1):47–57, 1981.

T. Christof and A. Löbel. Porta - polyhedron representation transformation algorithm. `http://typo.zib.de/opt-long_projects/Software/Porta/`, 1997–2000.

F.R.K. Chung and D. Mumford. Chordal completions of planar graphs. *Journal of Combinatorial Theory, Series B*, 62(1):96 – 106, 1994.

P. Coll, J. Marenco, I. Méndez Díaz, and P. Zabala. Facets of the graph coloring polytope. *Annals of Operations Research*, 116:79–90, 2002.

G.B. Dantzig. Maximization of a Linear Function of Variables Subject to Linear Inequalities, in Activity Analysis of Production and Allocation. In T.C. Koopmans, editor, *Activity Analysis of Production and Allocation*, chapter XXI, pages 339–347. John Wiley & Sons, New York, 1951.

G.B. Dantzig. Discrete-variable extremum problems. *Operations Research*, 5(2): pp. 266–277, 1957.

G.B. Dantzig, D.R. Fulkerson, and S.M. Johnson. Solution of a large-scale traveling-salesman problem. *Operations Research*, 2:393–410, 1954.

E.W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

C. Eijgenraam, R. Berkelmans, D. den Hertog, and K. Roos. Safe dike heights at minimal costs: The homogeneous case. Technical report, Tilburg University, 2010.

C. Feremans, M. Oswald, and G. Reinelt. A y-formulation for the treewidth. Technical report, Heidelberg University, June 2002.

M. Fischetti and A. Lodi. Local branching. *Mathematical Programming*, 98(1-3): 23–47, 2003.

F.V. Fomin, D. Kratsch, and I. Todinca. Exact (exponential) algorithms for treewidth and minimum fill-in. In J. Díaz, J. Karhumäki, A. Lepistö, and D. Sannella, editors, *Automata, Languages and Programming*, volume 3142 of *Lecture Notes in Computer Science*, pages 568–580. Springer Berlin Heidelberg, 2004.

D.R. Fulkerson and O.A. Gross. Incidence Matrices and Interval Graphs. *Pacific Journal of Mathematics*, 15:835–855, 1965.

P. Galinier, J.P. Hamiez, J.K. Hao, and D. Porumbel. Recent advances in graph vertex coloring. In I. Zelinka, V. Snášel, and A. Abraham, editors, *Handbook of*

Optimization, volume 38 of *Intelligent Systems Reference Library*, pages 505–528. Springer Berlin Heidelberg, 2013.

M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

F. Gavril. Algorithms for minimum coloring, maximum clique, minimum covering by cliques, and maximum independent set of a chordal graph. *SIAM Journal on Computing*, 1(2):180–187, 1972.

R.E. Gomory. Outline of an algorithm for integer solutions to linear program. *Bulletin of the American Mathematical Society*, 64(5):275–278, September 1958.

M. Grötschel, M. Jünger, and G. Reinelt. Facets of the linear ordering polytope. *Mathematical Programming*, 33(1):43–60, 1985.

P. Hansen, M. Labbé, and D. Schindl. Set covering and packing formulations of graph coloring: Algorithms and first polyhedral results. *Discrete Optimization*, 6(2):135 – 147, 2009.

P. Heggernes. Minimal triangulations of graphs: A survey. *Discrete Mathematics*, 306(3):297–317, 2006.

P. Hell, S. Klein, L.T. Nogueira, and F. Protti. Partitioning chordal graphs into independent sets and cliques. *Discrete Applied Mathematics*, 141(13):185 – 194, 2004. Brazilian Symposium on Graphs, Algorithms and Combinatorics.

IBM. IBM ILOG CPLEX Optimizer. `www.ibm.com/software/commerce/optimization/cplex-optimizer/`, Last retrieved June 2014.

L.V. Kantorovich. Mathematical methods of organizing and planning production. *Management Science*, 6(4):pp. 366–422, 1960.

N. Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing*, STOC '84, pages 302–311, New York, NY, USA, 1984. ACM.

R. Karp. Reducibility among combinatorial problems. In R. Miller and J. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

L.G. Khachiyan. Polynomial algorithms in linear programming. *USSR Computational Mathematics and Mathematical Physics*, 20(1):53 – 72, 1980.

M. Kubale. *Graph Colorings*. Contemporary mathematics v. 352. American Mathematical Society, 2004.

A.H. Land and A.G. Doig. An automatic method of solving discrete programming problems. *Econometrica*, 28(3):pp. 497–520, 1960.

S.L. Lauritzen and D.J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society. Series B (Methodological)*, 50(2):pp. 157–224, 1988.

E.L. Lawler. A note on the complexity of the chromatic number problem. *Information Processing Letters*, 5(3):66 – 67, 1976.

V. Lotfi and S. Sarin. A graph coloring algorithm for large scale scheduling problems. *Computers & Operations Research*, 13(1):27–32, January 1986.

H.M. Markowitz and A.S. Manne. On the solution of discrete programming problems. *Econometrica*, 25(1):pp. 84–110, 1957.

Munich RE. Floods dominate natural catastrophe statistics in first half of 2013. Press Release, July 2013. URL `http://www.munichre.com/en/media_relations/press_releases/2013/2013_07_09_press_release_en.pdf`.

A. Mehrotra and M.A. Trick. A column generation approach for graph coloring. *INFORMS Journal on Computing*, 8:344–354, 1995.

I. Méndez-Díaz and P. Zabala. A branch-and-cut algorithm for graph coloring. *Discrete Applied Mathematics*, 154(5):826 – 847, 2006.

I. Méndez-Díaz and P. Zabala. A cutting plane algorithm for graph coloring. *Discrete Applied Mathematics*, 156(2):159 – 179, 2008.

Rudolf Müller and Andreas S. Schulz. Transitive packing: A unifying concept in combinatorial optimization. *SIAM Journal on Optimization*, 13(2):335–367, June 2002. ISSN 1052-6234.

G.L. Nemhauser and G. Sigismondi. A Strong Cutting Plane/Branch-and-Bound Algorithm for Node Packing. *The Journal of the Operational Research Society*, 43(5):443–457, 1992.

G.L. Nemhauser and L.A. Wolsey. *Integer and Combinatorial Optimization*. Wiley-Interscience, New York, NY, USA, 1988.

G. Palubeckis. Facet-inducing web and antiweb inequalities for the graph coloring polytope. *Discrete Applied Mathematics*, 158(18):2075 – 2080, 2010.

C.M. Papadimitriou. *Computational complexity*. Addison-Wesley, Reading, Massachusetts, 1994.

D.J. Rose. Triangulated graphs and the elimination process. *Journal of Mathematical Analysis and Applications*, 32(3):597 – 609, 1970.

D.J. Rose. A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations. In *Graph theory and computing*, pages 183–217. Academic Press, New York, 1973.

R.E. Tarjan and M. Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM Journal on Computing*, 13(3):566–579, July 1984.

D. van Dantzig. Economic decision problems for flood prevention. *Econometrica*, 24(3):pp. 276–287, 1956.

Y. Villanger. Improved exponential-time algorithms for treewidth and minimum fill-in. In J. Correa, A. Hevia, and M. Kiwi, editors, *LATIN 2006: Theoretical Informatics*, volume 3887 of *Lecture Notes in Computer Science*, pages 800–811. Springer Berlin Heidelberg, 2006.

H.P. Williams and H. Yan. Representations of the all_different predicate of constraint satisfaction in integer programming. *INFORMS Journal on Computing*, 13(2):96–103, March 2001.

M. Yannakakis. Computing the minimum fill-in is np-complete. *SIAM Journal on Algebraic Discrete Methods*, 2(1):77–79, 1981.

J. Yuan and H. Zhang. Minimum fill-in problem of graphs. *Journal of Systems Science and Complexity*, 9(3):193, 1996.

P.J. Zwaneveld and G. Verweij. Safe dike heights at minimal costs: An integer programming approach, 2014.

Nederlandse samenvatting (Dutch summary)

Operationeel onderzoek maakt gebruik van wiskundige technieken om te helpen met de optimalisering van zowel operationele als strategische besluitvorming. Deze wiskundige tools omvatten wiskundige modellering, algoritmische gereedschappen, simulatie, statistiek, en besliskundige analyse. Een element dat wordt gedeeld door al deze tools is dat ze allemaal processen modelleren uit de bedrijfspraktijk. Hoewel deze modellen niet volledig de werkelijkheid kunnen beschrijven door de complexiteit van de praktijk, hebben ze hun nut bewezen in verschillende gebieden: operations research wordt vaak gebruikt in verschillende sectoren, zoals luchtvaart, spoorwegen, productie management, voorraadbeheer, transport en logistiek en telecommunicatie.

Problemen op voornoemde gebieden kunnen worden geformuleerd en opgelost met behulp van speciale representaties, zoals grafen en formuleringsmethoden zoals lineaire en geheeltallige programmering. De drie problemen besproken in dit proefschrift zijn combinatorische optimaliseringsproblemen. Bovendien hebben ze allemaal een onderliggende graafstructuur en worden opgelost met dezelfde oplossingsmethodiek, branch-and-cut genaamd.

Het eerste probleem dat we onderzoeken is minimale triangulatie van grafen, een probleem dat optreedt in Gaussische matrix eliminatie, probabilistische netwerken en computer science. Het tweede probleem is graafkleuring, dat vaak wordt gebruikt in de planning, verroostering, en telecommunicatienetwerken, in het bijzonder frequentie toewijzing in mobiele netwerken.

Het eerste probleem dat we bestuderen is minimale triangulatie van grafen. Een chordale graaf is een graaf waar elke cykel van meer dan 3 kanten een koorde heeft. Een triangulatie (chordalisatie) van een willekeurige graaf is de toevoeging van kanten aan de graaf, zodanig dat de nieuw verkregen graaf chordaal is. Het minimale triangulatie probleem is het probleem van het vinden van het kleinste aantal extra kanten om de graaf chordaal te maken. Het heeft toepassingen in Gaussische eliminatie van positief-definiete matrices, database management, kennissystemen, Bayesiaanse netwerken en kunstmatige intelligentie.

Het tweede probleem is graafkleuring. In het graafkleuringsprobleem dienen de punten van een graaf zodanig van een kleur voorzien te worden dat verbonden punten niet dezelfde kleur hebben. Het doel is om het aantal gebruikte kleuren zo klein mogelijk te krijgen. Het graafkleuringsprobleem is ontstaan uit de noodzaak om aardrijkskundige kaarten met een minimum aantal kleuren in te kleuren waarbij gebieden die een grens gemeen hebben verschillende kleuren dienden te krijgen. Later vond het probleem ook toepassingen in de planning, verroostering, toewijzing van frequenties aan GSM-masten, en volgorde bepaling van processen.

Het derde probleem onderzoekt het bepalen veilige dijkhoogten in Nederland waarbij een balans gevonden dient te worden van de kosten van dijkophoging enerzijds en van het veiligheidsrisico van overstromingen anderzijds. Het bepalen van veilige dijkhoogten in Nederland werd een probleem na de overstroming van 1953. Met de verandering van het klimaat (opwarming van de aarde en het hiermee samenhangende verhogen van de zeespiegel) wordt bescherming tegen overstromingen belangrijker dan ooit. Voor de kosten-baten analyse (KBA), maken we gebruik van modellen die verwachte kosten van verhoging van de dijken kunnen afwegen tegen de maatschappelijke koten van veiligheid (voorkomen van materile en immaterile schade) . Onze analyse stelt ons in staat om te gaan met verschillende scenarios voor zeespiegelstijging.

Valorization

The problems studied in this thesis have applications in various fields.

The minimum triangulation problem is used in Gaussian elimination of sparse positive definite matrices, database management, knowledge based systems, and Bayesian networks, artificial intelligence. The problem is also used in tensor-based recommender systems in order to determine the order of multiplication of sparse tensors.

Even though the graph coloring problem originates from the need to color the maps, it is also used in scheduling, timetabling, bandwidth allocation, and sequencing. In fact the literature on the applications of the problem precedes some of the literature on the graph coloring problem itself.

The optimal dike heights problem is used in order to determine safe dike heights in the Netherlands by balancing the cost of safety against the benefit of security from the floods. Whereas the previous literature focuses on mostly single dike scenarios, we were able to model different types of scenarios with respect to the relation of protection among dikes.

In addition to the good results we obtained using branch-and-cut algorithm, our branch-and-cut approaches for the problems are versatile in the sense that they can be used in combination with other approaches. Furthermore, we believe some of the results in this thesis can initiate further research on their respective problems.

Even though I humbly believe that the research carried in this thesis can be applied and has value in this sense, I strongly believe that trying to justify the value of research has the potential to decrease its value. In an age, where the value of research is strongly linked to publishing (a system that is flawed itself)

pushing for 'valuable' research risks concentrating research on popular topics. As researchers, we should remember that the questions are always there and most of the time we simply do not see them. I believe that the fact that we cannot see somethings does not take away their values. The graph coloring problem originates from the need to color the maps using the minimum number of colors. As it turned out the problem of coloring maps is not interesting (you can always do it with 4 colors) but the graph coloring problem is still researched to this day because of its other uses, which were not known at that time. The problem is used in compiler optimization, even though at that time the computers did not exist. We should remember that the problem would have still existed even if someone had said 'Just color the map!' because he or she could not see the value of the question at that time.

Short Curriculum Vitae

Birol Yüceoğlu was born in January 4, 1983 in İstanbul, Turkey. He attended Galatasaray Highschool, in İstanbul, between 1994 and 2002. After highschool, he studied Manufacturing Systems Engineering at Sabancı University, in İstanbul. After obtaning his B.S. degree in 2007, he started his M.S. at the same university in Industrial Engineering program. He graduated in 2009 and his M.S. thesis is entitled "Tactical crew planning in railways". In September 2009, he joined the Department of Quantitative Economics at Maastricht University as a Ph.D. candidate under the supervision of Stan van Hoesel. The results of his research is presented in this thesis.

His research interests include graph theory, mathematical modelling, integer programming, and algorithms.