Maximum Stable Set Problem: A Branch & Cut Solver

Diplomarbeit von

Steffen Rebennack

Betreut von

Prof. Dr. Gerhard Reinelt

Ruprecht-Karls-Universität Heidelberg

Fakultät für Mathematik und Informatik

Mai 2006

Für meine Eltern.

Acknowledgment

My special thanks is directed to my advisor Prof. Dr. Gerhard Reinelt, who gave me the possibility to write my diploma thesis with such an interesting and ambitious topic in the field of combinatorial optimization. Furthermore, I would like to thank him for his support of my study and research activities. Hanna Seitz was a very dedicated tutor and always had a sympathetic ear for my questions. In addition, I thank Dr. Marcus Oswald for his clear explanations and Dr. Dirk Oliver Theis for his countless suggestions.

In addition to the people associated with the group of combinatorial optimization at the University of Heidelberg, I would like to thank...

... Lothar Kountz, who sparked my interest in mathematics.

... Prof. Dr. Josef Kallrath, for his altruistic and engaged patronization.

... BASF cooperation which gave me the chance to work on a lot of interesting projects regarding optimization.

... Susumu Ikenoue, who provided me with an unprecedented and memorable time in Japan.

... Prof. Dr. Rossi and Prof. Dr. Smriglio, for their invitation to the University of L'Aquila and their interest in working together with me.

... Prof. Dr. Bartels and Prof. Dr. Freitag, who gave me the chance to work as a student assistant.

... my fellow student Thorsten Bonato for his inspiring discussions and his assistance with LaTeX.

... my father, who not only financed my complete study but also supported me from the very beginning.

... my girlfriend for her balance during the last stressful weeks.

... all who cross-checked my diploma thesis, especially Cara Cocking, Janine Erhard, Josef Kallrath, Marcus Oswald, Birgit Rebennack, Hanna Seitz, Dirk Oliver Theis, and Ingrid Tschernoch.

Introduction

Imagine you wanted to go to a fair with many different events. Of course, some of them take place at the same time and some are more interesting for you than others. To select the best events according to the preference list is a maximum stable set problem. In general, a set of objects (the events) is given with weights (the preference list) and conflicts between them (time overlap). The task of finding a maximum number of objects with respect to the weighting of the objects such that no conflicts occur is the stable set problem. According to its definition, this problem has a huge variety of applications in various fields—we will come back to that in Chapter 2. Therefore it is not surprising that the stable set problem is, next to the traveling salesman problem, one of the most important combinatorial optimization problems.

At first glance there seems to be no better way to solve the maximum stable set problem exactly than to use a brute force method. This means to enumerate all possible stable sets and to select the best one. But as the number of solutions grows exponentially in the size of the number of objects and conflicts, this method is inefficient for large problem instances. Unfortunately, unless $\mathcal{P} = \mathcal{NP}$, there is no polynomial time algorithm which solves the maximum stable set problem. Therefore, also at second glance there is no essentially computationally better solution algorithm than the brute force method in the general case. Nevertheless, there are a lot of ideas which help to speed up the solver for many instances. The stable set problem has been widely studied with tens of thousands of publications, thousands of implementations of exact solvers, and heuristic approaches. The development of Branch & Cut solvers was driven by MANNINO and SASSANO though their introduction of the edge projection. This work was adapted to separation routines by ROSSI and SMRIGLIO in 2001. Their resulting Branch & Cut algorithm still provides one of the best dual bounds and their handling of the Branch & Bound tree has no competition. Therefore it is used as a benchmark. However, presently there was no Branch & Cut algorithm using all the separation routines provided by the theory, and this was the motivation for this diploma thesis.

This work has the following structure. We start with the introduction of the basic definitions and results needed for the following chapters. Relevant aspects of graph theory are thoroughly reviewed as there is no uniform nomenclature in the literature. The next sections give an overview of mathematical programming and the Branch & Cut solver which is based on polyhedral theory. Chapter 2 introduces the stable set problem formally and shows various connections to other problems in the field of graph theory. That chapter is closed by a list of applications. We formulate the stable set problem as a mathematical optimization model in Chapter 3. In order to solve this model via a Branch & Cut algorithm we study the structure of the underlying polytope. The theory will lead us to several classes of inequalities, for instance the odd-cycle and clique inequalities. As a by-product we will see two cases in which the stable

set problem turns out to be polynomially solvable. The theoretical aspects of the Branch & Cut modules are presented in Chapter 4. We start with the discussion of preprocessing for the stable set problem using some basic observations and the theory provided by NEMHAUSER and TROTTER. We show that the generalization to some other inequalities is not possible, which is a new result. Section 4.2 is a central part in this diploma thesis and presents several separation routines. We begin with the separation of the odd-cycle inequalities and continue with the clique separation where the idea of lifting 3-cycles to maximal cliques is introduced. Next, the edge projection based on the results of MANNINO, SASSANO, ROSSI and SMRIGLIO is presented. This theory is extended by polyhedral investigations. We consider some examples and provide a sufficient condition when an inequality after the projection is facet-defining. At the end of that section an extension of the edge projection to general inequalities is provided. The section about the rank inequalities is followed by two types of general inequalities which can be adopted to any integer program formulation. We start with the local cuts which have been introduced for the traveling salesman problem. In the next section, the mod-k cuts are adapted to the stable set problem. The 4th chapter is closed with the review of the branching strategy for the stable set problem introduced by BALAS and YU. Chapter 5 focuses on some implementation details of a Branch & Cut algorithm. Some aspects of the theory of Chapter 4 are discussed again and modifications are presented. In particular, we discuss a new modification of the odd-cycle inequalities which leads to a separation routine for the odd-hole inequalities. The lower bounds in the Branch & Cut algorithm will be provided by two heuristics, which will be discussed in Section 5.3. In particular, a new improvement heuristic is introduced. The implemented Branch & Cut algorithm is tested on some DIMACS benchmark graphs and uniform random graphs. The computational results are presented and discussed in Chapter 6. We will see that the dual bounds for the implemented algorithms are quite good. This diploma thesis is closed with a summary and an outlook for further research activities in Chapter 7.

At the very end of this diploma thesis one can find lists of figures, algorithms, and symbols. The bibliography is followed by an index which may help looking something up. All implemented algorithms, together with their documentation, the test instances, the computational results, as well as the LaTeX code of this document are contained on the CD which can be found in the inside of the back cover.

Contents

Chapter 1

Mathematical Preliminaries

This chapter introduces the terminology and basic results used in this diploma thesis. We start with a few notations. Section 1.1 gives an introduction into some definitions in the field of graph theory. After that we get a rough idea about complexity of algorithms. Section 1.3 deals with the concepts of polyhedral theory. Afterwards, Section 1.4 introduces the ideas of linear programming with some important results about duality. The change from continuous to integer domain leads us to integer programming. In order to solve such problems we introduce the Branch & Cut algorithm in Section 1.6.

For matrices we use capital letters. Vectors are presented by lower case letters. A vector v is always considered as a column vector. The i-th unit vector is abbreviated with e_i and $\mathbb{1}$ is the vector consisting of ones. By $\mathbb{N}$, $\mathbb{Z}$ and $\mathbb{R}$ we denote the positive natural, integer and real numbers, respectively. If each of the sets have a superscript "+" we mean the non negative numbers, including zero. With "**iff**" we shorten "if and only if". iff

1.1 Graphs

This chapter deals with several definitions and notations in the field of graph theory. Unfortunately, there is no uniform terminology in graph theory. The following is based on [Rei01, Gal05, Die00, GLS88, Wes00].

An **undirected graph** G is a pair (V, E) consisting of a nonempty, finite set V and a finite (possibly empty) set E of unordered pairs of distinct elements of V. As we consider only undirected graphs we call them just **graphs**. The elements of V are called **nodes** of the graph G, the elements of E are **edges**. An edge $e = \{u, v\}$ is usually written as uv, which is equal to vu. In our context the edge set E contains no multiple edges. The **order** of a graph G is the **cardinality** of its node set V, which is the number of elements of V. It is denoted by $|\mathrm{G}|$. The **complement** $\overline{\mathrm{G}}$ of a graph G is the graph with the same node set as G and the **complement edge set** $\overline{E}$, containing only the edges which are not in E. Two graphs $\mathrm{G} = (V, E)$ and $H = (W, F)$ are called **isomorphic**, denoted by $\mathrm{G} \cong H$, if there is a bijection $\phi\colon V \to W$ such that $uv \in E \Leftrightarrow \phi(u)\phi(v) \in F$. If a graph G has a weighting function $c\colon V \to \mathbb{R}$ of its nodes, we call G together with c a **node-weighted graph**. In case of a weighting function $c\colon E \to \mathbb{R}$ of the edges of G, we say that G together with c is an **edge-weighted graph**. If it is clear from graph, node, edge, cardinality, complement, isomorphic

weighted graph

context which weighting is meant, we just call it a **weighted graph** and abbreviate it as a triple $G_c = (V, E, c)$.

incident
endnodes
adjacent
neighbors
neighborhood
degree
isolated

A node v is **incident** to an edge e if $e = uv$. The two nodes incident to an edge are its **endnodes**. Two nodes u, v of a graph G are **adjacent** or **neighbors** if uv is an edge of G. For a node set $W \subseteq V$ we define the set of neighbors $\Gamma(W)$ as the set of all nodes in $V \setminus W$ which are adjacent to at least one node in W and call it the **neighborhood**. For a single node we write $\Gamma(v)$ instead of $\Gamma(\{v\})$. The **degree** of a node v, denoted by $\delta(v)$, in a graph G is the number of edges incident with v, or equivalently the number of adjacent nodes to v, which is the cardinality of $\Gamma(v)$. A node is called **isolated** if its degree is zero.

subgraph
induced subgraph

Let $G = (V, E)$ and $H = (W, F)$ be graphs. We call H a **subgraph** of G and write $H \subseteq G$, when $W \subseteq V$ and $F \subseteq E$ is the set of edges of graph G with both endnodes in W. For a node set $W \subseteq V$, the graph $G - W$ denotes the graph obtained by removing all nodes of W and all edges adjacent with at least one node of W. In the special case when W contains only one node v, we simply write $G - v$. $G[W] := (W, E(W))$ denotes the **induced subgraph** of G with $E(W)$ containing all edges of graph G with both endnodes in W.

complete
clique
triangle
bipartite
bipartition

A graph is said to be **complete** if it contains an edge to each pair of its nodes. A complete graph with n nodes is denoted by K_n. A **clique** is the node set of a complete subgraph. If it has n nodes, it is called n-clique, whereas a 3-clique is called **triangle**. We call a graph G **bipartite** if its node set V can be partitioned into two disjoint sets V_1, V_2 with $V = V_1 \cup V_2$ such that neither two nodes of set V_1 nor two nodes of set V_2 are neighbors. The node sets V_1, V_2 are called **bipartition** of V.

walk
length
trail
path
closed
circuit
cycle
n-cycle
chord

Let u and v be two nodes of a graph G. A u-v **walk** W of G from u to v is a finite alternating sequence $W := (u = u_0, e_1, u_1, e_2, u_2, \ldots, u_{k-1}, e_k, u_k = v)$ of nodes and edges, beginning with node u and ending with node v, such that $e_i = u_{i-1}u_i$ for $i = 1, 2, \ldots, k$. The number k is called the **length** of walk W. A walk in which no edge is repeated is called a **trail**, while a **path** is a walk without node repetition. A walk, trail or path is **closed** if u is equal to v. Let W be a walk, trail or path, then we denote by $V(W)$ the set of nodes which are contained in W and by $E(W)$ the set of edges for which both incident nodes are a member of set W. A **circuit** is a non-trivial closed trail. If all nodes of a circuit are distinct, then it is called a **cycle**. This means that a cycle has no repetition of edges and nodes. A cycle is even / odd if its length is even / odd. A cycle of length n is an n-**cycle**. An edge which joins two nodes of a cycle but is not itself an edge of the cycle is a **chord** of that cycle.

connected
connected component

A node u is said to be **connected** to a node v in a graph G if there is a path from u to v in G. If each pair of nodes of G is connected, then G is said to be **connected**, otherwise G is **disconnected**. The **connected components** of a graph G are the connected non-empty inclusion-maximal subgraphs of G.

node-edge incidence matrix
adjacency matrix
adjacency list

We look at three different possibilities to store a graph. The **node-edge incidence matrix** is a $|V| \times |E|$ matrix. Each row represents a node, while each column is associated with an edge. This matrix has value one in entry i, j iff node i is incident to edge j, otherwise zero. The **adjacency matrix** is of dimension $|V| \times |V|$ and has entry one iff the two nodes are adjacent and otherwise zero. A third method of storing a graph is via an **adjacency list**. In this case only the adjacent nodes to each node are stored.

At the end of this section we quote two fundamental results which will be needed in Chapter 3. We begin with a characterization of bipartite graphs.

Theorem 1.1.1 (KÖNIG 1916)**.** *[KV91] A graph* G *is bipartite iff it contains no circuit of odd length.*

Bipartite graphs have a very special structure. One is stated by the next proposition. But first we need the following definition. A matrix A is called **totally unimodular** if the determinant of each square submatrix of A has the value of 0, 1 or -1. Now we are ready to understand

totally unimodular

Proposition 1.1.2. *[CCPS98] The node-edge incidence matrix of a graph* G *is totally unimodular, iff* G *is bipartite.*

1.2 Complexity

We briefly review the most basic notations of complex theory, inspired by the summaries of [The05, Chr97]. For a thorough description we refer to [GJ79, CLRS01].

By $O(g(n))$ we denote an asymptotic upper bound for the running time $f(n)$ of an algorithm if there is a number $n_0 \in \mathbb{N}$ such that for all $n \geq n_0$ exists a constant $c > 0$ with the property $0 \leq f(n) \leq c \cdot g(n)$.

We distinguish between a decision problem, which requires a "yes" or "no" answer, and an optimization problem, in which one has to optimize a certain objective function. A problem is called **polynomially** solvable if there is a deterministic algorithm which solves the problem in polynomial time. This is an algorithm whose running time in terms of number of elementary steps is bounded by a fixed polynomial in the input size. The class containing the polynomially solvable decision problems is denoted by $\boldsymbol{\mathcal{P}}$. If each positive / negative answer of a decision problem can be checked in polynomial time, the decision problem is in the class $\boldsymbol{\mathcal{NP}}$ / $\boldsymbol{co\text{-}\mathcal{NP}}$. An optimization problem is called $\boldsymbol{\mathcal{NP}\text{-}hard}$ if every decision problem in $\mathcal{NP}$ can be polynomially reduced to it. This means that from the optimum of an optimization problem an answer for the decision problem can be computed in polynomial time.

polynomial

$\mathcal{P}$
$\mathcal{NP}$
$co\text{-}\mathcal{NP}$
$\mathcal{NP}\text{-}hard$

1.3 Polyhedra

Before we start with linear optimization, we want to get familiar with some concepts of polyhedral theory. The following definitions can be found in [GLS88, Brø83, Zie95].

A **linear combination** of vectors $x_1, \ldots, x_n \in \mathbb{R}^d$ is a vector of the form $\lambda_1 x_1 + \ldots + \lambda_n x_n$ with real scalars $\lambda_1, \ldots, \lambda_n$. If a linear combination $\lambda_1 x_1 + \ldots + \lambda_n x_n$ satisfies $\sum_{i=1}^{n} \lambda_i = 1$, then it is called **affine combination**. A **convex combination** of points from $\mathbb{R}^d$ is an affine combination with non-negative coefficients $\lambda_1, \ldots, \lambda_n$. These combinations are called **proper** if neither $\lambda = 0$ nor $\lambda = e_j$ for all $j \in \{1, 2, \ldots, k\}$ with $\lambda = (\lambda_1, \ldots, \lambda_n)^\top$.

linear combination
affine combination
convex combination
proper

A subset $I \subseteq \mathbb{R}^d$ is called **linearly** or **affinely independent** if none of its members is a proper linear or affine combination of the remaining elements of I. For any set $I \subseteq \mathbb{R}^d$ the **affine rank** of I, denoted by $\operatorname{arank}(I)$, is the cardinality of the largest affinely independent subset of I. For any subset $I \subseteq \mathbb{R}^d$ the **dimension** of I, denoted by $\dim(I)$, is the cardinality of a largest affinely independent subset of I minus one, $\dim(I) = \operatorname{arank}(I) - 1$. A set $I \subseteq \mathbb{R}^d$ with $\dim(I) = d$ is called **full-dimensional**.

linearly independent
affinely independent
affine rank
dimension
full-dimensional

affine hull

convex hull
polyhedron
polytope

The **affine hull** of a set of points $x_1, \dots, x_n \in \mathbb{R}^d$ is the set of its affine combinations. It is denoted by $\operatorname{aff}(x_1, \dots, x_n)$. Accordingly, the set of convex combinations of some points is called **convex hull** and is written as $\operatorname{conv}(x_1, \dots, x_n)$. A **polyhedron** is the intersection of a finite number of half-spaces. If a polyhedron is bounded, we call it **polytope**. An important equivalence is that each polytope is the convex hull of a finite set of points from $\mathbb{R}^d$. For instance, a bounded solution set $\{x \in \mathbb{R}^d \mid Ax \leq b\}$ of a system of linear inequalities is a polytope. The dimension of a polytope is the dimension of its affine hull.

valid
face

vertex
facet
facet-defining inequality
slack

A linear inequality $\beta^\top x \leq b_0$ is denoted by (β, b_0). In case β is $(0,1)$-valued, we write $x(W)$ as $\sum_{v_i \in W} x_i$ for a set $W \subseteq V$. A linear inequality $\beta^\top x \leq b_0$ is called **valid** with respect to a polyhedron P if P is a subset of $\{x \mid \beta^\top x \leq b_0\}$. A set $F \subseteq P$ is called a **face** of P if there is a valid inequality (β, b_0) for P such that $F = \{x \in P \mid \beta^\top x = b_0\}$. If v is a point in a polyhedron P such that $\{v\}$ is a face of P, then v is called a **vertex** of P. A **facet** is a nonempty face of P with dimension $\dim(P) - 1$ and its inequality is called **facet-defining inequality** for P. Let x^* be a vector satisfying (β, b_0). In this case the **slack** of (β, b_0) is defined as the difference between b_0 and $\beta^\top x^*$.

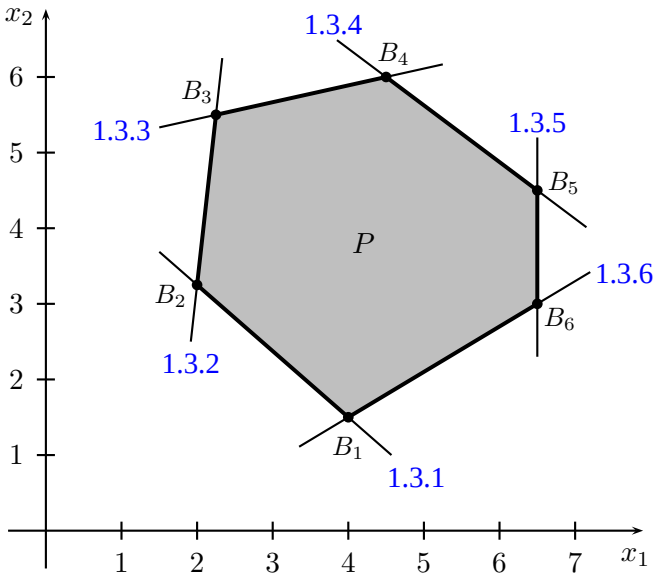


Figure 1.3.1: Polytope

After all these definitions, let us look at an example. First consider the following system of linear inequalities

$$\begin{aligned} -7x_1 - 8x_2 &\leq -25, && (1.3.1)\\ -36x_1 + 4x_2 &\leq -27, && (1.3.2)\\ -2x_1 + 9x_2 &\leq 45, && (1.3.3)\\ 6x_1 + 8x_2 &\leq 61, && (1.3.4)\\ 2x_1 &\leq 11, && (1.3.5)\\ 6x_1 - 10x_2 &\leq 13. && (1.3.6) \end{aligned}$$

The polytope P described by this inequalities is $P = \{x \in \mathbb{R}^2 \mid x \text{ satisfies } (1.3.1) \text{ to } (1.3.6)\}$.

In Fig. 1.3.1 one can see polytope P and the six inequalities bounding it. The intersections of the lines are the vertices of P which are labeled with B_1 to B_6.

We want to close this section with a fundamental result about the vertices of a special polyhedron.

Theorem 1.3.1. *[Sch03] Let A be a totally unimodular $m \times n$ matrix and let $b \in \mathbb{Z}^m$. Then the vertices of the polyhedron $P := \{x \mid Ax \leq b\}$ are integer.*

Theorem 1.3.1 includes as an important assumption that matrix A is totally unimodular. This is a quite strict requirement. The entries of A can only have the values of 0, 1 and -1 which is only a necessary condition. The result of Theorem 1.3.1 has an interesting consequence as we will see in Section 1.6.

1.4 Linear Programming and Duality

In the last section we got familiar with polytopes and linear inequalities. Now we want to optimize over polytopes resulting from linear inequalities. In the following, we will give some definitions and concepts on linear optimization. The theory has been widely studied and we can only give a very short introduction into the key ideas. The following is based on [KW97, GLS88].

Linear Programming

Given an $m \times n$ matrix A, a vector $b \in \mathbb{R}^m$, and a vector $c \in \mathbb{R}^n$. The task of a **linear program (LP)** is to find a vector of a set $P = \{x \in \mathbb{R}^n | Ax \leq b\}$ which maximizes the linear function $c^T x$ over P. We will write it in the following short form

linear program

$$\begin{aligned} \max \quad & c^\top x \\ \text{s.t.} \quad & Ax \leq b \\ & x \in \mathbb{R}^n. \end{aligned} \tag{1.4.1}$$

A **feasible solution** of the linear function above is a vector which satisfies the inequality system $Ax \leq b$. A feasible solution $\bar{x}$ which satisfies $c^\top \bar{x} \geq c^T x$ for all $x \in P$ is called **optimal solution**. This is not uniquely determined in general and the set of all optimal solutions define a face of P. The linear function $c^\top x$ is called **objective function**. In formulation (1.4.1), the number of variables is n while the size of vector b gives the number of inequalities. We call system (1.4.1) the **standard form**. In literature, there are also other definitions for the standard form. Indeed, every linear program can be transformed into the above form. Minimization, equalities and bounds on the variables can be expressed with the above system and we call it therefore still an LP. For details of these transformations see for instance [Sch00].

feasible solution

optimal solution

objective function

standard form

Consider again Fig. 1.3.1 of the last section. We have already studied the polytope with its defining inequalities (1.3.1) to (1.3.6). To obtain a linear program we add the objective function $f(x_1, x_2) := -x_1 + 2x_2$ to the constraints. This results in the following linear program

$$
\begin{array}{lrcl}
\max & -x_1 + 2x_2 & & \\
\text{s.t.} & -7x_1 - 8x_2 & \leq & -25 \\
& -36x_1 + 4x_2 & \leq & -27 \\
& -2x_1 + 9x_2 & \leq & 45 \\
& 6x_1 + 8x_2 & \leq & 61 \\
& 2x_1 & \leq & 11 \\
& 6x_1 - 10x_2 & \leq & 13 \\
& x \in \mathbb{R}^2. & &
\end{array}
\tag{1.4.2}
$$

The level curve of $f(x_1, x_2)$ for the value -3.9 can be seen in Fig. 1.4.1 as the dashed black line. The black arrow targets in the direction of the gradient of $f(x_1, x_2)$ and therefore in the direction where the objective function value increases the most. Thus, to obtain the optimal solution, we move the level curves of $f(x_1, x_2)$ parallel in this direction. If any further movement of $f(x_1, x_2)$ only leads into infeasible solutions, an optimal solution has been found. In our case the optimal solution is vertex $B_3 = (2.25, 5.5)$ of the polytope P. The gray dashed line is the level curve of the optimal objective function which has value 8.75.

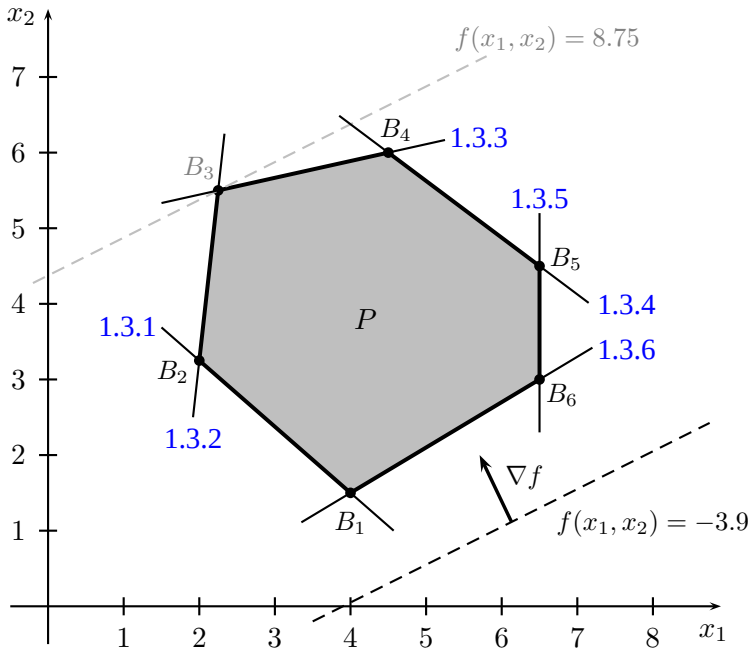


Figure 1.4.1: Linear program

Duality

To every linear program of the standard form we define its **dual (D)**, which reads: dual

$$\begin{aligned} \min \quad & y^\top b \\ \text{s.t.} \quad & A^\top y = c \\ & y \geq 0. \end{aligned} \tag{1.4.3}$$

The coefficients of the objective function of LP (1.4.1) have been exchanged by the right-hand side of its dual. The constraint matrix is transposed and inequalities have been changed to equalities. The variable vector y has to be non-negative and the number of inequalities of the original LP is its dimension. The original linear program is sometimes also called **primal** program. We consider now two famous theorems which connect the primal with its dual. Both proofs can be found in [Sch00]. primal

Theorem 1.4.1 (**Duality Theorem**). *[GLS88] Let (P) be a linear program of form (1.4.1) and (D) be its dual (1.4.3). If (P) and (D) both have feasible solutions, then both problems have optimal solutions and the optimum values of the objective functions are equal.*
If one of the programs (P) or (D) has no feasible solution, the other is either unbounded or has no feasible solution. If one of the programs (P) or (D) is unbounded then the other has no feasible solution.

Theorem 1.4.2 (**Complementary Slackness Theorem**). *[GLS88] Suppose u is a feasible solution for the primal linear programming problem (P) and v is a feasible solution for its dual (D). A necessary and sufficient condition for u and v to be optimal for (P) and (D), respectively, is that for all i*

$$\begin{aligned} v_i > 0 \quad &\text{implies} \quad A_i u = b_i, \\ \text{or equivalently } A_i u < b \quad &\text{implies} \quad v_i = 0. \end{aligned}$$

Solution methods for linear programs

We solved the linear program (1.4.2) in the way that we moved the level curve of the target function until we reached an extreme point. This was a two-dimensional example and its geometrical illustration. However, this leads to just a weak algorithm for higher dimensions. If we instead consider again Fig. 1.4.1, we recognize that the optimal solution lies in a vertex of the polytope P. In fact, this is generally true which means that we can always find an optimal solution of an LP in a vertex of the polytope, cf. [Sch00].

One of the best known algorithms for solving LPs is the **simplex algorithm** developed by GEORGE B. DANZIG in 1947. The simplex algorithm consists of two main steps. In the first step it computes an initial feasible vertex solution as a starting point. The second step computes from the starting point an optimal solution by moving from one vertex of the polyhedron to another vertex which has a better objective function value. We want to neglect technical details and especially the degenerated case. Since the number of vertices of a polyhedron is finite, the simplex algorithm terminates after a finite number of iterations. Unfortunately, the number of vertices of a polyhedron can be very large and in worst case the running time may simplex algorithm

ellipsoid method
interior-point methods

grow exponentially. Nevertheless on many real-world problems the simplex algorithm performs better[1] than polynomial time algorithms such as the **ellipsoid method** applied for solving LPs by KHACHIYAN in 1979, see for example [Sch00, KV91], or as **interior-point methods** forced by KARMARKAR in 1984, see [Kar84]. Since there are polynomial time algorithms for linear programs, LP-feasibility is in the class $\mathcal{P}$.

1.5 Integer Programming

In the following we consider a slight change of the domain of a linear program which has enormous consequences for the running time of a solution algorithm. This section is based on [KW97].

integer program

Let x be the variable of an LP of type (1.4.1). If we change the domain of variable x to be integer, we get an **integer program (IP)** of the form

$$\begin{array}{ll} \max & c^\top x \\ \text{s.t.} & Ax \leq b \\ & x \in \mathbb{Z}^n. \end{array} \tag{1.5.1}$$

The variables are no longer continuous. Instead their domain is integer. We also consider the special case of x being a proper subset of $\mathbb{Z}^n$. In the following chapters, we confine ourselves to the case that the entries of x have only the values 0 and 1. This change of the domain has a great impact as we will see in this and the next section.

We again consider the linear program (1.4.2). Now we require that the solution x is integer. The blue dots in Fig. 1.5.1 show the feasible solutions after adding the integrality constraint. It is obvious that the number of feasible solutions is no longer infinite. We also recognize the change of the optimal solution and the decrease of its objective function value.

explicit enumeration

Clearly, all feasible solutions of a linear program are within a polyhedron P. This implies that the number of solutions is infinite as the variables are continuous. It is quite remarkable that the change to integer solutions makes the problem much harder to solve even though the number of solutions is finite if the polyhedron is bounded. In fact, to determine a solution for general IPs is $\mathcal{NP}$-$hard$. An intuitive idea for solving IPs leads to **explicit enumeration**. This method computes all possible solutions. When a solution is feasible, the objective function value is stored and once we have all of them, we can pick out the best solution. However, this approach cannot be used in practice since the number of combinations grows exponentially. In the case of n binary variables, we have 2^n possible combinations. As $2^n = 10^{M \cdot n}$ with $M = \ln(2)/\ln(10) \approx 0.301$, 2^{50} is already a number with 16 digits. The best idea to solve such a problem is to use a Branch & Cut algorithm. In the following chapter we will have a closer look at this solution technique.

[1] see [KW97]

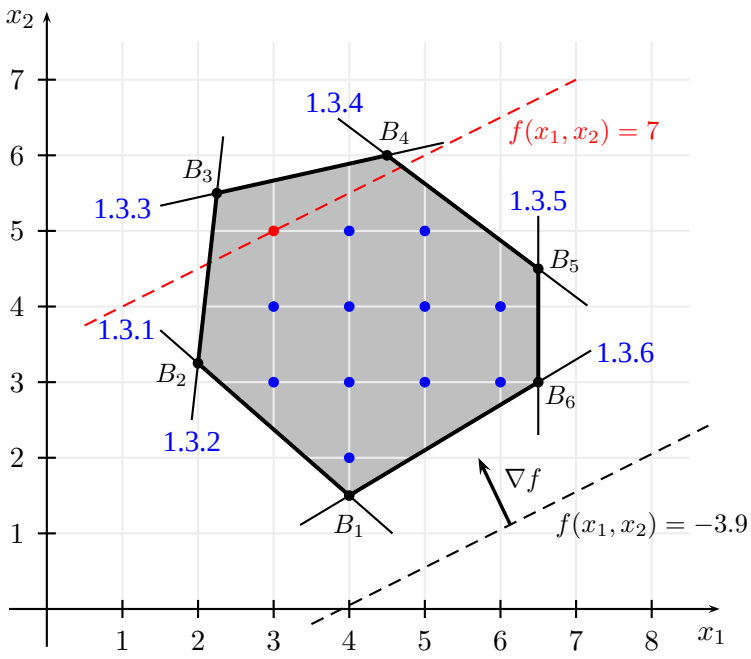


Figure 1.5.1: Integer program

1.6 Branch & Cut

In the last two sections we saw that there are polynomial time algorithms for linear programs. As mentioned above, the change from linear to integer constraints makes the problem much harder. But what is the reason? The reason why we can solve LPs in polynomial time even though they have an infinite number of solutions, is the concept that we only have to consider the vertices of its polytope. If we want to adopt this idea to integer programs, we have to find the convex hull of all feasible solutions of the IP, which is $P_I := \text{conv}\{x \in \mathbb{Z}^n \mid Ax \leq b\}$. If we can describe this polytope with a polynomially sized list of inequalities, we can compute the optimum in polynomial time. Recalling Theorem 1.3.1, we see that for totally unimodular constraint matrices A the polytope P is equal to the convex hull of all integer solutions. Therefore, the optimum lies in an integer vertex of the polytope and can be found in polynomial time. Unfortunately, in general the number of facets of a polytope P_I increases exponentially. If we consider Fig. 1.6.1, we recognize the polytope P_{IP} and the optimal solution x^*_{IP}. But in order to compute the optimal solution, it is not necessary to know all facets. Instead, the two dashed dark-green ones are enough. To compute only such necessary facets is one principle idea of the Branch & Cut algorithm. The use of that method was first published by GRÖTSCHEL, JÜNGER and REINELT [GJR84] in 1984. Its name was introduced by PADBERG and RINALDI [PR87]. This section is inspired by [Chr97, The05] and mainly based on [EGJR01].

In order to solve the system (1.5.1), we omit the condition that x has to be integer and obtain a so-called **LP-relaxation** or more precisely an LP-domain-relaxation. Its solution x^*_{RLP} may be fractional, but it provides an upper bound (UP) of IP (1.5.1). By adding special inequalities to the system $Ax \leq b$, the LP-relaxation polytope gets more and more close to P_{IP}. "Special" means that, if x^* is a vector satisfying $Ax^* \leq b$, we want a valid inequality (β, b_0), a so-called

LP-relaxation

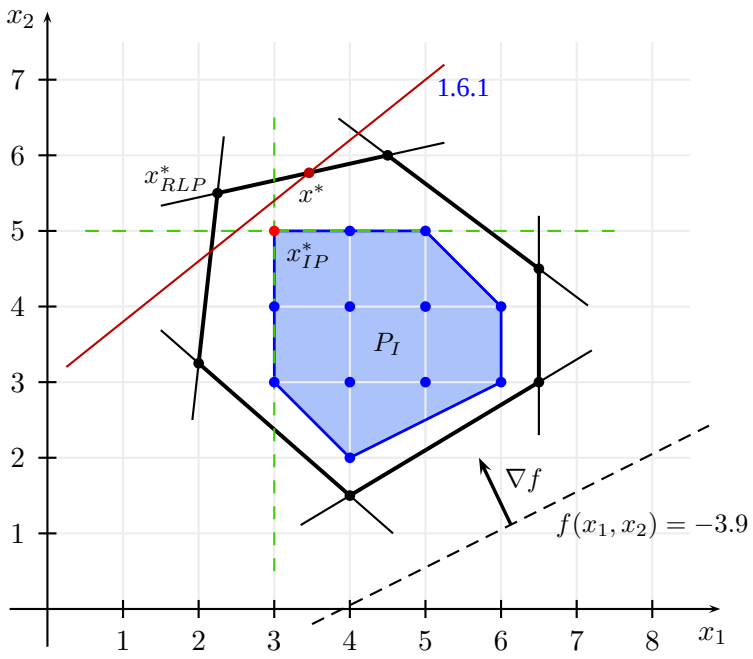


Figure 1.6.1: Branch & Cut

cutting-plane
separation problem

cutting-plane, separating x^* from P_{IP}. The task to decide if such an inequality exists and to compute one if possible, is called the **separation problem**. For instance, in Fig. 1.6.1 inequality

$$-4x_1 + 5x_2 \leq 15 \tag{1.6.1}$$

is a cutting plane for x^*_{RLP}. An important theorem in this context is that, under some technical conditions, the optimization problem can be solved in polynomial time, iff the separation problem is polynomial [GLS88]. In order to solve an IP, one could generate such cutting-planes until the optimal solution is found. In practice, however, it turned out that a combination of cutting-planes with the *Branch & Bound* technique is more successful. Its conjunction is called

Branch & Cut

Branch & Cut.

The core of a Branch & Cut algorithm is displayed in Fig.[2] 1.6.2. After the initialization and preprocessing this algorithm uses a tree to maintain the subproblems. It begins the **bounding process** with its root. After solving the LP-relaxation, it is checked whether the solution is LP feasible. In the case of feasibility the **local upper bound** (LUB) is updated. If the LUB is smaller than the **lower bound** (LB), which is at the beginning minus infinity, the subproblem is **fathomed**. Otherwise, the algorithm checks if the solution is integer feasible. Clearly, if it is an integer solution, the best solution for this subproblem has been computed and this node of the tree can be fathomed. Mostly, this is not the case and heuristics are used to compute feasible solutions, which provide the lower bounds. This is done in step "expand LP". The separation routine computes cutting-planes which are added to the LP-relaxation. If improvement of the LP solution after adding new constraints is too small—**tailing off**—or the separation routine could not compute a new constraints, the algorithm **branches** in new subproblems, e.g. by setting a variable. If the selection of a subproblem of the tree is successful, which means that

bounding
local upper bound
lower bound
fathom
tailing off
branching

[2] This figure is based on [The05] and [EGJR01].

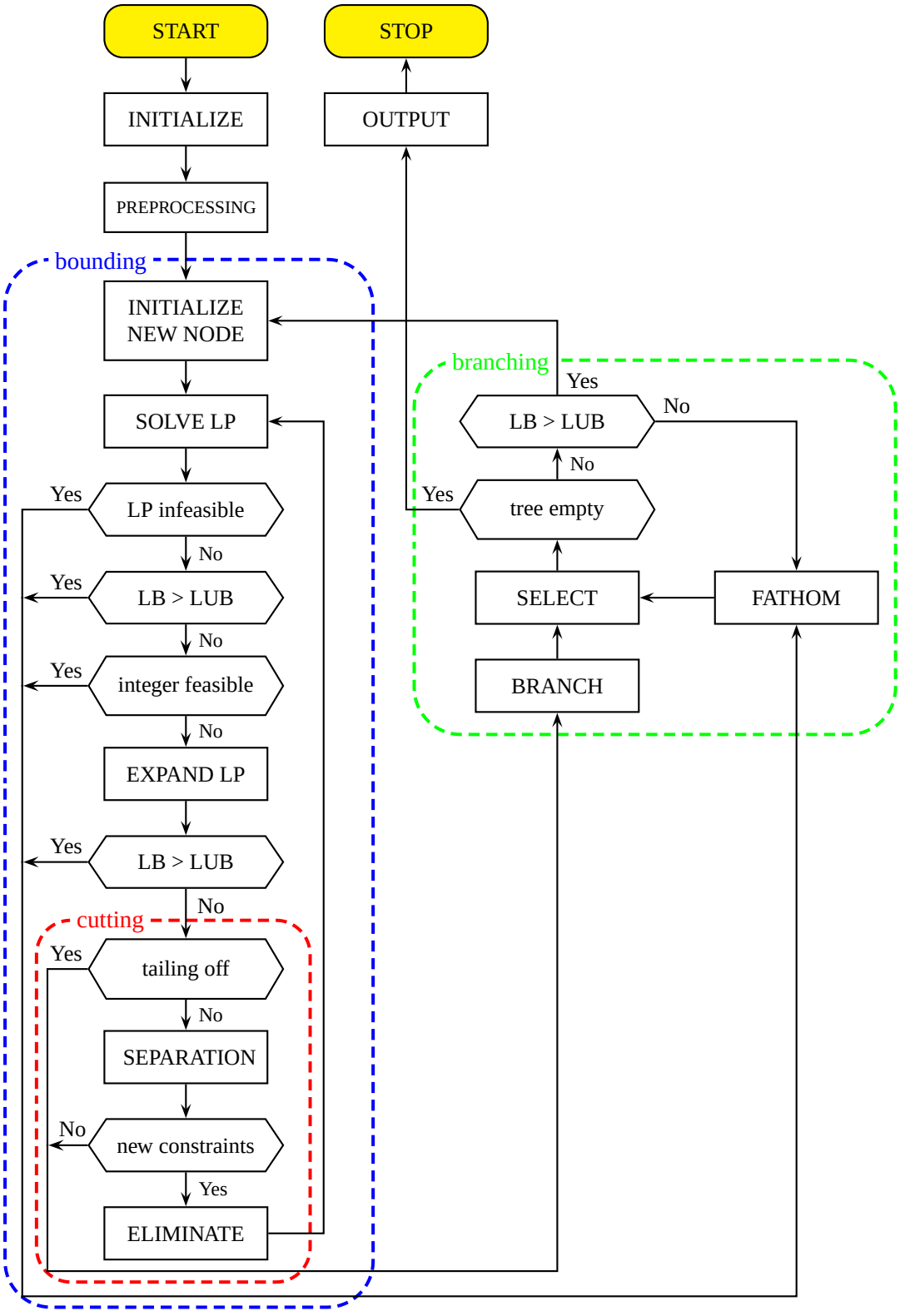


Figure 1.6.2: Flowchart of a Branch & Cut algorithm

the tree is not empty and that the UB is greater than the LB, the algorithm reenters the bounding phase. If the selected subproblem does not bring new information, we fathom it and select a new one. This process terminates when the tree is empty.

global upper bound

While the LP-relaxation of (1.5.1) provides a **global upper bound** (GUB), each subproblem computes a local upper bound which is only valid for all its sons in the tree. In some cases the problems are too difficult to solve them exactly and therefore it is a good idea to stop the algorithm after a certain number of iterations. The advantage of a Branch & Cut algorithm is that this algorithm provides next to a solution also a quality of it. This quality is measured by the **gap**, which is calculated by

gap

$$\frac{GUB - LB}{GUB}.$$

It turns out that it is not helpful to add more and more inequalities to the LP-relaxation. The inequality system $Ax \leq b$ may grow too big, which slows down the computation time. Therefore after each separation phase, some inequalities are eliminated and stored in a pool. This pool can also be checked once in a while, if it contains a violated constraint which is then added again. This process is called **pool separation**.

pool separation

Chapter 2

Stable Set

In this chapter we will give an introduction to the basic problem of the diploma thesis. After some definitions and a small example, we consider relationships to other problems and give an overview of the wide field of applications for the stable set problem. This chapter is mainly based on [GLS88, BP76, Rei01, HV77, Wes00].

2.1 Definition and Introduction

We start with the definition of a stable set.

> A **stable set** of a graph G is a set of nodes S with the property that the nodes of S are pairwise non adjacent.

stable set

A stable set is also known as *independent set*, *vertex packing*, *co-clique* or *anticlique*. We abbreviate a stable set with $\mathcal{S}$. Let G_c be a node-weighted graph. A stable set of G_c, which maximizes $\sum_{v \in \mathcal{S}} c(v)$, is called a **maximum stable set**, or a *maximum-weight stable set* and is denoted by the symbol $\mathcal{S}^*$. A maximum stable set is called *maximum-cardinality stable set* or *maximum-size stable set* in the special case when the weighting function is equal to $\mathbb{1}$. The weight of a maximum stable set is denoted by $\alpha(\mathrm{G}_c)$ and in the case of $c = \mathbb{1}$, it is the cardinality of a maximum stable set of the corresponding unweighted graph. In this case we neglect c and write more simply $\alpha(\mathrm{G})$. The size of a maximum-cardinality stable set is called the **stability number** or the *stable set number*. We distinguish strictly between a maximum stable set and a **maximal stable set** which is an inclusion-maximal stable set. This means that there is no stable set which contains the maximal stable set and has a higher weight.

maximum stable set

stability number

maximal stable set

Let us come back to the example of the introduction. Hence, the task is to find the best set of events to attend in a fair with respect to a priority list. We model this as a maximum-weight stable set problem on a graph G_c. Each event will be represented as a node of G. Whenever there is a conflict between two events, an edge between their corresponding nodes is added. The priority of the events is transformed into a node weighting c. The task of finding a best set of events to join is then the problem of finding a maximum stable set in G_c. This construction of G is a general concept. Such a graph is called a **conflict graph**. We will come back to conflict

conflict graph

graphs in Section 2.3. Let us now look at Fig. 2.1.1 which shows us a more theoretical example. This weighted graph has eight nodes, $v_1, \ldots, v_8$, and their corresponding weights, which are written as a tuple for each node. The set $\{v_2, v_8\}$, for instance, is a stable set of G. The red colored nodes v_2,v_4 and v_7 build a maximum stable set $\mathcal{S}^*$. The weight of $\mathcal{S}^*$ is 6 which is $\alpha(\mathrm{G}_c)$. It can be seen from this graph that a maximum stable set is not uniquely determined. Set $\{v_2, v_4, v_8\}$ is also a maximum stable set with weight 6. The yellow nodes v_3 and v_5 build a maximal stable set of G with weight 5, because $\Gamma(\{v_3, v_5\}) \cup \{v_3, v_5\} = V$.

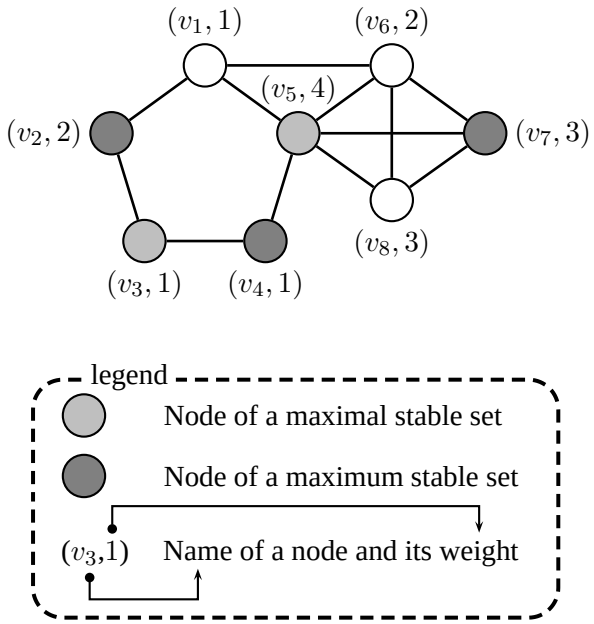


Figure 2.1.1: Weighted graph with legend

2.2 Relationships to Other Problems

Before we have a look at the numerous applications of the stable set problem, we summarize some relationships of the stable set problem to other graph theoretical problems.

clique number

Let $\mathrm{G}_c = (V, E, c)$ be a connected node-weighted graph. The **clique number** is the maximum weight of a clique in G and is denoted by $\omega(\mathrm{G}_c)$. Note that a clique in G corresponds to a stable set in $\overline{\mathrm{G}}$ with the same weight and vise versa. Hence there is a bijection between the stable sets in G and the cliques in $\overline{\mathrm{G}}$, which gives

$$\alpha(\mathrm{G}_c) = \omega(\overline{\mathrm{G}}_c). \tag{2.2.1}$$

node partitioning

weighted node partitioning

This shows that the maximum stable set problem and the problem of determining a maximum clique are equivalent. A **node partitioning** is a subset $W \subseteq V$ such that every edge has exactly one endnode in W. A node partitioning does not exist for all graphs, for instance not for K_n with $n > 2$ and not for graphs induced by odd cycles. The **weighted node partitioning** problem asks for a minimum node partitioning with respect to a node weighting c. We show that the weighted node partitioning problem is equivalent to the weighted stable set problem if a node partitioning

exists. Clearly, a minimum node partitioning of the weights $-c$ is a maximum-weight stable set On the other hand, a maximum stable set with $c_i + \delta(x_i)\theta$ as the components of the weighting function corresponds to a minimum node partitioning. One has to choose θ sufficiently large enough, for instance $\theta = \sum_{i=1}^{n} |c_i|$. With this choice, all edges of G are covered by a stable set. Since exactly m edges are covered, the weight of a minimum node partitioning is $\alpha(\mathrm{G}_c) - \theta \cdot m$.

In the following we consider some cardinality problems on a graph $\mathrm{G} = (V, E)$. An **edge covering** is a subset $F \subseteq E$ such that every node is incident to at least one edge in F. The minimum cardinality of an edge covering is the **edge covering number** and is denoted by $\rho(\mathrm{G})$. A stable set cannot have more nodes than an edge covering has edges in a graph without isolated nodes. This is stated by the following inequality

edge covering

edge covering number

$$\alpha(\mathrm{G}) \leq \rho(\mathrm{G}).$$

The triangle K_3 provides one example where this inequality is tight. The stability number and the edge covering number are equal, if G is bipartite and has no isolated nodes. This was proven by KÖNIG in 1916. A **node covering**, in contrast to an edge covering, is a subset $W \subseteq V$ such that every edge has at least one endnode in W. The minimum cardinality of a node covering is called the **node covering number** and is denoted by $\tau(\mathrm{G})$. If $\mathcal{S}$ is a stable set on G, $\overline{\mathcal{S}} = V \setminus \mathcal{S}$ is a node covering of G, since every edge has two endnodes of which one is in $\overline{\mathcal{S}}$. Otherwise, if $\overline{\mathcal{S}}$ is a node covering, $\mathcal{S}$ defines a stable set, because no two nodes of $\mathcal{S}$ can be adjacent. Thus, finding a maximum cardinality stable set is equivalent to finding a minimum node covering. This shows

node covering

node covering number

$$\alpha(\mathrm{G}) + \tau(\mathrm{G}) \quad = \quad |V|$$

which was primary proven by GALLAI in 1959. A **matching** is a collection of pairwise disjoint edges. If in a matching M every node of G is incident with exactly one edge in M, then it is a **perfect matching**. The maximum cardinality of a matching in G is called the **matching number** and is denoted by $\nu(\mathrm{G})$. For a graph G without isolated nodes, GALLAI showed in 1959 the following connection between matchings and edge coverings

matching

perfect matching

matching number

$$\nu(\mathrm{G}) + \rho(\mathrm{G}) \quad = \quad |V|.$$

A **coloring** of G is a partition of V into disjoint stable sets, while a **clique covering** is a partition into disjoint cliques. The **coloring number**, which is denoted by $\chi(\mathrm{G})$, is the smallest number of stable sets needed for a coloring of G. Analogously, the smallest number of cliques for a clique covering of G is called **clique covering number** and is abbreviated with $\overline{\chi}(\mathrm{G})$.

coloring

clique covering

coloring number

clique covering number

Consider the following inequalities

$$\chi(\mathrm{G}) \quad = \quad \overline{\chi}(\overline{\mathrm{G}}), \tag{2.2.2}$$

$$\omega(\mathrm{G}) \quad \leq \quad \chi(\mathrm{G}), \tag{2.2.3}$$

$$\alpha(\mathrm{G}) \quad \leq \quad \overline{\chi}(\mathrm{G}). \tag{2.2.4}$$

Since the number of stable sets needed to cover a graph is equal to the number needed to cover the complement with cliques, inequality (2.2.2) is true. To partition the node set of a graph G into disjoint stable sets, one needs at least the number of a maximum clique in G. This is stated by inequality (2.2.3). In a 5-cycle a maximum clique has size two, but the coloring number is three. This shows that the coloring number and the clique number are not in general equal. Inequality (2.2.4) is the consequence of (2.2.1), (2.2.2), (2.2.3) and the observation that the complement of $\overline{\mathrm{G}}$ is again G.

2.3 Applications

In the last section we have seen various connections of the stable set problem to other problems in graph theory. Now we want to consider some more applications of the stable set problem beyond the motivating example.

In *coding theory*, one can ask to construct a (n, M, d) code. This is a set of M binary vectors of length n such that any two vectors differ in at least d places, and d is the largest number with this property. When n and d are given, the aim is to compute a maximum number of codewords. Geometrically, a codeword can be identified as a binary vector with length n representing a vertex of a unit cube in n dimensions. In this interpretation, a (n, M, d) code is just a subset of these vertices. It can be shown that the minimum distance d is equivalent to the property that two codewords have a Euclidean distance greater than or equal to $\sqrt{d}$. With this information we construct a graph G as follows. The nodes are the codewords and two nodes are adjacent iff their Euclidean distance is smaller than $\sqrt{d}$. The problem of finding a maximum number of codewords and computing them is then just a maximum-cardinality stable set problem on graph G, cf. [MS96a].

One recognizes the similarity of the construction of graph G in the application in coding theory and the modeling of the motivating example. Again, graph G is a conflict graph. This concept is quite strong and has many applications besides the provided examples. For instance in...

... case-based reasoning systems [Igl01],

... combinatorial auctions [dVV03],

... computer vision [BB82],

... economics [AB62],

... fault diagnosis [BP90],

... forest planning [BWE92],

... geometric tilding [CS90],

... graph coloring [MT96],

... integer programming [ANS00],

... map labeling [SVA00, AvKS98],

... molecular biology [SBS05, VP95, GAW98, GARW93, MARW89],

... pattern recognition [HS89],

... scheduling [JSW97, BSSW05],

... stock cutting [Pie70].

Chapter 3

Stable Set Polytope

In this chapter we look at some polyhedral aspects of the maximum stable set problem. We start with the definition of the stable set polytope and develop an integer program formulation for the stable set problem. It turns out that the stable set problem is $\mathcal{NP}$-$hard$ and we have to consider a relaxation. We will see that this first relaxation is very weak in general and the underlying polytope describes the stable set polytope only for bipartite graphs. To strengthen this relaxation for general graphs, several classes of inequalities valid for the stable set problem are considered. We begin with the so-called odd-cycle inequalities in Section 3.2 and discuss the lifting procedure. A very important inequality class for the stable set polytope are the clique inequalities, which are presented in Section 3.3. We review a theorem of PADBERG which implies that they are facet-defining if and only if the corresponding clique is maximal. The clique inequality will lead us to perfect graphs for which the stable set problem is proven to be polynomially solvable. An overview of some more inequalities valid for the stable set polytope is given in Section 3.4. A large class of inequalities containing the odd-cycle and clique inequalities is considered next to a facet-producing graph. This graph will be used in Chapter 4.2.3. Besides where otherwise noted, the following is mainly based on [GLS88, GL06, NW88]. Throughout this chapter we assume that all graphs are connected.

3.1 Introduction

To understand the definition of the stable set polytope, we first have to define an **incidence vector** $\chi^{\mathcal{S}}$ of a stable set $\mathcal{S}$ of graph G as a $|\mathrm{G}|$-dimensional vector with the following components

incidence vector

$$\chi_i^{\mathcal{S}} := \begin{cases} 1, & \text{if } v_i \in \mathcal{S} \\ 0, & \text{otherwise.} \end{cases}$$

Now, we are ready for the following definition.

Definition 3.1.1. *[Sch03] The **stable set polytope** of a graph $\mathrm{G} = (V, E)$ is the convex hull of the incidence vectors of all stable sets in G. It is denoted by*

stable set polytope

$$P_{STAB}(\mathrm{G}) := \text{conv}\left\{\chi^{\mathcal{S}} \mid \mathcal{S} \subseteq V \text{ stable set }\right\}.$$

If is is clear from the context which graph is meant, we sometimes just write P_{STAB} instead of $P_{STAB}(G)$. The stable set polytope is bounded by the n-dimensional unit cube and therefore it is a polytope. The definition of a stable set implies that the unit vectors are always stable sets. The zero vector is trivially a stable set—the empty set—and therefore, the stable set polytope is full-dimensional. This implies that all facets of P_{STAB} are inequalities and hence we do not have to consider equalities.

A maximum stable set corresponds to a vertex of P_{STAB} and the incidence vector $\chi^{\mathcal{S}}$ to a binary variable x. Since the stability number is equal to $c^\top x$, it is a candidate for the objective function of an IP formulation. To complete the IP formulation, we need inequalities which define P_{STAB}. Consider the following inequality system

$$x_i \geq 0 \qquad \forall i \in V, \tag{3.1.1}$$
$$x_i + x_j \leq 1 \qquad \forall ij \in E. \tag{3.1.2}$$

non-negativity inequality

The definition of an incidence vector of a stable set implies the so-called **non-negativity inequalities** (3.1.1). They are always facet-defining for P_{STAB}, since there are $n-1$ affinely independent solutions which have value zero in one entry. The inequalities (3.1.2) ensure that there cannot be a pair of adjacent nodes in one stable set which is a direct consequence of its definition. This type of constrains is called **edge inequality**. Hence, inequalities (3.1.1) and (3.1.2) are both valid for the stable set polytope. Inequalities (3.1.2) are not generally facet-defining. We get a criterion for that in Section 3.3. Recognize that all integer solutions of the inequality system (3.1.1) and (3.1.2) are incidence vectors of a stable set of G. This is only true if graph G contains no isolated nodes which is satisfied as we assume a connected graph. However, if the graph is disconnected, each variable corresponding to a isolated node is unbounded. To avoid this, one can limit it with value 1. Therefore we consider in this diploma thesis only graphs without isolated nodes. Inequalities (3.1.2) together with the non-negativity inequalities are called **trivial inequalities**. The last observations lead us to a linear integer program formulation for the maximum stable set problem

edge inequality

trivial inequality

$$\begin{aligned} \mathcal{IP}: \quad \max \quad & c^\top x \\ \text{s.t.} \quad & Ax \leq \mathbb{1} && (3.1.3) \\ & x \in \{0,1\}^{|V|}. && (3.1.4) \end{aligned}$$

In this integer program, vector x corresponds to an incidence vector of a stable set and matrix $A^\top$ is the node-edge incidence matrix of G. Inequalities (3.1.3) are equivalent to (3.1.2).

It is well known that it is $\mathcal{NP}$-$hard$ to determine a maximum stable set in an arbitrary graph. For a proof see for instance GAREY and JOHNSON [GJ79]. This is still true in the special case when the weighting function is equal to $\mathbb{1}$. Furthermore, it can be shown that for fixed $\varepsilon > 0$ there is no polynomial time algorithm for approximating the stability number within a factor of $|V|^\varepsilon$ under the assumption that $\mathcal{P} \neq \mathcal{NP}$, cf. [AS92, FMK95]. Hence, it is also $\mathcal{NP}$-$hard$ to find a solution of $\mathcal{IP}$. Nevertheless, we will use the Branch & Cut algorithm to solve this problem and therefore we will formulate a linear program and see how well it describes the stable set polytope. The reason why $\mathcal{IP}$ is so challenging to solve, are the harmless looking inequalities (3.1.4). Therefore, a natural way how to create a polynomially solvable problem is

to substitute the inequalities (3.1.4) by (3.1.1), which results in the LP-relaxation

$$\begin{aligned} \mathcal{LP}: \quad \max \quad & c^\top x \\ \text{s.t.} \quad & Ax \leq \mathbb{1} \qquad (3.1.5) \\ & x \geq 0. \qquad (3.1.6) \end{aligned}$$

Its polytope is described by

$$P_{RSTAB}(\mathrm{G}) := \left\{ x \in \mathbb{R}^{|V|} \,|\, Ax \leq \mathbb{1}, x \geq 0 \right\}$$

and called **stable set polytope relaxation**. In order to get a feeling for the quality of P_{RSATB} compared to P_{STAB}, we solve $\mathcal{LP}$ for the graph given in Fig. 2.1.1, for example with the simplex method. We get the solution vector $x^* = (0, 1, 0, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2})$ with objective function value 8.5. Clearly, this vector is not an incidence vector of a stable set. As we have seen in Chapter 2.1, a maximum stable set for that graph has weight 6. Thus the absolute difference of the two objective function values is 2.5. The percentage difference gets even larger if we solve the maximum-cardinality stable set problem only for the triangle built by nodes v_5, v_6 and v_7. In this case, the $\mathcal{LP}$ solution has value 1.5 while a maximum stable set has cardinality 1, which is a difference of 50%. In general, the objective value of $\mathcal{LP}$ for an arbitrary graph with weighting $\mathbb{1}$ is greater or equal than $\frac{n}{2}$—independent from its structure. This shows that the stable set polytope relaxation is very week in general. Obviously, we have to put in some effort if we want to solve the maximum stable set problem for this and other graphs. We recognize from our example that the solution is $(0, \frac{1}{2}, 1)$-valued. This is not a coincidence, indeed there is a structure behind it. Consider the following corollary, which was indicated by BALINSKI [Bal70].

stable set polytope relaxation

Corollary 3.1.2. *The vertices of $P_{RSTAB}(\mathrm{G})$ are $(0, \frac{1}{2}, 1)$-valued.*

Proof. Based on [NT74]. Let x be a vertex of $P_{RSTAB}(\mathrm{G})$ and define the two sets

$$\begin{aligned} U_{-1} &:= \left\{ v_i \,|\, 0 < x_i < \tfrac{1}{2}, 1 \leq i \leq n \right\}, \\ U_1 &:= \left\{ v_i \,|\, \tfrac{1}{2} < x_i < 1, 1 \leq i \leq n \right\}. \end{aligned}$$

We have to show that they are both empty. Therefore we define the following two vectors

$$y_i := \begin{cases} x_i - \varepsilon, & \text{if } v_i \in U_{-1} \\ x_i + \varepsilon, & \text{if } v_i \in U_1 \\ x_i, & \text{otherwise,} \end{cases}$$

$$z_i := \begin{cases} x_i + \varepsilon, & \text{if } v_i \in U_{-1} \\ x_i - \varepsilon, & \text{if } v_i \in U_1 \\ x_i, & \text{otherwise} \end{cases}$$

for all i with $1 \leq i \leq n$ and an $\varepsilon > 0$. From the construction of y and z follows that $x = \frac{y+z}{2}$. Now assume that at least one of the two sets U_{-1} or U_1 is not empty. One possible choice for ε such that y and z are in $P_{RSTAB}(\mathrm{G})$ is

$$\varepsilon := \min\left\{ \tfrac{1}{2} - x_i, 1 - x_j \,|\, v_i \in U_{-1}, v_j \in U_1 \right\} > 0.$$

From the construction of ε, the vectors y and z are non-negative in each component. Hence, they satisfy the non-negativity inequalities (3.1.1). For each edge v_iv_j, the following holds for the components of vector y

$$y_i + y_j = \begin{cases} x_i + x_j - 2\varepsilon, & \text{if } v_i, v_j \in U_{-1} \\ x_i + x_j, & \text{if } (v_i \in U_{-1} \wedge v_j \in U_1) \vee (v_j \in U_{-1} \wedge v_i \in U_1) \\ & \quad \vee (v_i, v_j \notin U_{-1} \cup U_1) \\ x_i + x_j - \varepsilon, & \text{if } (v_i \in U_{-1} \wedge v_j \notin U_{-1} \cup U_1) \vee (v_j \in U_{-1} \wedge v_i \notin U_{-1} \cup U_1) \\ x_i + \varepsilon, & \text{if } (v_i \in U_1 \wedge v_j \notin U_{-1} \cup U_1) \vee (v_j \in U_1 \wedge v_i \notin U_{-1} \cup U_1) \end{cases}$$

because x is a vertex of P_{STAB}, the case v_i and $v_j \in U_1$ cannot occur and $y_i + y_j \leq 1$. With the same argument, vector z satisfies the edge inequalities. As the three vectors x, y and z are pairwise different, the convex combination of x by y and z is contradictory to the assumption that x is a vertex. This implies that $U_{-1} = U_1 = \emptyset$, which closes the proof. □

To motivate the next theorem, we look again at the graph of Fig. 2.1.1, but in this case we delete the three edges v_1v_5, v_5v_7 and v_6v_8 and denote the graph by $\widetilde{\mathrm{G}}$. Solving $\mathcal{LP}$ for that graph leads to the following interesting result: $x^* = (0, 1, 0, 0, 1, 0, 1, 0)$. In this case it is a $(0, 1)$-valued solution which indicates the stable set $\mathcal{S} = \{v_2, v_5, v_7\}$. Again, this has a special reason. The new graph $\widetilde{\mathrm{G}}$ is bipartite. In Fig. 3.1.1 one can see the bipartition V_1 and V_2 of $\widetilde{\mathrm{G}}$.

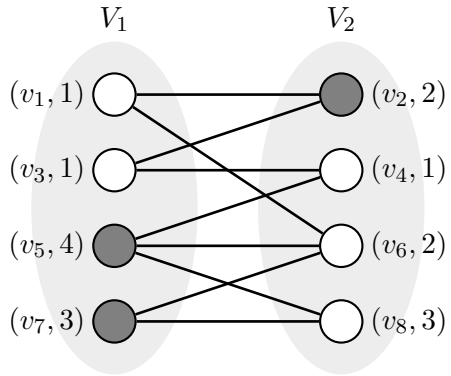


Figure 3.1.1: Bipartite graph

The following theorem summarizes this observation.

Theorem 3.1.3. *[GLS88] The non-negativity inequalities (3.1.1) together with the edge inequalities (3.1.2) are sufficient to describe $P_{STAB}(\mathrm{G})$ iff G is bipartite and has no isolated nodes.*

Proof. " $\Rightarrow$ " Let G be non bipartite. Then we know that G contains at least one odd cycle C, cf. Theorem 1.1.1. Consider vertex $N := (0, \ldots, 0, \frac{1}{2}, \ldots, \frac{1}{2}, 0, \ldots, 0)$ where the zero values stand in all vertices not according to cycle C. Obviously, according to its definition N satisfies all inequalities (3.1.1) and (3.1.2). Next, we show that N lies not in $P_{STAB}(\mathrm{G})$. First, recognize that each integer vertex of the stable set polytope has maximal $\lfloor \frac{|C|}{2} \rfloor$ times a 1-entry in a vertex of C. Since the vertices of $P_{STAB}(\mathrm{G})$ are $(0, 1)$-valued, you have only two vertices from $P_{STAB}(\mathrm{G})$ to combine vertex N convex. Two arbitrary vertices of $P_{STAB}(\mathrm{G})$

have in the nodes of C only $2 \cdot \left\lfloor \frac{|C|}{2} \right\rfloor = 2 \cdot \frac{|C|-1}{2} = |C| - 1$ times entry 1, which implies that $N \notin P_{STAB}(\mathrm{G})$.

" $\Leftarrow$ " The set of all solutions of the inequality systems (3.1.1) and (3.1.2) is given through

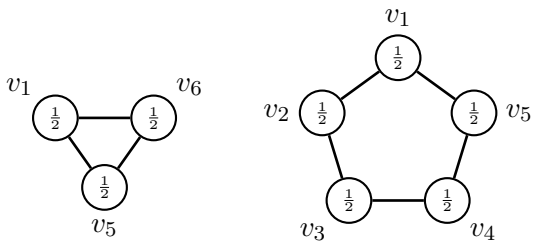
$$L := \{x \mid x \geq 0,\, x^\top M \leq \mathbb{1}^\top\}.$$

In this case M is the node-edge incidence matrix with dimension $|V| \times |E|$. If we can show that $L = P_{STAB}(\mathrm{G})$ we are done. It has already been mentioned that each stable set of G is described by the non-negativity and edge constraints. Therefore $P_{STAB}(\mathrm{G})$ is a subset of $L(\mathrm{G})$. On the other hand, every vertex of L is integer, since matrix M is totally unimodular, compare Proposition 1.1.2, and set L is bounded, because graph G has no isolated nodes. Now we know that all vertices of the polytope of L are integer. Since they are also the vertices of the stable set polytope, the proof is closed. □

Theorem 3.1.3 has the following important implication. It states that the maximum stable set problem for bipartite graphs can be solved in polynomial time, by solving $\mathcal{LP}$. As a consequence, a Branch & Cut algorithm using $\mathcal{LP}$ as the first relaxation, will terminate for bipartite graphs after one iteration. Therefore, there is no need in a Branch & Cut framework to check whether the graph is bipartite or not in order to get a fast solver. Nevertheless, it can be checked in linear time if a graph is bipartite or not. With the use of some graph theory, the polynomial time algorithm designed by HOPCROFT and KARP in 1973 can be adapted to solve the maximum stable set problem for bipartite graphs, cf. [GLS88].

3.2 Odd-Cycle Inequalities

In the last section we have seen that $P_{RSTAB}(\mathrm{G})$ and $P_{STAB}(\mathrm{G})$ are only equal if G is bipartite. We also considered example 2.1.1 where the inequalities (3.1.1) and (3.1.2) are not sufficient to describe the stable set polytope. One of the simplest not bipartite graphs are the odd cycles. A 3-cycle and a 5-cycle can be seen in Fig. 3.2.1. These graphs are induced subgraphs of G of Fig. 2.1.1. We have already used the observation in the proof of Theorem 3.1.3 that point $N = (\frac{1}{2}, \ldots, \frac{1}{2})$ lies in P_{RSTAB} but not in P_{STAB}; in particular, N is a vertex of P_{RSTAB}. The values in the nodes of the two odd cycles are optimal values of the corresponding variables in the $\mathcal{LP}$-model.



(a) 3-cycle (b) 5-cycle

Figure 3.2.1: Two odd cycles

odd-cycle inequality

This suggests a new class of inequalities for $P_{STAB}(\mathrm{G})$, which are called **odd-cycle inequalities**

$$\sum_{v_i \in V(C)} x_i \leq \frac{|V(C)|-1}{2} \qquad \text{for each odd cycle } C \text{ in G.} \tag{3.2.1}$$

It is obvious that the odd-cycle inequalities are valid for the stable set polytope, since the cardinality of a stable set in an odd cycle can be maximal the greatest integer which is smaller than half length of the cycle. The polytope satisfying the non-negativity, edge and odd-cycle inequalities is called **cycle-constraint stable set polytope** of G and is denoted by

cycle-constraint stable set polytope

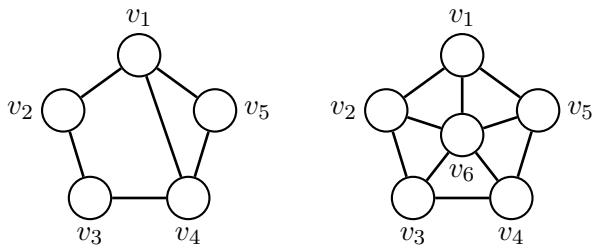
$$P_{CSTAB}(\mathrm{G}) := \left\{x \in \mathbb{R}^{|V|} \mid x \text{ satisfies } (3.1.1), (3.1.2) \text{ and } (3.2.1)\right\}.$$

t-perfect

A graph G is called **t-perfect**[1] if $P_{CSTAB}(\mathrm{G}) = P_{STAB}(\mathrm{G})$, which means that the inequalities of P_{CSTAB} are sufficient to describe the stable set polytope. Examples for t-perfect graphs are bipartite and almost bipartite graphs[2]. The problem of checking whether a graph is t-perfect or not belongs to $co\text{-}\mathcal{NP}$: a non-integer vertex of P_{CSTAB} would do this. Like in the previous section, the special structure of t-perfect graphs helps finding a maximum stable set. This is stated by the next corollary.

Corollary 3.2.1. *The maximum stable set problem in a t-perfect graph can be solved in polynomial time.*

We will see in Chapter 4.2 that the odd-cycle separation is in $\mathcal{P}$. This proofs together with [GLS88] Corollary 3.2.1.



(a) 5-cycle with chord (b) 5-cycle with extra node

Figure 3.2.2: Not facet-defining odd-cycle inequalities

We are mainly interested in facets of P_{STAB} since they are not dominated by any valid inequality of P_{STAB}. The odd-cycle inequalities can only be facet-defining if its odd cycle is chordless. Therefore consider Fig. 3.2.2 (a). The chord v_1v_4 cracks the 5-cycle in the 3-cycle $(v_1, v_1v_4, v_4, v_4v_5, v_5, v_5v_1, v_1)$, and the 4-cycle $(v_1, v_1v_2, v_2, v_2v_3, v_3, v_3v_4, v_4, v_4v_1, v_1)$. In general, if there is a chord, one gets a smaller odd-cycle and an even cycle. The smaller odd-cycle inequality together with the edge inequalities dominate the odd-cycle inequalities which shows that they cannot be a facet. In our case

$$\begin{array}{rrrrr} & x_1 & & + x_4 + x_5 & \leq 1 \\ + & & x_2 + x_3 & & \leq 1 \\ \hline & x_1 + x_2 + x_3 + x_4 + x_5 & & & \leq 2. \end{array}$$

[1] The "t" stands for "trou", which is the French word for "hole".

[2] A graph G is called almost bipartite if there is a node v such that graph $\mathrm{G} - v$ is bipartite.

A graph which is a chordless cycle is called **hole**. If an odd cycle induces an odd hole, the corresponding odd-cycle inequality is called **odd-hole inequality**. Consider the following

hole

odd-hole inequality

Corollary 3.2.2. *[NT74] Let* G *be an odd hole. Then* $\sum_{v_i \in V} x_i \leq \frac{|V|-1}{2}$ *is facet-defining for* $P_{STAB}(G)$.

Proof. Let L be a facet of $P_{STAB}(\mathrm{G})$ defined through inequality (β, b_0) and

$$F := \left\{ x \in P_{STAB}(\mathrm{G}) \,\middle|\, \sum_{v_i \in V} x_i \leq \tfrac{|V|-1}{2} \right\} \subseteq L.$$

Label graph G with indices $i = 1, \ldots, n = |V|$ such that

$$v_i v_j \in E \Longleftrightarrow |i - j| \equiv 1 \pmod{n-2}.$$

The incidence vector χ of a maximum stable set on G has now alternating entries of value 0 and 1, except for one case where the entries are a sequence of two zeros. Define the index of the corresponding variable to such a 0 as i, which is followed by a 1, modulo n. The index of this variable is denoted by j. Now, let x_{ij} be a variable corresponding to such an incidence vector. The value of entry i and j are exchanged in variables x_{ij} and x_{ji}, but they are equal in all other components. By construction, both vectors are in $P_{STAB}(\mathrm{G})$ and insertion of them into inequality (β, b_0) yields in

$$\begin{aligned} \beta x_{ij} = b_0 = \beta x_{ji} \\ \Rightarrow \beta_j = \beta_i, \end{aligned}$$

which is true for all neighbors i and j, or more precisely $\forall i, j : |i - j| \equiv 1 \pmod{n-2}$. Thus, there exists $\eta \in \mathbb{R}$ with $\beta = \eta \cdot \mathbb{1}$. Since all components of β are positive, otherwise inequality (β, b_0) would be dominated by a non-negativity inequality, η has to be greater than zero. Consider the right-hand side

$$\beta \cdot x_{uv} = \eta \cdot x_{uv} = \eta \cdot \tfrac{|V|-1}{2}.$$

This implies

$$(\beta, b_0) = (\eta \cdot \mathbb{1}, \eta \cdot \tfrac{|V|-1}{2}) = \eta \cdot (\mathbb{1}, \tfrac{|V|-1}{2}),$$

which shows that $F = L$. □

Corollary 3.2.2 has the strict assumption that G has to be an odd hole. In all other cases it depends on the structure of the graph whether the odd-cycle inequalities (3.2.1) are facet-defining for $P_{STAB}(\mathrm{G})$ or not. We will see in Section 3.4 that the two odd-hole inequalities indicated by Fig. 3.2.1 are facet-defining for the graph of Fig. 2.1.1. On the contrary, consider Fig. 3.2.2 (b) which shows a chordless 5-cycle with an additional node. Obviously, inequality

$$x_1 + x_2 + x_3 + x_4 + x_5 + 2x_6 \leq 2 \tag{3.2.2}$$

is valid and dominates the odd-cycle inequality $\sum_{i=1}^{5} x_i \leq 2$, which shows that this odd-cycle inequality has to be lifted to get a facet-defining inequality.

Lifting

lifting

The process called **lifting** is the extension of a valid inequality for a polytope P to a valid inequality for a higher dimensional polytope $\tilde{P} \supseteq P$. Thereby, one must compute appropriate coefficients for the other variables missing. If all the coefficients are zero, we call the process **trivial lifting**. This is the simplest way and leads in any cases to a valid inequality of P_{STAB}. However, the main purpose of lifting is to receive a facet-defining inequality of $\tilde{P}$ from a facet-defining inequality (β, b_0) for P. As (β, b_0) is a face of polytope P, its dimension has to be increased. Consider again odd-cycle inequality $\sum_{i=1}^{5} x_i \leq 2$ for the graph of Fig. 3.2.1, which is a facet of the odd hole induced by the nodes $v_1, \ldots, v_5$. We have already seen that this inequality does not define a facet of the whole graph. The addition of $2x_6$ gave us inequality (3.2.2). This is a facet-defining inequality, since the characteristic vectors corresponding to the stable sets $\{v_1, v_3\}, \{v_1, v_4\}, \{v_2, v_4\}, \{v_2, v_5\}, \{v_3, v_5\}$ and $\{v_6\}$ are linearly independent and satisfy (3.2.2) at equality. Note that for coefficient 1 of variable x_6, inequality (3.2.2) is still valid, but does not define a facet and if the coefficient is greater than 2, the inequality is no longer valid. The next theorem shows that there are always such integer coefficients for P_{STAB}.

trivial lifting

Theorem 3.2.3. *[NT74] Let $\mathrm{G} = (V, E)$ be a graph and $W \subseteq V$. Suppose*

$$\sum_{v_i \in W} \beta_i x_i \leq b_0$$

is facet-defining for $P_{STAB}(\mathrm{G}[W])$. Then there are $\beta_i \in \mathbb{N}_0$ for all $v_i \in V \setminus W$ such that

$$\sum_{v_i \in V} \beta_i x_i \leq b_0$$

is facet-defining for $P_{STAB}(\mathrm{G})$.

We neglect all the details of the proof but instead look at a construction scheme for the coefficients. Consider first the special case with $V \setminus W = \{v_j\}$. Let

$$\begin{aligned} IP: \quad z^* = \max \quad & \sum_{v_i \in W} \beta_i x_i \\ \text{s.t.} \quad & Ax \leq \mathbb{1} - A_{\cdot,j} \\ & x \in \{0,1\}^{|W|} \end{aligned}$$

be a modified weighted stable set problem where $A_{\cdot,j}$ denotes the j-th column of node-edge incidence matrix A. Then $\beta_j := \max\{0, b_0 - z^*\}$. One observes that if v_j is not adjacent to any node of W, the coefficient $\beta_n = 0$. For the case of $|V \setminus W| > 1$ each coefficients can be computed in the way described. Afterwards it has to be added to set W. Hence, the new coefficients may depend on the previous computed one. Note that the sequence of the computed coefficients might lead in different results which means that one gets different facets. As the subproblems are hard to solve, this method is more of theoretical interest.

sequential lifting

This idea of lifting was primarily introduced by PADBERG in 1972, [Pad73]. It is called **sequential lifting**. PADBERG used it for the case where set W indicates an odd cycle in G. This is a special case of Theorem 3.2.3 and provides the information that an odd-cycle inequality can be lifted to a facet, whenever the odd cycle is chordless and therefore induces an odd hole in G. A slight modification can be found in [GL06].

3.3 Clique Inequalities

As on bipartite graphs the non-negativity and edge inequalities are sufficient to describe P_{STAB} and a graph is only not bipartite if it contains odd cycles, one could think that, in general, the odd-cycle inequalities are enough to describe P_{STAB}. Unfortunately, there is already a graph with four vertices which is not t-perfect. Consider the example of Fig. 3.3.1. It provides the induced subgraph $\mathrm{G}[\{v_5, \ldots, v_8\}]$ of Fig. 2.1.1.

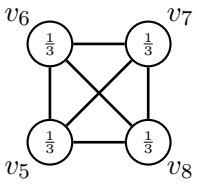


Figure 3.3.1: K_4 is not t-perfect

We recognize that the graph of Fig. 3.3.1 is complete and hence a clique. This suggest the following inequalities

$$\sum_{v_i \in Q} x_i \leq 1 \qquad \text{for each clique } Q, \tag{3.3.1}$$

which are called **clique inequalities**. We consider

clique inequalities

Theorem 3.3.1. *[Pad73] Let* G *be a graph with node set* V *and* $Q \subseteq V$*. Inequality* (3.3.1) *is valid for* $P_{STAB}(\mathrm{G})$*. An inequality* $\sum_{v_j \in Q} x_j \leq 1$ *is a facet of* P_{STAB}*, iff* Q *is a maximal clique in* G*.*

Proof. [Pad73] Let Q be a clique in $\mathrm{G} = (V, E)$. Since there is an edge $v_i v_j \in E$ for all nodes $v_i, v_j \in Q$, a stable set of G can only contain one node of Q which implies that inequality $\sum_{v_j \in Q} x_j \leq 1$ holds for all $x \in P_{STAB}$. Therefore, it is valid for P_{STAB}.

"⇒" Let Q be a clique in G which is not maximal. Then there is a node $v_j \in V \setminus Q$ such that $Q \cup \{v_j\}$ is still a clique in G. Then inequality $\sum_{v_k \in Q \cup \{v_j\}} x_k \leq 1$ is tight for all stable sets, satisfying $\sum_{v_k \in Q} x_k = 1$. In addition, there is at least one solution more, namely

$$x_l = \begin{cases} 1, & \text{if } l = j \\ 0, & \text{otherwise,} \end{cases}$$

where the clique inequality for $Q \cup \{v_j\}$ has slack zero. This shows that inequality

$$\sum_{v_k \in Q} x_k \leq 1$$

is dominated by

$$\sum_{v_k \in Q \cup \{v_j\}} x_k \leq 1$$

and therefore not facet-defining for $P_{STAB}(\mathrm{G})$.

"$\Leftarrow$" We will see that there are $|V|$ linearly independent $(0,1)$-valued solutions of the stable set problem which satisfy the clique inequality at equality. By definition, they are also affinely independent and the affine rank of their face has dimension $|V|-1$ which implies that they are facet-defining. First we construct $|Q|$ solutions by setting $x_j = 1$ for exactly one node of Q at a time and $x_j = 0$ otherwise. If $V = Q$, we have already constructed $|V|$ solutions. Otherwise, for every $v_k \in N \setminus Q$ there must exist at least one $v_j \in Q$ such that $v_k v_j \notin E$, since Q is the node set of a maximal complete subgraph. Therefore we have

$$x_l = \begin{cases} 1, & \text{if } l = k \vee l = j \\ 0, & \text{otherwise.} \end{cases}$$

This implies that we get $|V \setminus Q| + |Q| = |V|$ solutions. To demonstrate that they are linearly independent, we consider the following $|V| \times |V|$ matrix

$$M := \begin{pmatrix} A & B \\ C & A \end{pmatrix}$$

with the $|Q| \times |Q|$ submatrix

$$A := \begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & 1 \end{pmatrix}$$

and the zero matrix C. Matrix B has in each row exactly one entry with value 1 corresponding to a node which is not adjacent to the node selected by matrix A. The dimension of B is $|V \setminus Q| \times |V \setminus Q|$. Since M is an upper triangular matrix, $\det M = 1$, and the corresponding vectors to each row of M are linearly independent. □

Theorem 3.3.1 shows that the edge inequalities (3.1.2) are only facet-defining for P_{STAB}, if they build a maximal clique—hence, they are dominated by the clique inequalities. Note that for triangles, the clique inequality and the odd-cycle inequality are equal. We define the so called **clique-constraint stable set polytope** as

clique-constraint stable set polytope

$$P_{QSTAB}(\mathrm{G}) := \left\{ x \in \mathbb{R}^{|V|} \mid x \text{ satisfies } (3.1.1), (3.1.2) \text{ and } (3.3.1) \right\}.$$

perfect

A graph G is called **perfect** if $P_{QSTAB}(\mathrm{G}) = P_{STAB}(G)$.

Perfect Graphs

We have already seen in Section 2.2 that the coloring number is greater or equal to the clique number. In 1961, BERGE called a graph G perfect, if

$$\omega(\widetilde{\mathrm{G}}) = \chi(\widetilde{\mathrm{G}}) \qquad \text{for each } \widetilde{\mathrm{G}} \subseteq \mathrm{G}. \tag{3.3.2}$$

It can be shown that the polyhedral definition, $P_{STAB}(\mathrm{G}) = P_{QSTAB}(\mathrm{G})$, is equivalent to the graph theoretical one, cf. [GLS88]. The next theorem gives a characterization of the perfect graphs. It was already posed by BERGE in 1962 but primary proven in 2002 by CHUDNOVSKY, ROBERTSON, SEYMOUR and THOMAS, cf. [CRST04].

Theorem 3.3.2 (**Strong Perfect Graph Theorem**). *[RAR01] A graph is perfect iff it, or its complement, does not contain an odd hole of length at least five as an induced subgraph.*

This theorem can be stated more simply as the assertion that a graph is perfect, iff it contains no odd hole and no odd **antihole**, which is the complement of an odd hole. In honor of BERGE a perfect graph is sometimes also called Berge Graph. Examples for the wide class of perfect graphs are bipartite graphs, line graphs of bipartite graphs[3] or triangulated graphs[4]. Especially, each complement of a perfect graph is a perfect graph, which is known as the **Weak Perfect Graph Theorem**, proven by LOVÁSZ in 1972. Consider the following

antihole

Theorem 3.3.3. *[GLS88] The maximum stable set problem for perfect graphs can be solved in polynomial time.*

The proof of Theorem 3.3.3 uses an infinite class of inequalities which are called **orthonormal representation inequalities**. They take the form $\sum_{v_i \in V}(c^\top \cdot u_i)^2 x_i \leq 1$ with real vectors u_i satisfying $\|u_i\| = 1$ and $u_i^\top u_j = 0$ for all $v_i v_j \notin E$ and an arbitrary vector $d \in \mathbb{R}^n$ with $\|d| = 1$. In this case $\|\cdot\|$ denotes the Euclidean norm. The convex set of all vectors satisfying the non-negativity and the orthonormal representation inequalities is called **theta body**. It defines a polytope, iff graph G is perfect. It can be shown that the orthonormal representation inequalities generalize the clique inequalities. For the special case of a perfect graph, the theta body is equal to P_{QSTAB} and P_{STAB}. Furthermore, it can be shown that the separation problem for the orthonormal representation inequalities can be solved in polynomial time. This implies that the maximum stable set problem for perfect graphs can also be solved in polynomial time. This is quite remarkable, as the clique separation problem is $\mathcal{NP}$-$hard$. Indeed, to determine a solution for P_{QSTAB} is in general $\mathcal{NP}$-$hard$, too, and only for perfect graphs proven to be polynomial. We have to neglect more details here and refer the interested reader for instance to [GLS88].

orthonormal representation inequalities

theta body

To check whether a graph is perfect can be done in polynomial time, but the algorithms are quite sophisticated. Recently, there is some progress but the best known algorithms have still a running time of $O(V^6)$, cf. [CCL+05]. For further studies of perfect graphs we recommend the book [RAR01].

3.4 Further Inequalities

In this section we look at some additional inequalities, valid for the stable set polytope of a graph G. The following is based on [Sch03, GL06, BP76].

Consider Fig. 3.4.1 (a). It shows an antihole with 7 nodes which is the complement graph of an odd hole. We recognize the so-called **antihole inequalities**

antihole inequality

$$\sum_{v_i \in A} x_i \leq 2 \qquad \text{for each } A \text{ inducing an antihole in G,} \tag{3.4.1}$$

which are valid for $P_{STAB}(\mathrm{G})$. Note that an antihole with 5 nodes is isomorphic to an odd hole with 5 nodes. The corresponding inequality to an antihole with 6 nodes is equal to the sum of its

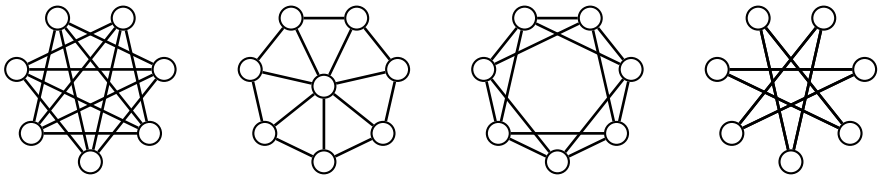
[3] The line graph of G is the graph whose node set is the edge set of G and two nodes are adjacent, iff their corresponding edges are incident to a same node in G.

[4] A graph is called triangulated, if it does not contain a chordless cycle of length at least four.

two triangle inequalities. For an antihole with $n \geq 6$ nodes, adding all its triangle inequalities leads to the following inequality

$$\left(\tfrac{(n-5)(n-4)}{2} + (n-5)(n-3)\right) \sum_{i=1}^{n} x_i \leq \tfrac{n(n-5)(n-4)}{2}. \tag{3.4.2}$$

Insertion of $n = 7$ in this inequality, gives value 34 for each component of x and as left hand side value 90. Dividing the resulting inequality by 34 leads to the right hand side $\frac{90}{34}$ which is fractional. As all coefficients of this inequality are integer, it can be rounded down resulting in the odd-antihole inequality. Formula (3.4.2) shows that this concept does not work for an antihole with more than 7 nodes. Note, the odd-cycle inequalities are also valid for even cycles, but they are equal to the sum of the edge inequalities. In contrast to this, the antihole inequalities bring new information for antiholes of an even order greater than 4.



(a) Odd antihole (b) Odd wheel (c) (7,3)-web (d) (7,3)-antiweb

Figure 3.4.1: Further inequalities

wheel

odd-wheel inequality

In the section about lifting we considered an odd hole having an additional node which is adjacent to all other nodes. Such a graph is called **wheel**. For the special case of a wheel with 5 nodes we have already recognized a facet-defining inequality for that particular graph. This type is known as **odd-wheel inequality** and reads in general

$$\sum_{v_i \in V(C)} x_i + \tfrac{|C|-1}{2} x_u \leq \tfrac{|C|-1}{2} \tag{3.4.3}$$

with C as an odd cycle and $u \in V \setminus C$ with $uv \in E$ for all $v \in C$. In the case of $|C| = 3$ it is a clique inequality. Inequality (3.4.3) is valid for $P_{STAB}(\mathrm{G})$ and defines a facet if G is isomorphic to an odd wheel. For an example of an odd wheel with 8 nodes consider Fig. 3.4.1 (b).

web

antiweb

Let p and q be integers satisfying $p > 2q + 1$ and $q > 1$. A graph G is called a **web** if G is isomorphic to the graph consisting of the nodes $\{v_1, \ldots, v_p\}$ with an edge $v_i v_j$, iff $|i-j| \equiv r < q$ modulo $(n-2)$. A web is abbreviated with $W(p,q)$. A graph is called **antiweb**, denoted by $AW(p,q)$, iff $AW(p,q) \cong \overline{W}(p,q)$. Examples can be seen in Fig. 3.4.1 (c) and (d), respectively. The following inequalities

$$\sum_{v_i \in V(W(p,q))} x_i \leq q, \tag{3.4.4}$$

$$\sum_{v_i \in V(AW(p,q))} x_i \leq \left\lfloor \tfrac{p}{q} \right\rfloor \tag{3.4.5}$$

web inequalities

antiweb inequalities

are called **web inequalities** and **antiweb inequalities**, respectively. Both types of inequali-

ties are valid for $P_{STAB}(\mathrm{G})$. The web inequalities (3.4.4) define facets if p and q are relatively prime[5] and $\mathrm{G} = W(p,q)$, while the antiweb inequalities (3.4.5) are facet-defining for $P_{STAB}(AW(p,q))$, if there is no $k \in \mathbb{N}$ with $p = k * q$.

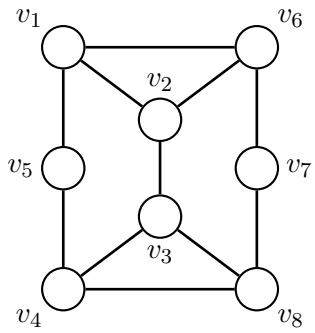


Figure 3.4.2: Facet-producing graph

Let us come back to graph G defined through Fig. 2.1.1. If we add the 5-cycle, the triangle and the 4-clique inequalities to P_{RSTAB}, it can be shown that it is equal to P_{STAB}. Unfortunately, the inequalities that have been discussed until now are not sufficient to describe P_{STAB} in general. Therefore, consider graph G of Fig. 3.4.2. We observe two triangles and two 5-cycles. Adding their corresponding inequalities to P_{RSTAB} leads to a polytope with 35 vertices[6]. Exactly one vertex is not integer, which reads

$$\left(\frac{1}{3}, \frac{1}{3}, \frac{1}{3}, \frac{1}{3}, \frac{2}{3}, \frac{1}{3}, \frac{2}{3}, \frac{1}{3}\right).$$

This implies that the added inequalities are not sufficient to describe $P_{STAB}(\mathrm{G})$. In order to calculate a description of the facets of G, we use PORTA [por]. The output can be seen on the next page. "DIM" stands for the number of variables and the vector after "VALID" lies inside the polytope described by the inequalities of "INEQUALITIES_SECTION", which shows that the polytope is not empty. Let us have a closer look at the inequalities. Constraints (1) to (8) are the non-negativity inequalities (3.1.1) and the inequalities (9) to (13) are some edge inequalities (3.1.2). (14) and (15) are inequalities corresponding to a maximal clique, (16) and (17) are 5-cycle inequalities. Inequality (18) does not belong to any inequality which have been studied until now. Nevertheless, it defines a facet.

[5] Two natural numbers are called relatively prime if their greatest common divisor is 1, or in formula: gcd(p,q)=1.
[6] This is not obvious, but they can be calculated for instance with PORTA.

```
DIM = 8

VALID
1 0 1 0 0 0 1 0

INEQUALITIES_SECTION
(  1) -x1                          <= 0
(  2)    -x2                       <= 0
(  3)       -x3                    <= 0
(  4)          -x4                 <= 0
(  5)             -x5              <= 0
(  6)                -x6           <= 0
(  7)                   -x7        <= 0
(  8)                      -x8     <= 0
(  9)                   +x7+x8     <= 1
( 10)                +x6+x7        <= 1
( 11)          +x4+x5              <= 1
( 12)    +x2+x3                    <= 1
( 13) +x1          +x5             <= 1
( 14)       +x3+x4          +x8    <= 1
( 15) +x1+x2          +x6          <= 1
( 16)    +x2+x3       +x6+x7+x8    <= 2
( 17) +x1+x2+x3+x4+x5              <= 2
( 18) +x1+x2+x3+x4+x5+x6+x7+x8     <= 3

END
```

rank inequality

Inequality (18) belongs to the large class of so-called **rank inequalities**. Let $\mathrm{G} = (V, E)$ be a graph and $W \subseteq V$, then these inequalities read

$$x(W) := \sum_{v_i \in W} x_i \leq \alpha(\mathrm{G}[W]). \tag{3.4.6}$$

From their construction, inequalities (3.4.6) are valid for $P_{STAB}(\mathrm{G})$. The edge, odd-cycle, clique, antihole, web and antiweb inequalities belong to the class of rank inequalities. Therefore, these inequalities are not facet-defining for $P_{STAB}(\mathrm{G})$ in general. The inequality of an odd-wheel with 5 or more nodes is no rank inequality, for instance.

Chapter 4

Branch & Cut Modules

In the last chapter we introduced the stable set polytope and some of its relaxations. Now, we want to use the polyhedral results to develop a Branch & Cut solver for the maximum stable set problem. We start with a preprocessing phase which exploits the special structure of the linear program $\mathcal{LP}$. Several generalizations for other LPs are discussed and examples are given which show that the structure of $\mathcal{LP}$ is lost in general after the addition of clique or odd-hole inequalities. In Section 2 we consider separation procedures. After a polynomial time separation algorithm for the odd-cycle inequalities is developed, we consider the clique inequalities. This is followed by a discussion of the edge projection which is a method to shrink a graph in order to call separation routines in the shrunk graph. We review the results from the literature and present several extensions of the theory as a sufficient criteria for facet-defining inequalities and a generalization of the edge projection to general inequalities. At the end of Section 4.2 we consider two types of general inequalities. The local cuts are adopted from the traveling salesman problem and the mod-k cuts are considered as a special case of the CHVÁTAL-GOMORY cuts. We will close this chapter with a famous branching strategy for the stable set problem.

4.1 Preprocessing

"Given a formulation, preprocessing refers to elementary operations that can be performed to improve or simplify the formulation by tightening bounds on variables, fixing values and so on. Preprocessing can be thought of as a phase between formulation and solution." ([NW88], p. 17)

The citation above includes the main ideas of a preprocessing phase. In the case of a Branch & Cut algorithm for the maximum stable set problem, we focus on eliminating or fixing of variables and on some structure properties of maximal stable sets. As mentioned above, preprocessing uses mostly fast algorithms, which need only linear time, or at most polynomial time. It is also typical that small problem instances can be solved immediately in the preprocessing phase. In the following, we will look at some special structures and results on how to improve a Branch & Cut algorithm via preprocessing.

4.1.1 Basic Preprocessing

There are some fundamental ideas for preprocessing for the maximum stable set problem. Let $\mathrm{G}_c = (V, E, c)$ be a node-weighted graph.

- One can delete all nodes of the graph with *negative* or *zero weight*:

  $$\forall c_i \text{ with } 1 \leq i \leq |V| \text{ and } c_i \leq 0 \Longrightarrow \alpha(\mathrm{G}) = \alpha(\widetilde{\mathrm{G}}) \text{ with } \widetilde{\mathrm{G}} := \mathrm{G} - v_i.$$

  This can be done in $O(|V|)$.

- If the graph is not connected one can solve the maximum stable set problem by solving the problem for each *connected component* separately:

  $$\exists \mathrm{G}_j \text{ for some } 1 < j < \infty \text{ with } \mathrm{G} = \cup_j \mathrm{G}_j \Longrightarrow \alpha(\mathrm{G}) = \sum_j \alpha(\mathrm{G}_j).$$

  The connected components can be identified for instance with the Depth-First-Search algorithm in linear time with respect to the size of the adjacency structure, cf. [Rei01]. After all maximum stable set problems are solved independently, they have to be merged to a maximum stable set for graph G. This can be done in linear time.

- As a special case of the last observation, all *isolated nodes* with positive weight are part of a (all) maximum stable set(s):

  $$\forall v_i \text{ with } 1 \leq i \leq |V| \text{ and } \delta(v_i) = 0 \wedge c_i > 0 \Longrightarrow \alpha(\mathrm{G}_c) = \alpha(\mathrm{G}_c - v_i) + c_i.$$

  This can be done in $O(|V|)$, if stored in an adjacency list.

- If the *weight* of a node is *greater* or equal than the weight of the sum of all its *neighbors*, it can be fixed:

  $$\forall v_i \text{ with } 1 \leq i \leq |V| \text{ and } c_i \geq \sum_{v_j \in \Gamma(v_i)} c_j \Longrightarrow \alpha(\mathrm{G}_c) = \alpha(\mathrm{G}_c - (\{v_i\} \cup \Gamma(v_i)) + c_i.$$

Due to this observations we assume from now on that the graph G is connected and has only positive weights.

4.1.2 Fixing of Cliques

According to our definition in Chapter 1.1, we call a node set a clique if its induced subgraph is complete. Hence, we consider also the degenerated cases when the clique consists of one or two nodes. Imagine there is a clique Q and a node $v \in Q$, whose neighborhood is a subset of the clique. If the weight of this node is greater or equal to the weight of all other nodes of the clique, this node can be added to a stable set and the whole clique can be deleted from the graph. This is summarized in the following lemma.

Lemma 4.1.1. *Let $\mathrm{G}_c = (V, E, c)$ be a node-weighted graph. Exists a node $v_i \in V$ and a clique Q of G with the property $\Gamma(v_i) \subseteq Q$ and $c_i \geq c_j$ for all $v_j \in Q$, then there is a maximum stable set of G which contains node v_i.*

The proof is a by-product of Prop. 4.1.4 and is discussed there.

4.1.3 Fixing of Nodes

In Chapter 3, we became familiar with the stable set polytope relaxation. This polytope is defined by the trivial and edge inequalities. The resulting structure was considered in Corollary 3.1.2. It states that the vertices are all $(0, \frac{1}{2}, 1)$-valued. This will enable us to prove

Theorem 4.1.2. *[NT74] Suppose x^* is an optimal $(0, \frac{1}{2}, 1)$-valued solution of $\mathcal{LP}$. There is a maximum stable set in* G *that contains $\mathcal{S} := \{v_j \in V \mid x_j^* = 1\}$.*

Theorem 4.1.2 is the main result of this paragraph. It expresses that all nodes which have value 1 in an optimum solution of $\mathcal{LP}$ can be fixed without changing the optimum value. This can be used in order to solve a maximum stable set problem. Whenever an optimal solution of $\mathcal{LP}$ has the value 1, the corresponding node can be fixed. After that, this node and all its neighbors can be deleted from the graph. This leads to a reduction of the order, which might help to cope with the complexity.

In the following, we look more closely at the assumptions of Theorem 4.1.2 and discuss possiblities to generalize the result. First we have to examine a

Proof of Theorem 4.1.2

This theorem was first noticed and proven by NEMHAUSER and TROTTER [NT75] and is the template of the following proof. In order to prove Theorem 4.1.2 we consider two propositions. We start with several notations.

Let $G_c = (V, E, c)$ be a node-weighted graph and let $V_1, V_2 \subseteq V$. Define the **incidence** of V_1 in V_2 by $V_2(V_1) := V_2 \cap \Gamma(V_1)$. Let $\mathcal{S}$ be a stable set on G and denote $\overline{S} := V \setminus S$ the complement of $\mathcal{S}$ in V. A node set $I \subseteq \overline{\mathcal{S}}$ is called an **augmenting subset** to $\mathcal{S}$ if I is a stable set on G with the property that $c\big(\mathcal{S}(I)\big) < c(I)$.

incidence

augmenting subset

Proposition 4.1.3. *[NT74] Let $\mathcal{S}$ be a stable set in graph G_c. Then $\mathcal{S}$ is not a maximum stable set in G_c, iff some $I \subseteq \overline{\mathcal{S}}$ is augmenting to $\mathcal{S}$.*

Proof. "⇒" If there is an $I \subseteq \overline{\mathcal{S}}$ augmenting to $\mathcal{S}$, then $(\mathcal{S} \cup I) \setminus \mathcal{S}(I)$ defines a stable set on G. The additivity of the weighting function and the property of an augmenting subset implies

$$c\big((\mathcal{S} \cup I) \setminus \mathcal{S}(I)\big) = c(\mathcal{S}) + c(I) - c\big(\mathcal{S}(I)\big) > c(\mathcal{S})$$

which shows that $\mathcal{S}$ is no maximum stable set.

"⇐" If $\mathcal{S}$ is not a maximum stable set on G_c, then there is a stable set $\widetilde{\mathcal{S}}$ with the property $c(\widetilde{\mathcal{S}}) > c(\mathcal{S})$. Define $I := \widetilde{\mathcal{S}} \setminus \mathcal{S}$. Obviously, $I \subseteq \overline{\mathcal{S}}$ and I is a stable set. I is augmenting to $\mathcal{S}$ because of

$$\begin{aligned} c(I) &= c(\widetilde{\mathcal{S}} \setminus \mathcal{S}) \\ &= c(\widetilde{\mathcal{S}}) - c(\mathcal{S}) + c(\mathcal{S} \setminus \widetilde{\mathcal{S}}) \\ &> c(\mathcal{S} \setminus \widetilde{\mathcal{S}}) \\ &\overset{(1)}{\geq} c(S(I)). \end{aligned}$$

Inequality (1) holds because $\widetilde{\mathcal{S}}$ is a stable set with the property $(\widetilde{\mathcal{S}} \cap \mathcal{S}) \cap \Gamma(I) = \emptyset$ which shows that $\mathcal{S}(I) \subseteq S \setminus \widetilde{\mathcal{S}}$. Since the weighting c is positive we have closed the proof. □

For the next proposition we define $\widehat{\mathrm{G}}_{\mathcal{S}}$ as induced subgraph of G by $\mathcal{S} \cup \Gamma(\mathcal{S})$.

Proposition 4.1.4. *[NT74] If $\mathcal{S}$ is a maximum stable set in graph $\widehat{\mathrm{G}}_{\mathcal{S}}$, then there is a maximum stable set $\mathcal{S}^*$ in G with $\mathcal{S} \subseteq \mathcal{S}^*$.*

Proof. Let $\mathcal{S}$ be a maximum stable set in $\widehat{\mathrm{G}}_{\mathcal{S}}$. We construct a stable set $\mathcal{S}^*$ for G and show that there is no augmenting subset. Therefore, consider the following notations:

- $\widetilde{\mathrm{G}} := \mathrm{G}[\widetilde{V}]$ with $\widetilde{V} := V \setminus (\Gamma(\mathcal{S}) \cup \mathcal{S})$,
- $\widetilde{\mathcal{S}}$ denotes a maximum stable set in $\widetilde{\mathrm{G}}$,
- $\mathcal{S}^* := \mathcal{S} \cup \widetilde{\mathcal{S}}$, which implies that $\mathcal{S}^*$ is a stable set of G,
- $\widetilde{\mathcal{S}}^c := \widetilde{V} \setminus \widetilde{\mathcal{S}}$ denotes the complement of $\mathcal{S}$ in $\widetilde{V}$,
- $\overline{\mathcal{S}}^* := \Gamma(\mathcal{S}) \cup \widetilde{\mathcal{S}}^c$ is the complement of $\mathcal{S}^*$ in V, which implies $\overline{\mathcal{S}}^* \cap \mathcal{S}^* = \emptyset$,
- $I \subseteq \overline{\mathcal{S}}^*$ is a stable set for G,
- $I_1 := \Gamma(\mathcal{S}) \cap I$ and $I_2 := \widetilde{\mathcal{S}}^c \cap I$.

Since $\widetilde{\mathcal{S}}$ is maximum stable set in $\widetilde{\mathrm{G}}$, $\widetilde{\mathcal{S}} \cup \Gamma(\widetilde{\mathcal{S}}) = \widetilde{V}$ and $I_2 \subseteq \Gamma(\widetilde{\mathcal{S}})$ we have

$$c(I_2) \leq c\big(\widetilde{\mathcal{S}}(I_2)\big).$$

As $\Gamma(\mathcal{S}) \cap I_2 = \emptyset$ we know that $\Gamma(I_2) \cap \mathcal{S} = \emptyset$, which implies

$$c\big(\widetilde{\mathcal{S}}(I_2)\big) = c\big(\mathcal{S}^*(I_2)\big).$$

Analogue, optimality of $\mathcal{S}$ in graph $\widehat{\mathrm{G}}_{\mathcal{S}}$ and $\Gamma(I_2) \cap \mathcal{S} = \emptyset$ implies

$$c(I_1) \leq c\big(\mathcal{S}(I_1)\big) \leq c\big(S^*(I_1) \setminus \mathcal{S}^*(I_2)\big).$$

Combining the relations above and using that $I = I_1 \cup I_2$, $\Gamma(I_1 \cup I_2) = \Gamma(I_1) \cup \Gamma(I_2)$ and $c > 0$ leads to

$$\begin{aligned} c(I) &= c(I_1) + c(I_2) \\ &\leq c\big(\mathcal{S}^*(I_1) \setminus \mathcal{S}^*(I_2)\big) + c\big(\mathcal{S}^*(I_2)\big) \\ &= c\big(\mathcal{S}^*(I)\big). \end{aligned}$$

Thus, no subset of $\overline{\mathcal{S}}^*$ augments $\mathcal{S}^*$ and, by Prop. 4.1.3, $\mathcal{S}^*$ is a maximum stable set in G. □

One can use Proposition 4.1.4 to proof Lemma 4.1.1. Node v_i is then a maximum stable set in $\widehat{\mathrm{G}}_{\{v_i\}}$. Recognize that it is not sufficient for Lemma 4.1.1 that $c(v_i) > c(v_j)$ for all $v_j \in \Gamma(v_i)$. Instead, we have seen that $c(v_i)$ must be greater than the sum of the weights of its neighbors. This can also be proven using Proposition 4.1.4. We want to point out that Proposition 4.1.4 is not true for all maximum stable sets $\mathcal{S}^*$ in G in general. Rather, it says that there is at least one maximum stable set with that property.

Now, we are prepared to proof the main result.

Proof of Theorem 4.1.2. Suppose $\mathcal{S} \neq \emptyset$. With proposition 4.1.4 it is sufficient to show that $\mathcal{S}$ is a maximum stable set in subgraph $\widehat{\mathrm{G}}_{\mathcal{S}}$ of G. If $\mathcal{S}$ is optimal in $\widehat{\mathrm{G}}_{\mathcal{S}}$, there cannot be an augmenting subset to $\mathcal{S}$ according to Proposition 4.1.3. We use a proof by contradiction.

Assume there is an augmenting subset $I \subseteq \Gamma(\mathcal{S})$ to $\mathcal{S}$ in $\widehat{\mathrm{G}}_{\mathcal{S}}$. By definition $c\big(\mathcal{S}(I)\big) < c(I)$. Let x^* be an optimal solution of $\mathcal{LP}$ and define vector x as

$$x_j \quad = \quad \begin{cases} \frac{1}{2}, & \text{if } v_j \in I \cup \mathcal{S}(I) \\ x^*_j, & \text{otherwise .} \end{cases}$$

According to the assumption and Corollary 3.1.2 vector x^* is $(0, \frac{1}{2}, 1)$-valued. Since $x_j = 1$ implies $v_j \in \mathcal{S} \setminus \mathcal{S}(I)$, x is also a feasible solution to $\mathcal{LP}$. Because x and x^* differ only in set $I \cup \mathcal{S}(I)$ and $I \cap \mathcal{S}(I) = \emptyset$, we have

$$\begin{aligned} \sum_{v_j \in V} c_j x_j - \sum_{v_j \in V} c_j x^*_j &= \frac{1}{2} c\big(I \cup \mathcal{S}(I)\big) - c\big(I \cup \mathcal{S}(I)\big) \\ &= \frac{1}{2} c\big(I \cup \mathcal{S}(I)\big) - c\big(\mathcal{S}(I)\big) \\ &= \frac{1}{2} c(I) + \frac{1}{2} c\big(\mathcal{S}(I)\big) - c\big(\mathcal{S}(I)\big) \\ &= \frac{1}{2} \Big(c(I) - c\big(\mathcal{S}(I)\big) \Big) \\ &> 0. \end{aligned}$$

This contradicts the optimality of x^* which shows that $\mathcal{S}$ is a maximum stable set in $\widehat{\mathrm{G}}_{\mathcal{S}}$. □

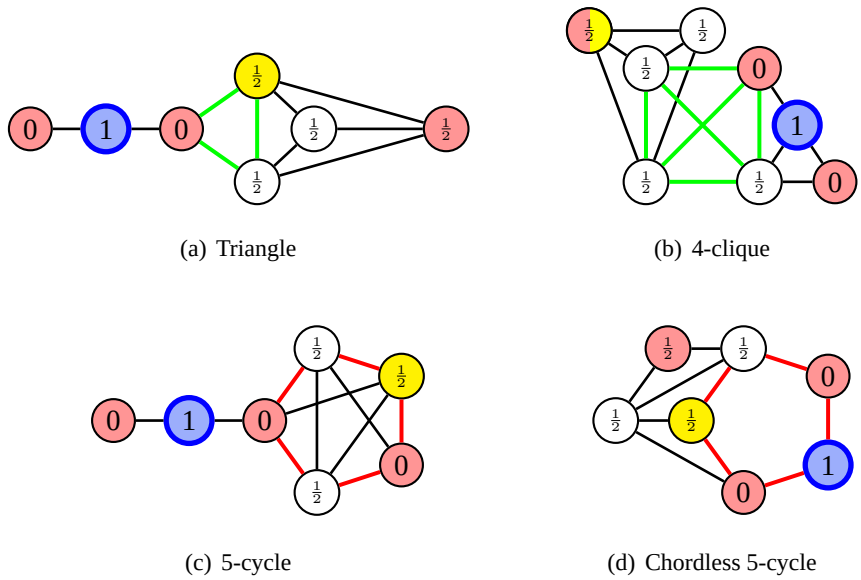
Generalization

Consider again Theorem 4.1.2 and recognize that it has basically one assumption. It requires that the linear program $\mathcal{LP}$ consists only of the edge and trivial inequalities. The second condition that the optimum of $\mathcal{LP}$ is $(0, \frac{1}{2}, 1)$-valued is implied by the first assumption which shows Corollary 3.1.2.

In order to solve a maximum stable set problem via Branch & Cut, one adds additional inequalities to $\mathcal{LP}$. In the last chapter we have seen that the solution of $\mathcal{LP}$ loses the property of being $(0, \frac{1}{2}, 1)$-valued in general, if inequalities are added. Therefore the natural question arises if there is any possibility to strengthen Theorem 4.1.2 that it can be generalized to accept solutions of the linear program $\mathcal{LP}$ with additional

- odd-cycle inequalities,
- clique inequalities,
- odd-cycle and clique inequalities together,
- facet-defining inequalities,
- valid (general) inequalities.

Considering the proof of Theorem 4.1.2 gives no hint how to modify it accepting one of the above changes. The proof of NEMHAUSER and TROTTER strongly requires the special structure of the $\mathcal{LP}$ polytope. If a solution vector is not $(0, \frac{1}{2}, 1)$-valued, the construction of x in the above proof will fail. But even if we assume a $(0, \frac{1}{2}, 1)$-valued solution, we will see that the answers to all 5 questions above are negative. This is demonstrated by Fig. 4.1.1.



(a) Triangle (b) 4-clique

(c) 5-cycle (d) Chordless 5-cycle

Figure 4.1.1: Graphs with $(0, \frac{1}{2}, 1)$-valued LP solution. The green / red lines represent a clique / odd-cycle inequality added to $\mathcal{LP}$. The number in a node represents the solution value of $\mathcal{LP}$ after adding one odd-cycle or clique inequality. The blue colored nodes are the fixed nodes. The yellow colored nodes build a maximum stable set in the remaining graph after fixing a node while the red colored nodes build a maximum stable set for each whole graph.

The size of a maximum stable set after the fixing of a node with value 1 differs in all four graphs of Fig. 4.1.1 from the size of a maximum stable set of the whole graph. In each graph the number of yellow and blue nodes is two, which is smaller than the number of red nodes. Graph 4.1.1 (a) provides an example for a 3-clique which is also a 3-cycle. This example gives the answer to all 5 question above. The clique inequality highlighted with the green lines is facet-defining, because it is maximal—in this case even maximum. A clique which is not also an odd cycle can be seen in graph 4.1.1 (b). Graph 4.1.1 (c) and (d) provide examples for cases with the addition of an odd cycle and a chordless odd cycle, respectively. All this graphs show that the theorem of NEMHAUSER and TROTTER cannot be generalized in that way. One should notice that the solution vectors of the four examples correspond to vertices of the polytope. Hence, they are candidates for solutions of the simplex method.

One can ask why the proof of Theorem 4.1.2 does not work when adding an inequality and assuming that the solution is $(0, \frac{1}{2}, 1)$-valued. The problem is that the constructed vector x in the proof is generally no longer feasible since it does not respect the added inequality.

4.2 Separation

In Chapter 3 we described some classes of inequalities valid for the stable set polytope. The number of valid inequalities may grow exponentially and therefore it is not helpful to add all of them to a linear program relaxation. It is not necessary to describe the complete polytope to solve a particular problem. One would add only the "important" inequalities. This is done sequentially by solving an LP-relaxation and adding violated inequalities to the formulation. Finding these violated inequalities, is the separation problem. We start with the separation of the odd-cycle inequality, which is based on [CC97, GS86, GP81].

4.2.1 Odd-Cycle Inequalities

In order to separate the odd-cycle inequalities (3.2), one has to find an odd cycle violating the corresponding inequality or one has to proof that such cycles doe not exist. With other words, we have to find a minimum-weight odd cycle in a graph, with appropriate weighting function. If this cycle satisfies the corresponding inequality (3.2), it is proven that all odd-cycle inequalities are satisfied. Otherwise one has found a maximal violated odd-cycle inequality. Therefore, we first consider an algorithm which computes a minimum-weight cycle of an arbitrary graph with edge weighting. The second step is to define edge weights depending on a current LP solution. This will solve the separation problem for the odd-cycle inequalities. It is not quite clear, who has brought up the idea of the following algorithm, but it was published in 1986 by GERARDS and SCHRIJVER, cf. [GS86].

Algorithm 4.2.1 Minimum odd cycle in a graph

Input: Edge-weighted graph $\mathrm{G}_c = (V, E, c)$, weighting c is non-negative
Output: Minimum-weight odd cycle $\mathcal{C}$ of G_c with weight h

// Construct auxiliary bipartite graph $H := (V_H, E_H)$
$V_H := \{v^+, v^- \mid v \in V\}$ // Duplicate all nodes of graph G
$E_H := \{u^+v^-, v^+u^- \mid uv \in E\}$ // $u^+v^-, u^-v^+ \in E_H \Leftrightarrow u, v \in E$
$\forall\, uv \in E : c_H(u^+v^-) = c_H(v^+u^-) := c(uv)$ // Define the weights of H

// Initialize odd cycle $\mathcal{C}$ and its weight h
$\mathcal{C} := \emptyset$ and $h := \infty$

// Construct for each node of G_c a minimum-weight odd cycle
for all nodes $u \in V$ **do**
  Compute a minimum-weight path $P_H := (u^+, u^+u_1^-, u_1^-, u_1^-u_2^+, \ldots, u_k^+, u_k^+u^-, u^-)$ from node u^+ to node u^- in graph H with weight c_{P_H}; $0 \leq k \leq 2|V| - 2$, k even
  Receive the closed walk $W_{\mathrm{G}} := (u, uu_1, u_1, u_1u_2, u_2, \ldots, u_k, u_ku, u)$ in graph G
  Construct odd-cycle $C = (v_0, v_1, v_2, \ldots, v_m, v_0)$ with weight c_G from walk W_{G}; $0 < m \leq |V| - 1$, m even
  if $c_G < h$ **then**
    $\mathcal{C} := C$ and $h := c_G$ // Update minimum-weight odd cycle
  end if
end for

return Odd cycle $\mathcal{C}$ and its weight h.

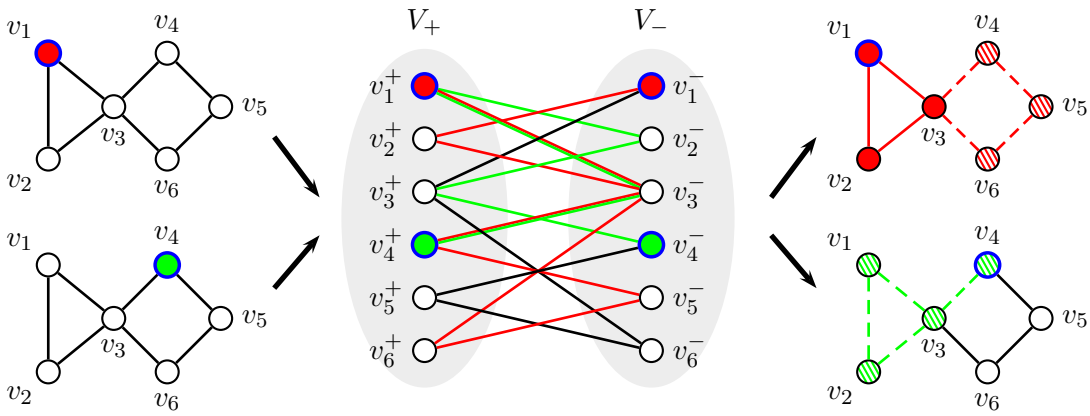
Consider now Algorithm 4.2.1. In the first three steps an auxiliary graph $H = (V_H, E_H)$ is constructed. Node set V is duplicated and the two copies are called V^+ and V^-. An edge u^+v^- or u^-v^+ is in E_H, iff edge $uv \in E$. Since there is no edge between two nodes of V^+ and the same for V^-, H is bipartite with $V_H = V_+ \dot\cup V_-$. The weights are copies of the weights of graph G. In step 6 a minimum-weight path from node u^+ to node v^- is computed. Since nodes u^+ and u^- are in two different sets of the bipartition, an odd number of edges are in each closed path. The corresponding odd walk in G is constructed in step 7 by deleting the indices $+$ and $-$, respectively. Note that this walk can have edge and node repetitions. In order to get an odd cycle in G, one has to delete the double nodes and edges. One idea can be to start with node u and mark all visited nodes and edges along the closed walk. If a marked node is visited again, all nodes and edges along this closed path can be removed from the walk. In the case of an edge repetition, the last visited node and the repeated edge have to be removed from the walk. After this is done, all node and edge repetitions have been eliminated. The resulting closed walk can have odd or even length or may be empty. If it is odd, one has found a minimum-weight odd cycle in G_c, since c is non-negative. In the other two cases, one of the removed closed paths has odd length. Each of these odd cycles has smaller or equal weight than any odd cycle containing u. Hence, we store one of them in set C. Since all edge weights are non-negative the remove of nodes from the path does not destroy the exactness of this algorithm. As each node and edge in W_G is marked only as its frequency, the procedure described above can be done in linear time in the length of W_G. Therefore, a minimum-weight path from u^+ to u^- in H, with respect to weighting c_H, corresponds to an odd cycle in G_c which has smaller or equal weight than any odd cycle containing node u. As in step 5 such an odd cycle is computed for all nodes u of graph G, algorithm 4.2.1 computes a minimum-weight odd cycle in G.

We recognize that the computational complexity is dominated by the for-loop of step 5 and the construction of a shortest path in step 6. Since all weights of auxiliary graph H are positive, one can use the algorithm of DIJKSTRA to calculate the shortest path, which has running time $O(E \log(V))$. We summarize all this observations in

Proposition 4.2.1. *A minimum-weight odd cycle in a graph* G *can be computed with Algorithm 4.2.1 in* $O(VE \log(V))$.

For an example, we consider Fig. 4.2.1. We begin with the graph on the top of sub figure (a). In this case, we say that all edges have weight zero. In sub figure (b) the auxiliary graph constructed by Algorithm 4.2.1 is shown. All edge weights are 0 by construction. We start to compute a minimum-weight odd cycle containing node v_1. A resulting shortest path from node v_1^+ to node v_1^- is represented by the red edges. The translation of this shortest path to a closed walk can be seen in sub figure (c) on the top graph. We recognize that node v_3 is contained twice in the walk. Using the described method leads to a remove of nodes v_4, v_5 and v_6 from the walk. In this case the remaining closed walk is an odd cycle containing node v_1. Computation of an odd cycle including v_4 is shown in the graph below. The corresponding path is labeled green. Again there is a node repetition and we remove nodes v_1 and v_2. After that, we recognize that edge v_3v_4 is contained twice in the walk. Elimination of it shows that there is no odd cycle containing v_4 and the resulting walk gets empty. But as a by-product we receive an odd cycle containing nodes v_1, v_2 and v_3.

Next, we want to use Prop. 4.2.1 to show that the separation problem for the odd-cycle inequalities can be solved in polynomial time. We therefore have to define an edge weighting. Let $x^* \in [0,1]^n$ be a vector satisfying the non-negativity (3.1.1) and edge inequalities (3.1.2),



(a) Two identical graphs (b) Auxiliary graph (c) Resulting odd cycles

Figure 4.2.1: Odd-cycle separation

for instance a solution of $\mathcal{LP}$. Define an edge weighting of graph G depending on x^* as

$$\begin{aligned} c\colon E &\longrightarrow [0,1] \\ c(v_iv_j) &:= \frac{1-x_i^*-x_j^*}{2}. \end{aligned}$$

Suppose that C is an odd cycle in G. Then the weight of C, with respect to c, is

$$\begin{aligned} c(C) &:= \sum_{v_iv_j\in E(C)} c(x_ix_j) \\ &= \sum_{v_iv_j\in E(C)} \frac{1-x_i^*-x_j^*}{2} \\ &= \tfrac{|C|}{2} - \tfrac{1}{2} \sum_{v_iv_j\in E(C)} (x_i^*+x_j^*) \\ &= \tfrac{|C|}{2} - \sum_{v_i\in V(C)} x_i^*. \end{aligned}$$

An odd-cycle inequality in G is violated by vector x^*, iff

$$\begin{aligned} &\exists C \text{ with } V(C)\subseteq V, C \text{ odd cycle and } \sum_{v_i\in V(C)} x_i > \tfrac{|C|-1}{2} \\ \iff\ &\exists C \text{ with } V(C)\subseteq V, C \text{ odd cycle and } \sum_{v_iv_j\in E(C)} c(v_iv_j) < \frac{1}{2}. \end{aligned}$$

Therefore a most violated odd-cycle inequality corresponds to an odd cycle in G, having minimum weight with respect to c. This leads to Algorithm 4.2.2. In step 1 the edge weights for graph G are calculated and in step 2 a minimum-weight odd cycle is computed. The check in step 3 provides the information whether all odd-cycle inequalities are satisfied or not. As a computer has double precision only, it is quite important to check the violation with a small tolerance ε. However, for the theoretical point of view, ε has to be fixed to zero to get an exact separation routine. We get the following theorem.

Theorem 4.2.2. *The separation problem for the class of odd-cycle inequalities can be solved in polynomial time.*

After all the observation, we only have to show that Algorithm 4.2.2 has a polynomial running time in order to proof Theorem 4.2.2. We observe that the running time is dominated by the computation of the minimum-weight odd cycle. As the constructed weights are all non-negative, we can use Algorithm 4.2.1 and get the running time $O(VE\log(V))$. Note that it is quite important that the trivial and the edge inequalities are met by vector x^*. Otherwise the weights c would get negative, which has the consequence that DIJKSTRA's algorithm does not work anymore. One could use for instance the algorithm of BELLMANN-FORD, which deals with negative weights. Unfortunately, negative cycle could occur which makes the problem unsolvable. But actually, this is no problem as the number of edge and trivial inequalities is linear in E. Therefore one would separate them before executing Algorithm 4.2.2.

Theorem 4.2.2 is quite remarkable as the number of odd cycles in a graph can be exponential. For instance, K_n has

$$\sum_{\substack{k=3 \\ k\equiv 1 \pmod 2}}^{n} \binom{n}{k}$$

odd cycles. The clue behind is that there is no need to check all odd cycles. Computing a very small number of them is enough.

Algorithm 4.2.2 Odd-cycle separation

Input: Graph $\mathrm{G} = (V, E)$, $x \in [0,1]^{|V|}$ satisfies edge inequalities, $\varepsilon_{minViol}$
Output: Maximum violated odd-cycle inequality

// Calculate weight c.
1: $c(u_iu_j) := \frac{1-x_i^*-x_j^*}{2}$ for all $u_iu_j \in E$
// Compute minimum-weight odd cycle in G
2: Use algorithm 4.2.1 to compute $\mathcal{C}$ and h from G_c
3: **if** $h > \frac{1}{2} - \varepsilon_{minViol}$ **then**
4: **return** There is no violated odd-cycle inequality in G.
5: **else**
6: **return** $\sum_{v_i \in \mathcal{C}} x_i \leq \frac{|\mathcal{C}|-1}{2}$ is a maximum-violated odd-cycle inequality in graph G with respect to vector x^*.
7: **end if**

The first known algorithm for the separation of the class of odd-cycle inequalities goes back to GRÖTSCHEL and PULLEYBLANK in 1981, cf. [GP81]. This algorithm also makes use of an auxiliary graph. In this case it is no longer bipartite and the minimum-weight odd cycle is computed via a perfect matching. Unfortunately, it is quite time-consuming to calculate a perfect matching which can be done for instance in $O(|E|^3)$. This algorithm will then have a running time of $O(|E|^4)$. This shows the dominance of Algorithm 4.2.2. For the details, we refer to [GP81].

4.2.2 Clique Inequalities

Finding a maximum clique is $\mathcal{NP}$-$hard$, as it is equivalent to finding a maximum stable set in the complement graph; compare Chap. 2.2. Computing an arbitrary maximal clique as well as an arbitrary maximal stable set can be done in linear time. The separation problem for the clique inequalities asks to find a violated clique inequality in a particular graph G with a given LP solution or to state that all clique inequalities are satisfied. This is equivalent to finding a maximum clique in G with the LP solution as node-weighting c. Clearly, a maximum clique in G_c solves the separation problem and to certify that no clique inequality in G is violated one has to consider a maximum clique in G_c. The consequence is that we have to expect that any exact separation will be too time-consuming for a fast Branch & Cut algorithm. However, computational tests show that the clique inequalities are very important for polyhedral approaches to the stable set problem, cf. [RS01]. One idea could be to fix the size of the cliques to be separated, as then the problem becomes polynomially solvable. Another observation is that it is enough to consider maximal cliques and their corresponding clique inequality dominates clique inequalities with smaller cliques, see Chap. 3.3. We will come back to this in Chapter 5.2.2 where we discuss some heuristics.

4.2.3 Rank Inequalities

In the following, we introduce the method due to MANNINO and SASSANO to find violated rank inequalities, cf. [MS96b]. The appealing idea is to reduce the size of the graph and to make it denser at the same time. This process, which is called **edge projection**, should fasten the separation routines for a class of rank inequalities for the resulting graph, for instance the clique separation. Once a violated inequality for that smaller graph has been found, it is adjusted to the whole graph. This process is called **anti-projection**. After reviewing the latest results of ROSSI and SMRIGLIO, [RS01], we will discuss some new polyhedral aspects.

edge projection

anti-projection

In the following let $\mathrm{G} = (V, E)$ be a graph. We want to start with an exact definition of the projection of a graph. For a given edge $e = uv$ the new graph is constructed in the following way. Both endnodes u, v of edge e and their **common neighborhood** $\Gamma_{uv} := \Gamma(u) \cap \Gamma(v)$ are deleted from the graph. This has the effect that the edges in the set

common neighborhood

$$E_{uv} := \{v_i v_j \in E \mid v_i \text{ or } v_j \in \Gamma_{uv} \cup \{u, v\}\}$$

are removed, too. In addition, new edges, the so-called **false edges**, are added. In the new graph, every node which is a neighbor of node u but not of v is adjacent to all nodes which are neighbors of v but not of u. The corresponding edge set is defined as

false edge

$$\overline{E}_{uv} := \{v_i v_j \mid v_i v_j \in V \setminus (\{u, v\} \cup \Gamma_{uv}) \text{ and } \Gamma(\{v_i, v_j\}) \supseteq \{u, v\}\}.$$

The elements e of $\overline{E}_{uv}$ are called false edges, if $e \notin E$. We get the formal

Definition 4.2.3. *The graph* $\mathrm{G}|e := (V|e, E|e)$ *with* $V|e := V \setminus (\Gamma_{uv} \cup \{u, v\})$ *and edge set* $E|e := E \setminus E_{uv} \cup \overline{E}_{uv}$ *is called the* ***projection*** *of* e *in* G.

projection

In order to have a chance to make a rank inequality of $\mathrm{G}|e$ valid for the polytope of the original graph G, we only consider edges e with a special property. An edge $e = uv \in E$ is

projectable
strongly projectable

called **projectable** in G, if there is a maximum stable set $\mathcal{S}^*$ in G such that $\mathcal{S}^* \cap \{u, v\} \neq \emptyset$. It is called **strongly projectable** in G if it is projectable in every induced subgraph of G containing both u and v. These definitions indicate the following

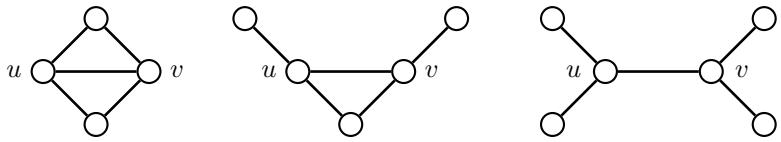
Lemma 4.2.4. *[MS96b] Let $e = uv$ be a projectable edge in G. Then $\alpha(\mathrm{G}) = \alpha(\mathrm{G}|e) + 1$.*

This implies that a valid rank inequality for the projection $\mathrm{G}|e$ can be anti-projected by adding the deleted nodes of the projection step. This is stated in the following theorem.

Theorem 4.2.5. *[RS01] Let $e = uv$ be a projectable edge in G and $W \subseteq V|e$. If $x(W) \leq l$ is a valid rank inequality for $P_{STAB}(\mathrm{G}|e)$, then $x(W) + x(\Gamma_{uv}) + x_u + x_v \leq l + 1$ is valid for $P_{STAB}(\mathrm{G})$.*

The next question is how the projectable edges can be characterized. Therefore we consider first the case of strongly projectable edges.

Theorem 4.2.6. *[RS01] An edge $e = uv \in E$ is strongly projectable in G, iff it is not the central edge of an induced subgraph isomorphic to a diamond, Fig. 4.2.2 (a), a bull, Fig. 4.2.2 (b), or a double fork, Fig. 4.2.2 (c).*



(a) Diamond (b) Bull (c) Double Fork

Figure 4.2.2: Diamond, Bull and Double Fork

Theorem 4.2.6 indicates the following necessary condition for a strongly projectable edge.

Corollary 4.2.7. *If edge $uv \in E$ is strongly projectable in G, then the common neighborhood Γ_{uv} is a clique.*

Proof. Let $uv \in E$ be a strongly projectable edge and Γ_{uv} be no clique. Then there are two nodes v_i, v_j in set Γ_{uv} which are not adjacent. This implies that the graph indicated by the nodes v_i, v_j, u and v is a diamond with central edge uv which is a contradiction to Theorem 4.2.6. □

According to its definition, every strongly projectable edge is projectable. This implies

Theorem 4.2.8. *[MS96b] An edge $e = uv \in E$ is projectable in G, if it is not the central edge of an induced subgraph isomorphic to a diamond or a bull or a double fork.*

Note that in this case we only have one direction. In fact, it is $\mathcal{NP}$-$hard$ to check whether an edge is projectable or not, cf. [MS96b]. In contrast, Theorem 4.2.6 provides a method to test strong projectability in polynomial time. For practical use however, it turns out that the number of strongly projectable edges for the relevant graphs is quite small. A check of the DIMACS Challenge benchmark graphs, see for this graphs also Chap. 6, shows that they do

not contain any strongly projectable edge[1]. The MANN graphs form an exception according to there special structure arising from the transformation of the set covering problem. In these 4 graphs half of the edges are strongly projectable. To use the projection nevertheless, we help us with a trick. The following corollary will enable us to make any edge uv strongly projectable in a smaller subgraph. The idea is to remove some edges which destroys the strong projectablity. Now consider

Corollary 4.2.9. *[RS01] Let $e = uv \in E$. If $\Gamma(u) \setminus \{v\}$ is a clique, then uv is strongly projectable.*

To make an edge uv strongly projectable, one can select a clique Q from set $\Gamma(u) \setminus \{v\}$ and delete all edges uw where w is in set $\Gamma(u) \setminus (\{v\} \cup Q)$—the same could be done for node v. The drawback is that the deletion of edges changes the structure of the resulting graph and decreases the density which negatively effects the projection. To reduce this drawback, one idea could be to compute a large clique. Unfortunately, the computation of a maximum clique in an arbitrary graph is $\mathcal{NP}$-$hard$ and one has to use a heuristic.

Now consider an example. In Fig. 4.2.3 (a) we select the edge v_3v_5 as a candidate for the projection. As this edge is not the central edge of an induced subgraph isomorphic to a diamond, bull or double fork, it is strongly projectable. We recognize that $\Gamma(v_3) \setminus \{v_5\} = \{v_2, v_4\}$ is no clique but the criteria of Cor. 4.2.9 holds for node v_5. The projection of v_3v_5 removes nodes v_3, v_4 and v_5 and adds the false edge v_2v_6. This can be seen in Fig. 4.2.3 (b). A separation routine could find the clique inequality $x_1 + x_2 + x_6 \leq 1$. Note that this inequality is not valid for the whole graph G. The anti-projection adds the deleted nodes to the inequality and increases the right hand side by value one. We obtain inequality $\sum_{i=1}^{6} x_i \leq 2$ which is facet-defining for $P_{STAB}(\mathrm{G})$. Removing of edge v_4v_6 changes the problem completely. Consider therefore Fig. 4.2.3 (c). The computed inequality is no longer valid as the edge v_3v_5 is not strongly projectable any more. It is the central edge of a bull. We use the idea indicated by Cor. 4.2.9 and compute a (maximum) clique Q. In this case we choose $Q = \{v_2\}$. Edge v_3v_4 is removed and after the projection we obtain the graph of Fig. 4.2.3 (d). Using again the clique inequality indicated by v_1, v_2 and v_6 leads to inequality $x_1 + x_2 + x_3 + x_5 + x_6 \leq 2$ which is now valid for $P_{STAB}(\mathrm{G})$.

In both examples of Fig. 4.2.3 the rank inequality valid for $P_{STAB}(\mathrm{G}|e)$ cannot be trivially lifted, because it is not valid for $P_{STAB}(\mathrm{G})$ without anti-projection. The reason is that the graph induced by the inequality contains a false edge. Clearly, a rank inequality (β, b_0) in the graph $\mathrm{G}|e$, whose induced subgraph does not contain any false edge, is also valid for $P_{STAB}(\mathrm{G})$ without anti-projection. Observe that the anti-projected inequality is dominated by (β, b_0) and $x(\Gamma_{uv}) \leq 1$, according to Cor. 4.2.7. Hence, it is better not to anti-project the inequality (β, b_0).

Let us now consider the case that we have found a facet-defining inequality in $\mathrm{G}|e$, for instance a maximal clique. This is illustrated in Fig. 4.2.4 (a) and (b). The inequality after

[1]Tested graphs: brock200_1, brock200_2, brock200_3, brock200_4, brock400_1, brock400_2, brock400_3, brock400_4, brock800_1, brock800_2, brock800_3, brock800_2; c-fat200-1, c-fat200-2, c-fat200-5, c-fat500-1, c-fat500-2, c-fat500-5, c-fat500-10; hamming6_2, hamming6_4, hamming8_2, hamming8_4, hamming10_2, hamming10_4; johnson8-2-4, johnson8-4-4, johnson16-2-4, johnson32-2-4; keller4, keller5; p-hat300-1, p-hat300-2, p-hat300-3, p-hat500-1, p_hat500-2, p_hat500-3, p_hat700-1, p_hat700-2, p_hat700-3, p_hat1000-1, p_hat1000-2, p_hat1000-3, p_hat1500-1, p_hat1500-2, p_hat1500-3; san200_0.7_1, san200_0.7_2, san200_0.9_1, san200_0.9_2, san200_0.9_3, san400_0.5_1, san400_0.7_1, san200_0.7_2, san200_0.7_3, san200_0.9_1, san1000; sanr200_0.7, sanr200_0.9, sanr400_0.5, sanr400_0.7; C125.9, C250.9.

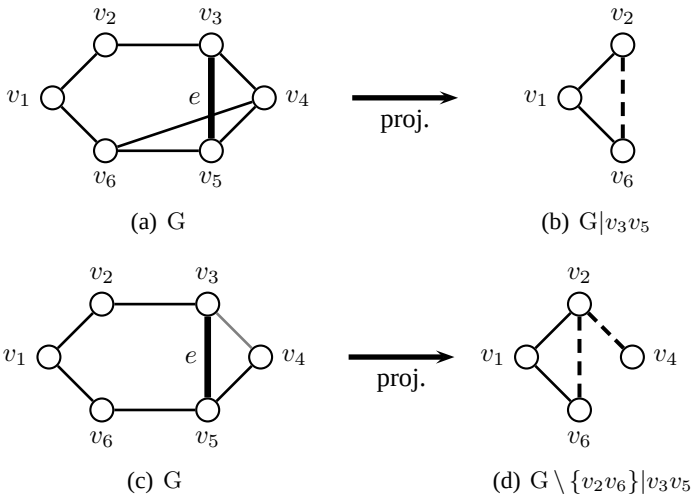


Figure 4.2.3: Edge projections

the anti-projection is not facet-defining for $P_{STAB}(\mathrm{G})$, as it is dominated by the two edge-inequalities $x_1 + x_4 \leq 1$ and $x_2 + x_3 \leq 1$. In addition, it is not even a necessary condition for a facet-defining rank inequality of G that the inequality on G|e before the anti-projection is facet-defining. This is shown by Fig. 4.2.4 (c) and (d). The inequality in (d) is dominated by the two clique inequalities $x_1 + x_4 + x_5 \leq 1$ and $x_6 + x_7 + x_8 \leq 1$, while the inequality according to (c) is facet-defining, compare Chap. 3.4. Let us summarize this in the following lemma.

Lemma 4.2.10. *A facet-defining inequality for $P_{STAB}(\mathrm{G}|e)$ is not in general facet-defining for $P_{STAB}(\mathrm{G})$. It is not necessary for a rank inequality to be facet-defining for $P_{STAB}(\mathrm{G})$ that the inequality before the anti-projection is facet-defining for $P_{STAB}(\mathrm{G}|e)$.*

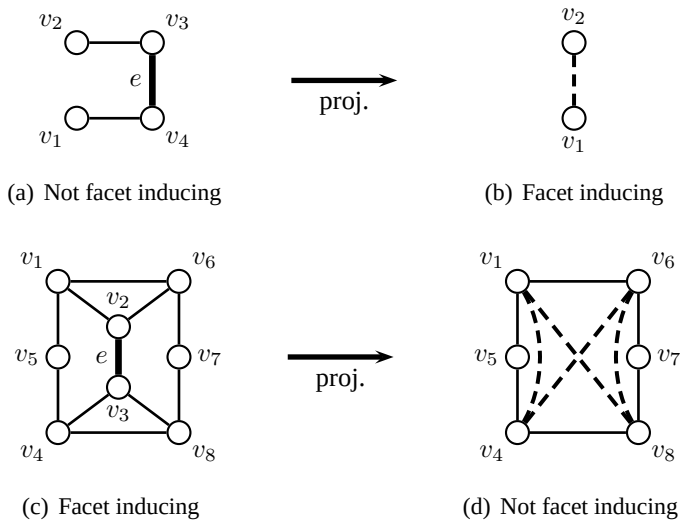


Figure 4.2.4: Edge projections and facet-defining inequalities

We want to look more closely at the polyhedral structure of the edge projection. For the case of an inequality being facet-defining on $\mathrm{G} - \Gamma_{uv}$ Theorem 3.2.3 of NEMHAUSER and TROTTER

can be used to lift it to a facet-defining inequality for $P_{STAB}(\mathrm{G})$. Therefore, let $W \subset V$ and $x(W) \leq l$ be a facet-defining rank inequality for $P_{STAB}(\mathrm{G} - \Gamma_{uv})$ for a strongly projectable edge $uv \in E$. Then, the inequality

$$x(W \cup \{u, v\}) + \sum_{v_i \in \Gamma_{uv}} a_i x_i \leq l + 1$$

with coefficients

$$a_i = l + 1 - \alpha(\mathrm{G}[W \setminus \Gamma(v_i)])$$

is facet-defining for $P_{STAB}(\mathrm{G})$. Note that in this case the coefficients for Γ_{uv} are independent of each other. They are uniquely determined by the objective function value of a stable set problem on a subgraph of G. As the coefficient depends on the stability number of an induced subgraph, it may become too time-consuming to calculate it exactly. However, to strengthen the inequality $x(W) \leq l$ it would be enough to have an upper bound $\widetilde{\alpha}$ for $\alpha(\mathrm{G}[W \setminus \Gamma(v_i)]))$. The new coefficient is then $a_i = \max\{1, l + 1 - \widetilde{\alpha}\}$. Hence, if $\widetilde{\alpha} < l + 1$ the resulting inequality is already strengthened. As this subproblem is a maximum-cardinality stable set problem, one could use for instance a clique covering for the upper bound $\widetilde{\alpha}$.

We have seen that an anti-projected inequality can be lifted to a facet, whenever its restriction on $\mathrm{G} - \Gamma_{uv}$ was facet-defining. In the following, we introduce a sufficient condition for an inequality to be facet-defining for $P_{STAB}(\mathrm{G} - \Gamma_{uv})$ after the anti-projection. What we need is a special edge. We call a false edge $\tilde{e}$ **critical** for an inequality (β, b_0) and a strongly projectable edge e, if inequality (β, b_0) is not valid any more for $P_{STAB}(\mathrm{G}|e)$ if $\tilde{e}$ is deleted from $\mathrm{G}|e$.

critical false edge

Proposition 4.2.11. *Let $e = uv \in E$ be strongly projectable, $W \subseteq V|e$, and $x(W) \leq l$ be facet-defining for $P_{STAB}(\mathrm{G}|e)$. If for e and $x(W) \leq l$ there is a critical edge, and a stable set $\widetilde{\mathcal{S}} \subseteq W$ in $\mathrm{G}|e$ such that $\widetilde{\mathcal{S}} \cap \Gamma(u) = \widetilde{\mathcal{S}} \cap \Gamma(v) = \emptyset$ and $|\widetilde{\mathcal{S}}| = l$, then the inequality $x(W \cup \{u, v\}) \leq l + 1$ is facet-defining for $P_{STAB}(\mathrm{G} - \Gamma_{uv})$.*

Proof. Let $\beta \in \mathbb{R}^n$, $b_0 \in \mathbb{R}$ with

$$\text{for all stable sets } \mathcal{S} \text{ in } \mathrm{G} - \Gamma_{uv} : \chi^{\mathcal{S}}(W \cup \{uv\}) = l + 1 \Longrightarrow \beta^\top \chi^{\mathcal{S}} = b_0. \tag{4.2.1}$$

We have to show that there is a constant $\xi \in \mathbb{R}$ with $\beta = \xi \cdot \chi^{W \cup \{u,v\}}$ and $b_0 = \xi \cdot (l+1)$. This will be done in three steps.

First step. We will show that $\beta_u = \beta_v$. Let $\widetilde{\mathcal{S}}$ be as assumed. Then $\widetilde{\mathcal{S}} \cup \{u\}$ is a stable set on G with cardinality $l+1$. This implies together with (4.2.1) that $\beta^\top(\chi^{\mathcal{S}} + \chi^{\{u\}}) = b_0$. The same can be shown for v which gives $\beta^\top(\chi^{\mathcal{S}} + \chi^{\{v\}}) = b_0$. These two identities imply $\beta_u = \beta_v$.

Second step. The existence of a constant $\xi \in \mathbb{R}$ with $\beta|_{V|e} = \xi \cdot \chi^W$ will be shown where $\beta|_{V|e}$ means the restriction of β to the set $V|e \subset V$. According to the definition of the projection $\mathrm{G}|e$, the addition of one of the two nodes u or v to an arbitrary stable set $\mathcal{S}$ in $\mathrm{G}|e$ will lead to a stable set in G (this is only valid for one of the two nodes in general). Otherwise $\mathcal{S}$ would contain a node $w_1 \in \Gamma(u)$ and a node $w_2 \in \Gamma(v)$ which is a contradiction as w_1 and w_2 are adjacent in $\mathrm{G}|e$. Therefore for all stable sets $\mathcal{S} \subseteq W$ in $\mathrm{G}|e$ with cardinality l there exists a node $w \in \{u, v\}$ with

$$(\chi^{\mathcal{S}} + \chi^w)(W \cup \{u, v\}) = l + 1.$$

This implies together with (4.2.1) that $\beta^\top(\chi^{\mathcal{S}} + \chi^w) = b_0$. With the result of the first step we get that $\beta^\top|_{V|e}\chi^{\mathcal{S}} = b - \beta_u$ for all stable sets $\mathcal{S}$ in $\mathrm{G}|e$ satisfying $\chi^{\mathcal{S}}(W) = l$. Since $x(W) \leq l$ is facet-defining on $P_{STAB}(\mathrm{G}|e)$ there is a constant $\xi \in \mathbb{R}$ with $\beta|_{V|e} = \xi \cdot \chi^W$.

Third step. It will be shown that ξ from step two satisfies $\xi = \beta_u\ (= \beta_v)$. Let $\tilde{e} = w_1w_2$ be a critical false edge. Then there is a stable set $\mathcal{S} \subseteq W$ in $\mathrm{G}|e$ with size $l-1$ and the property that after deleting $\tilde{e}$ from $\mathrm{G}|e$ the extension $\mathcal{S} \cup \{w_1, w_2\}$ is a stable set. It can be assumed that $w_1 \in \Gamma(u)$. Then

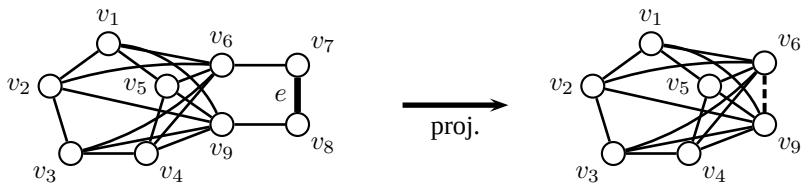
$$\begin{aligned} & \chi^{\mathcal{S}\cup\{w_1,w_2\}}(W \cup \{u,v\}) &&= l+1 \\ \text{and} \quad & \chi^{\mathcal{S}\cup\{w_1,v\}}(W \cup \{u,v\}) &&= l+1. \end{aligned}$$

This enables us to use (4.2.1) and we obtain $\beta^\top\chi^{\mathcal{S}\cup\{w_1,w_2\}} = b$ and $\beta^\top\chi^{\mathcal{S}\cup\{w_1,v\}} = b$. Comparison of the coefficients yields the equality $\beta_{w_1} + \beta_{w_2} = \beta_{w_1} + \beta_v$ which can be simplified to $\beta_{w_2} = \beta_v$. As w_2 is in $V|e$ we get from step two that $\xi = \beta_v$.

Up to this point we have shown that there is a constant ξ with $\beta = \xi \cdot \chi^{W\cup\{u,v\}}$. As the inequality $\chi^{W\cup\{u,v\}} \leq l+1$ is valid and the face is not empty, $b_0 = \xi \cdot (l+1)$ and the proof is complete.

□

The assumptions of Prop. 4.2.11 are satisfied if W is a maximal clique in $\mathrm{G}|e$ with size 3 or more and one of its edges $\tilde{e}$ is a false edge. In this case, $\tilde{e}$ is a critical edge in $\mathrm{G}|e$ and one can choose $\widetilde{\mathcal{S}} = \emptyset$. If in addition the nodes of Γ_{uv} are only adjacent to u, v, $\Gamma(u)$ or $\Gamma(v)$ the anti-projected clique inequality is facet-defining for $P_{STAB}(\mathrm{G})$. If $W = V|e$ induces an odd hole in $\mathrm{G}|e$ and $E|e$ contains a false edge, then the anti-projected inequality is facet-defining for $P_{STAB}(\mathrm{G} - \Gamma_{uv})$. For lifted odd-cycle inequalities in $\mathrm{G}|e$ this is not generally true. There is an counterexample with only 6 nodes.



(a) Not facet inducing rank inequality (b) 5 wheel with 7 nodes

Figure 4.2.5: Edge projection for general inequality

One observes that all the presented results were restricted to rank inequalities. The reason is that the anti-projection may fail for general inequalities. An example is given in Fig. 4.2.5. The graph of sub figure (a) is projected with the strongly projectable edge v_7v_8. The result can be seen in sub figure (b). This graph contains one 5-cycle and the two nodes v_7, v_8 which are adjacent to all nodes of the graph. The resulting wheel inequality reads

$$\sum_{i=1}^{5} x_i + 2x_6 + 2x_9 \leq 2.$$

The anti-projection with Theorem 4.2.5 would result in the inequality

$$\sum_{i=1}^{5} x_i + 2x_6 + x_7 + x_8 + 2x_9 \leq 3.$$

This inequality is no longer valid since $\{v_6, v_9\}$ is a stable set in G which violates the inequality. With this example it is obvious that the anti-projection fails whenever there is a false edge with the property that the coefficients of both of its endnodes have a value greater than 1. The next proposition deals with these cases and extends the edge projection to general inequalities. We define $\max \emptyset := 0$.

Proposition 4.2.12. *Let $e = uv \in E$ be a strongly projectable edge and $\sum_{v_i \in V|e} a_i x_i \leq l$ be valid for $P_{STAB}(\mathrm{G}|e)$ and not dominated by a non-negativity facet. The inequality*

$$\sum_{v_i \in V} a_i x_i \leq l + a \text{ with } a_j := a := \max \{a_i \mid v_i \text{ is an endnode of a false edge } \}$$

for all $v_j \in V \setminus V|e$ is valid for $P_{STAB}(\mathrm{G})$.

Proof. Without loss of generality we can assume that $a_i \in \mathbb{N}_0$ for all $v_i \in V|e$. Let $\mathcal{S}$ be a stable set on G_c with a node-weighting c which is given through the coefficients a_i of the inequality

$$\sum_{v_i \in V} a_i x_i \leq l + a. \tag{4.2.2}$$

We construct a stable set $\widetilde{\mathcal{S}}$ on $\mathrm{G}|e$ and show that if $\widetilde{\mathcal{S}}$ satisfies the inequality

$$\sum_{v_i \in V|e} a_i x_i \leq l, \tag{4.2.3}$$

then $\mathcal{S}$ satisfies inequality (4.2.2). Therefore we consider the following cases and assume first that $\Gamma_{uv} = \emptyset$.

Case 1: $u \in \mathcal{S}$. This implies that $v \notin \mathcal{S}$ and that there exists no $v_i \in \Gamma_u$ with $v_i \in \mathcal{S}$. Hence, the set $\widetilde{\mathcal{S}} := \mathcal{S} \setminus \{u\}$ is a stable set on $\mathrm{G}|e$ with $\alpha(\mathrm{G}_c[\mathcal{S}]) = \alpha(\mathrm{G}_c[\widetilde{\mathcal{S}}]) + a_u$. This shows that $\mathcal{S}$ satisfies (4.2.2) if $\widetilde{\mathcal{S}}$ satisfies (4.2.3).

Case 2: $u \notin \mathcal{S}$ and $\exists v_i \in \Gamma_v$ with $v_i \in \mathcal{S}$. Let $w \in \Gamma_u$ be in $\mathcal{S}$. If there is no such w the set $\mathcal{S}$ is a stable set in $\mathrm{G}|e$ and there is nothing to show. Since edge e is strongly projectable it cannot be the central edge of a double fork which implies that

$$\begin{aligned} & (\mathcal{S} \setminus \{v_i, w\}) \cap \Gamma_v = \emptyset \\ \text{or} \quad & (\mathcal{S} \setminus \{v_i, w\}) \cap \Gamma_u = \emptyset. \end{aligned}$$

Without loss of generality let $(\mathcal{S} \setminus \{v_i, w\}) \cap \Gamma_v = \emptyset$. Then $\widetilde{\mathcal{S}} := \mathcal{S} \setminus \{v_i\}$ defines a stable set in $\mathrm{G}|e$ which shows that in this case inequality (4.2.2) is valid for $\mathcal{S}$.

We only have to discuss the case $\Gamma_{uv} \neq \emptyset$ as all other cases are trivial or equivalent to the two cases discussed above. Let $w \in \Gamma_{uv}$ be in $\mathcal{S}$. We claim that $\widetilde{\mathcal{S}} := \mathcal{S} \setminus \{w\}$ is a stable set on $\mathrm{G}|e$. First of all $\widetilde{\mathcal{S}} \subseteq V|e$ as $\Gamma_{uv} \cup \{u, v\}$ is a clique, compare Lemma 4.2.7. It defines a stable set on $\mathrm{G}|e$ because e is strongly projectable and therefore not the central edge of a bull. This closes the proof. □

In case the computed inequality in the projected graph does not contain any false edge, Proposition 4.2.12 uses the observation that this inequality is also valid for the graph before projection. In this case inequalities (4.2.2) and (4.2.3) are identical, as $a = 0$. Consider again Fig. 4.2.5. The use of Proposition 4.2.12 gives $\sum_{i=1}^{5} x_i + \sum_{i=6}^{9} 2x_i \leq 4$. This inequality is valid for $P_{STAB}(\mathrm{G})$ and even facet-defining.

We want to close this section with two remarks. Up to this point we have considered only one edge projection but it can also be done iteratively. The theory can be adopted straight forward without any new result or restriction but it becomes a little bit technical. For all its details we refer again to [RS01]. The difficulties occur if one wants to implement this iterative edge projection. We will discuss this in the next chapter together with other implementation details. The second remark is on the weighted stable set problem. Since the computed inequalities are valid for the stable set polytope, they can also be used by separation routines of a maximum-weight stable set problem.

4.2.4 General Inequalities

In the following we consider two types of inequalities which can be adopted to any integer program. We start with the so-called local cuts which use polyhedral aspects. We arrange a linear program and see how a violated inequality can be computed from its dual coefficients. This method is followed by an introduction about mod-k cuts which belong to the CHVÁTAL-GOMORY cuts. We focus on the so-called maximal violated mod-k cuts and consider two examples.

Local Cuts

The principle of this method goes back to APPLEGATE, BIXBY, CHVÁTAL and COOK in 2001, cf. [ABCC01]. They introduced the idea for the traveling salesman problem. It has been changed and adjusted to the form we present it by OSWALD, cf. [Osw05].

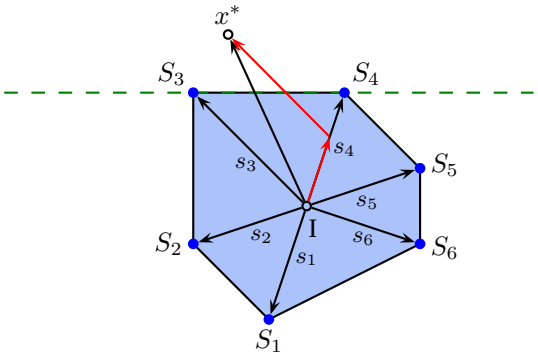


Figure 4.2.6: Local cut

We explain the principle with Fig. 4.2.6. If one knows all feasible solutions $S_1, \ldots, S_m$ of a particular problem, for instance the stable set problem, one could check if a vector x^* lies in its convex hull or not. If it lies outside, the aim is to compute a facet which separates this point

from the convex hull. In the picture this is the dashed line. To get it, one has to find an interior point I of the convex hull and calculate all vectors s_j from I to the feasible solution S_j. Let i be the vector according to point I. A cone combination of $x^* - i$ with the vectors s_j provides the information, if I is inside or outside the convex hull. Therefore, consider the following LP.

$$\begin{aligned} \min \quad & \mathbb{1}^\top \lambda \\ \text{s.t.} \quad & (s_1, \ldots, s_m)\lambda = x^* - i \\ & \lambda \in \mathbb{R}^m_+. \end{aligned} \tag{4.2.4}$$

The objective function value (4.2.4) provides the information about the position of x^*. If it is greater than one, the cone combination is not convex and x^* lies outside the polytope. In order to compute the facet separating x^* from the convex hull, we consider its dual program.

$$\begin{aligned} \max \quad & (x^* - i)^\top y \\ \text{s.t.} \quad & \begin{pmatrix} s_1^\top \\ \vdots \\ s_m^\top \end{pmatrix} y \leq \mathbb{1} \\ & y \in \mathbb{R}^n. \end{aligned}$$

The complementary slackness provides the following identities

$$\begin{aligned} & (\mathbb{1}^\top - y^\top(s_1, \ldots, s_m))\lambda = 0 \\ \iff \quad & y^\top(s_1, \ldots, s_m)\lambda = \mathbb{1}^\top \lambda \\ \iff \quad & y^\top(x^* - i)\lambda = \mathbb{1}^\top \lambda. \end{aligned}$$

We know that x^* lies inside the polytope, iff $\mathbb{1}^\top \lambda \leq 1$. This leads to the following inequality

$$y^\top(x^* - i) \leq 1 \tag{4.2.5}$$

which defines a facet of the polytope, which is not proven here. The left hand side of (4.2.5) depends on i and the dual variables may be fractional. As this inequality is facet-defining, the coefficients have a greatest common divisor, which may be fractional, too. We denote it by gcd and get

$$\widetilde{y}^\top x \leq \frac{1}{gcd} + \widetilde{y}^\top i \tag{4.2.6}$$

with $\widetilde{y} := \frac{1}{gcd} y$. Inequality (4.2.6) has now the integer coefficients $\widetilde{y}$. The right hand side may be still fractional and one can strengthen this inequality by rounding its ride hand side down. This provides the so-called **local cut**

local cut

$$\widetilde{y}^\top x \leq \left\lfloor \frac{1}{gcd} + \widetilde{y}^\top i \right\rfloor. \tag{4.2.7}$$

The complexity of this method is dominated by the enumeration of all feasible solutions and by the solution time for the linear program. The dual variables are a by-product of the solution of the linear program with the simplex method. They can be calculated in linear time from the optimal solution x^* of the primal LP, cf. [Kos04].

For the stable set problem, one maximal stable set can be computed in linear time but the computation of all of them is $\mathcal{NP}$-$hard$. Therefore, it is obvious that such a local cut (4.2.7) will not be computed for the whole graph. In this case the explicit enumeration of all solutions would already provide the optimum. Hence, one has to select a subgraph and compute the cut there. This explains the name "local cut" since it is "local" facet-defining for the smaller graph. The computation of the smaller graph will be discussed in Chapter 5.2.4.

Mod-k Cuts

The appealing idea of the mod-k cuts is to find a multiplier μ such that a particular inequality system $Ax \leq b$ multiplied with this vector μ can be strengthened by dividing it by a positive integer k. The following is due to [CFL00].

Let $k > 1$ be integer and suppose that we have given a system of linear inequalities $Ax \leq b$ with integral coefficients. We multiply each inequality r with a positive integer multiplier μ_r. Let μ be the vector with the positive integer entries μ_r and appropriate dimension such that

$$\begin{aligned} \mu^\top A &\equiv 0 \pmod{k} \\ \mu^\top b &\equiv k-1 \pmod{k}. \end{aligned}$$

According to this construction, $\mu^\top A$ and $\mu^\top b - (k-1)$ are divisible by k without fraction. This leads to the following inequality

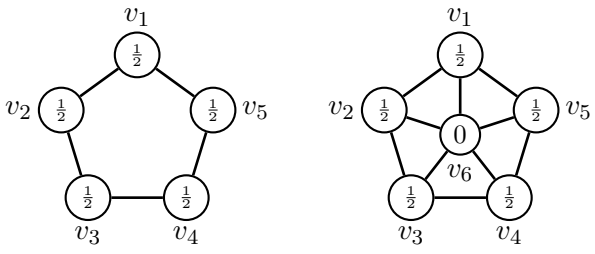
$$\mu^\top Ax \leq \mu^\top b - (k-1) \tag{4.2.8}$$

mod-k inequality

which is valid for the polytope defined through $\text{conv}\{x \in \mathbb{Z}^n \mid Ax \leq b\}$. As constructed inequality (4.2.8) can be divided by k which leads to the so-called **mod-k inequality**

$$\frac{1}{k}\mu^\top Ax \leq \frac{1}{k}(\mu^\top b - (k-1)). \tag{4.2.9}$$

Let x^* be a fractional solution satisfying the inequality system $Ax \leq b$. Vector x^* can violate (4.2.8) by at most $k-1$. This maximal violation can only be achieved if $\mu^\top Ax = \mu^\top b$. One can reach that if $\mu_r = 0$ for all r with $A_r x^* < b_r$, which means that only those inequalities are considered which are tight for x^*.



(a) Odd hole (b) Odd wheel

Figure 4.2.7: Mod-2 cuts

Let us consider now some special cases for the stable set problem. We choose $k = 2$ and consider Fig. 4.2.7 (a). Adding of the edge inequalities corresponding to that graph and

multiplying them with factor 1 leads to inequality

$$\sum_{i=1}^{5} 2x_i \leq 5.$$

This inequality can be divided by 2 which yields to the mod-2 cut

$$\sum_{i=1}^{5} x_i \leq \tfrac{1}{2}(5-1) = 2. \tag{4.2.10}$$

We recognize that inequality (4.2.10) is an odd-cycle inequality (3.2.1). Therefore, if one chooses the edge inequalities for the inequality system $Ax \leq b$ to be checked, one can separate the odd-cycle inequalities with the method described above. A fractional solution which is tight for the edge inequalities is the value $\frac{1}{2}$ for each variable corresponding to a node of the graph. Note that the violation for an odd-cycle inequality with tight edge inequalities is indeed $\frac{1}{2}$. Next, consider the wheel with 5 nodes shown in Fig. 4.2.7. The corresponding wheel inequality can be computed by addition of four clique inequalities and one edge inequality. We obtain

$$\sum_{i=1}^{5} 2x_i + 4x_6 \leq 5.$$

Dividing by factor 2 implies inequality

$$\sum_{i=1}^{5} x_i + 2x_6 \leq 3. \tag{4.2.11}$$

A possible solution vector is labeled in Fig. 4.2.7. Note that there is no fractional solution for the odd wheel with 5 nodes satisfying the four clique inequalities and one edge inequality tight and satisfying in addition a 5-cycle inequality. Vector

$$\left(\frac{1}{3}, \frac{1}{3}, \frac{1}{3}, \frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right)$$

is valid and satisfies all mentioned inequalities except for the odd wheel. This suggests that in this case inequality 4.2.11 is not a maximal violated mod-2 cut.

4.3 Branching

One time in a Branch & Cut algorithm the separation routines might not find any violated inequality or the improvement of the addition of new inequalities is too small and one decides to branch. The standard branching idea for a problem with binary variables is to generate two subproblems. One (fractional) variable is set to value 1 in one problem and in the other subproblem it is set to value 0. For the maximum stable set problem this branching strategy leads to a very unbalanced Branch & Bound tree. To set a variable to value 1 has an impact to all nodes of its neighborhood as they are all set to value 0. In contrast, setting a variable to value 0 has no consequence for all other variables of the graph. To avoid this drawback, one could

think to set in each subproblem of the tree at least one variable to value 1. In the following we will introduce a branching strategy which has this property. The idea goes back to BALAS and YU, [BY86].

Let $G' = (V', E')$ be the subgraph induced by the set of nodes which are not fixed in a current subproblem. The goal in each subproblem is to find a maximum stable set in the particular graph of the tree or to proof that $\alpha(G') < LB$ with the lower bound LB. Let $W \subseteq V'$ and assume that we can show that $\alpha(G[W]) \leq LB$. Clearly, if $W = V'$ the subproblem can be fathomed. Otherwise, if $\alpha(G') > LB$ any maximum stable set must contain at least one node of set $Z := V' \setminus W = \{v_1, \ldots, v_p\}$. On the base of that observation BALAS and YU showed that every maximum-cardinality stable set with greater weight than the lower bound must be contained in one of the sets

$$V'_i := \{v_i\} \cup V' \setminus (\Gamma(v_i) \cup \{v_{i+1}, \ldots, v_p\}) \quad \text{for } i = 1, \ldots, p.$$

Clearly, this is also true for the weighted case with $c \neq \mathbb{1}$. This branching leads to p new subproblems in one branching step. In each subproblem node v_i is set to value 1 and all nodes of $\Gamma(v_i) \cup \{v_{i+1}, \ldots, v_p\}$ are set to value 0.

The size of W and the ordering of the nodes in Z can effect the total number of subproblems that must be solved. Of course, the larger W is, the fewer subproblem will be generated in that state. The size of W is strongly effected by the quality of the computed lower bound. To determine W is quite crucial and can be done for the cardinality stable set problem for instance with a clique covering, cf. [BY86, RS01]. Also other methods as matchings or holes [Sew98] have been considered. In addition, the choice of the branching variable has also a great impact to the number of subproblems being solved. The size of the tree can be reduced by branching on nodes with a high degree, which was empirically shown by CARRAGHAN and PARDALOS [CP90]. The reason is the previously mentioned observation that with the branching node its neighborhood is also set. To sort the nodes in each subproblem in ascending order of degree seems to be computational expensive as the degree of all nodes has to be calculated in each branching step prior to sorting. However, SEWELL [Sew98] showed experimentally that for sparse graphs this is still convenient.

Chapter 5

Implementation Details

In this chapter we consider details of the implementation of a Branch & Cut solver for the maximum stable set problem. We start with the development of an algorithm putting the presented ideas for the preprocessing together. The following section focuses on some aspects of the implementation of separation routines. A key point in the realization of the odd-cycle inequalities will be the removal of chords. As a by-product, we get an exact separation routine for the odd-hole inequalities. The discussion of two heuristics for the clique inequalities is followed by some aspects of the rank inequalities. We outline the implementation difficulties for an iterative edge projection and introduce a data structure for coping with them. The section about the separation routines is closed with some aspects of the local cuts and mod-k cuts. Two heuristics providing upper bounds are presented in Section 5.3. The first one is a rounding heuristic using an LP solution. The second heuristic algorithm tries to improve a stable set.

5.1 Preprocessing

In Chapter 4.1 we introduced the main ideas of a preprocessing for the stable set problem. We will focus now on the algorithmic aspects of it.

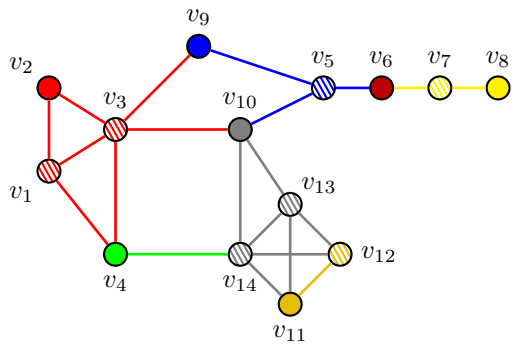


Figure 5.1.1: Preprocessing

We start with an algorithm which fixes cliques in a graph, according to Lemma 4.1.1. Consider Algorithm 5.1.1. It manages a node set M, which is at the beginning the node set of graph G. In the while loop of step 5, each weight of node $v \in M$ is checked if it is greater than all

the weights of its neighbors and if its neighborhood is a clique. If both is true, one can put v into a maximum stable set and remove the clique $Q \cup \{v\}$ from G. This is done in steps 18 to 22. In order to find all nodes which can be fixed in that way, one has to update M. All neighbors of clique Q are new candidates to be fixed. Therefore, they have to be checked again which is done in step 16. The mode of operation of Algorithm 5.1.1 can be seen in Fig. 5.1.1. The completely filled nodes are members of $\mathcal{S}$ of Algorithm 5.1.1. The nodes and edges with the same color symbolize a clique which has been fixed. In this example we recognize that the clique Q contains one, two or three nodes. This shows that Algorithm 5.1.1 also considers isolated nodes and paths. Furthermore, we see that node v_6 would not be fixed if just the nodes in the increasing order were considered and no update of set M were performed. In this graph the fixing of cliques already solved the maximum stable set problem. Of course, this cannot be generally true because Algorithm 5.1.1 uses only local information.

Algorithm 5.1.1 Clique fixing

Input: Graph $\mathrm{G}_c = (V, E, c)$
Output: Graph $\widetilde{\mathrm{G}}_c \subseteq \mathrm{G}_c$, set of nodes $\mathcal{S}$ and its weight h

// Initialize
1: $\mathcal{S} := \emptyset$, $h := 0$ // Stable set and its weight
2: $Q := \emptyset$ // Clique
3: $M := V$ // Set to administrate the nodes to be checked
4: $\widetilde{\mathrm{G}}_c := \mathrm{G}$ // Resulting graph after remove of nodes

// Check each node of set M if it can be fixed
5: **while** $M \neq \emptyset$ **do**
6: Select node $v \in M$
7: Delete node v from M

// Check if the neighborhood of v is a clique and if v has maximal weight
8: **for all** $w \in \Gamma(v)$ **do**
9: **if** $c(v) \geq c(w) \wedge \forall x \in Q\,(xw \in E)$ **then**
10: $Q = Q \cup \{v\}$
11: **else**
12: $Q = \emptyset$, goto 5
13: **end if**
14: **end for**

// Q contains a maximal clique which can be fixed
// Update...
15: **for all** $w \in Q$ **do**
16: $M = M \cup (\Gamma(w) \setminus Q)$ // ...the nodes to be checked
17: **end for**
18: $\widetilde{\mathrm{G}}_c = \widetilde{\mathrm{G}}_c \setminus (Q \cup \{v\})$ // ...graph $\widetilde{G}$
19: $M = M \setminus Q$ // ...the nodes to be checked
20: $Q := \emptyset$ // ...clique Q
21: $\mathcal{S} = \mathcal{S} \cup \{v\}$ // ...stable set
22: $h = h + c(v)$ // ...weight of stable set
23: **end while**
24: **return** $\widetilde{G}_c$, $\mathcal{S}$ and h

We are now ready to put together the observations of Chapter 4.1 into one algorithm. The splitting into connected components and the fact that a node can be fixed if its weight is greater than the sum of its neighbors have been neglected. Consider Algorithm 5.1.2. In step 2, it deletes all nodes which have negative weight as they are not part of a maximum stable set. Fixing of cliques is done in step 4 with the use of Algorithm 5.1.1. After all cliques have been deleted, the $\mathcal{LP}$ consisting of edge and non-negativity inequalities is generated and solved; for the $\mathcal{LP}$ see also Chap. 3.1. We have seen that nodes whose corresponding solution vector has value 1 can be fixed. This node is added to set $\mathcal{S}$ and the graph is updated. It is obvious that Algorithm 5.1.1 can only fix new nodes if graph $\widetilde{\mathrm{G}}$ has been updated and hence the fixing process is repeated until no node could be fixed.

One can ask for a justification of step 6. The question could be if there are any nodes to be fixed which could not be found by the clique fixing. Therefore, consider a graph which is induced by a chordless even cycle. On the one hand, in the cardinality case Algorithm 5.1.1 will not be able to fix one single node for this class of graphs. On the other hand, as this graph is bipartite, the edge inequalities together with the non-negativity inequalities define P_{STAB}. This implies that the $\mathcal{LP}$ solution indicates a stable set and the problem is solved in step 6. We recognize that in general, this preprocessing will solve the maximum stable set problem for bipartite graphs. For the running time it is quite important that the clique fixing precedes the fixing with the help of the $\mathcal{LP}$. The reasons are that all fixed nodes of a clique can potentially be found by the $\mathcal{LP}$ fixing and Algorithm 5.1.1 may reduce the order of the graph which speeds up the computational time of the LP solver. We will see in Chapter 6 how this preprocessing performs on test instances.

Algorithm 5.1.2 Preprocessing

Input: Graph $\mathrm{G}_c = (V, E, c)$
Output: Graph $\widetilde{\mathrm{G}}_c \subseteq \mathrm{G}$, set of nodes $\mathcal{S}$ and its weight h

// Initialize
1: $\widetilde{G} = G$
2: Delete all nodes from $\widetilde{\mathrm{G}}_c$ which have negative weight.
// Clique fixing and fixing of nodes corresponding to $\mathcal{LP}$ solution with value one
3: **repeat**
4: Use algorithm 5.1.1 with input data $\widetilde{G}_c$. Receive $\widetilde{\mathrm{G}}_c$, $\widetilde{\mathcal{S}}$ and weight $\widetilde{h}$. Union $\mathcal{S}$ with $\widetilde{\mathcal{S}}$ and add weight $\widetilde{h}$ to h
5: Construct $\mathcal{LP}$ of graph $\widetilde{\mathrm{G}}$
6: Solve $\mathcal{LP}$ with simplex method and get optimal vector x^*
7: **for all** $x_i^* = 1$ **do**
8: $\mathcal{S} = \mathcal{S} \cup \{v_i\}$
9: $h = h + c(v_i)$
10: $\widetilde{\mathrm{G}} = \widetilde{\mathrm{G}} \setminus (\Gamma(v_i) \cup \{v_i\})$
11: **end for**
12: **until** $\nexists i(x_i^* = 1)$
13: **return** Graph $\widetilde{G}_c$, stable set $\mathcal{S}$ and its weight h

5.2 Separation

This section focuses on some crucial points for the implementation of the separation routines discussed in Chapter 4.2. We start with the separation of the odd-cycle inequalities. Considered are some modifications to decrease the computational time and to delete the chords of an odd-cycle. This will lead to an exact separation for the odd-hole inequalities. The discussion of the odd-cycle inequalities will be closed by a modification which destroys the exactness but increases the speed of an algorithm. Together with a heuristic computing a maximal clique, the idea of lifting 3-cycles in linear time is presented in Section 5.2.2. Some details of an iterative edge projection algorithm are discussed in Section 5.2.3. The main part is the development of an efficient data structure to handle the projection and its anti-projection. The selection of the subgraph and the enumeration of all stable sets needed for the local cuts is presented in the section about general inequalities.

5.2.1 Odd-Cycle Inequalities

Consider now some details about the separation of the odd-cycle inequalities (3.2). If we assume that all trivial and edge inequalities are satisfied by a solution vector x^*, an odd-cycle inequality can maximal be violated by value $\frac{1}{2}$, independent of its size. Therefore, small odd-cycles should be preferred during separation. Recalling the definition of the edge weights $c(uv) = \frac{1-x_u^*-x_v^*}{2}$ provides for the case of $x_u^* + x_v^* = 1$ that the weights of the edges u^+v^- and u^-v^+ of auxiliary graph H are zero. Hence, a minimum-weight path in H may contain additional edges and nodes which are not required in order to calculate a shortest path. To avoid this phenomena, one can use the trick to add a small positive value to the weights which are zero, for instance 10^{-8}. In addition, this has the positive effect that in some cases unnecessary even cycles are avoided. For an example consider Fig. 4.2.1 (c) again.

A node whose corresponding variable has value 0 or 1 can be deleted from the auxiliary graph H used by Algorithm 4.2.1. The reason is that one single node with such a property implies that no odd cycle containing this node can violate the odd-cycle inequality. Therefore, let C be an odd cycle containing v_i with $x_i = 0$. As $x_k + x_l \leq 1$ holds for all $k, l \in C$,

$$c(C) = \sum_{\substack{v_j \in V(C) \\ j \neq i}} x_j \leq \frac{|C| - 1}{2}.$$

If a node of C has value 1, it can be reduced to the case above since their two neighbors in the cycle have value 0.

We already saw in Chapter 3.2 that it is a necessary condition to have a chordless odd-cycle inequality in order to be facet-defining for the stable set polytope. If we consider Fig. 3.2.2 (a) again and compute an odd-cycle starting with node v_2 or v_3, we see that the resulting cycle will contain a chord. This is independent whether we add a small value to its weights or not. The odd-cycle separation with Algorithm 4.2.2 will not deal with this case. Now, we want to discuss an extension of it with the aim to delete all chords. Consider therefore Algorithm 5.2.1. It requires an odd cycle C with mainly one property. Assumed is that the cycle has to be free from node and edge repetitions, which is already satisfied according to the definition of a cycle, compare Chap. 1.1. A real restriction is that cycle C does not contain a smaller sub cycle which

is odd and includes v_i. The computed odd cycle in Algorithm 4.2.1 will meet these assumptions if small values are added to all weights of H which are 0. Let us now consider the details of Algorithm 5.2.1. After relabeling the nodes of cycle C, it checks for each second node if it is adjacent to it, step 6. It is enough to check only nodes which have an odd distance in the cycle as otherwise C would contain an odd sub cycle. Therefore we need the assumption already discussed. If two nodes in step 6 are adjacent, a chord has been detected. In step 7 the even cycle containing node v_i is removed from $\widetilde{C}$. The procedure starts again with this smaller odd cycle, step 9. Observe that the assumptions are also met by the smaller cycle. In step 14 an odd hole is returned. The running time is linear in the size of C as each possible chord is only considered once. Therefore one could extend Algorithm 4.2.2 by calling Algorithm 5.2.1 after step 2. This leads to an polynomial separation routine for the odd-hole inequalities! Note that this separation is exact.

Algorithm 5.2.1 Delete chords

Input: Graph $G = (V, E)$, odd cycle C and start node v_i
Ensure that C has no node repetition and there is no odd-cycle in G$[V(C)]$ containing v_i

Output: Chordless odd cycle $\widetilde{C} \subseteq C$

// Relabel the nodes in C
1: $\widetilde{C} := \{v_i = w_1, w_1w_2, w_2, \ldots, w_{|C|}, w_{|C|}w_1, w_1\}$ with $w_i \in C$
2: $j := 2$
3: **for all** $j < |C|$ **do**
4: $k := j + 2$
5: **for all** $k < |C|$ **do**
6: **if** $w_jw_k \in E$ **then**
7: $C := \{w_j, w_jw_{j+1}, w_{j+1}, \ldots, w_k, w_kw_j, w_j\}$
8: $v_i := w_j$
9: goto 1
10: **end if**
11: $k = k + 2$
12: **end for**
13: $j = j + 2$
14: **end for**
15: **return** Chordless odd cycle $\widetilde{C}$

The odd-cycle separation has the aim to find the most violated odd-cycle inequality in a graph. In order to compute a solution for a stable set problem via a Branch & Cut algorithm, it is more interesting to get any violated inequality, within a tolerance ε. Therefore we consider each odd-cycle inequality resulting from Algorithm 4.2.2 in step 2 and check if its weight is smaller than $\frac{1}{2} - \varepsilon$. After that, we delete its chords with Algorithm 5.2.1 and look at its size. If the size is equal to 3 we lift it to a maximal clique, since we know that the resulting inequality is facet-defining; for details see the next section.

All of the last modifications did not change the exactness of the separation routine. Now we consider the following change: Compute only odd cycles in G for a node u, if it is not contained in an odd cycle already computed during this separation. Clearly, this is a heuristic procedure. Consider therefore Fig. 5.2.1. Using the method described with ascending order to

the indices of the nodes will compute only three 3-cycles. The sum of the solution vector for each of them is equal to 1 and hence the according odd-cycle inequalities are satisfied. The odd cycle containing nodes v_7, v_8 and v_9 is not detected. Hence, this separation routine would fail to find any violated odd-cycle inequality for our problem. Nevertheless, the speed up of the separation routine is quite large and we decided to use this modification.

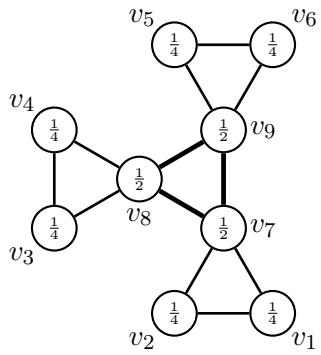


Figure 5.2.1: Not exact odd-cycle separation

5.2.2 Clique Inequalities

It is not a good idea to solve $\mathcal{NP}\text{-}hard$ problems as a subproblem. Therefore we do not separate the clique inequalities exact. Instead, we use a heuristic. It selects randomly an edge and computes step by step a maximal clique containing this edge. In each step one node which is adjacent to all selected nodes is added to the set. This added node is also selected randomly. It is done until the set of adjacent nodes is empty. By construction, the clique is maximal and defines a facet of the stable set polytope. If one is lucky, the clique inequality is violated by the current solution and it can be added to the linear program formulation.

A second approach is to sequentially lift a 3-cycle to a maximal clique. Whenever the odd-cycle separation recognizes a violated 3-cycle inequality, one can compute a maximal clique containing this triangle. As in the heuristic clique separation, step by step randomly selected nodes are added until a maximal clique is found. The running time is linear, since each edge is checked at most twice if the graph is stored in a sorted adjacency list. Hereby, it is important that we only compute one maximal clique and not all maximal cliques which contain the 3-cycle. This cannot be done in linear time anymore. We will discuss the effect of that in the next chapter.

5.2.3 Rank Inequalities

In this section we want to focus on the implementation of the edge projection. We outline again that the strength of edge projection lies in the reduction of the size of the graph and the increase of the density of the smaller graph. This gets even stronger, when the edge projection is used iteratively. We already mentioned that the theory does not bring any new information, but the implementation itself is quite challenging. As the separated inequalities have to be anti-projected, we have to store the information how this inequality is computed. We decided to use a tree structure. Each node of the tree stores the information of one projection. Once an

inequality has been found, it can be anti-projected by walking up the tree. We want look at this in more details now.

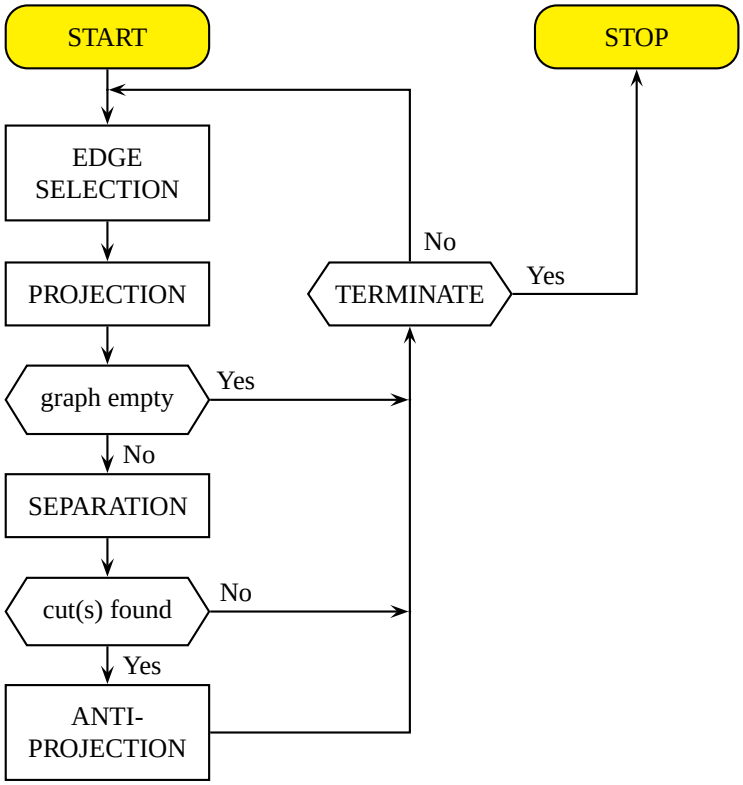


Figure 5.2.2: Flowchart of edge projection

The edge projection is shown as a flowchart in Fig. 5.2.2. First, an edge has to be selected which is the candidate for a projectable edge. The selection is quite crucial. We have seen that any violated inequality which contains false edges have to be anti-projected. This implies that the projectable edges are added to the inequality and the ride hand side is increased. If the weights of these added nodes are too small, a violated inequality in the smaller graph may be satisfied after the anti-projection. A possible choice could be to select only nodes where the sum is greater than 0.6. As the strongly projectable edges for the interesting graphs are very rare, the neighborhood of one of the endnodes of the edge has to be modified, for more details see Chap. 4.2.3. This is another crucial step. Once a strongly projectable edge uv has been found, the graph has to be projected. This means that the nodes u and v as well as their common neighborhood Γ_{uv} have to be removed from the graph and the false edges have to be added. In order to manage this update quickly, one could store the graph in an adjacency matrix. Each update step can than be done in constant time. If the projection is complete, a separation routine can be called—one has also the freedom to project again. It can be any separation procedure which computes rank inequalities. ROSSI and SMRIGLIO suggested the clique separation, [RS06]. Maybe, the reasons are that the new graph is denser which can be exploited by this separation. In addition, a heuristic procedure is also quite fast compared to other separation routines. The use of the clique separation is also justified by the theory, as there

is a good chance that the anti-projected inequalities of maximal cliques are facet-defining. The tricky part of the edge projection is the anti-projection as the data of the projection step(s) are required. The tree structure can handle this quite efficient as already mentioned. After some inequalities have been computed one can project again to find more violated inequalities or can terminate the program. It is also possible to go backwards in the tree. The idea is to project until the tree is empty and to go a few steps back before calling the separation procedure.

5.2.4 General Inequalities

The implementation of a maximally violated mod-k generator is not discussed in this diploma thesis. We used a foreign code implemented by FRICKE, [Fri06]. For details we refer for instance to [CFL00]. Next, we consider the

Local Cuts

In order to compute a local cut, one has to enumerate all stable sets explicitly. As the number of stable sets grows with $O(2^n)$ this cannot be done for the whole graph. Therefore, a subgraph has to be selected. Before we present a heuristic for its computation, we look at some properties which such a subgraph should meet. The subgraph should be connected. Otherwise one could tighten the computational time by partitioning the problem into several subproblems. The size of the graph is a crucial point. Together with the density it effects the number of stable sets in the graph. This is not only crucial for the enumeration but also for the time needed to solve the LP, the number of constraints is the number of feasible stable sets. The current LP solver can handle LPs with up to 10000 inequalities quite fast providing our magnitude. Hence, the selected graph should contain 13 to 16 nodes. Another point is the LP value of the nodes in the subgraph. As already mentioned in other separation routines, variables with a value 0 or 1 are not interesting. Most promising for a high violation are variables which have values close to 0.5. They should be preferred in the selection of the nodes for a subgraph.

Now, we present the idea of the selection of a subgraph. Select a node which LP solution has a value close to 0.5 and add it to the graph. Select now some nodes of its neighborhood and prefer again the nodes which LP value is close to 0.5. This is done iteratively for the added nodes until the whished size of the subgraph is reached.

After the subgraph has been found, we have to enumerate all stable sets. This can be done for instance in generating all 0-1 permutations and checking if they build a stable set in G or not. We used a slight modification where we construct a stable set from the previous generated stable set.

The interior point is calculated as a convex combination of all solutions. The choice of the interior point effects the resulting facet but it does not effect if a facet is found or not. We have to compute the greatest common divisor in order to get an inequality with integer coefficients, compare 4.2.4, which is calculated in the standard way, see Algorithm 5.2.2.

Algorithm 5.2.2 Greatest common divisor

Input: Real number a and b
Output: gcd(a,b)

1: **while** $a > (b + \epsilon) \vee (a > (b - \epsilon))$ **do**
2: **if** $a > b$ **then**
3: $a = a - b$
4: **else**
5: $b = b - a$
6: **end if**
7: **end while**
8: **return** a

5.3 Lower Bounds

During a Branch & Cut algorithm, the computation of lower bounds for a maximization problem is very important to decrease the running time. The branching and separation process provide upper bounds which decrease continuously. The only way to get lower bounds for a Branch & Cut algorithm is to receive them from an integer solution of an LP. This means that in the optimum vertex of the polytope all facets are known. Since all feasible solutions provide a lower bound, one could also use heuristics. We will present two of them.

5.3.1 Rounding Heuristic

After an LP has been solved one can check if the solution is integer feasible. If this test fails, the separation or branching step is called. Between these steps one can use heuristics to increase the global lower bound. With the information of the LP solution, we construct a maximal stable set with Algorithm 5.3.1. The heuristic puts all nodes in a set $\mathcal{S}$ which solution vector has the value 1. This set is a stable set as it is assumed that solution x satisfies the edge inequality. The neighbors of the added nodes are marked that they cannot get a member of $\mathcal{S}$, step 7. All nodes which are not added to $\mathcal{S}$ are put into a set F together with a weight. This is the LP solution value multiplied with the node weight of function c. After this is finished, node set F is sorted in ascending order with respect to the weighting, step 10. Step by step more nodes are added to set $\mathcal{S}$ until no nodes are left. Before a node is added it is checked if it is marked. This guarantees that $\mathcal{S}$ is in each step a stable set. The computational time is dominated by the sorting in step 10. Hence, Algorithm 5.3.1 has a running time of $O(V \log(V))$.

5.3.2 Improve Stable Set

Consider the graph of Fig. 5.3.1 which is a copy of the graph of Fig. 5.1.1. Assume that we have maximum-cardinality stable set problem and that we are given a maximal stable set $\mathcal{S}$ as indicated by the black filled nodes v_1, v_5, v_7 and v_{11}. We recognize that the shown maximal stable set $\mathcal{S}$ is not of maximum size. One could for instance put nodes v_9 and v_{10} into $\mathcal{S}$ and remove node v_5. The result is again a stable set but the size has increased by value 1. One reason

why the new set defines a stable set is that the distance of node v_5 to all other nodes of set $\mathcal{S}$ is greater than 3. Here, the distance of two nodes is defined as the size of a shortest path connecting them. The second reason is that there are the two shortest paths $(v_5, v_5v_9, v_9, v_9v_3, v_3v_1, v_1)$ and $(v_5, v_5v_{10}, v_{10}, v_{10}v_{13}, v_{13}v_{11}, v_{11})$ connecting v_5 with v_1 and v_{11} which have no node in common. Since the two nodes v_9 and v_{10} are not adjacent, they can be exchanged in $\mathcal{S}$ with v_5.

Algorithm 5.3.1 Rounding Heuristic

Input: Graph $G = (V, E)$, LP solution vector $x \in [0,1]^{|V|}$ and weight $c \in \mathbb{R}_+^{|V|}$
Ensure that x satisfies the edge inequalities
Output: Maximal stable set $\mathcal{S}$

// Initialize
1: $\mathcal{S} = \emptyset$, $F = \emptyset$, $y_i = -1 \; \forall i \in V$

// Add all variables which have value 1 to $\mathcal{S}$
2: **for all** $v_i \in V$ **do**
3: **if** x_i = 1 **then**
4: $\mathcal{S} = \mathcal{S} \cup \{v_i\}$
5: $\forall v_j \in \Gamma(v_i) : y_j = 0$
6: **else**
7: $F = F \cup \{(v_i, x_i * c_i)\}$
8: **end if**
9: **end for**

10: Sort F in the second argument

// Add nodes to $\mathcal{S}$ until the stable set is maximal
11: **while** $F \neq \emptyset$ **do**
12: Select element $(v_i, \cdot)$ from set F
13: $F = F \setminus \{(v_i, \cdot)\}$
14: **if** $y_i \neq 0$ **then**
15: $\mathcal{S} = \mathcal{S} \cup \{v_i\}$
16: $\forall v_j \in \Gamma(v_i) : y_j = 0$
17: **end if**
18: **end while**
19: **return** Maximal stable set $\mathcal{S}$

The described method is generally true and we want to use this idea to formulate Algorithm 5.3.2. Three arbitrary nodes of a stable set are selected in step 3. Node v_{top} is a candidate to be exchanged from the stable set. Therefore two shortest paths of length 3 from node v_{top} to u and v_{top} to v have to be calculated. This is done in step 4 and 7. If these paths exist, one has to check if the exchange of node v_{top} with the two nodes u_1 and v_1 of the paths leads to a stable set. Obviously, the nodes v_1 and v_2 have to be non adjacent. Another condition is that no node of the current stable set different to v_{top} is adjacent to node v_1 or v_2. These conditions are checked in step 7. Clearly, if they are met, the stable set $\widetilde{\mathcal{S}}$ can be increased, step 8 and 9. After such an exchange it is possible that the resulting stable set $\widetilde{\mathcal{S}}$ is not maximal, even if it is greater than the original set. The reason is that one of the neighbors of the removed node v_{top} can have no neighbor in $\widetilde{\mathcal{S}}$. This is exploited in step 11.

The complexity of an exact check of all possible combinations is too high and we randomly

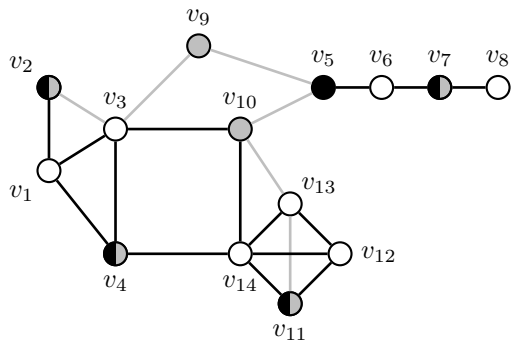


Figure 5.3.1: Improve stable set

selected some cases and bound the running time by a time limit t_{max}, step 2. In order to do it exactly, one would have to check for each node of $v_{top} \in \mathcal{S}$ all $\frac{|\mathcal{S}|^2-|\mathcal{S}|}{2}$ possibilities for u and v and compute all combinations of the two paths with length 3. If δ_{max} denotes the maximal degree in G, then there are at most δ_{max}^2 such paths for each pair of nodes. This yields in a total running time of $O(|\mathcal{S}| \frac{|\mathcal{S}|^2-\mathcal{S}}{2} \delta_{max}^4 \log(\delta_{max})) = O(|\mathcal{S}|^3 \delta_{max}^4 \log(\delta_{max}))$.

Algorithm 5.3.2 Improve Stable Set

Input: Graph $G_c = (V, E, \mathbb{1})$, stable set $\mathcal{S}$ and time limit t_{max}
Output: Stable set $\widetilde{\mathcal{S}}$ with $|\widetilde{\mathcal{S}}| \geq |\mathcal{S}|$

// Intitialize
1: $\widetilde{\mathcal{S}} = \mathcal{S}$; start CPU timer t_{CPU}
// Try to improve until time limit is reached
2: **while** $t_{CPU} < t_{max}$ **do**
3: Select three nodes v_{top}, u and v from stable set $\widetilde{\mathcal{S}}$, arbitrarily
4: Calculate path $P_u = (v_{top}, v_{top}u_1, u_1, u_1u_2, u_2, u_2u, u)$ with Algorithm 5.3.3
5: **if** $P_u \neq \emptyset$ **then**
6: Calculate path $P_v = (v_{top}, v_{top}v_1, v_1, v_1v_2, v_2, v_2v, v)$ with Algorithm 5.3.3
// Check the sufficient conditions for the exchange
7: **if** $(P_v \neq \emptyset) \wedge (u_1v_1 \notin E) \wedge (\nexists\, w \in \widetilde{\mathcal{S}}, w \neq v_{top} : (u_1w \in E \vee v_1w \in E))$ **then**
8: $\widetilde{\mathcal{S}} = \widetilde{\mathcal{S}} \setminus \{v_{top}\}$
9: $\widetilde{\mathcal{S}} = \widetilde{\mathcal{S}} \cup \{u_1, v_1\}$
10: **end if**
11: Add all possible nodes of $\Gamma(v_{top})$ to $\widetilde{\mathcal{S}}$.
12: **end if**
13: **end while**
14: **return** $\widetilde{\mathcal{S}}$

Observe that Algorithm 5.3.2 can be generalized to the weighted case with $c \neq \mathbb{1}$. The exchange of the nodes of a stable set leads then to a higher weight, if $c(v_{top}) < c(u_1) + c(v_1)$. This condition has to be added to step 7.

Consider once more Fig. 5.3.1. The maximal stable set computed with Algorithm 5.3.2 is still not maximum. But in this case the presented improvement algorithm will fail to find a better stable set which shows that this method is indeed a heuristic procedure. However, one

Algorithm 5.3.3 Compute Path

Input: Graph $G_c = (V, E, c)$, node $v_{start} \in V$ and node $v_{end} \in V$
Output: Path P with length 3 from node v_{start} to node v_{end}

// Check all neighbors of node v_{start} and v_{end} if they are adjacent
1: **for all** $u_1 \in \Gamma(v_{start})$ **do**
2: **for all** $u_2 \in \Gamma(v_{end})$ **do**
3: **if** $u_1u_2 \in E$ **then**
4: **return** $P = (v_{start}, v_{start}u_1, u_1, u_1u_2, u_2, u_2v_{end}, v_{end})$
5: **end if**
6: **end for**
7: **end for**
8: **return** $\emptyset$

could have expected that as the stable set problem is also hard to approximate, compare Chap. 3.1. If we assume $\mathcal{P} \neq \mathcal{NP}$ there is even no guarantee for the quality of this algorithm.

This improvement algorithm can be slightly modified. For instance the paths P_u and P_v in step 4 and 6 should not be calculated independently. The conditions of step 7 can be checked when computing them. In addition the nodes in step 1 and 2 of Algorithm 5.3.3 should be selected randomly. We will come back to the performance of that method in the next chapter where we present some computational results.

Chapter 6

Computational Results

Before we consider some computational results, we summarize the implemented Branch & Cut solver combining the methods discussed in the previous chapters. It starts with the preprocessing phase using Algorithm 5.1.2. The first LP-relaxation is set up via a clique covering. For each node of the graph one maximal clique containing it is constructed. Of course, this relaxation is stronger than $\mathcal{LP}$ as the clique inequalities dominate the edge inequalities, compare Theorem 3.3.1. After an LP has been solved, all edge inequalities are tested to see if they are satisfied. Whenever an edge inequality is violated, a maximal clique containing this edge is computed and added to the LP. If all edge inequalities are satisfied it is tested whether the LP solution is integer feasible. Obviously, if this solution is an incidence vector of a stable set then this (sub)problem can be fathomed. Otherwise, the rounding and improving heuristics are called. The time limit for the improvement algorithm is set to 0.002 seconds. If the gap cannot be closed by the heuristics the separation process starts. The Branch & Cut solver begins with the separation of the odd-hole inequalities and lifts the computed 3-cycles to maximal cliques, cf. Chap. 5.2.1 and 5.2.2. This is followed by the clique heuristics, the edge projection, the separation of mod-$\{2, 3, 5, 7\}$ cuts and the local-cut method, see Chapters 5.2.2, 5.2.3, and 5.2.4. Whenever less violated cuts than $|\mathrm{G}|/10$ are computed, the next separation procedure in this list is called. If none of these algorithms can find any violated cut, a so-called large rank inequality is computed. This means that the edge projection is iteratively executed until a graph of at most 75 nodes is reached. Then, this Branch & Cut solver is again used to solve the maximum-cardinality stable set problem on this smaller graph. The optimum value together with the nodes of the graph imply a rank inequality which has a high support after the anti-projection. All computed inequalities are stored in a cut pool. Each inequality is only added if it is not contained in the pool. After an LP has been solved, all inequalities which have a slack $\neq 0$ are removed from the formulation. If no cuts during the separation can be computed or if the improvement of the LP solution is not better than 10^{-4}, the branching phase starts. The method of BALAS and YU, cf. Chap. 4.3, is used. The nodes of the induced subgraph of the current subproblem are sorted in ascending order of their degree. After that a covering of these nodes is constructed with the use of the rank inequalities stored in the pool. Step by step the inequality which maximizes the ratio of the right hand side and the number of nodes containing it is added. The Branch & Bound tree is scanned with the *bestBoundFirst* strategy.

Table 6.1: Computational results on DIMACS graphs

Instances				Branch & Cut										
				LB_{root}	LP value in root			# B & B nodes			# LP	Time		
Name	$\|V\|$	Dens.	α	BC	BC	BC_{rank}	BC_{clique}	BC	BC_{rank}	BC_{clique}	BC	BC	BC_{rank}	BC_{clique}
brock200_2	200	0.50	12	12	20.99	22.01	22.17	1002	1957	1510	2726	477	292	424
brock200_4	200	0.34	17	14	29.93	31.54	33.87	8227	8695	23457	13855	2128	661	683
brock400_2	400	0.25	29	22	63.84	67.66	70.89	161363[b]	+++	+++	37746[b]	45207[b]	+++	+++
brock400_4	400	0.25	33	23	63.89	67.98	71.85	186247[b]	+++	+++	48882[b]	45714[b]	+++	+++
c_fat200-1	200	0.92	12	12	12.71	12.86	12.98	1	1	1	3	1	33	27
c_fat200-2	200	0.84	24	22	24.00	24.00	24.00	1	1	1	5	1	38	25
c_fat200-5	200	0.57	58	58	58.89	65.25	66.66	1	1	42	90	16	146	92
c_fat500-1	500	0.96	14	11	14.00	14.98	14.90	1	1	1	16	323	697	1012
c_fat500-2	500	0.93	26	26	26.97	57.78	90.51	1	130	304	13	81	1038	1171
c_fat500-5	500	0.81	64	64	64.70	67.08	67.58	1	2	2	10	26	1751	1754
c_fat500-10	500	0.63	126	118	126.00	223.29	223.32	1	672	547	5	4	209	211
C125.9	125	0.10	34	34	41.26	37.40	43.15	1097	80	3136	2234	29	18	28
C250.9	250	0.10	44	43	69.76	58.30	71.91	207462[b]	141954	+++	83612[b]	49047[b]	115099	+++
hamming8-4	256	0.36	16	9	16.00	16.00	16.00	1	1	1	3	1	89	89
keller4	171	0.35	11	11	14.83	14.95	14.96	990	2070	4256	2085	136	187	261

Table 6.1: Continued

san200_0.7_1	200	0.30	30	16	30.00	30.71	30.69	1	1	1	4	1	17	24
san200_0.7_2	200	0.30	18	14	18.50	19.18	19.24	577	38	38	347	11	11	15
san200_0.9_1	200	0.10	70	69	70.00	70.00	70.00	1	1	1	4	1	2	2
san200_0.9_2	200	0.10	60	38	60.00	60.00	60.00	1	1	1	6	1	6	6
san200_0.9_3	200	0.10	44	39	44.00	44.80	45.68	10620	1	46	5359	74	8	5
san400_0.5_1	400	0.50	13	10	13.24	17.14	27.68	228	139	194	416	348	147	315
san400_0.9_1	400	0.10	100	73	100.00	100.40	100.40	1	1	1	12	2	131	131
p_hat300-2	300	0.51	25	17	33.81	34.19	34.42	1124	686	844	2327	912	1015	973
p_hat300-3	300	0.26	36	30	54.12	53.19	55.55	28064	32756	5667	44749	11703	8226	13228

+++ Could not be solved to optimality
^{b} Statistics collected at CPU time limit

The Branch & Cut algorithm is implemented in C++ with the ABACUS framework, cf. [aba]. We used CPLEX as the LP solver, cf. [cpl]. The processor of the test run is an Intel Xeon with 2.8 GHz and 2.0 GB RAM. The CPU time limit is set to 12 hours. We tested the implemented algorithm on *DIMACS Challenge benchmark graphs* [dim] and *uniform random graphs*, cf. Chap. 4.1 in [Sew98]. As benchmark we chose the Branch & Cut algorithms of ROSSI and SMRIGLIO, [RS01]. They have compared two Branch & Cut approaches using only the clique inequalities, BC_{clique}, or the rank inequalities, BC_{rank}. The DIMACS machine benchmarks have shown that our computer is approximately 6.6 times faster than the one used by ROSSI and SMRIGLIO in 2001.

All computed inequalities are checked with an independent program. For each inequality a stable set problem on its induced subgraph is solved with the CPLEX mixed-integer program solver to validate its feasibility. In addition, for all problems which could be solved in the root node, an LP with the computed inequalities is solved. This LP solution always lies between the optimum value and the LP solution of the Branch & Cut solver. The difference is due to the removal of inequalities with slack zero after an LP has been solved.

Now consider Table 6.1 and 6.2. These tables contain the computational results of the implemented Branch & Cut algorithm which is in the following abbreviated by "BC". In the first 4 columns there is the information about the problem instances. The density, "Den.", is the ratio of $\frac{2|E|}{|V|^2-|V|}$ and α is the stability number of the particular graph. The column "LB_{root}" provides the (global) lower bound in the root of the Branch & Bound tree. The next three columns give the LP solution value before branching. This is rounded down equal to the dual bound. The values of BC are stated next to the results of the algorithm of the literature. This is followed by the information about the number of nodes in the Branch & Bound tree which is the number of subproblems. The column "# LP" gives the number of solved linear programs. The last three columns provide the needed CPU time in seconds of the three algorithms. The graphs of Table 6.1 are the inverse graphs of the DIMACS clique benchmark graphs.

Let us look at the test results for the DIMACS graphs in Table 6.1. The comparison of the dual bounds of the three algorithms shows that BC is always better than BC_{clique}. This is not astonishing as BC_{clique} is mainly based on clique separation which is only one separation routine in BC. However, the results of BC_{clique} can be used to evaluate the other separation routines of BC, as one could expect that the results of BC would be similar to BC_{clique} if all other separation routines of BC were turned off. For the whole algorithm, it is more interesting to compare BC with the results of BC_{rank}. Observe that in most cases the dual bounds of BC are even better than the bounds of BC_{rank}. An exception builds the two graphs "C125.9" and "C250.9". The last graph could not even be solved to optimality by BC. The upper bound after 12 CPU hours was still 57 and the gap 25.5%. All four "brock" graphs have quite good bounds compared to BC_{rank}. The "c_fat" graphs are all solved in the root node. Especially "c_fat500-2" and "c_fat500-10" have a very good result. Interestingly, the number of computed inequalities is quite small, see Table 6.3. We again point out that all inequalities are checked. The LP with all computed inequalities for "c_fat500-2" has value 26.48 and the solution for "c_fat500-10" is integer feasible. The optimality proof for all "san" graphs can be done in the root node if the optimum is known. Unfortunately, the lower bounds for "san200_0.7_2", "san200_0.9_3" and "san400_0.5_1" differ from the optimum and the branching phase has to be entered. The gaps for the "p_hat300" graphs are quite big. Nevertheless they can both be solved to optimality.

Table 6.2: Computational results on uniform random graphs

Instances				Branch & Cut										
				LB_{root}	LP value in root			# B & B nodes			# LP	Time		
Name	$\|V\|$	Dens.	α	BC	BC	BC_{rank}	BC_{clique}	BC	BC_{rank}	BC_{clique}	BC	BC	BC_{rank}	BC_{clique}
G100_10	100	0.10	30.9	30.2	34.93	–	–	233	–	–	405	4	–	–
G100_20	100	0.20	19.9	19.3	25.73	–	–	346	–	–	763	12	–	–
G125_10	125	0.10	34.4	33.5	41.55	–	–	1634	–	–	2274	35	–	–
G125_20	125	0.20	21.7	20.5	30.40	–	–	1766	–	–	3552	116	–	–
G150_10	150	0.10	37.2	36.1	48.05	–	–	8078	–	–	12890	350	–	–
G150_20	150	0.20	23.2	21.7	34.46	–	–	8448	–	–	13994	1211	–	–
g100_10	100	0.10	31.4	–	–	33.30	36.42	–	16	323	–	–	4	5
g100_20	100	0.20	19.2	–	–	23.13	26.66	–	26	676	–	–	32	18
g125_10	125	0.10	33.3	–	–	36.92	42.87	–	144	4046	–	–	23	75
g125_20	125	0.20	21.2	–	–	26.07	31.11	–	222	3006	–	–	123	101
g150_10	150	0.10	36.8	–	–	42.14	48.14	–	803	19160	–	–	361	702
g150_20	150	0.20	22.3	–	–	30.34	35.54	–	1121	44081	–	–	475	976

– Data not available

To some extent the downside of the good upper bounds is an increase in the CPU time. The separation of the mod-k cuts has no effect on the upper bounds but were quite time consuming, see 6.3. The computation of the local cuts and the large rank inequalities are also very time consuming with only very little contribution to closing the gap.

The computational results of the uniform random graphs can be seen in Table 6.2. The graphs are randomly constructed where each edge is selected with a fixed probability to be in the graph. As the results of BC with BC$_{rank}$ and BC$_{clique}$ are not 100% comparable they are listed separately. Indeed, the average size of α differs. We recognize that the problems become harder to solve if the density grows from 0.1 to 0.2. This is confirmed by the results in literature, cf. [RS01, Sew98, MS96b]. The lower bounds turn out to be close to the optimal solution on average. However, the upper bounds are not so strong which explains the high number of subproblems and solved linear programs.

A first impression of the performance of the two heuristics is given through the value of "LB$_{root}$" in Table 6.1 and 6.2. It shows that in several cases the optimal solution is already found in the root of the Branch & Bound tree. This is very important for the total running time of the algorithm. The lower bounds not only affect the gap but also have a strong impact on the size of the branching tree. Especially for the random graphs the improvement heuristic is very effective and finds the optimal solution in many cases. For the graphs with 100 nodes and a probability of 10%, "G100-10", in 20 out of 25 cases the improvement heuristic found the optimum. From the much harder graphs "G150-20" still 13 instances were solved by this heuristic. In all other cases the optimum has been computed via an integer solution. This changes for the DIMACS graphs. Here, the rounding heuristic performed better than the improvement heuristic.

The preprocessing with Algorithm 5.1.2 fails for the presented graphs. Not a single node can be fixed. Only the two graphs "hamming6-2" and "hamming8-2" of the DIMACS graphs are solved to optimality in the preprocessing step (not shown in Table 6.1). It turns out that the clique fixing works for random graphs with up to 80 nodes and 0.15 density. The fixing due to the structure of the $\mathcal{LP}$ is only effective for bipartite graphs. However, for the computation of the large rank inequality this preprocessing is useful. As the graphs have a size equal to or less than 75 nodes and the graphs have a different structure than the original graphs, some nodes can be fixed which reduces the running time.

Now consider the generated cuts for all tested graphs which is shown in Table 6.3. It has already been mentioned that whenever an edge inequality is violated, a maximal clique containing this edge is computed. The number of these cliques is given in column two. In the next column the number of odd-hole inequalities with support greater than or equal to 5 can be seen. This is followed by the number of lifted 3-cycles to maximal cliques. The number of clique inequalities computed by the clique heuristics can be seen in column "clique". This is followed by the number of computed rank inequalities, mod-k cuts, local cuts and large inequalities. The smooth numbers in the table come from the restriction of the separation routines. For instance the clique separation is stopped if $|\mathrm{G}|/5$ inequalities have been computed at the latest. Each inequality counts only once in Table 6.3, even if it is computed several times.

Let us interpret the results. One observes that the number of odd-holes with size greater than 3 is not so large compared to the total number of computed odd-cycles, which is the sum of column three and four. In most cases the odd-cycles turn out to be triangles which can be lifted to maximal cliques. To get the total number of computed clique inequalities, one has to add the numbers given in columns two, three, and five. This shows that the total number of inequalities

Table 6.3: Number of generated cutting-planes for the tested graphs

Instances	# Inequalities							
Name	edge	hole	3-cycle	clique	rank	mod-k	local	large
brock200_2	3043	0	567	22476	23753	0	45	40
brock200_4	66426	3450	10537	52037	57399	0	121	40
brock400_2	322701	12716	138114	437211	325870	0	177	46
brock400_4	428397	15019	181752	591953	399548	0	205	23
c_fat200-1	100	0	8	400	0	0	0	0
c_fat200-2	102	0	0	800	0	0	0	0
c_fat200-5	2327	0	3092	6572	4056	0	0	0
c_fat500-1	250	0	0	3607	423	0	0	0
c_fat500-2	250	0	3	3600	0	0	0	0
c_fat500-5	504	0	51	2700	0	0	0	0
c_fat500-10	998	0	0	1199	0	0	0	0
C125.9	271	641	3	254	219	0	17	16
C250.9	861	148832	172	1434	15041	0	143	34
hamming8-4	112	0	32	488	0	0	0	0
keller4	1674	147	745	4612	11775	0	2	0
san200_0.7_1	184	0	0	549	0	0	0	0
san200_0.7_2	1849	57	82	2537	163	0	0	0
san200_0.9_1	177	0	0	217	0	0	0	0
san200_0.9_2	341	0	0	373	0	0	0	0
san200_0.9_3	922	100	0	250	0	0	0	0
san400_0.5_1	5399	4	112	61611	3699	0	0	0
san400_0.9_1	1120	0	0	1711	0	0	0	0
p_hat300-2	14093	1397	3308	34492	5761	0	109	60
p_hat300-3	351214	23087	5493	437683	31589	0	48	21
g100_10	239	241	1	87	30	0	3	0
g100_20	319	626	72	423	940	0	9	0
g125_10	303	762	3	208	117	0	16	15
g125_20	533	3194	163	781	2509	50	24	16
g150_10	367	3380	8	365	340	0	17	16
g150_20	788	12285	285	1364	8608	150	88	19

is close to the sum of computed clique inequalities and rank inequalities. An exception is the two graphs "C125.9" and "C250.9". The number of computed rank inequalities is quite high. One should remember that this separation is only called if the odd-cycle and clique separation could not compute enough inequalities. Not only the quantity of the generated clique and rank inequalities but also their quality is very high. Their impact in reducing the upper bound is much more important than all other generated inequalities. The maximally violated mod-$\{2, 3, 5, 7\}$ cuts separation is inefficient for the tested graphs. Only for random graphs with density 0.2 have they been helpful. Interestingly, it can be observed that the maximally violated mod-k cut separation finds violated cuts very late in the separation process. This means that if the tailing off is reduced and the instances get hard, this separation will compute cuts. However, these cuts are very weak as their effect on the LP solution is small. For the local cuts and the large rank cuts we recognize something similar. Their contribution to the reduction of the gap is very small compared to their running time.

Chapter 7

Summary and Outlook

In this thesis a strong Branch & Cut solver for the maximum stable set problem is proposed. To develop it we considered some polyhedral aspects of the stable set polytope and examined some of its facets. After a discussion of preprocessing we focused on the separation routines. The odd-hole separation was combined with a clique heuristic. We discussed the edge projection and introduced several new aspects. In particular, a sufficient criteria has been established for an inequality to be facet-defining after the anti-projection. As a special case we saw that, under some restrictions to the common neighborhood of the projected edge, the inequalities corresponding to maximal cliques are facet-defining after the anti-projection. This was confirmed by the computational results of the edge projection. In addition, two general separation methods for integer programs were tested. We also presented rounding and improving heuristics.

Let us summarize the main ideas of this thesis:

- Preprocessing by fixing of variables.
- A new improvement heuristic.
- Sophisticated lifting of 3-cycles to maximal cliques.
- First adaption and computational results on mod-k cuts and local cuts for the stable set problem.
- Improved edge projection algorithm compared to ROSSI and SMRIGLIO [RS01] along with new theoretical contributions to edge projection.

The implemented algorithm was tested on benchmark and random graphs. The computational results showed that the fraction of 3-cycles in the odd-hole separation is quite large, which supports the idea of lifting. The key computational results are the following:

- The upper bounds produced by the algorithm proposed in this thesis are considerably better than those in the state of the art Branch & Cut software discussed in [RS01].
- The improvement heuristic together with the rounding heuristic produced very satisfactory lower bounds.
- The lifting of 3-cycles is effective with regard to upper bounds. Moreover, it works very fast.

- This thesis gives evidence that mod-k cuts are of very little use for the stable set problem. The local cuts help to reduce the upper bounds a bit but their separation is very time consuming.
- The edge projection arguably provides good cuts and is an important part of the implemented Branch & Cut algorithm.

Further Branch & Cut solvers could focus on a combination of clique separation and edge projection. Another result of this thesis is that the odd-cycle separation should be replaced by an exact separation of the triangles only, which should then be lifted to maximal cliques. As the number of the computed rank inequalities is very high, a promising idea could be to strengthen these inequalities. Since the exact lifting method is practically inefficient, a clique covering should be used to compute better coefficients. Another idea could be to generate some clique coverings and store them. They could then be used to compute a covering for the induced subgraph of the inequalities. This would lead to a (non exact) lifting in linear time. A third approach could be to combine the edge projection with the separation of general inequalities.

We draw the conclusion that the maximum stable set problem is indeed very tough to solve. The computational results depend strongly on the structure of the underlying graphs. Inequalities for which combinatorial polynomial time separation algorithms are known, like odd holes, are so special in structure that the impact is very limited. On the other hand, general purpose methods, such as mod-k or local cut, which work well on other combinatorial optimization problems turn out to be inefficient for the stable set problem. Even though many investigations into the stable set problem have been made, the breakthrough is still missing and the stable set problem remains one of the major challenges in the field of combinatorial optimization.

List of Figures

List of Algorithms

List of Symbols

Bibliography

[AB62] G. Avondo-Bodeno. Economic applications of the theory of graphs. *Gordon and Breach, Science Publishers, New York*, 1962.

[aba] ABACUS—A Branch And CUt Solver, Version 2.3.0. http://www.informatik.uni-koeln.de/abacus/.

[ABCC01] David Applegate, Robert Bixby, Vašek Chvátal, and William Cook. TSP Cuts Which Do Not Conform to the Template Paradigm. *Computational Combinatorial Optimization*, LNCS 2241:157–222, 2001.

[ANS00] A. Atamtürk, G. L. Nemhauser, and M. W. P. Savelsbergh. Conflict graphs in solving integer programming problems. *European Journal of Operational Research*, 121:40–55, 2000.

[AS92] S. Arora and S. Safra. Probabilistic Checking of Proofs; a new Characterization of NP. In *Proceedings 33rd IEEE Symposium on Foundations of Computer Science*, pages 2–13. IEEE Computer Society, Los Angeles, CA, 1992.

[AvKS98] Pankaj K. Agarwal, Marc J. van Kreveld, and Subhash Suri. Label Placement by Maximum Independent Set in Rectangles. *Computational Geometry*, (3-4):209–218, 1998.

[Bal70] M. L. Balinski. On Maximum Matching, Minimum Covering and their Connections. In H.W. Kuhn, editor, *Proceedings of the Princeton symposium on mathematical programming*, pages 303–312. Princeton University Press, Princeton, N.J., 1970.

[BB82] Dana H. Ballard and Christopher M. Brown. *Computer Vision*. Prentice-Hall, Englewood Cliffs, New Jersey, 1982.

[BP76] Egon Balas and Manfred W. Padberg. Set Partitioning: A Survey. *SIAM Review*, 18:710–761, 1976.

[BP90] P. Berman and A. Pelc. Distributed Fault Diagnosis for Multiprocessor Systems. In *Proceedings of the 20th Annual International Symposium on Fault-Tolerant Computing, Newcastle, UK*, pages 340–346, 1990.

[Brø83] Arne Brønsted. *An introduction to Convex Polytopes*. Graduate Texts in Mathematics. Springer, 1983.

[BSSW05] Ralf Borndörfer, Uwe Schelten, Thomas Schlechte, and Steffen Weider. A Column Generation Approach to Airline Crew Scheduling. Technical Report ZIB Report 05-37, Zuse-Institut Berlin, 2005.

[BWE92] Francisco Barahona, Andres Weintraub, and Rafael Epstein. Habitat Dispersion in Forest Planning and the Stable Set Problem. *Operations Research*, 40:S14–S21, 1992.

[BY86] Egon Balas and Chang Sung Yu. Finding a Maximum Clique in an Arbitrary Graph. *Siam Journal on Computing*, 14(4):1054–1068, 1986.

[CC97] Eddi Cheng and William H. Cunningham. Wheel Inequalities for Stable Set Polytopes. *Mathematical Programming*, 77:389–421, 1997.

[CCL+05] Maria Chudnovsky, Gérard Cornuéjols, Xinming Liu, Paul Seymour, and Kristina Vušković. Recognizing Berge Graphs. *Combinatorica*, 25:143–186, March 2005.

[CCPS98] William J. Cook, William H. Cunningham, William R. Pulleyblank, and Alexander Schrijver. *Combinatorial Optimization*. Wiley-Interscience Series in Discrete Mathematics and Optimization. Johnson Wiley & Sons, Inc., 1998.

[CFL00] A. Caprara, M. Fiscetti, and A. N. Letchford. On the Separation of Maximally Violated mod-k Cuts. *Mathematical Programming*, 87(1):37–56, 2000.

[Chr97] Thomas Christof. *Low-Dimensional 0/1-Polytopes and Branch-and-Cut in Combinatorial Optimization*. PhD thesis, Ruprecht-Karls-Universität, Heidelberg, 1997.

[CLRS01] Thomas H. Cormen, Charles E. Leisserson, Ronald L. Riverst, and Clifford Stein. *Introduction to Algorithms, Second Edition*. The MIT Press, 2001.

[CP90] R. Carraghan and P. M. Pardalos. An Exact Algorithm for the Maximum Clique Problem. *Operations Research Letters*, 9:375–382, 1990.

[cpl] ILOG CPLEX, Version 8.100. http://www.ilog.com/products/cplex/.

[CRST04] Maria Chudnovsky, Neil Robertson, Paul Seymour, and Robin Thomas. The Strong Perfect Graph Theorem. Typescript, revised June 4, 2004.

[CS90] K. Corradi and S. Szabo. A Combinatorial Approach for Keller's Conjecture. *Periodica Mathematica Hungarica*, 21:95–100, 1990.

[Die00] Reinhard Diestel. *Graph Theory*. Electronic Edition 2000. Springer-Verlag New York 1997, 2000.

[dim] Second DIMACS Challenge, 1992/1993. http://mat.gsia.cmu.edu/challenge.html.

[dVV03] Sven de Vries and Rakesh V. Vohra. Combinatorial Auctions: A Survey. *INFORMS Journal on Computing*, 15(3):284–309, 2003.

[EGJR01] Matthias Elf, Carsten Gutwenger, Michael Jünger, and Giovanni Rinaldi. Branch-and-Cut Algorithms for Combinatorial Optimization and Their Implementation in ABACUS. *Computational Combinatorial Optimization*, LNCS 2241:157–222, 2001.

[FMK95] Katsuku Fujisawa, S. Morito, and Miko Kubo. Experimental Analyses of the Life Span Method for the Maximum Stable Set Problem. *The Institute of Statistical Mathematics Cooperative Research Report*, 75:135–165, 1995.

[Fri06] Lars Fricke. Personal communication, 2006.

[Gal05] Gábor Galambos. Graph Theory. Lecture notes. Institut für Informatik, Universität Heidelberg, 2005.

[GARW93] H. M. Grindley, P. J. Artymiuk, D. W. Rice, and P. Willett. Identification of Tertiary Structure Resemblance in Proteins using a Maximal Common Sub-graph Isomorphism Algorithm. *Journal of Molecular Biology*, 229:707–721, 1993.

[GAW98] E. J. Gardiner, P. J. Artymiuk, and P. Willett. Clique-detection Algorithms for Matching three-dimensional Molecular Structures. *Journal of Molecular Graphics and Modeling*, 15:245–253, 1998.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability, A guide to the Theory of NP-Completeness*. A series of books in the mathematical sciences, Vicot Klee, Editor. W. H. Freeman and Company, 1979.

[GJR84] M. Grötschel, M. Jünger, and G. Reinelt. A Cutting Plane Algorithm for the Linear Ordering Problem. *Operations Research*, 32:1195–1220, 1984.

[GL06] Monia Giandomenico and Adam N. Letchford. Exploring the Relationship Between Max-Cut and Stable Set Relaxations. *Mathematical Programming*, 106:159–175, 2006.

[GLS88] Martin Grötschel, László Lovász, and Alexander Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Algorithms and Combinatorics 2. Springer, 1988.

[GP81] M. Grötschel and W. R. Pulleyblank. Weakly Bipartite Graphs and the Max-cut Problem. *Operations research letters*, 1:23–27, October 1981.

[GS86] A. M. H. Gerards and A. Schrijver. Matrices with the Edmonds-Johnson property. *Combinatorica*, 6:365–379, 1986.

[HS89] R. Horaud and T. Skordas. Stereo correspondence through feature grouping and maximal cliques. *IEEE Trans. Pattern Anal. Mach. Intell.*, 11(11):1168–1180, 1989.

[HV77] David J. Houck and R. Rao Vemuganti. An Algorithm for the Vertex Packing Problem. *Operations Research*, 26:773–787, September-October 1977.

[Igl01] Ioannis Iglezakis. The Conflict Graph for Maintaining Case-Based Reasoning Systems. *Proceedings of the 4th International Conference on Case-Based Reasoning (ICCBR), LNAI 2080, Springer*, pages 263–275, 2001.

[JSW97] K. Jansen, P. Scheffler, and G. Woeginger. The Disjoint Cliques Problem. *Operations Research*, 31:45–66, 1997.

[Kar84] N. Karmarkar. A new Polynomial Time Algorithm for Linear Programming. *Combinatorica*, 4:375–395, 1984.

[Kos04] Ekaterina Kostina. Algorithmische Optimierung II. Lecture notes. Proceeded by Christian Kirches and Joachim Ferreau. Lehrstuhl für Simulation und Optimierung, Universität Heidelberg, July 2004.

[KV91] Bernhard Korte and Jens Vygen. *Combinatorial Optimization: Theory and Algorithms*, volume 21 of *Algorithms and Combinatorics*. Springer, 1991.

[KW97] Josef Kallrath and John M. Wilson. *Business Optimization using Mathematical Programming*. Macmillan, 1997.

[MARW89] E. M. Mitchell, P. J. Artymiuk, D. W. Rice, and P. Willett. Use of Techniques derived from Graph Theory to compare Secondary Structure Motifs in Proteins. *Journal of Molecular Biology*, 212:151–166, 1989.

[MS96a] F. J. MacWilliams and N. J. A. Sloane, editors. *The Theory of Error-Correcting Codes*. North-Holland Mathematical Library Vol.16, 1996.

[MS96b] C. Mannino and A. Sassano. Edge Projection and the Maximum Cardinality Stable Set Problem. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 26:249–261, 1996.

[MT96] Anuj Mehrotra and Michael A. Trick. A Column Generation Approach for Graph Coloring. *INFORMS Journal on Computing*, 8(4):344–354, 1996.

[NT74] G. L. Nemhauser and L. E. Trotter, Jr. Properties of Vertex Packing and Independence System Polyhedra. *Mathematical Programming*, 6:48–61, 1974.

[NT75] G. L. Nemhauser and L. E. Trotter, Jr. Vertex Packings: Structural Properties and Algorithms. *Mathematical Programming*, 8:232–248, 1975.

[NW88] G. L. Nemhauser and Laurence A. Wolsey. *Integer and Combinatorial Optimization*. Wiley-Interscience Series in Discrete Mathematics and Optimization. Johnson Wiley & Sons, Inc., 1988.

[Osw05] Marcus Oswald. Personal communication, 2005.

[Pad73] Manfred W. Padberg. On the Facial Structure of Set Packing Polyhedra. *Mathematical Programming*, 5:199–215, 1973.

[Pie70] J. F. Pierce. Pattern sequencing and matching in stock cutting operations. *Tappi*, 53:668–678, 1970.

[por] PORTA—POlyhedron Representation Transformation Algorithm, Version 1.4.0. http://www.informatik.uni-heidelberg.de/groups/comopt/software/PORTA.

[PR87] M. W. Padberg and G. Rinaldi. Optimization of a 532 City Symmetric Traveling Salesman Problem by Branch and Cut. *Operations Research Letters*, 6:1–7, 1987.

[RAR01] Jorge L. Ramírez-Alfonsín and Bruce A. Reed, editors. *Perfect Graphs*. John Wiley & Sons, 2001.

[Rei01] Gerhard Reinelt. Effiziente Algorithmen 1. Lecture notes. Institut für Informatik, Universität Heidelberg, 2001.

[RS01] Fabrizio Rossi and Stefano Smriglio. A Branch-and-Cut Algorithm for the Maximum Cardinality Stable Set Problem. *Operations Research Letters*, 28:63–74, 2001.

[RS06] Fabrizio Rossi and Stefano Smriglio. Personal communication, 2006.

[SBS05] Dawn M. Strickland, Earl Barnes, and Joel S. Sokol. Optimal Protein Structure Alignment Using Maximum Cliques. *Operations Research*, 53:389–402, 2005.

[Sch00] Alexander Schrijver. *Theory of Linear and Integer Programming*. John Wiley & Sons, 2000.

[Sch03] Alexander Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*, volume 24 of *Algorithms and Combinatorics*. Springer, 2003.

[Sew98] E. C. Sewell. A Branch and Bound Algorithm for the Stability Number of a Sparse Graph. *INFORMS Journal on Computing*, 10(4):438–447, 1998.

[SVA00] T. W. Strijk, A. M. Verweij, and K. I. Aardal. Algorithms for maximum independent set applied to map labelling. Technical Report UU-CS-2000-22, Institute of Information and Computing Sciences, Utrecht University, 2000.

[The05] Dirk Oliver Theis. *Polyhedra and Algorithms for the General Routing Problem*. PhD thesis, Ruprecht-Karls-Universität, Heidelberg, 2005.

[VP95] M. Vingron and P. Pevzner. Multiple Sequence Comparison and Consistency on Multipartite Graphs. *Advances in Applied Mathematics*, 16:1–22, 1995.

[Wes00] Douglas B. West. *Introduction to Graph Theory, Second Edition*. Prentice Hall, 2000.

[Zie95] Günter M. Ziegler. *Lecture on Polytopes*. Graduate Texts in Mathematics. Springer, 1995.

Index