Cutting Planes for the Traveling Salesman Problem

Felipe Manuel Lorenzo Martínez
Born 31st of May 2003 in Boadilla del Monte, Madrid, Spain

15th July 2024

Bachelor's Thesis Mathematics

Advisor: Prof. Dr. Stefan Hougardy

Second Advisor: Prof. Dr. Stephan Held

Research Institute for Discrete Mathematics

Mathematisch-Naturwissenschaftliche Fakultät der

Rheinischen Friedrich-Wilhelms-Universität Bonn

Contents

1 Abstract

1.1 Zusammenfassung

Beim Traveling-Salesman-Problem (TSP) geht es darum, aus einer Liste von Städten den kürzesten Weg zu finden, um jede Stadt genau einmal zu besuchen und dort zu enden, wo man gestartet ist. Obwohl dieses Problem relativ trivial erscheinen mag, insbesondere bei einer kleineren Anzahl von Städten, ist es als NP-schweres Problem bekannt, für das seit fast einem Jahrhundert nach einer effizienten Lösungsmethode gesucht wird.

Einer der erfolgreichsten Ansätze für dieses Problem wurde von Dantzig, Fulkerson und Johnson [DFJ54] entwickelt. Sie schlugen vor, dieses Problem auf ein allgemeineres Problem der linearen Programmierung zu reduzieren, indem sie die Bedingung der Eliminierung von Untertouren einführten. Obwohl dieser Algorithmus das Problem sehr effizient löst, müssen wir eine ganzzahlige lineare Programmierung verwenden, um eine gültige Lösung zu garantieren. Auch dieses Problem ist bekanntermaßen NP-schwer. Wir können einen polynomialen Algorithmus erhalten, indem wir die Integralitätsbedingung weglassen. In einigen Fällen erhalten wir dann eine optimale Lösung, aber in anderen Fällen erhalten wir eine nicht ganzzahlige Lösung, die für das TSP ungültig ist.

Das ursprüngliche Modell von Dantzig, Fulkerson und Johnson wurde durch zusätzliche Bedingungen ergänzt, um die Wahrscheinlichkeit einer ganzzahligen Lösung zu erhöhen. Eine davon ist die Verwendung von Comb-Ungleichungen, die von Grötschel und Padberg [GP79] eingeführt wurden, allerdings ist kein polynomieller Algorithmus zur Erzeugung dieser Ungleichungen bekannt. Chvátal [Chv72] definierte auch eine Unterklasse dieser Ungleichungen, die als Chvátal Combs bekannt sind und effizienter generiert werden können, allerdings bieten sie eine geringere Wahrscheinlichkeit einer gültigen Lösung. Grötschel und Pulleyblank [GP86] definierten auch eine allgemeinere Klasse von Ungleichungen, die Clique-Tree-Ungleichungen, die das Konzept der Combs verallgemeinern. Diese Ungleichungen haben sich als sehr effizient für die Lösung des TSP erwiesen, sind aber auch sehr langsam zu generieren.

In den 90er Jahren wurde das Programm Concorde entwickelt, das eine Mischung aus verschiedenen Algorithmen verwendet, um TSP-Instanzen so effizient wie möglich zu lösen. Dieses Programm ist bis heute das effizienteste, da es eine ausgeklügelte Form von Branch and Bound verwendet, um exakte optimale Lösungen für das Traveling Salesman Problem zu erzeugen.

In dieser Arbeit arbeiten wir an einer Methode zur Lösung des TSP-Problems mit Hilfe von Cutting Planes. Dabei geht es nicht um die Lösung des TSP selbst, sondern vielmehr um den Vergleich der Lösungsräume, die sich durch die Verwendung verschiedener Klassen an Cutting Planes ergeben. Aus diesem Grund werden wir uns für exakte Brute-Force-Algorithmen entscheiden, da unser Schwerpunkt auf der Korrektheit der Lösung und nicht auf der Geschwindigkeit liegt, mit der wir sie berechnen können.

Diese Arbeit ist stark von der gleichnamigen Arbeit von Laura Bülte beeinflusst, in

der sie sich erstmals dieser Frage näherte, es aber aufgrund der Verwendung von Separationsheuristiken nicht schaffte, diese Zusammenhänge schlüssig zu zeigen. Wir werden uns bemühen, diese Implementierungsfehler zu korrigieren und die meisten der ursprünglich gegebenen Beispiele im Sinne ihrer Minimalität zu verbessern.

In Kapitel 2 werden wir das Traveling Salesman Problem und die Methode der Cutting Planes vorstellen. In Kapitel 3 werden wir Combs und Comb-Ungleichungen sowie die verschiedenen Unterklassen von Combs vorstellen. Wir werden auch zeigen, dass das Polytop der Lösungen für diese jeweiligen Comb-Ungleichungen streng ineinander enthalten sind, indem wir Beispiele geben, die sie voneinander trennen. In Kapitel 4 werden wir die Verallgemeinerung von Combs, die Clique Trees, und die Clique-Tree-Ungleichungen einführen. Wir werden auch ein Beispiel geben, um das Polytop, das von den Lösungen der Clique-Tree-Ungleichungen erzeugt wird, von dem der Comb-Ungleichungen zu trennen. In Kapitel 5 werden wir dann beweisen, dass alle zuvor gegebenen Beispiele minimal sind, indem wir das Konzept der trivial minimalen Beispiele einführen. In Kapitel 6 beschreiben wir die Methodik, die zur Erlangung der vorherigen Ergebnisse verwendet wurde, und liefern eine Beschreibung des implementierten Programms zum Auffinden von Combs und Clique Trees durch Enumeration. Kapitel 7 besteht aus weiteren Beispielen, die die in den Kapiteln 3 und 4 gegebenen Beispiele widerspiegeln, aber diesmal für das euklidische und metrische Traveling Salesman Problem.

1.2 Abstract

The Traveling Salesman Problem (TSP) asks the question of, given a list of cities, finding the shortest path to visit every city exactly once, ending where you started. Although this problem might seem relatively trivial, especially for a smaller amount of cities, it is known as an NP-hard problem, one for which an efficient solving method has been sought for the better part of the last century.

One of the most successful approaches to this problem was proposed by Dantzig, Fulkerson and Johnson [DFJ54]. They suggested to reduce this problem to a more general Linear Programming Problem by introducing the use of the subtour elimination constraint. Although this algorithm solves the problem very efficiently, it requires us to use Integer Linear Programming to guarantee a valid solution. This is also known to be an NP-hard problem. We can obtain a polynomial algorithm by leaving out the integrality constraint, and in some cases it will give us an optimal solution, however in others we will get a non-integral solution which will be invalid for the TSP. There have been additions to the original model by Dantzig, Fulkerson and Johnson that add additional constraints to increase the likelihood of achieving an integer solution. One of these are the use of comb inequalities introduced by Grötschel and Padberg [GP79], however no polynomial algorithm is known for generating these inequalities. Chvátal [Chv72] also defined a subclass of these inequalities known as Chvátal combs which can be generated more efficiently, however they provide a lesser likelihood of a valid solution. Grötschel and Pulleyblank [GP86] also defined a more general class of inequalities called clique tree inequalities which generalize the concept of combs. These have been proven to be very efficient for solving the TSP, however they are also very slow to generate.

In the 90s the program Concorde was developed which uses a mix of different algorithms for solving TSP instances as efficiently as possible. This program remains the most efficient one to date, using a sophisticated form of branch and bound to generate exact optimal solutions for the Traveling Salesman Problem.

In this thesis we will be working on a method for solving the TSP problem using Cutting Planes. We will not be interested in solving the TSP itself, but rather in the comparison of the solution spaces offered by the use of different cutting planes. For this reason we will be opting for more brute-force exact algorithms, as our focus will be on the correctness of the solution rather than the speed at which we can compute it.

This thesis is heavily influenced by Laura Bülte's work with the same title [Bül22] in which she first approached this question, however due to the use of separation heuristics she did not manage to conclusively show these relations. We will aim to correct these implementation errors as well as improving on most of the examples originally provided in the sense of their minimality.

In chapter 2 we will introduce the Traveling Salesman Problem and the cutting plane method. In chapter 3 we will introduce combs and comb inequalities, as well as the different subclasses of combs. We will also show that the polytope of solutions for these respective comb inequalities are strictly contained in each other by providing examples that separate them from each other. In chapter 4 we will introduce the generalization of combs known as clique trees and the clique tree inequalities, we will also provide an example to separate the polytope generated by the solutions of the clique tree inequalities from that of the comb inequalities. We will then prove in chapter 5,that all previous examples provided are minimal by introducing the concept of trivially minimal examples. In chapter 6 we will describe the methodology used for obtaining the previous results, providing a description of the program implemented for finding combs and clique trees through enumeration. Chapter 7 consists of further examples mirroring the ones given in chapters 3 and 4 but this time for the Euclidean and metric Traveling Salesman Problem.

2 Introduction

In this chapter we will be introducing the Traveling Salesman Problem and the motivation for the Cutting Planes approach analyzed in this thesis.

2.1 The Traveling Salesman Problem

Definition 2.1.1 (Traveling Salesman Problem)**.** Given a complete graph K_n and weights $c : E(K_n) \to \mathbb{R}_+$ find the Hamiltonian circuit T whose weight $\sum_{e \in E(T)} c(e)$ is minimal.

We will provide the following alternate definition of the Traveling Salesman Problem as proposed by Dantzig, Fulkerson and Johnson [DFJ54]:

Proposition 2.1.2 (TSP as Integer Linear Programming Problem)**.** *For a complete graph K_n and weights $c : E(K_n) \to \mathbb{R}_+$ the following Integer Linear Programming Problem is equivalent to the Traveling Salesman Problem:*

Let $x \in \mathbb{R}^{|E(K_n)|}$
$\min \sum_{e \in E(K_n)} c(e)x_e$ *s.t.:*

$$\sum_{e \in \delta(v)} x_e = 2 \qquad \forall v \in V(K_n) \tag{I}$$

$$\sum_{e \in E(K_n[S])} x_e \leq |S| - 1 \qquad \forall \emptyset \neq S \subsetneq V(K_n) \tag{II}$$

$$x_e \in \{0, 1\} \qquad \forall e \in E(K_n) \tag{III}$$

Where the edges $e \in E(K_n)$ such that $x_e = 1$ form the Hamiltonian circuit T.

Constraint I is called the degree constraint and ensures that our solution forms a tour (or multiple subtours). Constraint II is known as the subtour elimination constraint and ensures that no subset $S \subsetneq V(K_n)$ forms a sub-tour, making the resulting solution non-connected. Constraint III is the integrality constraint and ensures the integrality of our solution so that every edge $e \in E(K_n)$ is either included in the circuit T for $x_e = 1$ or excluded for $x_e = 0$.

The Traveling Salesman Problem or TSP is known to be strongly NP-hard; there also exists no possible p-factor approximation algorithm for all $p \geq 1$, assuming P $\neq$ NP, as any p-Approximation is also known to be NP-hard [KV17].

Definition 2.1.3 (Polyhedron/Polytope [KV17])**.** A polyhedron in $\mathbb{R}^n$ is a set of the form $P = \{x \in \mathbb{R}^n : Ax \leq b\}$ for some matrix $A \in \mathbb{R}^{m \times n}$ and some vector $b \in \mathbb{R}^m$. A bounded polyhedron is also called a polytope.

Definition 2.1.4 (Convex hull [KV17])**.** The convex hull $\text{conv}(X)$ of a set X is defined as the set of all convex combinations of points in X.

We will define the solution space polytope of the Traveling Salesman Problem as an Integer Linear Programming Problem:
$P^n_{TSP} \coloneqq \text{conv}\{x \in \mathbb{R}^{|E|} \mid x \text{ is the incidence vector of a Hamiltonian circuit}\}$.

2.2 The Cutting Plane Method

The cutting plane method is an iterative algorithm used to solve linear programming problems by successively refining the feasible region through the addition of linear inequalities, known as cuts. These cuts are designed to exclude portions of the solution space that do not contain optimal integer solutions, thereby progressively tightening the bounds on the feasible region until an optimal integer solution is identified. As long as our optimal solution x^* is invalid (in our case non-integer for example), there will be an inequality $cx \leq \delta$ that separates x^* from all valid solutions of $Ax \leq b$. We can add this inequality to our system and continue to the next iteration. Using this method we can solve the original inequality system $Ax \leq b$ using only a fraction of the original inequalities. In order to find these inequalities we can observe the following problem:

Definition 2.2.1 (Separation Problem)**.** For a given $x^* \in \mathbb{R}^n$ and an inequality system $Ax \leq b$ find an inequality $cx \leq \delta$ from $Ax \leq b$ so that $cx^* > \delta$ or determine that $Ax^* \leq b$.

When observing a problem and its relaxation we then come to the question of finding a class of inequalities that are fulfilled by all solutions of the original problem but may leave out certain solutions of its relaxation.

Our starting point will be the integrality-relaxation of the P^n_{TSP} Polytope.

Definition 2.2.2 (Subtour Elimination Polytope)**.** The Subtour Elimination Polytope P^n_{SEP} is defined as the solution space of the following Linear Programming Problem:

Let $x \in \mathbb{R}^{|E(K_n)|}$
$\min \sum_{e \in E(K_n)} c(e)x_e$ s.t.:

$$\sum_{e \in \delta(v)} x_e = 2 \qquad \forall v \in V(K_n) \tag{I}$$

$$\sum_{e \in E(K_n[S])} x_e \leq |S| - 1 \qquad \forall \emptyset \neq S \subsetneq V(K_n) \tag{II}$$

$$0 \leq x_e \leq 1 \qquad \forall e \in E(K_n) \tag{IV}$$

Normally constraint II is not directly solved, as there are an exponential number of such inequalities. The Cutting Plane Method is used to only solve the necessary number of subtour elimination inequalities. In our case we will be starting directly with all subtour elimination inequalities, as the examples we will be seeing are small enough and the subtour elimination ineqalities are greatly outnumbered by the other constraints we will be seeing.

3 Combs and Comb Inequalities

In this chapter we will be introducing different types of Comb inequalities as defined by Grötschel and Padberg [GP79].

3.1 Combs

Definition 3.1.1 ((General) Comb)**.** Let $H, T_1, \ldots, T_k \subset V(K_n)$. $C \coloneqq E[H] \cup \bigcup_{i=1}^{k} E[T_i]$ is a comb if:

(i) $|H \cap T_i| \geq 1$ for $i = 1, \ldots, k$

(ii) $|T_i \backslash H| \geq 1$ for $i = 1, \ldots, k$

(iii) $|T_i \cap T_j| = 0$ for $1 \leq i < j \leq k$

(iv) $k \geq 3$ odd.

Although (iv) was originally defined as k only being odd, Grötschel and Padberg [GP79] also show that the comb inequalities for $k = 1$ are redundant. For the sake of brevity we will refer to the comb as $C \coloneqq (H, T_1, \ldots, T_k)$ in most cases.

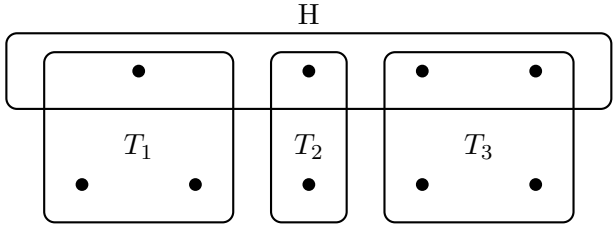

Figure 3.1: A standard comb with three teeth and a handle

Definition 3.1.2 (Comb Inequality)**.** Every comb defined by a handle H and teeth T_i for $i = 1, \ldots, k$ generates a comb inequality defined as:

$$x(H) + \sum_{i=1}^{k} x(T_i) \leq |H| + \sum_{i=1}^{k} (|T_i| - 1) - \frac{k+1}{2} \tag{3.1}$$

for $x \in \mathbb{R}^{E(K_n)}$, where $x(S)$ corresponds to $\sum_{e \in E[S]} x_e$ for $S \subseteq V(K_n)$.

We will define the Comb Polytope as:

$$P_C^n \coloneqq \{x \in P_{SEP}^n \mid x \text{ satisfies all comb inequalities in } K_n\}.$$

This is the Polytope generated by adding all comb inequalities induced by a comb in K_n to the Subtour Elimination Polytope.

Theorem 3.1.3 ([GP79])**.** *Every comb inequality is satisfied by every $x_{TSP} \in P^n_{TSP}$. Therefore $P^n_{TSP} \subseteq P^n_C$.*

Proof. Consider $x_{TSP} \in P^n_{TSP}$ and a comb $C = (H, T_1, \ldots, T_k)$. Let $x_{TSP}(S)$ denote the sum of the values of x_{TSP} in $E[S]$ for $S \subseteq V(K_n)$. Let $x_{TSP}(S')$ denote the sum of the values of x_{TSP} in S' for $S' \subseteq E(K_n)$. Then the following is true:

$$
\begin{aligned}
&2\left(x_{TSP}(H) + \sum_{i=1}^{k} x_{TSP}(T_i)\right) \\
&= 2x_{TSP}(H) + \sum_{i=1}^{k} \left(x_{TSP}(T_i) + x_{TSP}(T_i/H) + x_{TSP}(T_i \cap H) + x_{TSP}(\delta(H) \cap E(T_i))\right) \\
&= 2x_{TSP}\left(H\right) + \sum_{i=1}^{k} x_{TSP}\left(\delta(H) \cap E(T_i)\right) + \sum_{i=1}^{k} \left(x_{TSP}(T_i) + x_{TSP}(T_i/H) + x_{TSP}(T_i \cap H)\right) \\
&\leq \sum_{v \in H} x_{TSP}(\delta(v)) + \sum_{i=1}^{k} \left(x_{TSP}(T_i) + x_{TSP}(T_i/H) + x_{TSP}(T_i \cap H)\right) \\
&\leq \sum_{v \in H} 2 + \sum_{i=1}^{k} \left(|T_i| - 1 + |T_i/H| - 1 + |T_i \cap H| - 1\right) \\
&\leq 2|H| + \sum_{i=1}^{k} \left(|T_i| - 1 + |T_i/H| - 1 + |T_i \cap H| - 1\right) \\
&= 2\left(|H| + \sum_{i=1}^{k} \left(|T_i| - 1\right)\right) - k
\end{aligned}
$$

Here we first divided the second instance of $x_{TSP}(T_i)$ into the edges in the head, outside the head and connecting the two. Then we used the degree constraint I on the left and the subtour constraint II on the right.

After dividing by two we get the following inequality:

$$x_{TSP}(H) + \sum_{i=1}^{k} x_{TSP}(T_i) \leq |H| + \sum_{i=1}^{k} \left(|T_i| - 1\right) - \frac{k}{2}$$

Due to the integrality of every solution of the Traveling Salesman Problem, the left side of the inequality will always be integral for these solutions. Consequently we can round down the right side of the inequality while mantaining its validity for these solutions. P^n_{TSP} is defined as the convex hull of these solution and therefore these rounded down inequalities will also be valid for every $x_{TSP} \in P^n_{TSP}$. Since $-\frac{k}{2}$ is the only non-integral component we only need to round down this component.

$$x_{TSP}(H) + \sum_{i=1}^{k} x_{TSP}(T_i) \leq |H| + \sum_{i=1}^{k} (|T_i| - 1) - \frac{k+1}{2}$$

□

There exists no separation algorithm to date for efficiently separating combs in polynomial time. We will see further types of combs that are easier to separate but provide a worse (average-case) integrality gap. However we are not interested in a polynomial time separation and will be enumerating all combs in exponential time.

3.2 Simple Combs

Definition 3.2.1 (Simple Comb)**.** A simple comb is a comb $C = (H, T_1, \ldots, T_k)$ such that:

(i) $|H \cap T_i| \geq 1$ for $i = 1, \ldots, k$

(ii) $|T_i \backslash H| \geq 1$ for $i = 1, \ldots, k$

(iii) $|H \cap T_i| = 1$ or $|T_i \backslash H| = 1$ for $i = 1, \ldots, k$

(iv) $|T_i \cap T_j| = 0$ for $1 \leq i < j \leq k$

(v) $k \geq 3$ odd.

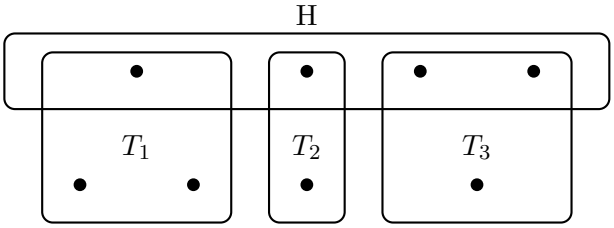

Figure 3.2: A simple comb with three teeth and a handle

We call inequality 3.1 a simple comb inequality if it is generated by a simple comb. We will define the Simple Comb Polytope as:

$$P^n_{SC} \coloneqq \{x \in P^n_{SEP} \mid x \text{ satisfies all simple comb inequalities in } K_n\}.$$

This is the polytope generated by adding all simple comb inequalities induced by a simple comb in K_n to the Subtour Elimination Polytope.

Theorem 3.2.2. $P_C^n \subsetneq P_{SC}^n$.

Proof. It is trivial that $P_C^n \subseteq P_{SC}^n$ as every simple comb is also a comb. Therefore it is sufficient to show that $P_{SC}^n \neq P_C^n$

Observe the following example proposed by Bülte [Bül22].

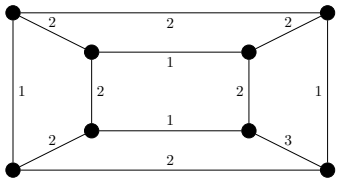

Figure 3.3: Weights of the graph

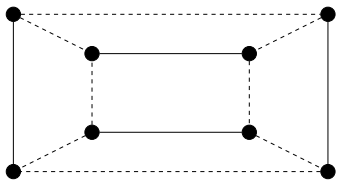

Figure 3.4: Solution with simple combs

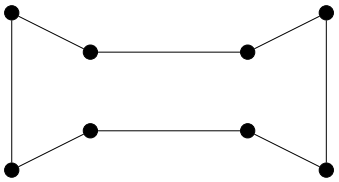

Figure 3.5: Solution with combs

In figure 3.3 we can see the weights of the edges where all non-depicted edges have the weight defined by the corresponding minimal path on the graph. In figure 3.4 we can see the optimal solution $x_{SC} \in P_{SC}^n$ for the Simple Comb Polytope, where every dashed edge represents a value of 0.5. In total these add up to a value of 12.5. In figure 3.5 we have the optimal solution $x_C \in P_C^n$ for the Comb Polytope, which has a value of 13.

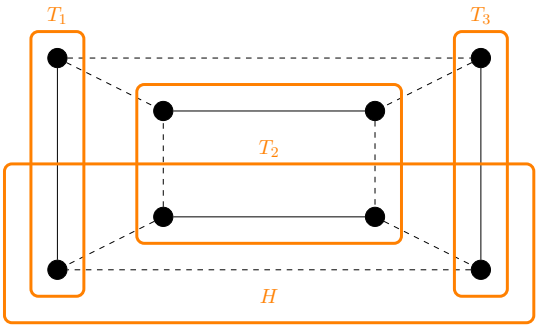

Figure 3.6: A comb with an unsatisfied inequality

As we can see in figure 3.6 the solution x_{SC} is not a valid solution for P_C^n. We have the values $x_{SC}(H) = 2.5$, $x_{SC}(T_1) = 1$, $x_{SC}(T_2) = 3$, $x_{SC}(T_3) = 1$, however as for the cardinality we have $|H| = 4$, $|T_1| = 2$, $|T_2| = 4$, $|T_3| = 2$. The inequality looks as follows:

$$x_{SC}(H) + \sum_{i=1}^{3} x_{SC}(T_i) = 7.5 > 7 = |H| + \sum_{i=1}^{3} (|T_i| - 1) - \frac{3+1}{2}.$$

By adding this inequality as a cutting plane to P_{SC}^n we get the optimal solution x_C with value 13. This example shows $P_{SC}^n \neq P_C^n$ and $P_C^n \subsetneq P_{SC}^n$ follows. □

In chapter 4 we will see that this example is also minimal; in the appendix we can also find an example for the euclidean TSP, proving that this theorem still holds when only considering Euclidean instances.

3.3 Chvátal Combs

Definition 3.3.1 (Chvátal Comb)**.** A Chvátal comb is a comb $C = (H, T_1, \ldots, T_k)$ such that:

(i) $|H \cap T_i| = 1$ for $i = 1, \ldots, k$

(ii) $|T_i \backslash H| \geq 1$ for $i = 1, \ldots, k$

(iii) $|T_i \cap T_j| = 0$ for $1 \leq i < j \leq k$

(iv) $k \geq 3$ odd.

This is not the original definition by Chvátal [Chv72], as it was originally defined without properties (iii) and (iv). However these have been added to conform better with our definition of a general comb. This makes no difference when defining the Chvátal Comb Polytope as shown by Grötschel and Padberg [GP79], as all Chvátal comb inequalities generated by a Chvátal comb with at least two teeth with a non-empty cut are redundant when considering only solutions $x \in P^n_{SEP}$.

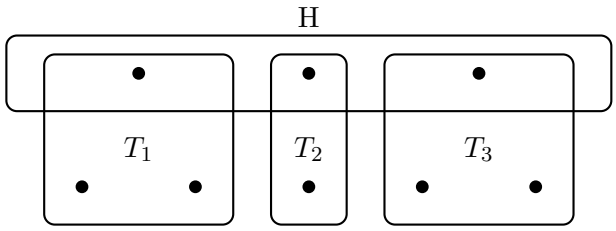

Figure 3.7: A Chvátal comb with three teeth and a handle

We call inequality 3.1 a Chvátal comb inequality if it is generated by a Chvátal comb. We will define the Chvátal Comb Polytope as:

$$P^n_{CC} \coloneqq \{x \in P^n_{SEP} \mid x \text{ satisfies all Chvátal comb inequalities in } K_n\}.$$

This is the Polytope generated by adding all Chvátal comb inequalities induced by a Chvátal comb in K_n to the Subtour Elimination Polytope.

Theorem 3.3.2. $P^n_{SC} \subsetneq P^n_{CC}$.

Proof. It is trivial that $P^n_{SC} \subseteq P^n_{CC}$ as every Chvátal comb is also a simple comb. Therefore it is sufficient to show that $P^n_{CC} \neq P^n_{SC}$.

Observe the example in the following page.

In figure 3.8 we can see the weights of the edges where all non-depicted edges have a set weight of 10. In figure 3.9 we can see the optimal solution $x_{CC} \in P^n_{CC}$ for the Chvátal Comb Polytope, where every dashed edge represents a value of 0.5. In total these add up to a value of 15. In figure 3.10 we have the optimal solution $x_{SC} \in P^n_{SC}$ for the Simple Comb Polytope, which has a value of 16.

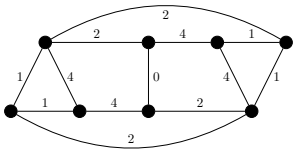

Figure 3.8: Weights of the graph

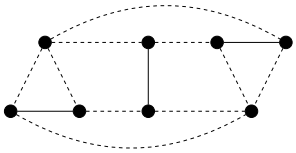

Figure 3.9: Solution with Chvátal combs

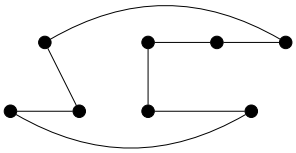

Figure 3.10: Solution with simple combs

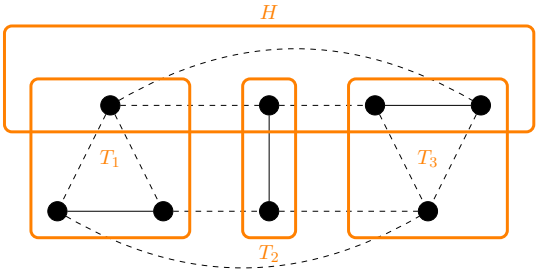

Figure 3.11: A comb with an unsatisfied inequality

As we can see in figure 3.11 the solution x_{CC} is not a valid solution for P^n_{SC}. We have the values $x_{CC}(H) = 2.5$, $x_{CC}(T_1) = 2$, $x_{CC}(T_2) = 1$, $x_{CC}(T_3) = 2$, however as for the cardinality we have $|H| = 4$, $|T_1| = 3$, $|T_2| = 2$, $|T_3| = 3$. The inequality looks as follows:

$$x_{CC}(H) + \sum_{i=1}^{3} x_{CC}(T_i) = 7.5 > 7 = |H| + \sum_{i=1}^{3} (|T_i| - 1) - \frac{3+1}{2}.$$

By adding this inequality as a cutting plane to P^n_{SC} (and several more iterations) we get the optimal solution x_{SC} with value 16. This example shows $P^n_{CC} \neq P^n_{SC}$ and $P^n_{SC} \subsetneq P^n_{CC}$ follows. □

3.4 Blossom Combs

Definition 3.4.1 (Blossom Comb)**.** A blossom comb is a comb $C = (H, T_1, \ldots, T_k)$ such that:

(i) $|H \cap T_i| = 1$ for $i = 1, \ldots, k$

(ii) $|T_i| = 2$ for $i = 1, \ldots, k$

(iii) $|T_i \cap T_j| = 0$ for $1 \leq i < j \leq k$

(iv) $k \geq 3$ odd.

We call inequality 3.1 a blossom comb inequality if it is generated by a blossom comb. We will define the Blossom Comb Polytope as:

$$P^n_{BC} \coloneqq \{x \in P^n_{SEP} \mid x \text{ satisfies all blossom comb inequalities in } K_n\}.$$

This is the Polytope generated by adding all blossom comb inequalities induced by a blossom comb in K_n to the Subtour Elimination Polytope.

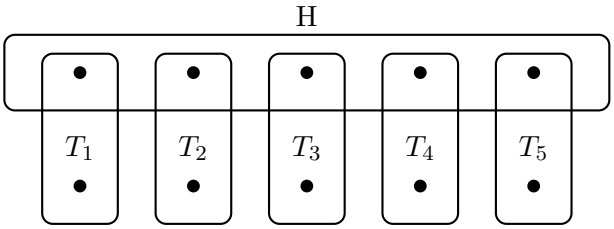

Figure 3.12: A blossom comb with five teeth and a handle

Theorem 3.4.2. $P^n_{CC} \subsetneq P^n_{BC}$.

Proof. It is trivial that $P^n_{CC} \subseteq P^n_{BC}$ as every blossom comb is also a Chvátal comb. Therefore it is sufficient to show that $P^n_{BC} \neq P^n_{CC}$

Observe the following example proposed by Bülte [Bül22].

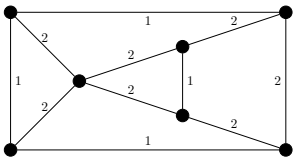

Figure 3.13: Weights of the graph

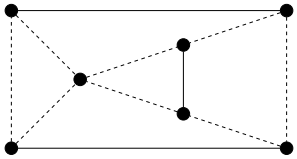

Figure 3.14: Solution with blossom combs

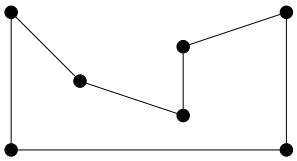

Figure 3.15: Solution with Chvátal combs

In figure 3.13 we can see the weights of the edges where all non-depicted edges the weight of the corresponding shortest path in the graph. In figure 3.14 we can see the optimal solution $x_{BC} \in P^n_{BC}$ for the Blossom Comb Polytope, where every dashed edge represents a value of 0.5. In total these add up to a value of 10.5. In figure 3.15 we have the optimal solution $x_{CC} \in P^n_{CC}$ for the Chvátal Comb Polytope, which has a value of 11.

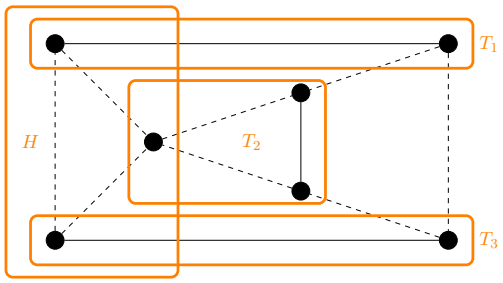

Figure 3.16: A comb with an unsatisfied inequality

As we can see in figure 3.16 the solution x_{BC} is not a valid solution for P^n_{CC}. We have the values $x_{BC}(H) = 1.5$, $x_{BC}(T_1) = 1$, $x_{BC}(T_2) = 2$, $x_{BC}(T_3) = 1$, however as for the cardinality we have $|H| = 3$, $|T_1| = 2$, $|T_2| = 3$, $|T_3| = 2$. The inequality looks as follows:

$$x_{BC}(H) + \sum_{i=1}^{3} x_{BC}(T_i) = 5.5 > 5 = |H| + \sum_{i=1}^{3} (|T_i| - 1) - \frac{3+1}{2}.$$

By adding this inequality as a cutting plane to P^n_{BC} (and several more iterations) we get the optimal solution x_{CC} with value 11. This example shows $P^n_{BC} \neq P^n_{CC}$ and $P^n_{CC} \subsetneq P^n_{BC}$ follows. □

Theorem 3.4.3. $P^n_{BC} \subsetneq P^n_{SEP}$.

Proof. It is trivial that $P^n_{BC} \subseteq P^n_{SEP}$ as it is included in the definition of the Blossom Comb Polytope. Therefore it is sufficient to show that $P^n_{SEP} \neq P^n_{BC}$

Observe the following example.

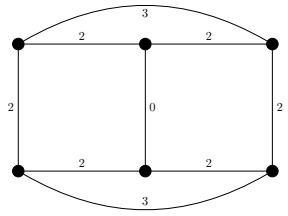

Figure 3.17: Weights of the graph

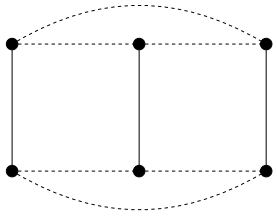

Figure 3.18: Solution with subtour elimination

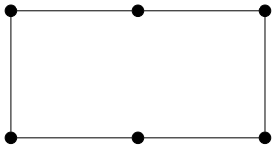

Figure 3.19: Solution with blossom combs

In figure 3.17 we can see the weights of the edges where all non-depicted edges have a set weight of 5. In figure 3.18 we can see the optimal solution $x_{SEP} \in P^n_{SEP}$ for the Subtour Elimination Polytope, where every dashed edge represents a value of 0.5. In total these add up to a value of 11. In figure 3.19 we have the optimal solution $x_{BC} \in P^n_{BC}$ for the Blossom Comb Polytope, which has a value of 12.

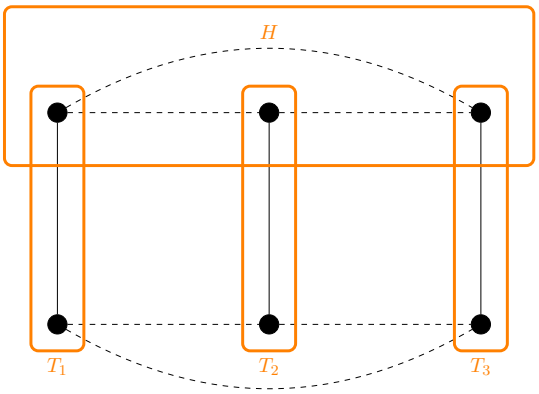

Figure 3.20: A comb with an unsatisfied inequality

As we can see in figure 3.20 the solution x_{SEP} is not a valid solution for P^n_{BC}. We have the values $x_{SEP}(H) = 1.5$, $x_{SEP}(T_1) = 1$, $x_{SEP}(T_2) = 1$, $x_{SEP}(T_3) = 1$, however as for the cardinality we have $|H| = 3$, $|T_1| = 2$, $|T_2| = 2$, $|T_3| = 2$. The inequality looks as follows:

$$x_{SEP}(H) + \sum_{i=1}^{3} x_{SEP}(T_i) = 4.5 > 4 = |H| + \sum_{i=1}^{3} (|T_i| - 1) - \frac{3+1}{2}.$$

By adding this inequality as a cutting plane to P^n_{SEP} (and several more iterations) we get the optimal solution x_{BC} with value 11. This example shows $P^n_{SEP} \neq P^n_{BC}$ and $P^n_{BC} \subsetneq P^n_{SEP}$ follows. □

4 Clique Trees and Clique Tree Inequalities

In this chapter we will be introducing clique trees, a generalization of combs with multiple handles, and clique tree inequalities as defined by Grötschel and Pulleyblank [GP86].

Definition 4.0.1 (Clique Tree)**.** A clique tree is a connected graph C, such that:

(i) The inclusion-wise maximal cliques can be partitioned into the sets of handles $\{H_1, \ldots, H_r\}$ and the set of teeth $\{T_1, \ldots, T_s\}$.

(ii) $|H_i \cap H_j| = 0$ for $1 \leq i < j \leq r$

(iii) $|T_i \cap T_j| = 0$ for $1 \leq i < j \leq s$

(iv) $2 \leq |T_i| \leq n - 2$ for $i = 1, \ldots, s$

(v) $\left|T_i \setminus \bigcup_{j=1}^{r} E(H_j)\right| \geq 1$ for $i = 1, \ldots, s$

(vi) $|\{j \in 1, \ldots, s : H_i \cap T_j \neq \emptyset\}| \geq 3$ odd for $i = 1, \ldots, s$

(vii) $T_i \cap H_j \neq \emptyset$ is an articulation set for $i = 1, \ldots, s;\ j = 1, \ldots, r$.

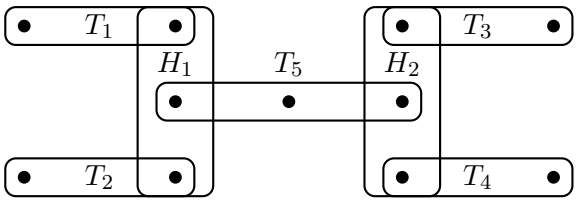

Figure 4.1: A clique tree with five teeth and two handles

Definition 4.0.2 (Clique Tree Inequality)**.** Every Clique Tree defined by Handles H_i for $i = 1, \ldots, r$ and Teeth T_j for $j = 1, \ldots, s$ induces a Clique Tree inequality defined as:

$$\sum_{i=1}^{r} x(H) + \sum_{j=1}^{s} x(T_j) \leq \sum_{i=1}^{r} |H| + \sum_{j=1}^{s} (|T_j| - t_j) - \frac{s+1}{2} \tag{4.1}$$

where t_j denotes the number of handles with which T_j has a non-empty cut.

We will define the Clique Tree Polytope as:

$$P_{CT}^n \coloneqq \{x \in P_{SEP}^n \mid x \text{ satisfies all clique tree inequalities in } K_n\}.$$

This is the Polytope generated by all clique tree inequalities induced by a clique tree in K_n. We can note that the clique tree inequalities with no handles are redundant, as they correspond to the subtour elimination inequalities already included in P^n_{SEP}.

We will see an equivalent definition for a clique tree from Grötschel and Pulleyblank [GP86] to show that all clique tree inequalities are satisfied for every $x_{TSP} \in P^n_{TSP}$.

Definition 4.0.3 (Glueing clique trees at a tooth)**.** Let C' and C'' be two clique trees with node sets V' and V''. Suppose $T := V' \cap V''$ is a tooth of both C' and C'', that T contains a node not in any handle of C' and C'', and that the handles of C' and C'' do not intersect. Then $C' \cup C'' = (V(C') \cup V(C''), E(C') \cup E(C''))$ is a clique tree called the clique tree obtained from C' and C'' by glueing at tooth T.

Definition 4.0.4 (Splitting a clique tree at a tooth and a handle)**.** Let C be a clique tree and T a tooth of C. Let H be a handle of C intersecting T. Delete the nodes $H \setminus T$ from C and let C'' be the component of $C \setminus (H \setminus T)$ containing T. Delete the nodes from C which are in handles meeting T but not in T or H and let C' be the component of this graph containing T. Then C' and C'' are clique trees called the clique trees obtained from C by splitting at T and H.

Definition 4.0.5 (Splitting a comb at a handle)**.** Let C be a clique tree and H a handle of C. Let $T, \ldots, T_k$ be the teeth of C which intersect H. For every tooth T_i, $i \in \{1, \ldots, k\}$, let C_i be the clique tree not containing H obtained from C by splitting at T_i and H. Then the clique trees $C_1, \ldots, C_k$ are called the clique trees obtained from C by splitting at H.

We shall call the left side of the comb inequality $x(C)$ and the right side of the comb inequality $s(C)$ or size of C. $x(C)$ should not be confused with $x(S)$, where S is a set of nodes and C is a graph, namely a clique tree, where only some edges are included.

Remark 4.0.6. There are certain trivial properties about the operations defined above:

- For a clique tree C obtained from glueing C' and C'' as in 4.0.3, it is true that
$$s(C') + s(C'') = s(C) + |T| - 1.$$
- For the clique trees C' and C'' obtained from splitting C on tooth T as in 4.0.4, it is true that
$$s(C') + s(C'') = s(C) + |T| - 1.$$
- For the clique trees $C_1, \ldots C_k$ obtained from splitting C on head H as in 4.0.5, it is true that
$$\sum_{i=1}^{k} s(C_i) = s(C) - |H| + \frac{k+1}{2}.$$

Definition 4.0.7 (Alternate Clique Tree Definition)**.** The following is an alternate definition for clique trees:

(i) Complete graphs with more than two and less than $n-2$ nodes are clique trees (with no handle). Combs are clique trees (with one handle).

(ii) If C' and C'' are two clique trees satisfying the requirements for glueing them at tooth T, then the resulting graph is a clique tree.

(iii) All graphs that can be generated using method (i) and (ii) are clique trees.

Theorem 4.0.8 ([GP86])**.** *Every clique tree inequality is satisfied for every $x \in P^n_{TSP}$. Therefore, $P^n_{TSP} \subseteq P^n_{CT}$.*

Proof. We can show this through induction on the number r of handles.

Initial Case: The initial case consists of the clique trees with one or zero handles. We know that this corresponds to the comb and subtour elimination inequalities, which have already been shown to be valid.

Induction Step: Assume that all clique tree inequalities with r handles are satisfied by every $x \in P^n_{TSP}$. Observe a clique tree C with $r+1$ handles. Let $C_1, \ldots, C_k$ be the clique trees generated by splitting C on a handle H_{r+1} as in 4.0.5. Call the remaining handles $H_1, \ldots, H_r$ and name the teeth intersecting H_{r+1} as $T_1, \ldots, T_k$ so that T_i is contained in C_i. Generate the combs $C^*_1, \ldots, C^*_k$ by replacing the tooth T_i in C_i with $T_i \setminus H$. From Remark 4.0.6 we have

$$\sum_{i=1}^{k} s(C_i) = s(C) - \mid H \mid + \frac{k+1}{2}$$

Therefore, it follows

$$\sum_{i=1}^{k} s(C^*_i) = s(C) - \mid H \mid + \frac{k+1}{2} - \sum_{i=1}^{k} \mid H \cap T_i \mid .$$

From this we arrive at

$$\begin{aligned} 2x(C) = 2\left(\sum_{i=1}^{r+1} x(H_i) + \sum_{j=1}^{s} x(H_j)\right) &\leq \sum_{i=1}^{k}\Big(x(C_i) + x(C^*_i) + x(E(H \cap T_i))\Big) + \sum_{v \in H_{r+1}} x(\delta(v)) \\ &\leq \sum_{i=1}^{k}\Big(s(C_i) + s(C^*_i) + \mid H \cap T_i \mid -1\Big) + 2 \mid H \mid \\ &= 2s(C) + 1 \end{aligned}$$

For every integer solution x of the TSP, the left side is an even integer, after dividing by two and rounding down we get inequality 4.1. The validity for all $x \in P^n_{TSP}$ follows as it is the convex hull of these solutions.

□

Theorem 4.0.9. *$P^n_{CT} \subsetneq P^n_C$.*

Proof. It is trivial that $P^n_{CT} \subseteq P^n_C$ as every simple comb is also a clique tree with only one handle. Therefore it is sufficient to show that $P^n_C \neq P^n_{CT}$

Observe the following example.

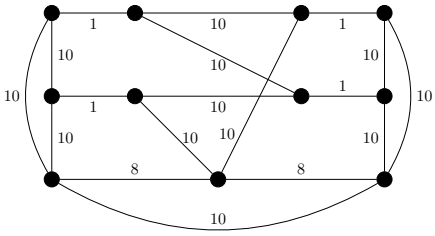

Figure 4.2: Weights of the graph

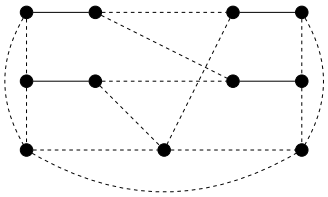

Figure 4.3: Solution with combs

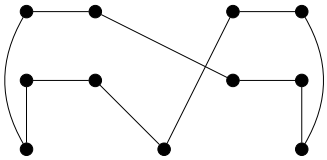

Figure 4.4: Solution with clique trees

In figure 4.2 we can see the weights of the edges where all non-depicted edges have a weight of 100. In figure 4.3 we can see the optimal solution $x_C \in P_C^n$ for the Comb Polytope, where every dashed edge represents a value of 0.5. In total these add up to a value of 72. In figure 4.4 we have the optimal solution $x_{CT} \in P_{CT}^n$ for the Clique Tree Polytope, which has a value of 74.

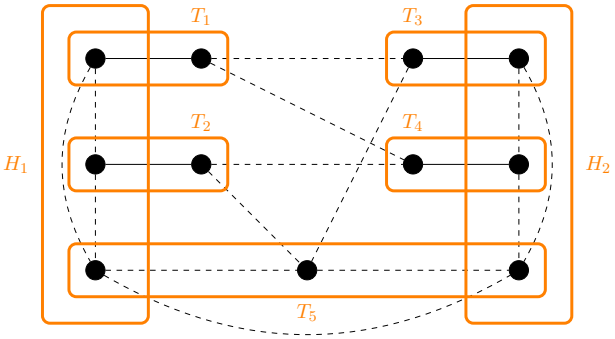

Figure 4.5: A comb with an unsatisfied inequality

As we can see in figure 4.5 the solution x_C is not a valid solution for P_{CT}^n. We have the values $x_C(H_1) = 1.5$, $x_C(H_2) = 1.5$, $x_C(T_1) = 1$, $x_C(T_2) = 1$, $x_C(T_3) = 1$, $x_C(T_4) = 1$ and $x_C(T_5) = 1.5$. However as for the cardinality we have $|H_1| = 3$, $|H_2| = 3$, $|T_1| = 2$, $|T_2| = 2$, $|T_3| = 2$, $|T_4| = 2$ and $|T_5| = 3$. The inequality looks as follows:

$$\sum_{i=1}^{2} x_C(H) + \sum_{j=1}^{5} x_C\left(T_j\right) = 8.5 > 8 = \sum_{i=1}^{2} |H| + \sum_{j=1}^{5} \left(|T_j| - t_j\right) - \frac{5+1}{2}.$$

By adding this inequality as a cutting plane to P_C^n (and after several more iterations finding new broken clique tree inequalities) we get the optimal solution x_{CT} with value 74. This example shows $P_C^n \neq P_{CT}^n$ and $P_{CT}^n \subsetneq P_C^n$ follows. □

5 Minimality

In chapters 3 and 4 we saw examples that separate the different classes of combs and clique trees in the context of the Traveling Salesman Problem. In this chapter we will see that the examples given in these chapters are minimal.

For the sake of brevity when refering to comb classes or combs in this chapter we will be including clique trees in this description. When talking about the class of smaller order we will be talking about the one that is contained in the class of higher order.

Definition 5.0.1 (Trivially minimal examples)**.** We will classify an example for separating two comb classes as trivially minimal when it is the same size as the smallest comb of the class of higher order that is not also a comb from the class of smaller order.

It is trivial to prove that a trivially minimal example, if it exists, is a minimal example.

Using this definition, we can see that most of the examples provided are trivially minimal. Example 3.3 for separating P_{SC} from P_C^n needs eight nodes to be trivially minimal, as we need a tooth with two nodes inside the handle as well as two nodes outside the handle. Example 3.13 for separating P_{BC} from P_{CC}^n needs seven nodes to be trivially minimal, as we need a tooth with three nodes. Example 3.17 for separating P_{SEP} from P_{BC}^n needs six nodes to be trivially minimal, as this is the minimum number of nodes for a comb. Example 4.2 for separating P_C from P_{CT}^n needs eleven nodes to be trivially minimal, as this is the minimum number of nodes for a clique tree with two handles. These are all trivially minimal examples. However example 3.8 for separating the Chvátal Comb Polytope from the Simple Comb Polytope is not. The amount of nodes for a trivially minimal example would be 7 nodes, as we only need a tooth with two nodes in the intersection with the handle. We will now see how to prove that a trivially minimal example is not possible.

In order to have an example for separating two comb classes we need a solution that satisfies all comb inequalities from the class of smaller order but breaks an inequality from a class of higher order. When considering trivially minimal examples there is a limited number of combs of a class of higher order that can be broken by our solution. In fact, if we ignore the enumeration of the nodes these are reduced to a single comb. Since all our inequalities disregard the enumeration of the nodes we only need to prove that one of these comb inequalities of higher order cannot be broken.

Definition 5.0.2 (Trivially Minimal Linear Inequality System)**.** Let C be the aforementioned smallest comb of higher order in a graph K_n so that every node is included in C. Let H be the union of all the handles in the comb and T the union of all the teeth in the comb. Define a_e as:

$$a_e = \begin{cases} 1, & \text{if } e \in H \setminus T \text{ or } e \in T \setminus H \\ 2, & \text{if } e \in H \cap T \\ 0, & \text{otherwise.} \end{cases}$$

The value a_e corresponds to the amount of times the edge e is counted when calculating $x(C)$. Let C_S be the class of combs of smaller order. The following Linear Inequality System will be called the Trivially Minimal Inequality System.

$\max \sum_{e \in C} a_e x_e$ s.t.:

$$\sum_{e \in \delta(v)} x_e = 2 \qquad \forall v \in V(K_n) \tag{I}$$

$$\sum_{e \in E(K_n[S])} x_e \leq |S| - 1 \qquad \forall \emptyset \neq S \subsetneq V(K_n) \tag{II}$$

$$0 \leq x_e \leq 1 \qquad \forall e \in E(K_n) \tag{IV}$$

$$x(C') \leq s(C') \qquad \forall C' \in C_S \tag{V}$$

Where constraint V refers to the comb inequality.

By looking at the optimal objective value of this LPP we can determine whether a trivially minimal example exists. If the value is greater than $s(C)$ then our solution satisifies all combs of smaller order but does not satisfy the comb inequality for the comb C. However if the value is equal to $s(C)$ there is no solution that satisfies all combs of smaller order but does not satisfy the comb inequality for the comb C. Therefore, there can be no trivially minimal example.

This is the case for the example 3.7 for separating the Chvátal Comb Polytope from the Simple Comb Polytope. If we compute the above described inequality system we get an optimal value of 6, with $s(C)$ also having a value of 6. This means that no trivially minimal example can exist. Our example with eight nodes, one more than the seven required for it to be trivially minimal, must then be a minimal example.

6 Methodology

In this chapter we will discuss the programs used to obtain the optimal solutions in chapters 3 and 4. For the programs we used the SoPlex library for exact solving of LPPs [EG24].

6.1 Combs

First we will see the algorithm for separating combs with some minor differences from the algorithm for separating clique trees. Although there exist polynomial algorithms for separating some classes of combs we will be enumerating them all in exponential time.

We start by generating the LPP corresponding to P^n_{SEP}. As discussed in chapter 2 we now want to find cutting planes to add to our solution; these cutting planes will be comb inequalities. For separating combs we use a simple enumeration algorithm which runs through all combs checking the validity of the previous solution. For each one we check whether the comb inequality is broken and if so add it to our LPP. Our algorithm does not differ depending on the class of combs as it is only optimised for general combs. We will only check in the last function of the algorithm whether our comb is of the wanted class.

The algorithm consists of several recursive functions that utilize backtracking to enumerate the combs in K_n. We will now describe the functions forming the algorithm.

Algorithm 1: Comb Relaxation Solver

```
Data: Complete Graph G with weights w_e
Result: Optimal Solution x_OPT for the Comb Relaxation Problem
Ax ≤ b ← Inequality System generated by the Subtour Relaxation
x ← Solve Ax ≤ b optimizing on the length in G
while x is non-integer solution do
    C ← findComb(G,x)
    if C is empty then
        break
    Ax ≤ b ← Inequality generated by C
    x ← Solve Ax ≤ b optimizing on the length in G
return x
```

Algorithm 1 generates the inequality system $Ax \leq b$ for optimizing on P^n_{SEP}. After solving this LPP with solution x we return that no comb exists if x is already integer and already solves the TSP. If this is not the case we will try to find a comb with a broken comb inequality to add to $Ax \leq b$. If no such comb exists we conclude that this is the optimal solution for the corresponding comb polytope.

Algorithm 2: findComb(G,x)

```
Data: Complete Graph G and Solution x
Result: Comb with a broken Comb Inequality
Initialize Comb C
if addHandle(G,x,C,-1) is true then
    return C
return empty Comb C
```

Algorithm 2 is merely a wrapper function which will gain utility on the algorithm for clique trees.

Algorithm 3: addHandle(G,x,C,v)

```
Data: Graph G, Solution x, Comb C, Node v
Result: Existence of Comb with broken inequality
if v is not -1 then
    add v to Handle
    if C is invalid then
        return false
    if Handle has more than 3 nodes then
        if addRoot(G,x,C,-1,0) is true then
            return true
for nonvisited node w in G do
    if addHandle(G,x,C,w) is true then
        return true
return false
```

Algorithm 3 tries adding and then skipping every node in order to the handle H. Once a valid Handle is found, meaning we have at least 3 nodes in our handle, we will continue to the next function to add roots to our handle. The comb C will be considered invalid if there are not suficient nodes outside of the handle to be able to build the mandatory minimum of three teeth. If it is not possible to generate a successful comb with the handle given to Algorithm 4 then we will backtrack and skip the last added node to the handle.

Algorithm 4: addRoot(G,x,C,v,yesno)

```
Data: Graph G, Solution x, Comb C, Node v, Bool yesno
Result: Existence of Comb with broken inequality
if v is not -1 then
    if yesno is 1 then
        add node v to Roots
    else
        mark node v
    if C is valid then
        if growTooth(G,x,C,-1,-1) is true then
            return true
    else
        return false
w ← nonmarked node in G
if w is empty then
    return false
if addRoot(G,x,C,w.1) is true then
    return true
if addRoot(G,x,C,w,0) is true then
    return true
return false
```

Algorithm 4 tries adding every root w as a root to the handle, thereby also adding a tooth containing (for now) only w. Once we have a valid number of roots, namely odd and greater or equal to 3 we will call the next function to grow the teeth generated by the roots.

If it is not possible to generate a successful comb from this set of roots, we will backtrack and mark the last non marked node in order to not consider it again. If it is not possible to generate a successful comb with any set of roots, then we shall return `false` to Algorithm 3 causing it to backtrack.

Algorithm 5: growTooth(G,x,C,v,T)

```
Data: Graph G, Solution x, Comb C, Node v, Tooth T
Result: Existence of Comb with broken inequality
if v is not -1 then
    if T is -1 then
        mark node v
    else
        add v to tooth T
    if C is valid then
        if C is a Comb and solution x is invalid for C then
            return true
    else
        return false
w ← nonmarked node in G
if C has an invalid tooth or w is -1 then
    return false
for Tooth T' in C do
    if growTooth(G,x,C,w,T') is true then
        return true
if growTooth(G,x,C,w,-1) is true then
    return true
return false
```

Algorithm 5 goes through every node trying to add it to every tooth in C in order. If C is already a Comb we will check if the inequality is satisfied for the solution x. If it is we will return the Comb C as our cutting plane. If it is not we will continue adding more nodes. The comb C is considered invalid if we no longer have enough unassigned nodes to add at least one node outside the handle to every tooth. If we cannot add any more nodes and have not found a successful comb we will backtrack by moving the last node added to the next tooth or marking it to not consider it anymore if we are already at the last tooth.

It is trivial to show that this algorithm enumerates all combs correctly, therefore if there were a comb with a broken comb inequality we would reach it before the end of the algorithm. Therefore, the algorithm works correctly on optimizing on the corresponding comb polytope.

6.2 Clique Trees

We will now see the algorithm for separating clique trees with some minor modifications from the one for clique trees to adapt to clique trees. This algorithm is based on Bültes algorithm [Bül22] with some modifications.

We start by generating the LPP corresponding to P^n_{SEP}. As discussed in chapter 2 we now want to find cutting planes to add to our solution, these cutting planes will be clique tree inequalities. For separating clique trees we used a simple enumeration algorithm which runs through all clique trees checking the validity of the previous solution. For each one we check whether the clique tree inequality is broken and if so add it to our LPP.

The algorithm consists of several recursive functions that utilize backtracking to enumerate the clique trees in K_n. We will now describe the functions forming the algorithm.

Algorithm 6: Clique Tree Relaxation Solver

```
Data: Complete Graph G with weights w_e
Result: Optimal Solution x_OPT for the Clique Tree Relaxation Problem
Ax ≤ b ← Inequality System generated by the Subtour Relaxation
x ← Solve Ax ≤ b optimizing on the length in G
while x is non-integer solution do
    C ← findCliqueTree(G,x)
    if C is empty then
        C ← findComb(G,x)
        if C is empty then
            break
    Ax ≤ b ← Inequality generated by C
    x ← Solve Ax ≤ b optimizing on the length in G
return x
```

Algorithm 6 generates the inequality system $Ax \leq b$ for optimizing on P^n_{SEP}. After solving this LPP with solution x we return that no clique tree exists if x is already integer and already solves the TSP. If this is not the case, we will try to find a clique tree, including combs, with a broken clique tree inequality to add to $Ax \leq b$. If no such clique tree exists we conclude that this is the optimal solution for the corresponding Clique Tree Polytope.

Algorithm 7: findCliqueTree(G,x)

```
Data: Complete Graph G and Solution x
Result: Clique Tree with a broken Clique Tree Inequality
Initialize Clique Tree C
for viable number of handles r do
    if addHandle(G,x,C,-1,r) is true then
        return C
return empty Clique Tree C
```

Algorithm 7 calls the next function to add nodes to the handle. We try to make a successful clique tree with every viable number of handles r. However, we will start with $r \geq 2$ since the 0 and 1 handle clique trees will be generated through subtour elimination and the Comb separation algorithm respectively.

Algorithm 8: addHandle(G,x,C,v,r)

```
Data: Graph G, Solution x, Clique Tree C, Node v, Handle H, int r
Result: Existence of Clique Tree with broken inequality
if v is not -1 then
    if H is -1 then
        mark v
    else
        add v to Handle H
        if C is invalid then
            return false
        if C is valid then
            if addRoot(G,x,C,-1,0) is true then
                return true
for nonmarked node w in G do
    for Handle H_i in C do
        if addHandle(G,x,C,w,H_i,r) is true then
            return true
    if addHandle(G,x,C,w,H_{i+1},r) is true then
        return true
    if addHandle(G,x,C,w,-1,r) is true then
        return true
return false
```

Algorithm 8 tries adding every node in order to consider every possible handle in C. If this is not successful we will try creating a new handle to which we add the node. If this also doesn't work, we will mark the node in order to ignore it in the future. We will consider C to have a valid set of handles if it has exactly r handles with at least 3 nodes each. When this is the case we will continue to the next function to add roots to our handles. The clique tree C will be considered invalid if there are not enough nodes outside of the handles to be able to build a suficient number of teeth for each handle. If it is not possible to generate a successful clique tree with the handle given to Algorithm 9 then we will backtrack and mark the last added node to the handle.

Algorithm 9: addRoot(G,x,C,v,yesno)

```
Data: Graph G, Solution x, Clique Tree C, Node v, Bool yesno
Result: Existence of Clique Tree with broken inequality
 1: if v is not -1 then
 2:     if yesno is 1 then
 3:         add node v to Roots
 4:     else
 5:         mark node v
 6:     if C is valid then
 7:         if combineTeeth(G,x,C,-1,-1) is true then
 8:             return true
 9:     else
10:         return false
11: w ← nonmarked node in a Handle in G
12: if w is empty then
13:     return false
14: if addRoot(G,x,C,w,1) is true then
15:     return true
16: if addRoot(G,x,C,w,0) is true then
17:     return true
18: return false
```

Algorithm 9 tries adding every root w as a root to every handle in C successively, thereby also adding a tooth containing (for now) only w. Once we have a valid number of roots, namely odd and greater or equal to 3 for each handle we will call the next function to combine the teeth generated by the roots. If it is not possible to generate a successful clique tree from this set of roots, we will mark we will backtrack and mark the last non marked node in order to not consider it again. If it is not possible to generate a successful clique tree with any set of roots, then we shall return `false` to Algorithm 8 causing it to backtrack.

Algorithm 10: combineTeeth(G,x,C,H_1,H_2)

```
Data: Graph G, Solution x, Clique Tree C, Handle H_1, Handle H_2
Result: Existence of Clique Tree with broken inequality
 1: if H_1 and H_2 are not -1 then
 2:     Combine Teeth H_1 and H_2 to form tooth H in C
 3:     if C is connected then
 4:         if growTooth(G,x,C,-1,-1) is true then
 5:             return true
 6:         else
 7:             return FALSE
 8: for unconnected pair of teeth H' and H'' do
 9:     if combineTeeth(G,x,C,H',H'') is true then
10:         return true
11: return false
```

Algorithm 10 combines the teeth in C in order to make C be connected as well as every handle and tooth being an articulation set. For this we will try to connect every possible pair of unconnected teeth in C until C is fully connected and then move on to the next function to grow the teeth. If it is not possible to get a successful clique tree from the connected teeth chosen, we will backtrack and try a different connection. If is not possible to get a successful clique tree from any set of connections, then we shall return `false` to Algorithm 9 causing it to backtrack.

Algorithm 11: growTooth(G,x,C,v,T)

```
Data: Graph G, Solution x, Clique Tree C, Node v, Tooth T
Result: Existence of Clique Tree with broken inequality
 1: if v is not -1 then
 2:     if T is -1 then
 3:         mark node v
 4:     else
 5:         add v to tooth T
 6:     if C is valid then
 7:         if C is a Clique Tree and solution x is invalid for C then
 8:             return true
 9:     else
10:         return false
11: w ← nonmarked node in G
12: if C has an invalid tooth or w is -1 then
13:     return false
14: for Tooth T' in C do
15:     if growTooth(G,x,C,w,T') is true then
16:         return true
17: if growTooth(G,x,C,w,-1) is true then
18:     return true
19: return false
```

Algorithm 11 goes through every node trying to add it to every tooth in C in order. If C is already a clique tree we will check if the inequality is satisfied for the solution x. If it is we will continue adding more nodes. The clique tree C is considered invalid if we no longer have enough unassigned nodes to add at least one node outside of every handle to every tooth. If we cannot add any more nodes and have not found a successful clique tree we will backtrack by moving the last node added to the next tooth or marking it to not consider it anymore if we are already at the last tooth.

The adaptation from combs to clique trees is based on the alternate definition for clique trees 4.0.7, where we show that by starting with combs and glueing them together we can reach all possible clique trees. Therefore, the algorithm must always find a clique tree with a broken inequality if one exists, as it enumerates all clique trees correctly. This means it optimizes correctly on the Clique Tree Polytope.

7 Euclidean Examples

In chapters 3 and 4 we saw examples for separating the different classes of combs and clique trees in the context of the Traveling Salesman Problem. However there are other important special cases of the Traveling Salesman Problem for which our examples are not valid. In this chapter I have compiled examples for the Euclidean case of the TSP, which also includes the metric TSP for those that did not have a metric example.

Remark 7.0.1. Although the instances in this chapter where calculated with the same exact method, the lengths given to the algorithm are not exact as they have to be stored in a double format. For our examples, where the longest edge is shorter than 254, this means we have at least 45 (binary) decimals. This leads to a rounding error of at most $2^{-45} \leq 10^{-13}$. We can perform a lower and upper bound by adding or subtracting 10^{-6}, since the maximum error is 11 times the rounding error for the lengths, while ideally maintaining the relation between the length of the two solutions. This is how we obtained the values given as a lower and upper bound. The solutions given will be optimal only for these rounded values, however the lower and upper bounds will also apply to the actual optimal solution using the exact weights.

7.1 Blossom Combs

Euclidean example for separating the Subtour Elimination Polytope and the Blossom Comb Polytope

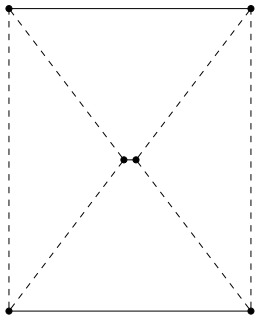

Figure 7.1: Solution with subtour elimination

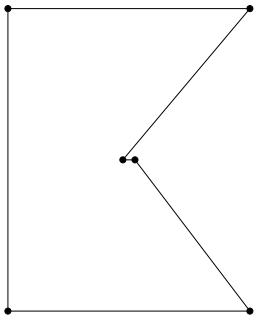

Figure 7.2: Solution with blossom combs

With points (0,5), (4,5), (1.9,2.5), (2.1,2.5), (0,0) and (4,0). The solution for the Subtour Elimination Polytope in figure 7.1 has a value of $13.2 + 2\sqrt{9.86}$. Therefore, the optimal solution must have a value of under 19.480129. The solution for the Blossom Comb Polytope in figure 7.2 has a value of $13.2 + \sqrt{9.86} + \sqrt{10.66}$. Therefore, the optimal solution must have a value of over 19.605027.

7.2 Chvátal Combs

Euclidean example for separating the Blossom Comb Polytope and the Chvátal Comb Polytope

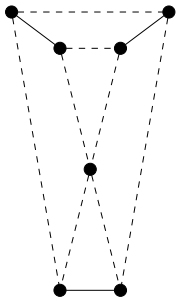

Figure 7.3: Solution with blossom combs

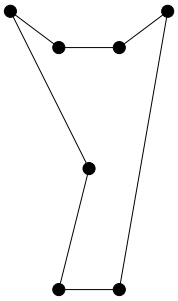

Figure 7.4: Solution with Chvátal combs

With points (1,4), (2,4), (0.2,4.6), (1.5,2), (2.8,4.6), (1,0) and (2,0). The solution for the Blossom Comb Polytope in figure 7.3 has a value of $4.8 + 2\sqrt{4.25} + \sqrt{21.8}$. Therefore, the optimal solution must have a value of under 13.592154. The solution for the Chvátal Comb Polytope in figure 7.4 has a value of $4 + \sqrt{4.25} + \sqrt{8.45} + \sqrt{21.8}$. Therefore, the optimal solution must have a value of over 13.637486.

7.3 Simple Combs

Euclidean example for separating the Chvátal Comb Polytope and the Simple Comb Polytope

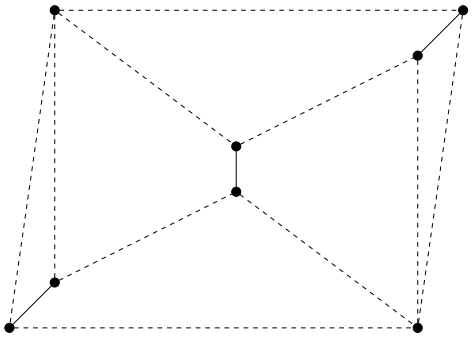

Figure 7.5: Solution with Chvátal combs

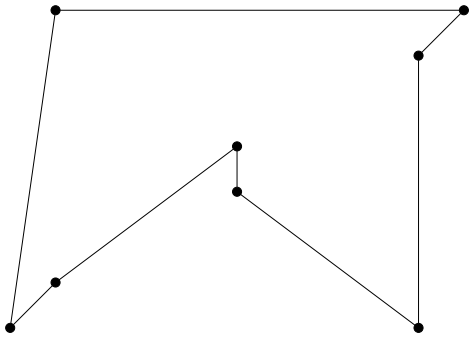

Figure 7.6: Solution with simple combs

With points (1,7), (5,4), (9,6), (10,7), (0,0), (1,1), (5,3) and (9,0). The solution for the Chvátal Comb Polytope in figure 7.5 has a value of $21 + 2\sqrt{2} + \sqrt{20} + \sqrt{50}$. Therefore, the optimal solution must have a value of under 35.371632. The solution for the Simple Comb Polytope in figure 7.6 has a value of $26 + 2\sqrt{2} + \sqrt{50}$. Therefore, the optimal solution must have a value of over 35.899492.

7.4 Combs

Euclidean example for separating the Simple Comb Polytope and the Comb Polytope

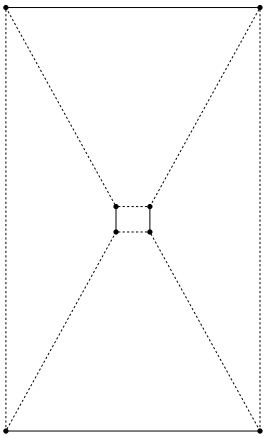

Figure 7.7: Solution with simple combs

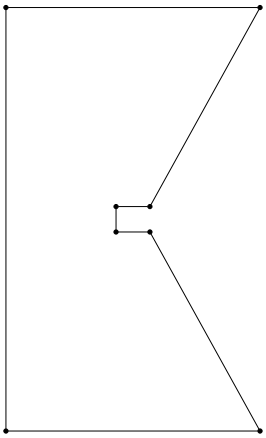

Figure 7.8: Solution with combs

With points (0,20), (6,20), (2.85,10.1), (3.15,10.1), (2.85,9.9), (3.15,9.9), (0,0) and (6,0). Figure 7.7 and Figure 7.8 are not properly scaled for the sake of clarity. The solution for the Simple Comb Polytope in figure 7.7 has a value of $32.7 + 2\sqrt{106.1325}$. Therefore, the optimal solution must have a value of under 53.304127. The solution for the Comb Polytope in figure 7.8 has a value of $32.8 + 2\sqrt{106.1325}$. Therefore, the optimal solution must have a value of over 53.404123.

7.5 Clique Trees

We were not able to find a minimal Euclidean example for separating combs and clique trees, however we can offer the following example with twelve nodes. In example 7.9 the dashed lines have a value of $\frac{1}{3}$ and the dotted lines have a value of $\frac{2}{3}$.

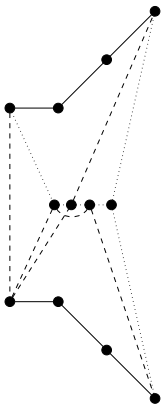

Figure 7.9: Solution with combs (not optimal)

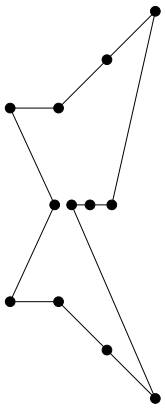

Figure 7.10: Solution with clique trees

With points (1, 4), (2, 4), (3, 5), (4, 6), (1.92, 2), (2.27, 2), (2.65, 2), (3.1, 2), (1, 0), (2, 0), (3, -1) and (4, -2).

The solution for the Comb Polytope in figure 7.9 has a value of $\frac{1309}{300} + 4\sqrt{2} + \sqrt{4.8464} + \frac{1}{3}\sqrt{5.6129} + \frac{1}{3}\sqrt{17.8225} + \frac{4}{3}\sqrt{16.81} + \frac{1}{3}\sqrt{18.9929}$. Therefore, the optimal solution must have a value of under 21.337947. The solution for the Clique Tree Polytope in figure 7.10 has a value of $2.83 + 4\sqrt{2} + 2\sqrt{4.8464} + \sqrt{16.81} + \sqrt{18.9929}$. Therefore, the optimal solution must have a value of over 21.347844.

Remark 7.5.1. It is important to note that although the solution in figure 7.9 has been shown to be a valid solution for the Comb Polytope, as it is the solution of the example provided by Laura Bülte [Bül22] for separating P_C^n and P_{CT}^n in the general case, the instance not been computed with our algorithm or shown to be optimal. An optimal solution could only possibly have a value lower than the solution given, which would only further strengthen our claim.

8 Conclusion

We will now briefly summarize what we have shown in this thesis.

In chapter 3 we introduced the different type of combs and comb inequalities: combs, simple combs, Chvátal combs and blossom combs. We defined the respective combs and their induced comb inequalities as well as the polytopes generated by these inequalities. We also showed the strict inclusions between these different polytopes. In chapter 4 we introduced clique trees and clique tree inequalities and showed the strict inclusion of the Comb Polytope in the Clique Tree Polytope.

We also showed in chapter 5 that all examples given in chapters 3 and 4 are minimal in respect to the amount of nodes they have. In chapter 7 we also showed the equivalent examples for the metric TSP and the Euclidean TSP which were also minimal, except for the separation of combs and clique trees which has one extra node compare to our general example.

We can depict the relationships shown for the TSP as well as the metric and Euclidean TSP as follows:

$$P^n_{TSP} \subseteq P^n_{CT} \subsetneq P^n_C \subsetneq P^n_{SC} \subsetneq P^n_{CC} \subsetneq P^n_{BC} \subsetneq P^n_{SEP}$$

In chapter 6 we described the algorithms used to achieve these results. We used simple enumeration algorithms together with SoPlex [EG24] , an exact LPP solver, to solve the examples given for each of the two polytopes. The corresponding program can be found in the memory stick provided. Thanks to this we can conclusively prove the relations proposed by Bülte [Bül22].

Bibliography

[Bül22] Laura Bülte. “Cutting Planes für das Traveling Salesman Problem”. Bachelor’s thesis. Rheinische Friedrich-Wilhelms-Universität Bonn, Forschungsinstitut für Diskrete Mathematik, 2022.

[Chv72] Václav Chvátal. “Edmonds Polyhedra and Weakly Hamiltonian Graphs”. In: *Mathematical Programming* 5 (1972), pp. 29–40.

[DFJ54] G. Dantzig, R. Fulkerson, and S. Johnson. “Solution of a Large-Scale Traveling-Salesman Problem”. In: *Journal of the Operations Research Society of America* 2 (Nov. 1954), pp. 393–410.

[EG24] Leo Eifler and Ambros Gleixner. *SoPlex: Sequential object-oriented simPlex*. github.com/scipopt/soplex. Version v.7.0.1. 2002-2024.

[GP79] Martin Grötschel and Manfred W. Padberg. “On the Symmetric Traveling Salesman Problem I: Inequalities”. In: *Mathematical Programming* 16 (1979), pp. 265–280.

[GP86] M. Grötschel and W. R. Pulleyblank. “Clique Tree Inequalities and the Symmetric Travelling Salesman Problem”. In: *Mathematics of Operation Research* 11 No. 4 (Nov. 1986), pp. 537–569.

[KV17] Bernhard Korte and Jens Vygen. *Combinatorial Optimization. Theory and Algorithms*. 6th ed. Springer, 2017.