Universidade Estadual de Campinas
Instituto de Computação

Breno Piva Ribeiro

Finding Geometric Structures with Minimum Stabbing Number

Encontrando Estruturas Geométricas com Número de Trespasse Mínimo

CAMPINAS
2016

Breno Piva Ribeiro

Finding Geometric Structures with Minimum Stabbing Number

Encontrando Estruturas Geométricas com Número de Trespasse Mínimo

Tese apresentada ao Instituto de Computação da Universidade Estadual de Campinas como parte dos requisitos para a obtenção do título de Doutor em Ciência da Computação.

Dissertation presented to the Institute of Computing of the University of Campinas in partial fulfillment of the requirements for the degree of Doctor in Computer Science.

Supervisor/Orientador: Prof. Dr. Cid Carvalho de Souza

Este exemplar corresponde à versão final da Tese defendida por Breno Piva Ribeiro e orientada pelo Prof. Dr. Cid Carvalho de Souza.

CAMPINAS
2016

Agência(s) de fomento e nº(s) de processo(s): CNPq, 147619/2010-6

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca do Instituto de Matemática, Estatística e Computação Científica
Maria Fabiana Bezerra Muller - CRB 8/6162

Piva, Breno, 1983-
P688f Finding geometric structures with minimum stabbing number / Breno Piva Ribeiro. – Campinas, SP : [s.n.], 2016.

Orientador: Cid Carvalho de Souza.
Tese (doutorado) – Universidade Estadual de Campinas, Instituto de Computação.

1. Otimização combinatória. 2. Geometria computacional. 3. Complexidade computacional. 4. Programação inteira. I. Souza, Cid Carvalho de,1963-. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

Informações para Biblioteca Digital

Título em outro idioma: Encontrando estruturas geométricas com número de trespasse mínimo
Palavras-chave em inglês:
Combinatorial optimization
Computational geometry
Computational complexity
Integer programming
Área de concentração: Ciência da Computação
Titulação: Doutor em Ciência da Computação
Banca examinadora:
Cid Carvalho de Souza [Orientador]
Alexandre Salles da Cunha
Cláudio Nogueira de Meneses
Fábio Luiz Usberti
Pedro Jussieu de Rezende
Data de defesa: 26-02-2016
Programa de Pós-Graduação: Ciência da Computação

Universidade Estadual de Campinas
Instituto de Computação

Breno Piva Ribeiro

Finding Geometric Structures with Minimum Stabbing Number

Encontrando Estruturas Geométricas com Número de Trespasse Mínimo

Campinas, 26 de fevereiro de 2016

Acknowledgements

Most of the acknowledgements have to be done in Portuguese.

Gostaria de agradecer ao CNPq pelo apoio financeiro. À Universidade Federal de Sergipe, em especial ao Departamento de Computação pelo apoio fornecido para a finalização deste trabalho.

Aos professores do Instituto de Computação da Universidade Estadual de Campinas, em particular ao meu orientador, Professor Cid C. de Souza, que ao longo dos quase nove anos que trabalhamos juntos, me ensinou muito mais do que eu poderia descrever aqui, além de ter me dado um exemplo de didática na sala de aula. E ainda ao Professor Pedro J. de Rezende que sempre esteve disponível para discutir e tirar dúvidas.

To Professor Sándor Fekete and the people from the algorithms group of the Technische Universität Braunschweig for all the support and for receiving me there so well and providing a good and productive time.

Aos colegas do IC, especialmente do LOCO, que sempre me ajudaram quando eu precisei, em especial para religar alguma máquina. E aos companheiros de república que me acolheram e mantiveram um ambiente tranquilo em casa no período do doutorado que passei em Campinas.

A todos os meus amigos, que sempre me deram apoio moral e psicológico e proveram distrações toda vez que precisei.

A toda minha família: meus avós (*in memorian*), meus irmãos (de sangue e de coração), meu padrasto Chiquinho e minha madrasta Emília, e, sobretudo, aos meus pais Jenival e Marta. Obrigado por fazerem parte da minha vida.

E, por último, aos meus amores Ana Maria, Luna, Guinness e Penélope que me dão alegria todos os dias e ficaram bem junto enquanto eu escrevia este trabalho.

Muito obrigado por tudo.

Resumo

Problemas de trespasse têm sido investigados há tempos em Geometria Computacional pois aplicações para eles são encontradas em uma grande variedade de áreas. Em geral, a entrada é formada por dois conjuntos de objetos geométricos: o conjunto, finito ou infinito, $\mathcal{L}$ de trespassadores e o conjunto $\mathcal{O}$. Uma solução viável é um subconjunto $\mathcal{O}'$ de $\mathcal{O}$ satisfazendo uma certa propriedade estrutural π. Dado $\mathcal{O}'$, o número de trespasse de $\ell \in \mathcal{L}$ é a quantidade de elementos de $\mathcal{O}'$ intersectados por ℓ. O número de trespasse de $\mathcal{O}'$ relativo a $\mathcal{L}$ é o número de trespasse máximo dentre qualquer $\ell \in \mathcal{L}$. O objetivo do problema é achar um subconjunto de $\mathcal{O}$ satisfazendo a propriedade π com o menor número de trespasse possível relativo a $\mathcal{L}$. Esta tese traz contribuições tanto teóricas quanto experimentais para alguns problemas de trespasse.

Em [17, 18], Fekete, Lübbecke e Meijer resolveram o problema aberto a respeito da complexidade de encontrar uma árvore geradora com número de trespasse mínimo. Eles também mostraram que achar um emparelhamento perfeito com número de trespasse mínimo é $\mathcal{NP}$-difícil. Modelos de programação inteira para os problemas foram apresentados. Porém, muito poucos experimentos computacionais foram realizados.

Nesta tese, estudamos modelos de programação inteira para encontrar emparelhamentos perfeitos, árvores geradoras e triangulação com número de trespasse mínimo. Com base nestas formulações, apresentamos algoritmos exatos e heurísticas Lagrangianas para resolvê-los. Estes algoritmos mostraram que as heurísticas Lagrangianas proveem boas soluções, frequentemente ótimas, em um breve tempo computacional.

De todos os dez problemas e variantes discutidos em [18], para apenas três deles a complexidade não foi provada: Triangulação com Número de Trespasse Mínimo, com trespassadores paralelos aos eixos e gerais, e Triangulação com Número de Cruzamento Mínimo, caso geral. Nesta tese, provamos que estes três problemas são $\mathcal{NP}$-difíceis.

Outro problema de trespasse mínimo é apresentado em [2] e também estudado em [16]. Este problema pede por uma partição retangular com número de trespasse mínimo em um polígono retilinear. Embora a complexidade do problema ainda seja desconhecida, em [2] um algoritmo de 3-aproximação é apresentado. Em [16] um modelo de programação inteira é dado e uma 2-aproximação reivindicada.

Nesta tese, fortalecemos a formulação introduzida em [16]. Também propomos um modelo alternativo e comparamos os dois teórica e computacionalmente. Além disso, mostramos que o algoritmo proposto em [16] não provê uma 2-aproximação para o problema.

Abstract

Stabbing problems have long been investigated in Computational Geometry since applications for them are found in a great variety of areas. In general, the input is formed by two sets of geometric objects: the finite or infinite set $\mathcal{L}$ of stabbers and a set $\mathcal{O}$. A feasible solution for the problem is a subset $\mathcal{O}'$ of $\mathcal{O}$ satisfying a given structural property π. Given $\mathcal{O}'$, the stabbing number of $\ell \in \mathcal{L}$ is the number of elements of $\mathcal{O}'$ that are intersected by ℓ. The stabbing number of $\mathcal{O}'$ relative to $\mathcal{L}$ is the maximum stabbing number of all $\ell \in \mathcal{L}$. The goal of the problem is to find a subset of $\mathcal{O}$ satisfying property π and leading to the smallest possible stabbing number relative to $\mathcal{L}$. This thesis brings both theoretical and experimental contributions to the investigation of some stabbing problems.

The works of Fekete, Lübbecke and Meijer [17, 18] solved the open problem relative to the complexity of finding a spanning tree with minimum stabbing number. They also showed that finding a perfect matching with minimum stabbing number is $\mathcal{NP}$-hard. Integer programming formulations for the problems were also presented. However, very few computational experiments were performed.

In this thesis, we study integer programming formulations for the problems of finding perfect matchings, spanning trees and triangulations with minimum stabbing number. Based on these formulations we present exact algorithms and Lagrangian heuristics to solve the problems. These algorithms show that the Lagrangian heuristics yield solutions with good quality, often optimal, in short computation time.

Of all the ten problems and variants discussed in [18], for only three of them the complexity was not proved: The Minimum Stabbing Triangulation, axis-parallel and general stabbers, and The Minimum Crossing Triangulation, general case. In this thesis, we prove that the three problems are $\mathcal{NP}$-hard.

Another problem of finding a structure with minimum stabbing number is presented in [2] and also studied in [16]. This problem asks for a rectangular partition with minimum stabbing number in a rectilinear polygon. Although the complexity of the problem is still unkown, in [2] a 3-approximation algorithm is presented. In [16] an integer programming formulation is given and a 2-approximation is claimed.

In this thesis, we strengthen the formulation introduced in [16]. We also propose an alternative model and compare the formulations both theoretically and computationally. Furthermore, we show that the algorithm proposed in [16] can not provide a 2-approximation for the problem.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

A problem of finding a structure with minimum stabbing number, in general, has its input formed by two sets of geometrical objects: the finite or infinite set $\mathcal{L}$ of stabbers and the set $\mathcal{O}$. A feasible solution for the problem is a subset $\mathcal{O}'$ of $\mathcal{O}$ satisfying a given structural property π. Given $\mathcal{O}'$, the stabbing number of $\ell \in \mathcal{L}$, as defined in [17, 18], is the total of elements of $\mathcal{O}'$ that are intersected by ℓ. The stabbing number of $\mathcal{O}'$ relative to $\mathcal{L}$ is the maximum stabbing number of all $\ell \in \mathcal{L}$. The goal of the problem is to find a subset of $\mathcal{O}$ satisfying property π and having the smallest possible stabbing number.

Related problems are those of finding structures with minimum crossing number. The input of this kind of problem is the same as that for stabbing problems, i.e., a set $\mathcal{L}$ and a set $\mathcal{O}$. A feasible solution for the problem is also given by a subset $\mathcal{O}'$ of $\mathcal{O}$ satisfying a given structural property π. According to the definition in [17, 18], given $\mathcal{O}'$, the crossing number of $\ell \in \mathcal{L}$ is the number of connected components in the intersection of ℓ and $\mathcal{O}'$. And as for the stabbing number, the crossing number of $\mathcal{O}'$ relative to $\mathcal{L}$ is the maximum crossing number of all $\ell \in \mathcal{L}$, while the goal of the problem is to find a subset of $\mathcal{O}$ satisfying property π and having the smallest possible crossing number.

Consider for instance the set of points $\mathcal{P}$ in Figure 1.1 (a). Let the set of stabbers $\mathcal{L}$ be the set of dashed lines in that figure and let $\mathcal{O}$ be the set of all line segments having points in $\mathcal{P}$ as its extremities. Let the property π be: being a single connected component. Then, let $\mathcal{O}'$ be the set of line segments having points of $\mathcal{P}$ as its extremities shown in Figure 1.1 (b). Since $\mathcal{O}'$ satisfies π it is a valid solution for the problem with stabbing number 7 (because line s stabs this number of segments in $\mathcal{O}'$ and no other line in $\mathcal{L}$ stabs more segments than s). This solution is said to be optimal if no other solution has a stabbing number smaller than 7.

In 2001, Mitchell and O'Rourke published the "Computational Geometry Column 42" [30] , containing a compendium of thirty previously published open problems in computational geometry. From this list, problem number 20 stated: "What is the complexity of computing a spanning tree of a planar point set having minimum stabbing number? The *stabbing number* of a tree T is the maximum number of edges of T intersected by a line. Any set of n points in the plane has a spanning tree of stabbing number $O(\sqrt{n})$, and this bound is tight in the worst case[1]. However, nothing is known about the complexity of

[1]i.e., there are instances for which the stabbing number of any spanning tree is at least $O(\sqrt{n})$

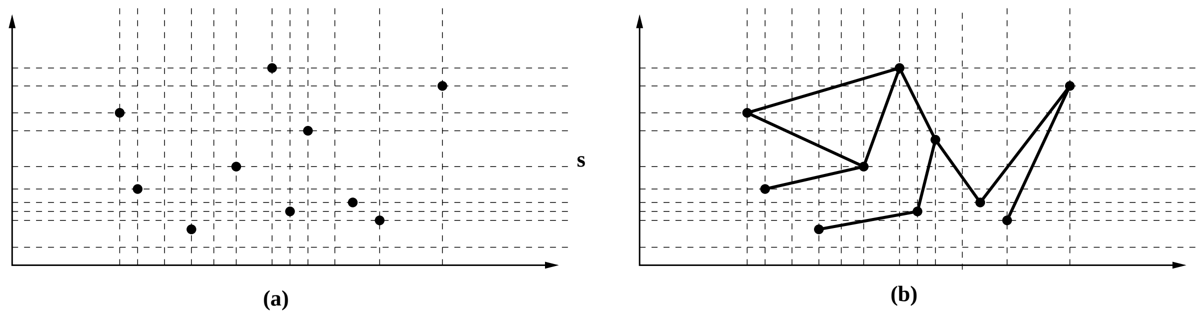

Figure 1.1: Instance of a problem of finding a structure with minimum stabbing number.

computing a spanning tree (or triangulation) of minimum stabbing number, exactly or approximately." [30]. This list then gave birth to the *Open Problems Project* [14], a list of problems without known solution by the time they were incorporated to the list.

Spanning Trees with low stabbing number can be used to construct data structures that have applications in computational geometry, computer graphics and virtual reality [43, 3]. The same is true for triangulations with low stabbing number [26, 25]. Usually, for these applications, guaranteeing a stabbing number $O(\sqrt{n})$ or $O(\log n)$ is enough and we are unaware of applications that require an optimal stabbing number. Notice, however that although stabbing problems have been known for a long time, the complexity of finding a **spanning tree with minimum stabbing number** (MSST) remained open until recently and it was open until now for the problem of finding a **triangulation with minimum stabbing number** (MSTR). Moreover, the cost measurement of a solution for the problem, i.e., its objective function, is not so usual in combinatorics, which makes the problem interesting by itself. Therefore, it should be noted that our primary interest in the problem is of a theoretical nature.

In [17, 18], Fekete, Lübbecke and Meijer studied problems of finding minimum stabbing number structures such as **perfect matchings** (MSPM), **spanning trees** (MSST) and **triangulations** (MSTR). They also considered the problems of finding the same structures with minimum crossing number (respectively, MCPM, MCSTand MCTR). In those papers they proved that finding a perfect matching or spanning tree with minimum stabbing or crossing number is $\mathcal{NP}$-hard in the general and axis-parallel cases. They also proved that finding a triangulation with minimum crossing number is $\mathcal{NP}$-hard in the axis-parallel case. The authors also presented integer programming (IP) formulations for the problems and a heuristic based on an iterated rounding procedure which was conjectured to define an approximation algorithm. Some computational experiments for the minimum stabbing perfect matching were also reported. While several contributions to minimum stabbing problems were given in [18], some problems were still left open, among them are the complexity of MSTR in both axis-parallel and general cases, and the complexity of MCTR in the general case.

Durocher and Mehrabi studied the problem of finding a rectangular partition of a rectilinear polygon with minimum stabbing number (RPST) [16]. The problem of finding a rectilinear decomposition with low stabbing number was introduced in [12] and the corresponding minimization problem was studied in [2] where a 3-approximation algorithm was presented for the problem. The paper by Durocher and Mehrabi caught our attention for two reasons: first it was about finding a structure with minimum stabbing number and second, it used integer programming to find an approximation algorithm for the conforming case of the problem which they proved to be $\mathcal{NP}$-hard (no complexity result was known before). In [16] the IP model was also extended for the general case, however no polyhedral study or computational experiments were performed.

In [13] and [8] the problem of finding a rectangular partition with minimum length (RGP) was studied. Two IP formulations for the problem were described and some algorithms were developed for it. As it turns out, the ideas used in the models studied for the RGP can also be applied to model the RPST. Moreover we can use the results obtained in those papers for the RGP to achieve similar results for RPST.

1.1 Contributions

The main contributions of this thesis are:

- We present the first integer programming formulations for MSTR and new formulations for MSST, based on the models introduced in [18].
- Computational results for MSTR are reported for the first time.
- We propose and experiment with Lagrangian heuristics for MSPM, MSST and MSTR.
- MSTR is shown to be $\mathcal{NP}$-hard both in the axis-parallel and general cases.
- We prove that MCTR is $\mathcal{NP}$-hard in the general case.
- We present computational results for an iterated rounding algorithm for MSTR.
- We perform a polyhedral study for the existing integer programming model of RPST, propose a new one and compare the strengths of these alternative formulations.
- Computational results for RPST are reported for the first time.
- We present a counterexample for a claimed 2-approximation algorithm for RPST proposed earlier in the literature.

1.2 Structure of the Thesis

This document is a compilation of the papers published or submitted to publication by the author with other researchers as a result of the investigation carried out during the doctoral program. Chapters 3, 4 and 5 correspond to those papers, [37], [38], [35], respectively. Following the rules of the graduate program of the University of Campinas, the papers are reproduced here without modification, except for the printing format. Chapter 6 corresponds to a technical note made public through `arXiv` [1]. The structure of this chapter is the same of the ones corresponding to published or submitted articles.

Each one of the chapters 4 to 6 are divided into three parts. The first part stands for a brief description of the paper informing, for instance, whether the paper is published or submitted. The second part is the text of the paper itself. Finally, the last part corresponds to the references of the original paper.

The next chapter summarizes some of the basic theoretical concepts and techniques necessary to understand the rest of the document.

Chapter 3 contains IP models for the stabbing problems described in [17]. These models are then used to develop exact Branch-and-Bound (B&B) and Branch-and-Cut (B&C) algorithms for the problems. Next, Lagrangian Relaxation (LR) of the models are utilized to produce heuristic algorithms which are then compared to the exact algorithms.

In Chapter 4, the complexity of the Minimum Stabbing Triangulation Problem and Minimum Crossing Triangulation Problem are studied. The axis-parallel case of MCTR was shown to be $\mathcal{NP}$-hard in [17], however, the complexity of the general case was left open. The complexity of MSTR was still unknown both in the general and axis-parallel cases. In this chapter we prove that these three problems are $\mathcal{NP}$-hard.

The problem of finding rectangular partitions of rectilinear polygons with minimum stabbing number is the subject of Chapter 5. In this chapter, we present IP models for the RPST and compare their strengths. We also show a relationship between RPST and RGP, this relationship is used to prove properties about the polyhedron defined by one of the IP models for RPST. Computational experiments are performed to compare the B&B algorithms derived from the different formulations.

Chapter 6 is dedicated to presenting a counterexample for the approximation algorithm proposed in [16] for the RPST. We analyse the proposed IP model and algorithm and show that it cannot lead to an approximation as claimed.

Finally, Chapter 7 presents some conclusions regarding the entire work and discusses possible directions for future work.

Chapter 2

Basic Concepts

The purpose of this chapter is to introduce basic concepts that will be necessary for the comprehension of the rest of this thesis.

All the problems treated in this text are combinatorial problems in graph theory and computational geometry. We approach these problems using integer programming and polyhedral combinatorics techniques. Moreover, we analyze the complexity of some of these problems. In Section 2.1 we present some definitions from graph theory and computational geometry. In Section 2.2 elements of computational complexity are introduced and, finally, Section 2.3 shows some important concepts from integer programming and polyhedral combinatorics. Notice that it is not our intention to write an exhaustive text on these subjects and very thorough texts can be found at [5, 7, 19, 31, 32, 40, 41, 44].

2.1 Graphs and Computational Geometry

Graphs are very versatile mathematical structures for modelling. Formally speaking, a **graph** G is composed by a set of vertices V (or $V(G)$) and a set of edges E (or $E(G)$), where $E \subseteq V \times V$. We use the notation $G = (V, E)$ to indicate the components of a graph G.

If $e = (u, v)$ is in E, we say that the vertices u and v are **adjacent** or **neighbours** and that u and v are the **extremes** of e. The **degree** of a vertex v is the number of vertices that are adjacent to v. The graphs used in this work are **simple graphs**, i.e., there are no edges of the form (v, v) and there is at most one edge for each pair of vertices. In this text, we are also dealing with **undirected graphs**, that means $(v, u) = (u, v)$ for every u and v in V.

A graph $G = (V, E)$ is said to be **weighted** if there is a function $w : E \to \mathbb{R}$ associating a real number (weight) to each edge of G.

A **subgraph** H of G, denoted by $H \subseteq G$, is a graph where $V(H) \subseteq V(G)$, $E(H) \subseteq E(G)$ and, since H is also a graph, for every edge $(u, v) \in E(H)$, u and v are in $V(H)$. Whenever $(u, v) \in E(G)$ for all $u \neq v \in V(G)$, we say the graph is **complete**. A complete subgraph of a graph is called a **clique**.

Given a graph G, a sequence $(v_0, v_1, ..., v_k)$ where $v_0, v_1, ..., v_k \in V(G)$ and for $i = 0, ..., k-1$, v_i and v_{i+1} are adjacent and $v_0 \neq v_k$ is called a **path**. If on the other hand

$v_0 = v_k$, then this sequence is called a **cycle**. If a graph has at least one cycle we say it is **cyclic**, otherwise it is **acyclic**.

A graph G is said to be **connected** if for every pair of distinct vertices u and v in $V(G)$, there is a path from u to v. If a graph is connected and acyclic, it is a **tree**.

Let G be a graph and $T \subseteq G$. If T is a tree and $V(T) = V(G)$, then T is a **spanning tree** of G.

Given a graph $G = (V, E)$ and a subset M of E where no two edges in M share a vertex. The set M is called a **matching** in G. If a vertex v is an extremity of some edge in M we say v is matched. A matching where all the vertices in $V(G)$ are matched is called a **perfect matching**. Obviously, a necessary condition for a perfect matching to exist is that $|V(G)|$ be even.

A **geometric graph** $G = (V, E)$ is a graph where each vertex in V is associated to a point in a coordinate system. We say a geometric graph has a **straight-line drawing** if its edges are represented by straight-line segments connecting the points associated to the extremities of the edge. The geometric graphs discussed in Chapters 3 and 4 are geometric graphs with straight-line drawings. The euclidean distance between the extremities of an edge is commonly used as a weight function for geometric graphs with straight-line drawings.

A **polygon** is a simple closed curve composed by a finite collection of line segments. A polygon with n vertices (or n segments) can be represented as a sequence of points in the plane where for $i = 0, ..., n-1$ the i-th and $i+1$-st points in the sequence are the extremities of one of the segments defining the polygon (addition is mod n). The sequence of segments along the closed curve defining a polygon P composes the **border** or **boundary** of P, denoted by $\delta(P)$[1].

The interior of a polygon can be partitioned into smaller polygons. A very common way of partitioning is a triangulation. As the name suggests, a **triangulation** is the partition of a polygon into triangles. A triangulation of a polygon P can be achieved by adding non-intersecting diagonal segments to the interior of P. A **diagonal** is a line segment connecting two vertices of P and contained in its interior. Hence, another way of defining a triangulation is as a maximal non-intersecting set of diagonals.

Notice that usually a triangulation is not unique. However, the number of diagonals and the number of triangles in any triangulation for a given polygon is always the same. A triangulation of a polygon with n vertices always has $n-2$ triangles and $n-3$ diagonals.

Triangulation can also be applied to a set of points in the plane. Given a set P of points in the plane, a triangulation of P is a maximal planar geometric graph with vertex set P, i.e., a geometric graph where no edge can be added connecting points in P without destroying its planarity. As in the triangulation of a polygon, a triangulation of a point set P also has a constant number of triangles and internal (not in the boundary) edges. If $|P| = n$ and the boundary of the smallest polygon containing P has k points in P, then a triangulation of P has $2n - 2 - k$ triangles and $3n - 3 - k$ internal edges.

A particular type of polygons are the **rectilinear polygons**, which are simply polygons where all the segments defining it are either horizontal or vertical. A common way

[1]Polygons defined like this are also called **simple polygons**. In this document, all the polygons are considered to be simple.

of partitioning a rectilinear polygon is by dividing its interior into rectangles. Unlike triangulations though, a rectangular partition does not always have the same number of rectangles.

2.2 Complexity Theory

In 1936, Alan Turing defined the Turing Machine, a mathematical model for computation. Simply put, a **Turing Machine** consists of an infinite tape for input and output, a control unit and a read/write head. The machine is initialized with its head on the leftmost symbol of the input which is written in the tape while the remainder of the tape is empty. The control unit contains a set of internal states, three among them are special states called the initial, acceptance and rejection states. Entering either of the latter two states stops the computation immediately. A configuration of a Turing machine is composed by its current state, position of the head and content of the tape. Depending on its configuration, a machine can write something to the current position on the tape, make a head movement to the left or to the right and change its internal state. We call these three actions, a step in the computation. If the machine stops in the acceptance state we say the input is accepted. If, on the other hand, the machine stops in the rejection state, the input is rejected.

Although extremely simple, the Turing Machine model is very powerful and we still accept the **Church-Turing thesis** that states that any algorithmically solvable problem can be modelled using a Turing machine. In other words, this thesis says that Turing Machines give a formal definition for what is an algorithm. Several other computational models were proposed over the years, but according to Church-Turing thesis, the most powerful of these models must be computationally equivalent to a Turing Machine. Computational equivalence means that the set of problems that can be solved by the models are the same.

One of these models is the **Non-deterministic Turing Machine**. This model is almost identical to regular Turing Machines, the only difference is that in a deterministic (regular) model, for each configuration there is exactly one possible step the machine can take. Meanwhile, in a non-deterministic model, several steps can be taken for each configuration and the machine executes all of them simultaneously. This process can be seem as if at each configuration where more than one step is possible, the machine creates copies of itself with the new configurations and continues executing all the copies in parallel.

The **time complexity of a Turing Machine** T is a function $f : \mathbb{N} \to \mathbb{N}$ where $f(n)$ is the maximum number of steps executed by T with an input of length n. Let $t : \mathbb{N} \to \mathbb{R}^+$ be a function, then $Time(t(n))$ is the **time complexity class** of all the problems that can be solved by a Turing Machine with time complexity $O(t(n))$.

Similarly, we can define the **time complexity of a Non-deterministic Turing Machine** NT is a function $f : \mathbb{N} \to \mathbb{N}$ where $f(n)$ is the maximum number of steps executed by NT in any of its possible computation paths with an input of length n. Let $t : \mathbb{N} \to \mathbb{R}^+$ be a function, then $NTime(t(n))$ is the **time complexity class** of all the problems that can be solved by a Non-deterministic Turing Machine with time complexity $O(t(n))$.

Now, we can define the class $\mathcal{P}= \bigcup_{k\in\mathbb{N}} Time(n^k)$, i.e., $\mathcal{P}$ is the class of all the problems that can be solved by a polynomial time complexity Turing Machine. Likewise, $\mathcal{NP}= \bigcup_{k\in\mathbb{N}} NTime(n^k)$, that is, $\mathcal{NP}$ is the class of all the problems that can be solved by a polynomial time complexity Non-deterministic Turing Machine. As said before, Turing Machines are a formal definition for algorithms, hence we can restate the definition above as: $\mathcal{P}$ is the class of problems having a **polynomial time algorithm** and $\mathcal{NP}$ is the class of problems that have a **polynomial time non-deterministic algorithm**.

Let A and B be two problems having, respectively, input (output) sets I_A and I_B (O_A and O_B). Hence, an algorithm M_A for A takes an instance $a \in I_A$ and produces $M_A(a) \in O_A$. Likewise, an algorithm M_B for B takes an instance $b \in I_B$ and outputs $M_B(b) \in O_B$. If there is an algorithm R having I_A as input set and I_B as output set, where $M_A(x) = M_B(R(x))$ for any $x \in I_A$, then we say R is a **reduction** (more precisely, a mapping reduction) from A to B. Moreover, if the time complexity of R is polynomial, we say that R is a **polynomial time reduction** from A to B and that A is **polynomially reducible** to B.

Reductions can be used to "transfer" properties from one problem to another. For instance, suppose A is a problem that has no polynomial time algorithm then, if there is a polynomial time reduction from A to B, then B cannot have a polynomial time algorithm either, otherwise we get a contradiction.

We say that a problem A is $\mathcal{NP}$-hard if every problem in $\mathcal{NP}$ is polynomially reducible to A. And a problem A is $\mathcal{NP}$-complete if A is $\mathcal{NP}$-hard and A is in $\mathcal{NP}$. Originally, the classes $\mathcal{NP}$ and $\mathcal{NP}$-complete were defined for **decision problems** (problems with yes or no outputs) however, it is common to see in many texts **optimization problems** (problems where the solution is maximum or minimum) been said to be $\mathcal{NP}$-complete. The idea behind the use of these terms is that an optimization problem is said to be $\mathcal{NP}$-complete if its decision version is $\mathcal{NP}$-complete. The decision version of an optimization problem is simply a version of the problem where instead of looking to maximize (minimize) some function, one is interested in deciding whether its value can be, for instance, greater or equal (less or equal) to some constant value.

The first problem proven to be $\mathcal{NP}$-complete was the satisfiability problem (SAT). Its $\mathcal{NP}$-completeness was proven by Cook in 1971 [9]. Cook's proof shows that the computation of any Non-deterministic Turing Machine can be translated to a logical formula in conjunctive normal form in polynomial time, hence, any problem in $\mathcal{NP}$ is polynomially reducible to SAT. Besides, SAT is in $\mathcal{NP}$. The existence of an $\mathcal{NP}$-complete problem was independently discovered by Levin in 1973 [29].

Knowing an $\mathcal{NP}$-hard problem, made it easier to prove that other problems were $\mathcal{NP}$-hard. We simply have to show that an $\mathcal{NP}$-hard problem is polynomially reducible to other problems. Since then, several problems have been shown to be $\mathcal{NP}$-hard. One of these problems is 3-SAT, proven $\mathcal{NP}$-complete in 1972 by Karp [28].

The importance of the $\mathcal{NP}$-hard and $\mathcal{NP}$-complete classes is that until this day, no deterministic polynomial time algorithm exists for solving the problems in these classes. However, finding such an algorithm for a single problem is enough to show that all the problems in $\mathcal{NP}$ are also in $\mathcal{P}$, i.e., $\mathcal{P} = \mathcal{NP}$. Likewise, if it is shown that a single problem in $\mathcal{NP}$-complete demands exponential time algorithms (every algorithm from now on is to be considered deterministic unless stated otherwise), then we have $\mathcal{P} \neq \mathcal{NP}$.

3-SAT is very commonly used to prove the $\mathcal{NP}$-hardness of other problems. An idea for such a proof is to transform each component of the input of 3-SAT, i.e., variables, literals and clauses, into structures of the target problem. These structures are called **gadgets**. Next, we have to connect these gadgets in order to simulate the relationship between variables, literals and clauses. Although it may seem strange to transform the input of a problem in logic to a problem in graphs or computational geometry for example, it has been shown to be an easier path for several problems. It has been done, for instance, for the clique problem and for MSST.

2.3 Integer Programming and Polyhedral Combinatorics

The work of Dantzig, published in 1947 is often considered a mark on the beginning of linear programming as a general tool for solving optimization problems [6, 10], although other works have used linear programming before. Linear programming have shown its usefulness for countless combinatorial optimization problems.

To model (or formulate) an optimization problem as a **linear programming problem** we must define three things: the set of variables, the set of linear inequalities describing the restrictions of the problem and a linear function that establishes the value of a solution, called the objective function. Therefore, usually a linear programming model have the following form:

$$z = \min \sum_{j=1}^{n} c_j x_j \tag{2.1}$$

$$\text{s.t.} \sum_{j=1}^{n} a_{ij} x_j \leq b_i, \qquad i = 1, ..., m \tag{2.2}$$

or in matrix notation: $z = \min\{cx : Ax \leq b, x \in \mathbb{R}^n_+\}$ where A is an m by n matrix, c an n-dimensional row vector, b an m-dimensional column vector and x an n-dimensional column vector.

Even though simplex was the first general method presented to solve linear programming problems and it is very useful in practice, until this day, every pivoting rule proposed for this method has a pathological case resulting in exponential time complexity. The first known polynomial time method presented for solving the linear programming problem was the ellipsoid method in 1979. This method was originally introduced by Yudin and Nemirovski (1976) and Shor (1977) in the context of non-linear programming. But Khachiyan proved that it could be used to solve linear programs in polynomial time. Despite its polynomial time complexity, the performance of the ellipsoid method in practice was worse than the simplex method. Only in 1984 a competitive method was presented by Karmarkar, the interior-point method [42].

To understand how an optimal solution can be found, we must first understand the structure of a set of valid solutions defined by a linear program and its properties. For that we need some definitions.

A **set** S **is convex** if for each pair of points x_1 and $x_2 \in S$ every **convex combination**, i.e., $x = \alpha x_1 + (1-\alpha)x_2, \forall\ 0 \leq \alpha \leq 1$, of x_1 and x_2 is also in S.

A set of points satisfying a finite number of linear inequalities is called a **polyhedron**. Hence, it is easy to see that a linear programming model defines a polyhedron. A polyhedron is a convex set. We call a point x in a polyhedron P a **vertex** if it cannot be defined as a convex combination of other points in $P \setminus \{x\}$. The **convex hull** of a set of points P ($\text{conv}(P)$) is the smallest convex set containing all points in P.

Now, concerning the values of solutions of a linear program, there is a theorem stating: if the optimal value of the objective function of a linear program is finite and the corresponding polyhedron is non-empty, then there is always a vertex that is an optimal solution for this linear program. If more than one vertex is an optimal solution, then every convex combination of these vertices is also optimal. This means that we only need to look at the vertices of the polyhedron for optimal solutions.

If we add integrality constraints to a linear programming model we obtain what is called a linear (mixed) **integer programming** model. Notice that although linear programming problems can be solved in polynomial time, a restricted version of the integer programming problem have already been proven to be $\mathcal{NP}$-hard by Karp in 1972 [28].

Notice that the set of feasible solutions for an integer programming problem can be defined by infinitely many different formulations as shown in Figure 2.1. Therefore, a natural question that arises is: how can we determine if a formulation is better than another? To answer this question, let us analyse the following situation. Let S be the set of valid solutions for an integer programming problem I. If a formulation for I defines a polyhedron $P = \text{conv}(S)$, then every vertex of P is a point in S. Then, it is possible to prove that we can abandon the integrality constraints and solve the problem as a linear programming problem and the solution obtained is a valid optimal solution for I.

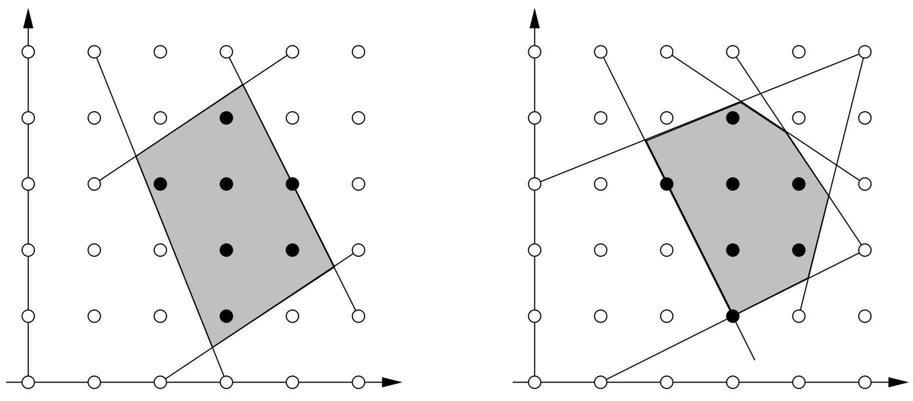

Figure 2.1: Different formulations for the same set of feasible solutions.

Therefore, the idea is to obtain a formulation that defines a polyhedron as close as possible to $\text{conv}(S)$. It is not always possible to obtain a formulation describing the convex hull of the solutions set, though. Then, in order to understand how good is a formulation we must perform a polyhedral study. The field interested in the study of the inequalities defining polyhedra is called polyhedral combinatorics. This area began with the work of Edmonds for the perfect matching polyhedron in 1965 [39].

To get a good formulation we need **strong valid inequalities**. An inequality is **valid** if every point in a solution set S satisfies the inequality. Every inequality defines a **face** of a polyhedron and the **strength of an inequality** depends on the **dimension** of the face characterized by it. Given a polyhedron $P \subseteq \mathbb{R}^n$ and an inequality $\pi x \leq \pi_0$ valid for P (where $\pi \in \mathbb{R}^n$ and $\pi_0 \in \mathbb{R}$) the inequality is said to define a face $F = P \cap \{x \in \mathbb{R}^n : \pi x = \pi_0\}$. If $F \neq \emptyset$ and $F \neq P$, then F is a proper face of P. Notice that from the definition of a polyhedron, F is also a polyhedron. In order to state what the dimension of a face (or a polyhedron) is, the definition of an **affinely independent** set is necessary.

A set of points $x_1, ..., x_n$ is affinely independent if the only solution to $\sum_{i=1}^{n} \alpha_i x_i = 0$, $\sum_{i=1}^{n} \alpha_i = 0$ with $\alpha_i \in \mathbb{R}$ is $\alpha_1 = \alpha_2 = ... = \alpha_n = 0$.

A polyhedron $P \subseteq \mathbb{R}^n$ has **dimension** $(dim(P))$ k if there are $k+1$ affinely independent points in P. A polyhedron is said to be full-dimensional if its dimension is the same as the one of the space containing it so, in this case, if $dim(P) = n$ we say P is **full-dimensional**. Since a face F of P is also a polyhedron, it is clear that $dim(F)$ is the number of affinely independent vectors in F. If F is a proper face of P, it is easy to see that the greatest possible value for $dim(F)$ is $dim(P) - 1$. If a face have dimension $dim(P) - 1$, it is called a **facet**.

It is noteworthy that the number of inequalities necessary to describe the convex hull of the set of solutions for an IP may be exponential. Hence, in these cases, it is impossible to use a formulation completely describing the convex hull of the problem in an algorithm. In this situation if we abandon the integrality constraints and use a linear programming algorithm the solution may not be an integral solution.

The formulation obtained from an integer program by abandoning its integrality constraints is called a **linear programming relaxation**. Given two problems $(RP) z^R = min\{f(x) : x \in T \subseteq \mathbb{R}^n\}$ and $(IP) z = min\{c(x) : x \in X \subseteq \mathbb{R}^n\}$, we say that (RP) is a relaxation of (IP) if $X \subseteq T$ and $f(x) \leq c(x) \forall x \in X$, then, it is easy to see that $z^R \leq z$. This means that a linear programming relaxation provides a **lower (dual) bound** [2] for the original IP problem.

Fortunately, we do not need the description of the entire convex hull to find an optimal solution, we only need the inequalities that are active in an optimal solution, see Figure 2.2. Therefore, we can start with a weaker formulation and include inequalities as they are needed. Algorithms that use this idea are called **cutting plane algorithms** (CPA). Such an algorithm works as follows: given an IP problem P, at each iteration a linear relaxation of P is solved. If the solution is integral, it must be optimal and the algorithm stops. Otherwise, a inequality $\pi x \leq \pi_0$ valid for P and violated by the solution (such inequality is called a cut) is added to the problem and the process is repeated. At each iteration the value of the linear relaxation obtained increases (for a minimization problem) and eventually it becomes integral and hence, optimal. Figure 2.3 shows a representation of an iteration of a CPA where an inequality is added to cut off a fractional solution.

[2]for a maximization problem the dual bound provided by the linear programming relaxation would be an upper bound.

The problem of finding a valid inequality that cuts off a fractional solution is called the **separation problem**. An algorithm to solve this problem is called a separation routine, i.e., a separation routine looks for valid inequalities that are violated by the current solution. Grötschel, Lovász and Schrijver showed the complexity equivalence between separation and optimization [22].

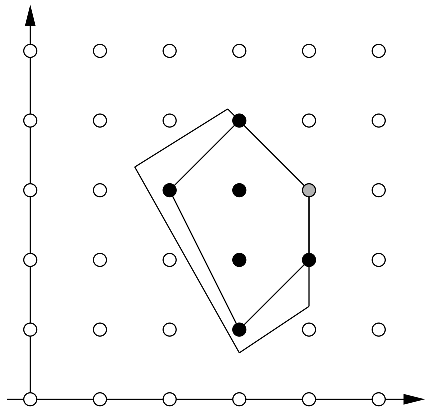

Figure 2.2: A formulation with inequalities that are active in the optimal solution.

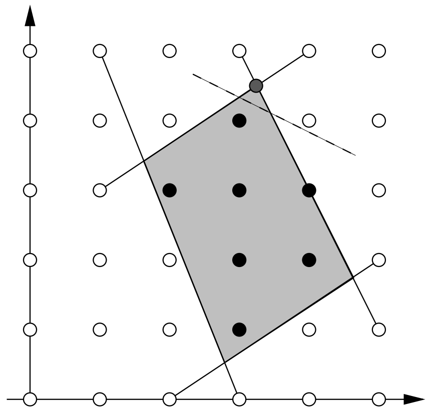

Figure 2.3: Formulation with a fractional optimal solution and a cut (represented by the dashed line).

The first cutting plane algorithm was introduced by Gomory in 1958 [39]. The cuts described by Gomory are called Gomory's cuts and although they guarantee to find an optimal solution in finite time, the original algorithm was very inefficient in practice.

Another commonly used technique to solve IP problems is **Branch-and-Bound** (B&B). The basic idea behind a B&B algorithm is to **decompose** the problem in smaller and easier to solve parts and afterwards, use this information to solve the original problem. For instance, let $z = min\{cx : x \in S\}$, we would like to partition S in $S = S_1 \cup ... \cup S_K$ and we have $z^k = min\{cx : x \in S_k\}$ for $k = 1, ..., K$ and $z = min\{z^k : k = 1, ..., K\}$. Notice that the partition can be constructed in an iterative fashion first dividing the set in a small number of subsets and then dividing these subsets and so on. Figure 2.4 depicts a representation of a partition of the set of feasible solutions.

A B&B algorithm can usually be represented by an **enumeration tree**. The partition in Figure 2.5 is obtained by fixing binary variables to its possible values. It is clear from

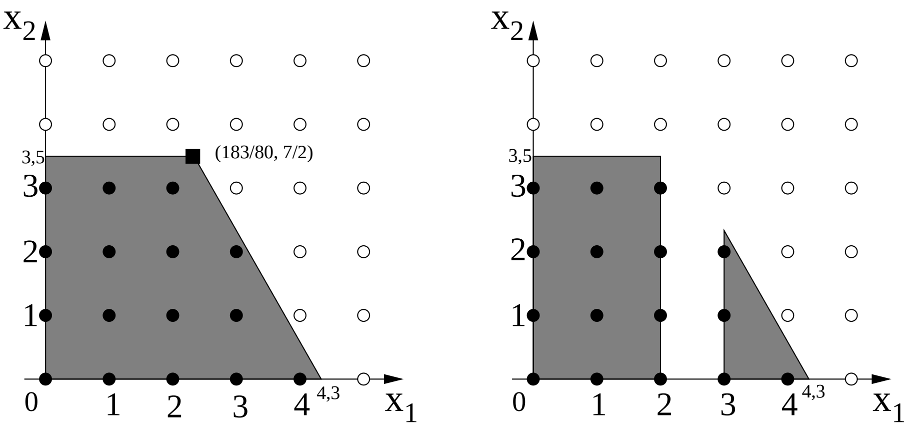

Figure 2.4: Partition of the set of feasible solutions in two sets. The first set corresponds to the solutions satisfying $x_1 \leq 2$ and the second $x_1 \geq 3$.

that image that a complete enumeration would take a number of steps that is exponential in the number of variables.

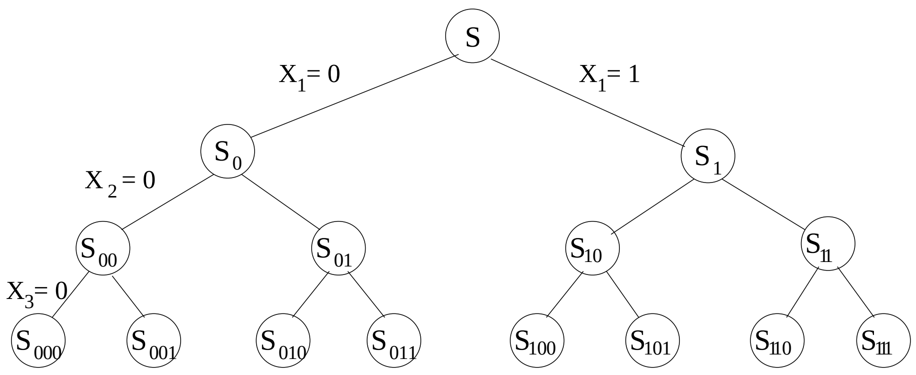

Figure 2.5: Enumeration tree of a B&B where the decomposition is done by fixing variables at different values. Figure extracted from [44]

Since a complete enumeration is impossible in practice, we try to make the **enumeration implicitly**. This is done by **pruning** the enumeration tree using bound information. To see why pruning is possible we just need to know the following property: let $S = S_1 \cup ... \cup S_K$ be a decomposition of S, let $z^k = min\{cx : x \in S_k\}$ for $k = 1, ..., K$, let $\bar{z}^k$ be an upper bound on z^k and $\underline{z}^k$ be a lower bound on z^k. Then $\underline{z} = min\{\underline{z}^k : k = 1, ..., K\}$ is a lower bound on z and $\bar{z} = min\{\bar{z}^k : k = 1, ..., K\}$ is an upper bound on z. In other words, considering a minimization problem, the minimum value among the lower bounds of all the nodes is a lower bound for the entire tree and the minimum value among the upper bounds in every node is an upper bound for the entire tree. Understanding this property, we can see that there are three types of possible pruning.

The first type of pruning is **by optimality**. This happens when the lower and upper bounds are the same in a given node. It means that the solution obtained is optimal and, therefore there is no reason to keep looking for a better solution in that sub-tree. An example of pruning by optimization is shown in Figure 2.6. Another type of pruning is **by bound**. This pruning happens when the global upper bound is smaller than the local lower bound of a given node. That means that no better bound can be produced by the corresponding sub-tree hence, it can be cut from the enumeration. Figure 2.7 shows this situation. The last type of pruning is **by infeasibility**, which happens when there is no feasible solution in a given node, making the entire sub-tree unfruitful.

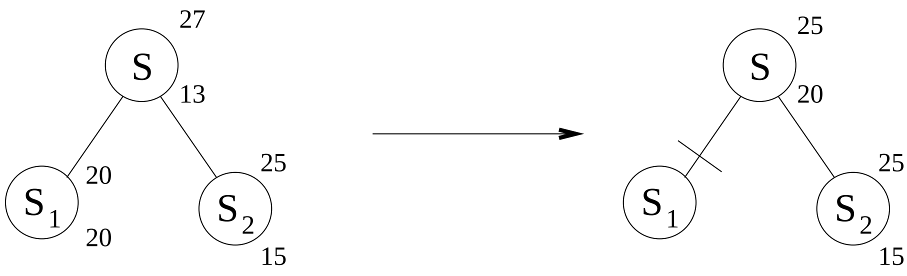

Figure 2.6: Pruning of an enumeration tree by optimality. Figure extracted from [44]

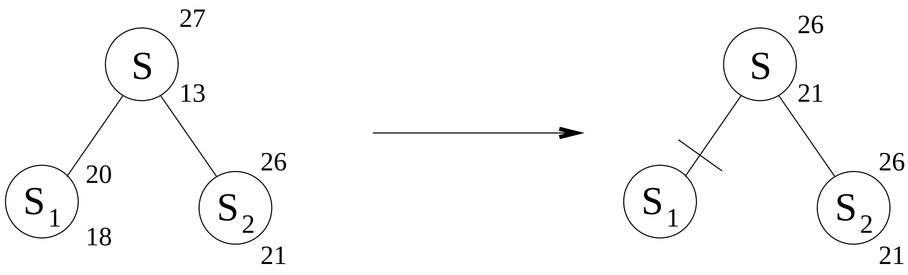

Figure 2.7: Pruning of an enumeration tree by bound. Figure extracted from [44]

When a B&B algorithm is combined with a CPA we obtain a **Branch-and-Cut** (B&C) algorithm. In this kind of algorithm, at each node of the enumeration tree, a separation routine is executed to find violated valid inequalities. Therefore, the idea of a B&C algorithm is to use the strengths (and weaknesses) of B&B and CPA at the same time.

Although linear relaxation is very commonly used, it is not the only kind of relaxation that exists. Another kind of relaxation is the **Lagrangian Relaxation**. Given an IP (IP):

$$
\begin{aligned}
(IP) \quad z = \min cx & \\
Ax \leq b, & \qquad (2.3)\\
Dx \leq d, & \qquad (2.4)\\
x \in \mathbb{Z}^n_+, &
\end{aligned}
$$

suppose $Ax \leq b$ is a set of “nice” restrictions while $Dx \leq d$ is a set of “hard” restrictions. The terms “nice” and “hard” here mean that if we remove the inequalities in $Dx \leq d$ from (IP), the resulting problem can be more easily solved. Then, in a LR the “hard” inequalities are dualized by adding the term $\lambda(Dx - d)$ to the objective function for a given vector $\lambda \geq 0$. The idea is to penalize the objective function whenever an inequality is violated. The resulting problem is:

$$\begin{aligned} (IP(\lambda)) \quad z(\lambda) = \min\, & cx + \lambda(Dx - d) \\ & Ax \leq b, \\ & x \in \mathbb{Z}^n_+, \end{aligned}$$

The problem $(IP(\lambda))$ is called the **Lagrangian Primal problem** and for all $\lambda \geq 0$, it is a relaxation of (IP), hence $z(\lambda) \leq z$. However, it would be interesting to obtain a value for λ so that $z(\lambda)$ is as great as possible, thus providing the best dual bound. This can be achieved by solving the **Lagrangian Dual problem**:

$$(LD) \quad \max\{z(\lambda) : \lambda \geq 0\}$$

It is possible to prove that the dual bound obtained by solving the Lagrangian Dual problem is at least as good as the one obtained from a linear relaxation. The Lagrangian Dual problem can be solved using a **Subgradient Method** (SGM) as described in [4, 44].

The Lagrangian multiplier method was introduced by Everett in 1963 [27], but it became popular after the works of Held and Karp in 1970 and 1971 [23, 24] solving large instances (at the time) of the travelling salesman problem.

It is also possible that an IP problem have an exponential number of variables. In this case, it is clearly not possible to solve the problem containing all the variables. So, instead we iteratively solve a partial problem with a subset of variables and try to find a variable that is not in the formulation and could improve the value of the solution. The problem of finding such variables is the **pricing problem**. An iterative algorithm as this is called a **column generation algorithm** (CGA).

The term **reduced cost** of a variable is usually used to describe how much the objective function has to improve before the corresponding variable can have a positive value in an optimal solution. Therefore, the pricing algorithms look for variables with negative reduced cost.

It is not hard to see that column generation is very similar to cutting plane algorithms. But while in CPA we have separation procedures, in CGA we have pricing procedures. In fact, CGA is the dual of CPA. Therefore, CGA can also be combined with B&B to produce what is named a **Branch-and-Price** (B&P) algorithm. CGA and B&P algorithms first appeared in the 60's in [11, 20, 21].

Chapter 3

Integer programming approaches for Minimum Stabbing Problems

This chapter corresponds to a paper published in Rairo-OR special issue of the 2nd International Symposium on Combinatorial Optimization (ISCO 2012) under DOI: 10.1051/ro/2014008 [37]. The original publication is available at www.rairo-ro.org and the copyright is owened by EDP Sciences. The paper was co-authored by Cid C. de Souza, Yuri Frota and Luidi Simonetti. In this paper, we present integer programming exact algorithms and lagrangian relaxation heuristics for the problems of finding perfect matchings, trees and triangulations with minimum stabbing number. The paper presented at ISCO 2012 that originated the article corresponding to this chapter is [33].

The problem of finding structures with minimum stabbing number has received considerable attention from researchers. Particularly, [10] study the minimum stabbing number of perfect matchings (MSPM), spanning trees (MSST) and triangulations (MSTR) associated to set of points in the plane. The complexity of the MSTR remains open whilst the other two are known to be $\mathcal{NP}$-hard. This paper presents integer programming (IP) formulations for these three problems, that allowed us to solve them to optimality through IP branch-and-bound (B&B) or branch-and-cut (B&C) algorithms. Moreover, these models are the basis for the development of Lagrangian heuristics. Computational tests were conducted with instances taken from the literature where the performance of the Lagrangian heuristics were compared with that of the exact B&B and B&C algorithms. The results reveal that the Lagrangian heuristics yield solutions with minute, and often null, duality gaps for instances with several hundreds of points in small computation times. To our knowledge, this is the first computational study ever reported in which these three stabbing problems are considered and where provably optimal solutions are given.

3.1 Introduction

Given a set of points P in the plane, the *geometric graph* associated to P is the graph $G(P) = (V, E)$ whose vertices are the points in P and whose edges are the straight line segments with both extremities in P. The *stabbing number of a line ℓ* passing through a geometric (sub)graph $G(P) = (V, E)$ is defined as the number of edges in E having a non-empty intersection with ℓ. Given a set L of straight lines, the *stabbing number of a (sub)graph* $G(P) = (V, E)$ is the maximum number of intersections between any line in L and the edges in E. The problem of finding a structure with minimum stabbing number can be defined for any kind of structure, e.g. Perfect Matchings, Spanning Trees, Triangulations etc. So, for example, the problem of finding the Minimum Stabbing Perfect Matching (MSPM) can be described as follows: given a set of points P, and a set of straight lines L, find a perfect matching in the geometric graph $G(P)$, among every possible perfect matchings in $G(P)$, having a stabbing number with minimum value. Two versions of the problem are presented in [9, 10] and are related to the choice of the set L. In the first version, here referred as the *general stabbing* one, L is defined as the infinite set formed by all straight lines that can be drawn in the plane. In the *axis parallel* version, L is the, also infinite, set composed solely by the vertical and horizontal lines in the plane. Figure 3.1 illustrates the two versions of the problem with a triangulation of stabbing numbers 14 and 9, respectively.

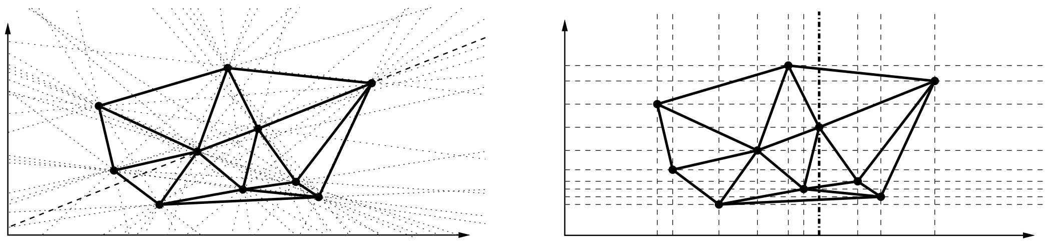
Figure 3.1: A triangulation with general (axis parallel) stabbing number 14 (9).

Motivation. Stabbing problems have received considerably attention in the Computational Geometry community. In 2001 Mitchell and O'Rourke published a list with thirty open problems in the field [16], given rise to *The Open Problems Project* [6], containing a list of geometric problems whose complexity, at that time, was unknown. The list, which is constantly updated, is an invaluable source of challenging problems in Computational Geometry. In [9, 10] general and axis parallel versions of the Minimum Stabbing Perfect Matching (MSPM), Minimum Stabbing Spanning Tree (MSST) - problem #20 of the aforementioned list - and Minimum Stabbing Triangulation (MSTR) were discussed. For the first two problems approximation algorithms were presented and $\mathcal{NP}$-hardness proofs were given for both versions of the problems. Computational results are presented for the MSPM. The complexity status of MSTR could not be established and no algorithms were developed or tested to solve it. Heuristics for the spanning tree, perfect matching and triangulation stabbing problems were investigated in [17]. These heuristics are mostly based on greedy and divide-and-conquer techniques. Contrarily to the Lagrangian heuristics proposed here, they are not able to provide the duality gap associated to the solution they yield. In [17] the limited amount of information about computational experiments refers exclusively to the spanning tree case. Other works related to finding geometric structures with minimum or low stabbing number include [4], [1], [24] and [26].

Our contribution. This paper presents two IP formulations for the MSTR based on the ideas described in [9, 10, 20] and one formulation for the MSST which explores the results given in [9, 10, 15]. Later, these formulations and a variation of the one described in [9, 10] for the MSPM are used to implement exact branch-and-bound (B&B) and branch-and-cut (B&C) algorithms for the corresponding problems, which allowed, for the first time in the literature, to obtain solutions with proven optimality. Besides, Lagrangian relaxation (LR) heuristics based on the IP models for the three problems are presented and appropriate subgradient methods are implemented. Computational results obtained by the Lagrangian algorithms are reported with instances taken from the literature and reveal that optimality or minute duality gaps are achieved in small computation times.

In the triangulation case, it was of paramount importance the realization of the relation existing between the Minimum Weight Triangulation (MWT) and the MSTR. This led to the development of strong IP models for the latter and also to the usage of effective algorithms to solve the MWT. As we will see later, such algorithms play an important role in our Lagrangian heuristic for MSTR.

Before continuing, we must observe that an early version of this paper appeared in the Proceedings of ISCO 2012 [22]. Thus, this work is to be seen as an extended and more complete version of that previous work.

Organization of the text. The remaining of this document is organized as follows. Section 3.2 presents IP models for the problems studied. Section 3.3 describes how to derive a LR heuristic for the problems from the IP models, whilst in Section 3.4 we present our computational results. At last, in Section 3.5 we draw some conclusions and indicate future research directions to be pursued.

3.2 Integer Programming Models

In the current section we present IP models for the three problems under consideration in this paper, where the model for the MSPM is extracted from [9, 10] and the models for the MSST and MSTR are based on the ideas presented in those papers. The formulations described here will be used in the implementation of exact B&B and B&C algorithms. Also, in Section 3.3, we show how to obtain LRs for each problem using the models introduced in this section, and use them to produce primal and dual bounds for the true optimum.

Stabbing Perfect Matchings. We first present the model for the MSPM. We are given the sets P and L of points and stabbing lines, respectively, and E denotes the set of edges of the geometric graph $G(P)$. Variable k denotes the stabbing number and, therefore, must be minimized. Variable x_{ij} is set to 1 when the edge ij is in the solution and 0 otherwise.:

$$
\begin{array}{llllll}
(MSPM) & z = \min k & & & & (3.1)\\
\text{subject to} & & & & & \\
& \sum_{ij \in E} x_{ij} & = & 1, & \forall\, i \in P, & (3.2)\\
& \sum_{ij \in E: i,j \in S} x_{ij} & \leq & (|S|-1)/2, & \forall\, S \subset P, |S| \text{ odd}, & (3.3)\\
& \sum_{ij \in E: ij \bigcap s \neq \emptyset} x_{ij} & \leq & k, & \forall\, s \in L. & (3.4)\\
& k \in \mathbb{Z}, x_{ij} \in \mathbb{B} & & & \forall\, ij \in E. & (3.5)
\end{array}
$$

In this formulation, constraints (3.2) and (3.3) guarantee that the solution is a perfect matching. The first enforces each vertex to have degree one and the second – although, satisfied by any integral solution and, therefore, not strictly necessary for the correctness of the model – strengthens the linear relaxation, as proved by Edmonds [8]. The third class of inequalities is formed by the stabbing inequalities and they state that the sum of the variables corresponding to the edges intersecting a given line $s \in L$ must always be smaller or equal to the stabbing number, k. Notice that, as observed in [9, 10], in principle, this formulation in not finite since there are infinitely many stabbing lines. However, considering the axis parallel version, when sweeping a stabbing line in a direction d, the stabbing number only changes at a point of P. For this reason, we only need to look at a linear number of stabbing lines, thus, making the model finite. Following a similar reasoning, when considering the general version, we only need to look at a quadratic number of lines, namely, those defined by each pair of points in P.

Stabbing Spanning Trees. There are a number of known IP formulations for the Minimum Spanning Tree Problem (MST), including some that define the convex hull of the points corresponding to integer solutions. So, in order to decide which one should be used to build a formulation for the MSST, we first implemented three of the strongest formulations described in [15] for the MST. After a few computational tests, we observed that the directed cut formulation had the best practical performance compared to the other alternatives. Hence, we decide to use this model as the basis for our MSST formulation described below.

Consider a digraph $D = (P, A)$, where A is the set of arcs connecting each pair of vertices in P, i.e., for each edge $ij \in E$ there is a pair of arcs (i, j) and (j, i). We arbitrarily set a vertex r as the root of the tree. The notation $\delta^+(C)$ refers to the cutset directed out of vertex set C and $\delta^-(C)$ to the cutset directed into the vertex set C. The variable $y_{ij} = 1$ if the tree contains arc (i, j) when rooted at r and $x_{ij} = 1$ if one of the arcs (i, j) or (j, i) is in the tree with r as root. The relationship between y and x variables is established by constraint (3.9).

$$
\begin{array}{llllll}
(MSST) & z = \min k & & & & (3.6)\\
\text{subject to} & \displaystyle\sum_{(i,j)\in\delta^+(C)} y_{ij} & \geq & 1, & \forall\, C \subset V \text{ with } r \in C & (3.7)\\
& \displaystyle\sum_{ij\in A} y_{ij} & = & |P|-1, & & (3.8)\\
& y_{ij} + y_{ji} & = & x_{ij}, & \forall ij \in E & (3.9)\\
& \displaystyle\sum_{ij\in E: ij \bigcap s \neq \emptyset} x_{ij} & \leq & k, & \forall\, s \in L. & (3.10)\\
& y_{ij} \in \mathbb{B} & & & \forall (i,j) \in A & (3.11)\\
& k \in \mathbb{Z}, x_{ij} \in \mathbb{B} & & & \forall\, ij \in E. & (3.12)
\end{array}
$$

As before, part of the formulation is composed by a set of constraints ((3.7), (3.8) and (3.9)) ensuring that the resulting solution is a geometric subgraph of the required type, in this case a spanning tree. The remaining constraints are stabbing inequalities (3.10), which have the same meaning as before. Constraint (3.8) guarantees that the solution has $|P|-1$ arcs, as required in a directed spanning tree. Finally, constraints (3.7) enforces that the solution is a directed connected graph.

Stabbing Triangulations. Next, the ideas used in the models above and the IP models for the MWT that can be found in [20] form the point of departure to build the Edge and Triangle Stabbing models for the MSTR. The first of these two models is simpler and, for this reason, easier to use in a Lagrangian Relaxation algorithm. The second, although more complicated, provides better bounds and, therefore, was used in a exact B&B algorithm.

In the Edge Stabbing model ($MSTE$), P_H is the set of vertices on the convex hull of P; a *crossing set* (Cr) is defined as a maximal set of edges which are pairwise intersecting (endpoints excluded); the set of all crossing sets in $G(P)$ is denoted by S_{Cr}; for an edge $pq \in E$, $Cr(pq)$ denotes the set of edges intersecting pq (again with endpoints excluded) plus pq itself; the rest of the notation stands for the same as before. For every $ij \in E$, $x_{ij} = 1$ if and only if the edge ij is in the triangulation. The variable k, once again, denotes the stabbing number. Then, the Edge Stabbing Model reads:

$$
\begin{aligned}
(MSTE) \quad & z = \min k && && && (3.13)\\
\text{subject to} \quad & \sum_{ij \in E} x_{ij} &=& \; 3|P| - |P_H| - 3, && && (3.14)\\
& \sum_{ij \in Cr} x_{ij} &\leq& \; 1, && \forall\, Cr \in S_{Cr}, && (3.15)\\
& \sum_{ij \in Cr(pq)} x_{ij} &\geq& \; 1, && \forall\, pq \in E, && (3.16)\\
& \sum_{ij \in E : ij \bigcap s \neq \emptyset} x_{ij} &\leq& \; k, && \forall\, s \in L. && (3.17)\\
& k \in \mathbb{Z}, x_{ij} \in \mathbb{B} && && \forall\, ij \in E. && (3.18)
\end{aligned}
$$

In this model, (3.14) guarantees that the solution has the right number of edges required for a triangulation of P. Constraint (3.15) states that only one edge in a crossing set can be in the solution, thus, ensuring planarity. Constraint (3.16) states that, either pq or at least one of the edges in $Cr(pq)$ must be in the solution, therefore, enforcing maximality (recall that a triangulation is a maximal planar subgraph of $G(P)$). It is noteworthy that constraint (3.16) is not strictly necessary for the formulation. However, as observed in [20], it greatly enhances the computational performance of the IP algorithms. Constraint (3.17) states that, for each stabbing line s in L, the number of edges from triangulation that intersect s is bounded from above by the stabbing number.

Another way to represent a triangulation using IP is to assign variables to the set of triangles with vertices in P. This idea was discussed in [5] and in [20], where it was shown that the dual bounds generated by the relaxation of the resulting IP dominate those produced by the previous formulation on edge variables. In the description of the Triangle Stabbing Model below, $\Delta(P)$ is the set of empty triangles over P, i.e., triangles that do not contain any point P in their interior; $L^+(ij)$ and $L^-(ij)$ are the two half-planes defined by the line containing ij; E_H is the set of edges on the convex hull of P. For every triangle $ijl \in \Delta(P)$, $x_{ijl} = 1$ if and only if the triangle ijl is in the triangulation. The variable k has the same meaning as in the previous models.

$$
\begin{aligned}
(MSTT) \quad & z = \min k && && && (3.19)\\
\text{subject to} \quad & \sum_{\substack{ijl \in \Delta(P)\,:\\ ijl \subset L^+(ij)}} x_{ijl} &=& \sum_{\substack{ijl \in \Delta(P)\,:\\ ijl \subset L^-(ij)}} x_{ijl}, && \forall ij \in E \setminus E_H, && (3.20)\\
& \sum_{ijl \in \Delta(P)} x_{ijl} &=& \; 1, && \forall\, ij \in E_H, && (3.21)\\
& \sum_{ijl \in \Delta(P) : ijl \bigcap s \neq \emptyset} c^s_{ijl} x_{ijl} &\leq& \; k, && \forall\, s \in L. && (3.22)\\
& k \in \mathbb{Z}, x_{ijl} \in \mathbb{B} && && \forall\, ijl \in \Delta(P). && (3.23)
\end{aligned}
$$

In the model above, constraint (3.20) states that the number of triangles containing an edge ij (which is not in E_H) must be the same in both half-planes defined by the line containing ij. As the edges in E_H are present in every planar triangulation, constraint (3.21) ensures that a triangle containing one such edge is in the triangulation. Constraint (3.22) states that the sum of the coefficients c^s_{ijl} of the triangles ijl intersecting a line s of L can not be larger than the stabbing number. A triangle ijl intersecting a line s has coefficient $c^s_{ijl} = \beta^s_{ij} + \beta^s_{il} + \beta^s_{jl}$, where $\beta^s_{ij} = 1$ if ij intersects s and is on the convex hull, $\beta^s_{ij} = 0.5$ if ij intersects s but is not on the convex hull and $\beta^s_{ij} = 0$ if ij does not intersect s.

Later we will see that both models presented in this section for the MSTR are used in our implementations: $(MSTT)$ in the B&B (exact) algorithm and $(MSTE)$ in the Lagrangian heuristic.

3.3 Lagrangian Relaxation

Using the IP formulations from the previous section, we now derive Lagrangian relaxation (LR) models for the three stabbing problems. We solve the dual of this relaxation via the subgradient method (SGM), which allows us to obtain a lower bound for the optimal value of the problems. Besides, at each iteration of the SGM, we compute the primal Lagrangian problem whose solution is a minimum perfect matching, spanning tree and triangulation, respectively for the MSPM, MSST and MSTR, and, thus, can be used to obtain upper bounds for these problems. For the basic theory of Lagrangian relaxation the reader is referred to [27].

The presentation of our LR is based on a model for a generic stabbing problem $(STAB)$, presented below. This model is composed by the generic constraints (3.25) that define the form of the subgraph of $G(P)$ to be found (in our case either a perfect matching, a spanning tree or a triangulation) and the constraints (3.26) which define that the stabbing number of the subgraph is greater than or equal to the stabbing number of any line.

$$(STAB) \qquad z = \min k \tag{3.24}$$

subject to

$$Ax \leq B, \tag{3.25}$$

$$\sum_{ij \in E : ij \bigcap s \neq \emptyset} x_{ij} \leq k, \qquad \forall\, s \in L. \tag{3.26}$$

$$k \in \mathbb{Z}, x_{ij} \in \mathbb{B} \qquad \forall\, ij \in E. \tag{3.27}$$

To obtain the LR $(STAB(u))$ of problem $(STAB)$ we simply dualize the constraints (3.26), penalizing them in the objective function. This operation results in the following model for the Lagrangian primal problem:

$$(STAB(u)) \qquad z(u) = \min k - \sum_{s \in L} u_s(k - \sum_{ij \in E: ij \bigcap s \neq \emptyset} x_{ij}) \tag{3.28}$$

subject to

$$Ax \leq B, \tag{3.29}$$

$$k \in \mathbb{Z}_+, x_{ij} \in \mathbb{B} \qquad \forall\, ij \in E. \tag{3.30}$$

Notice that the constraints (3.25) that remain in the model are those that define the subgraphs of interest. Also, since the constraints being dualized are in the "$\leq$" form, u_s is non-negative for all $s \in L$. As a consequence, the Lagrangian primal problem is equivalent to the problem of finding one such subgraph having minimum weight (the weight of the subgraph being defined as the sum of its edge weights). In the Lagrangian case, the weight of edge ij is given by

$$c_{ij} = \sum_{s \in L: s \bigcap ij \neq \emptyset} u_s. \tag{3.31}$$

From the Lagrangian theory, we know that whenever the primal problem can be solved in polynomial time, as is the case for the MSPM and MSST, we are able to obtain a dual bound for the original problem in short computation times. However, when the primal problem is $\mathcal{NP}$-hard, one may wonder if the relaxation is useful after all. This is precisely the situation with the MSTR since the MWT was proven to be $\mathcal{NP}$-hard in [19]. However, as we shall see later in Section 3.4, there are highly effective algorithms to compute large subsets of optimal MWT solutions. As a result, one can expect to solve instances of the MWT with several hundreds of points very quickly. Our approach relies on this observation and the results reported in this paper confirmed our expectations.

Now, as $(STAB(u))$ is a relaxation of $(STAB)$, we know that $z(u) \leq z$ and, since we want to find the best possible bound, we must find the value of u that maximizes $z(u)$, i.e., we must solve the Lagrangian dual problem given by

$$(DL) \;\; v_{DL} = \max\{z(u) : u \geq 0\}. \tag{3.32}$$

Problem (DL) can be solved using the SGM as described in [27, 2]. To this end, the multipliers u_s are initialized with null values and are updated at iteration t by the formula:

$$u_s^t = max(0, u_s^{t-1} - \mu G_s^{t-1}). \tag{3.33}$$

with μ given by

$$\mu = \frac{\pi(dist \times ub - lb)}{\sum_{s \in L}(G_s^{t-1})^2}, \tag{3.34}$$

and G_s^{t-1}, the s-th component of the subgradient of $z(u)$ in u^{t-1}, given by

$$G_s^{t-1} = k - \sum_{ij \in E: ij \bigcap s \neq \emptyset} x(u^{t-1})_{ij}. \tag{3.35}$$

In the formulas above, ub and lb are, respectively, an upper and a lower bound for the optimal value, *dist* is a perturbation factor (arbitrarily set to 1.05 in our experiments) and π is the step size (in our experiments initialized at 2 and halved every 30 iterations without improvement in the lower bound). The solution of the Lagrangian primal problem is denoted by $x(u)$ and the superscripts indicate the iteration at which each variable is been considered (e.g., u^t is the Lagrangian multipliers vector at iteration t).

Now, notice that, after dualizing constraints (3.26), the objective function of $(STAB(u))$ can be rewritten as:

$$z(u) = \min k(1 - \sum_{s\in L} u_s) + \sum_{ij\in E} x_{ij} \sum_{s\in L: s\bigcap ij\neq\emptyset} u_s. \tag{3.36}$$

Therefore, if $\sum_{s\in L} u_s > 1$, the first term of that equation would have a negative value and, hence, the larger the value of k, the smaller the value of $z(u)$. As a result, when optimizing the (primal) Lagrangian problem, if the cost of variable k is negative, the lower bound $z(u)$ is unlimited and hence useless. Analogously, if the cost of k is non negative, the obvious solution is to set k to zero. However, by doing so, we may waste the opportunity to produce a better dual bound for z. To overcome these situations, we proceed in the following way. In the solution of $(STAB(u))$, k is set, respectively, to the best upper (ub) or lower (lb) bound available for z depending on whether its cost is negative or not. In fact, in our implementation, when the cost is non negative, k is set to $\lceil lb\rceil/2$ rather than to lb to avoid an early convergence of the SGM. This tends to increase the number of iterations of the method, augmenting the chances of the Lagrangian heuristic to obtain a better feasible solution.

Notice that the dual bound obtained by setting k to $\lceil lb\rceil/2$ or ub, depending on whether $(1 - \sum_{s\in L} u_s)$ is negative or non-negative, is valid. This is so because the model for the primal Lagrangian problem remains correct if the constraint requiring that k belongs to $\mathbb{Z}_+$ is replaced by one that forces k to be in an interval between proper lower and upper bounds. It turns out that $\lceil lb\rceil/2$ and ub are, respectively, valid lower and upper bounds for k, ensuring the correctness of the computation of the dual bounds for $z(u)$.

The termination criteria implemented in our SGM are achieved when one of the following situations occur: the difference between the upper and lower bounds is smaller than 1 (one), the value of π is smaller than 0.005, or yet, a predefined time limit is reached.

Lagrangian Heuristic. Each iteration of the SGM solves a minimum weight problem (a MWPM, a MST, or a MWT, whichever is the case). The solution of this problem is a subgraph of $G(P)$ satisfying the property of interest (i.e., it is a perfect matching, a spanning tree, or a triangulation) and, therefore, is also feasible for the original stabbing problem. Thus, an upper (primal) bound for the optimal value of the stabbing problem can be immediately obtained by computing the stabbing number of this subgraph.

Solving the Lagrangian Primal. For the MSTR, $(STAB(u))$ corresponds to a MWT. As cited before, the MWT is known to be $\mathcal{NP}$-hard but there are algorithms to find subsets of optimal solutions. One of these algorithms is the one to find a Locally Minimum Triangulation Skeleton (LMT-skeleton) [7, 3]. This algorithm is based on the local minimality property of line segments (edges).

Given a planar triangulation T, let ij be an edge of T that is not in the convex hull. Then, ij must be the side of two empty triangles ijk and ijl in T. These two triangles together form a quadrilateral $ijkl$ having ij and kl as its diagonals. We say that ij is locally minimum with respect to $ijkl$ if this quadrilateral is not convex or, else, if the weight of ij is smaller than the weight of kl. Figure 3.2 illustrates this definition. If for

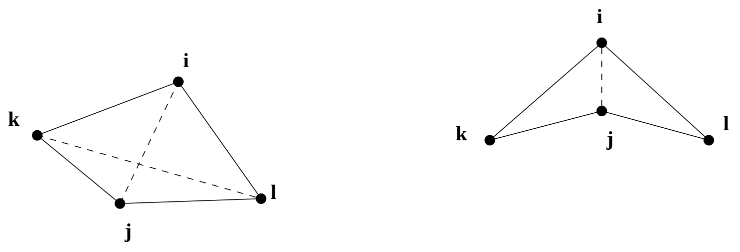

Figure 3.2: In both cases ij is locally minimum with respect to the quadrilateral $ijkl$.

any pair of points $\{k, l\}$ in $P - \{i, j\}$ the edge ij is locally minimum with respect to the quadrilateral $ijkl$, then ij is said to be locally minimum. When all the edges in a planar triangulation are locally minimum, we say that the triangulation itself is locally minimum. Clearly, any minimum weight triangulation is locally minimum. However, not all locally minimum triangulations have minimum weight. The LMT-skeleton is the subset of edges that are present in every locally minimum triangulation and, thus, is also a subset of any minimum weight triangulation.

In [7] the authors proposed a polynomial algorithm to find a LMT-skeleton and in [3] the algorithm was improved. The computational experiments performed with these algorithms showed that, together with a dynamic programming algorithm to find a MWT for convex polygons, it was capable to find the MWT of instances with thousands of points in quite small running times. The source code for this last algorithm written by Mulzer is available online at [18].

Therefore, we can make use of the LMT-skeleton algorithm to solve the Lagrangian Primal Problem through the following steps. First we determine three subsets T_m, T_p and T_f of edges which, respectively, are mandatory (the locally minimum ones), forbidden (those intersected by an edge in T_m) and uncertain (the remaining edges) in a optimal solution, using a LMT-skeleton algorithm [7, 3]. Then, we are left with a constrained MWT problem where all edges of T_m are forced to be in the solution, the ones in T_f are eliminated from the solution and those in T_p are the ones for which we have to make a decision. Typically, after fixing the appropriate variables to one or zero, the size of the MWT models reduces dramatically. This renders the usage of an IP solver to compute the model via a standard B&B algorithm a viable option, even for instances containing hundreds of points. Later we will see that this procedure is capable to solve the Lagrangian primal problems for MSTR in an extremely effective fashion in practice.

To conclude this section, we recall that the Lagrangian primal problems for the MSPM and MSST are, respectively, the MWPM and the MST. To solve the first one we use the

Blossom V algorithm described in [14], whose source code is publicly available. The MST problem is solved by a simple implementation of Prim's algorithm, which can be found in several textbooks on Algorithms.

3.4 Computational Results

We now describe the experiments we carried out to test the performance of the algorithms discussed in the previous sections. As mentioned earlier, we implemented exact B&C algorithms for the MSPM and MSST. An implementation of an exact B&B algorithm for the MSTR was also done. All these exact algorithms were based on the IP models discussed in Section 3.2. We also implemented LR algorithms for all the models using the ideas discussed in Section 3.3. All the experiments described in this section consider the axis parallel version of the problem.

Computational Environment. To perform the experiments, we used a computer with an Intel Core 2 Quad 1.60GHz, 4096 KB cache, 4GB of RAM memory and a Ubuntu 10.04.4 OS. The programming language used was `C/C++` with `gcc` 4.4.3 compiler and every program was compiled with `-O5` optimization flag. We also used the `XPRESS-Optimizer 64-bit v22.01.09` IP solver. The default cuts, heuristics and preprocessing were turned off. Also, the optimizer was set to use a single processor core.

3.4.1 MSPM Experiments

In order to evaluate the performance of our algorithms for the MSPM, we executed experiments with both, the exact B&C algorithm and the LR algorithm and then we tried to compare the results, although this kind of comparison is sometimes tricky, since the algorithms are different in nature.

For the exact B&C algorithm the model was initially loaded using only the degree inequalities (3.2) and stabbing inequalities (3.4). The heuristic proposed in [12] was implemented to separate violated inequalities (3.3). Only when the heuristic fails to find a cutting plane, we resort to the Padberg-Rao exact algorithm described in [21]. We also use a family of conditional cuts [11] that are not guaranteed to be valid for the problem, but can be used as a cutting plane as follows. Suppose an upper bound U_b of the problem is available. One can note that during the search for the optimal solution of the MSPM, we are looking for solutions of value better (lesser) than U_b. In this sense, any inequality can be used as a cutting plane, provided that is satisfied by every feasible solution of value less than U_b. In this vein, we considered the following family of conditional cuts:

$$\sum_{ij \in E[V_s^+]} x_{ij} \geq \left\lceil \frac{|V_s^+| - U_b + 1}{2} \right\rceil, \quad \forall s \in L, \tag{3.37}$$

$$\sum_{ij \in E[V_s^-]} x_{ij} \geq \left\lceil \frac{|V_s^-| - U_b + 1}{2} \right\rceil, \quad \forall s \in L, \tag{3.38}$$

where V_s^+ and V_s^- are sets composed by vertices of V in the interior of one of the two half-planes defined by the line s. Besides, the sets $E[V_s^+]$ and $E[V_s^-]$ are formed by all the edges with both endpoints in V_s^+ and V_s^-, respectively. It can be seen in inequalities (3.37) that a solution of value U_b has at most U_b edges crossing s (each one connected with a vertex in V_s^+). Hence, there are $(|V_s^+| - U_b)$ disconnected vertices in V_s^+ that need $\lceil (|V_s^+| - U_b)/2 \rceil$ edges in $E[V_s^+]$ to complete a matching. Then, it follows that (3.37) can be used as a conditional cut because no solution of value U_b (or greater) is feasible in (3.37). Similar arguments lead to an analogous conclusion for inequalities (3.38).

The cutting plane strategy adds the inequalities with the highest percentage of violation, as long as this value is at least 1% (to control the tailing off effect). No more than 50 inequalities are added per iteration. As for the branching strategy, we select 5 variables whose values in the current linear relaxation are closest to 0.5 and use strong branching to select which variable to branch on.

The primal heuristic used in B&C is based on the linear relaxation of the problem. From a relaxed solution $\overline{x}$, the method attempts to find a matching $M \subseteq E$ maximizing $\sum_{ij \in M} \overline{x}_{ij}$. The method begins with an empty set M and builds a matching, one edge at a time. At each iteration, one edge $(i,j) \in E \backslash M$ is greedily chosen according to the value of $\overline{x}_{ij}$ (prioritizing the highest ones) and inserted into M. The procedure is repeated until a perfect matching is reached. In a second phase, the matching M may be improved by a local search procedure. The neighborhood of the current solution M is defined as the set of all feasible matchings obtained by exchanging pairs of edges (i,j) and (l,m) by edges (i,l) and (j,m). The procedure iteratively replaces the current solution by the one with minimum cost within its neighborhood, halting when no better solution is found in that way. This primal heuristic is applied at every node of the search tree.

For the LR algorithm, a Lagrangian relaxation of the model described for the MSPM in Section 3.2 is obtained (see Section 3.3). The standard subgradient method is then executed to compute the Lagrangian dual problem. As said before, the Lagrangian primal problem is solved by an implementation of the Blossom V algorithm whose code is available for download in the web. It is worth noting that this program only deals with instances having integer weights. However, in the usual Lagrangian scheme, the edge weights are often not integer. To circumvent this difficulty, we multiplied all the edge weights in the Lagrangian primal problem by 10^6 before calling the routine. This is not expected to create major numerical problems and, in the end, is not more harmful to computation than the tolerance of 10^{-6} that we set for the IP solver.

As we will see in the results part of this subsection, the Lagrangian algorithm produces good bounds with small computation times. This suggests that it can be used together with the exact B&C algorithm to obtain better results. We used the primal bound from the LR algorithm to warm start the B&C algorithm. Our tests showed that, for the three problems studied, the use of primal bounds from LR algorithm to warm start the exact algorithms yielded better overall results. For this reason, we decided to use these results and compare them with the pure Lagrangian results.

Instances. For the MSPM, we experimented with the same instances tested in [9] (except for five TSPLIB instances [23] that are obviously infeasible since they have an odd number of vertices). These include 5 instances from TSPLIB, 16 from the clustered C1 and C2 classes of Solomon's Vehicle Routing Problem benchmark [25], 25 regular grid instances (5×5 to 20×20 grids with 20% of its points randomly removed) and 11 instances with up to 100 random points in the plane.

For the three problems under investigation, a time limit of $1,800$ seconds was set for the execution of any algorithm. Notice, however, that in the tables 3.4 to 3.7, occasionally the time is bigger than this limit. This happens for two reasons, first, the times presented for warm started exact algorithm (WSEA) are the sum of the time spent by the Lagrangian and the B&C or B&B algorithms, therefore could go up to $3,600$. Second, the time limit is verified at certain points in the program codes and, it could be that the time elapsed between two check-points is not negligible. This situation arises, for example, when the model of a big instance is being uploaded by the IP solver. In our experiments an additional timeout script running on the operating system level was used that forces the process to halt after $2,000$ seconds. In case the process ends naturally, a bound is always produced. On the other hand, if the process is killed by the timeout script, no output is produced. The latter situation is signalized in the tables by the symbol ‡. Also, duality gaps were computed through the formula $100 \times (ub - lb)/ub$, where ub and lb denote, respectively, the upper and lower bounds yielded by the algorithm.

Results. As we previously stated, all the WSEA outperformed the cold started exact algorithms and, for this reason, we compare the WSEA against the LR algorithms. Obviously, it does not make sense to just compare the times of these two kinds of algorithms because, first, as said before, the time of the WSEA is the sum of the LR algorithm and the B&C or B&B algorithm, thus, is always greater than the LR alone. Second, the algorithms are different in nature. So, the purpose of our comparison is to determine whether the WSEA can improve the bounds obtained by the LR algorithm, how much and how fast.

Our analysis of the results will be done in three parts: the first for the TSP and clustered instances, the second for the random instances and the third for the grid instances.

The results for the first set of instances are summarized in Table 3.1. We observe that the B&C algorithm proved optimality in all the cases within the fixed time limit. The Lagrangian SGM always converged, proving optimality in all but one case (berlin52), where there is an absolute gap of one unit (25.0%). For this set of instances the WSEA provided an average improvement of 1.19% in the relative gap with an average increasing of 4.48 seconds in time when compared to the LR algorithm.

Results for the random instances can be seen in Table 3.2. Once again the LR algorithm always converged. However, whilst the exact algorithm proves optimality for all instances, the Lagrangian failed to prove optimality in four cases, where gaps of one unit remain. The average improvement in the relative gap obtained from the WSEA was 8.64% and the average time increasing was 1.74 seconds.

The results for the grid instances are displayed in Table 3.3. This benchmark was the one for which the LR heuristic had the worst performance. The Lagrangian heuristic was

Table 3.1: Results for MSPM TSP and clustered instances.

Instance	LB		UB		Time		GAP%	
	LR	B&C	LR	B&C	LR	B&C	LR	B&C
a280	11	11	11	11	0.83	13.34	0.00	0.00
berlin52	3	4	4	4	0.86	1.23	25.00	0.00
lin318	9	9	9	9	29.17	52.43	0.00	0.00
pcb442	17	17	17	17	27.71	78.79	0.00	0.00
ulysses22	2	2	2	2	0.00	0.03	0.00	0.00
c101	7	7	7	7	0.05	0.41	0.00	0.00
c102	7	7	7	7	0.05	0.43	0.00	0.00
c103	7	7	7	7	0.05	0.41	0.00	0.00
c104	7	7	7	7	0.05	0.43	0.00	0.00
c105	7	7	7	7	0.06	0.43	0.00	0.00
c106	7	7	7	7	0.05	0.42	0.00	0.00
c107	7	7	7	7	0.05	0.43	0.00	0.00
c108	7	7	7	7	0.06	0.42	0.00	0.00
c201	6	6	6	6	0.08	0.55	0.00	0.00
c202	6	6	6	6	0.09	0.54	0.00	0.00
c203	6	6	6	6	0.09	0.55	0.00	0.00
c204	6	6	6	6	0.08	0.55	0.00	0.00
c205	6	6	6	6	0.09	0.55	0.00	0.00
c206	6	6	6	6	0.08	0.53	0.00	0.00
c207	6	6	6	6	0.08	0.53	0.00	0.00
c208	4	4	4	4	1.15	1.89	0.00	0.00

unable to prove optimality in 11 out of 25 cases, leaving gaps of one unit in 10 cases and two units in 1 case. The exact algorithm, on the other hand, was able to prove optimality for all of the grid instances. The improvement in the relative gap achieved using the exact algorithm was 4.85% and the average increasing of time was 8.95 seconds.

Therefore it is possible to say that the LR algorithm have a very nice performance for these sets of instances. Also, the price in time necessary to prove optimality using the warm started B&C algorithm seems rather small. We recall that B&C is an exact algorithm while LR is an heuristic. So, when comparing their performances, one has to bear in mind that they are rather different in nature.

In order to compare our results against those presented in [9] we implemented the model presented in that paper and executed a B&C algorithm in the same computational environment used to test ours. This experiment showed that the algorithm using the model from [9] was unable to prove optimality in six, cases among all the instances tested for the MSPM, within a time limit of $1,800$ seconds. Considering all the test cases for the MSPM, the average time of our WSEA was 5.91 seconds while the implementation of the algorithm from [9] had an average time of 213.10 seconds.

3.4.2 MSST Experiments

To analyze the performance of our algorithms for the MSST, again we implemented an exact B&C algorithm. Once more, we found that warm starting the B&C algorithm with

Table 3.2: Results for MSPM random instances.

Instance	LB		UB		Time		GAP%	
	LR	B&C	LR	B&C	LR	B&C	LR	B&C
rand10a	2	2	2	2	0.00	0.00	0.00	0.00
rand10b	2	2	2	2	0.00	0.00	0.00	0.00
rand10c	2	2	2	2	0.00	0.00	0.00	0.00
rand10d	2	2	2	2	0.00	0.01	0.00	0.00
rand10e	2	2	2	2	0.00	0.01	0.00	0.00
rand50a	3	3	3	3	0.15	0.67	0.00	0.00
rand50b	3	3	3	3	0.64	1.18	0.00	0.00
rand50c	3	4	4	4	0.62	1.20	25.00	0.00
rand50d	3	4	4	4	0.64	1.15	25.00	0.00
rand50e	3	4	4	4	0.77	1.32	25.00	0.00
rand100a	4	5	5	5	6.40	22.85	20.00	0.00

Table 3.3: Results for MSPM grid instances.

Instance	LB		UB		Time		GAP%	
	LR	B&C	LR	B&C	LR	B&C	LR	B&C
grid5a	4	4	4	4	0.00	0.01	0.00	0.00
grid5b	4	4	4	4	0.00	0.01	0.00	0.00
grid5c	4	4	4	4	0.01	0.02	0.00	0.00
grid5d	4	4	4	4	0.00	0.01	0.00	0.00
grid5e	4	4	4	4	0.00	0.02	0.00	0.00
grid8a	6	6	6	6	0.10	0.15	0.00	0.00
grid8b	6	6	6	6	0.06	0.12	0.00	0.00
grid8c	5	5	6	5	0.19	0.28	16.67	0.00
grid8d	6	6	6	6	0.00	0.06	0.00	0.00
grid8e	6	6	7	6	0.30	0.35	14.29	0.00
grid10a	7	7	7	7	0.22	0.43	0.00	0.00
grid10b	6	6	7	6	0.64	0.83	14.29	0.00
grid10c	7	7	8	7	0.69	2.04	12.50	0.00
grid10d	7	7	7	7	0.19	0.41	0.00	0.00
grid10e	7	7	8	7	0.59	1.73	12.50	0.00
grid15a	10	10	10	10	1.59	3.61	0.00	0.00
grid15b	10	10	11	10	5.45	50.42	9.09	0.00
grid15c	10	10	10	10	1.32	3.28	0.00	0.00
grid15d	10	10	10	10	2.94	4.96	0.00	0.00
grid15e	10	10	10	10	1.77	4.04	0.00	0.00
grid20a	13	13	15	13	25.65	111.31	13.33	0.00
grid20b	13	13	14	13	26.28	40.70	7.14	0.00
grid20c	13	13	14	13	28.16	47.46	7.14	0.00
grid20d	13	13	14	13	24.06	39.43	7.14	0.00
grid20e	13	13	14	13	31.02	63.31	7.14	0.00

the primal bound obtained from the Lagrangian SGM gives us better results than simply executing the B&C. Therefore, all comparisons in this subsection are made between the WSEA and the LR algorithm.

For the exact algorithm we used the model described in Section 3.2. Initially the model was loaded without constraints (3.7). In the branch-and-cut method, at each node of the search tree, the linear relaxation of MSST is solved. If in the optimal solution all variables are integral, the node is pruned by optimality. Otherwise, the solution is fractional and violated valid inequalities are sought by solving a separation problem. The polynomial-time algorithm presented in [13], based on the minimum edge cut problem in graphs, is used to separate the Steiner cut inequalities (3.7).

As for the LR algorithm, the implementation was done as described in Section 3.3, with the primal Lagrangian problem been solved by a simple implementation of Prim's algorithm for the MST.

Instances. As a test suite we used 25 instances from TSPLIB [23] and the 25 regular grid instances used in [9] for the Minimum Stabbing Perfect Matching Problem. The choice of these instances is based on the fact that the TSPLIB is a well known test library for geometric problems and, besides, some TSPLIB and all grid instances were also used in [9] for the MSPM. The choice of the instance sizes was made seeking tests that were hard enough to provide meaningful computation times, allowing a more precise comparison of the algorithms.

Results. We divide our analysis into two parts, one for the TSP instances and another for the grid instances.

The results for the TSP part are displayed in Table 3.4. One can see that the LR algorithm converged in all the cases within the time limit, proving optimality in 11 of the 25 of them. The WSEA was unable to yield any output within the time limit for just one of the test instances. Among the 24 remaining instances, the B&C algorithm proved optimality in 16 cases. It is interesting to notice that the SGM was able to prove optimality in one case where the B&C was unable to do so (despite the warm start), while the opposite occurred 6 times. For this set of instances, when compared with the LR algorithm, the improvement in the relative gap provided by the WSEA was 2.38% and the necessary extra time to achieve this improvement was 857.79 seconds.

Analyzing the results for the second group of instances given in Table 3.5, we observe that the performance of the LR algorithm is not as good as for the TSP instances, since optimality was achieved in fewer cases. The B&C failed to declare optimality in only 3 out of the 25 grid instances while the SGM failed in 14 other cases. In the grid instances, the execution of the WSEA improved the relative gap by 4.59% at the cost of 391.88 more seconds, both in average.

The analysis of the improvement relative to the Lagrangian SGM algorithm and of the additional time spent to obtain such gain when using WSEA points to a remarkable performance of the LR algorithm.

Table 3.4: Results for MSST TSP instances.

Instance	**LB**		**UB**		**Time**		**GAP%**	
	LR	B&C	LR	B&C	LR	B&C	LR	B&C
berlin52	6	6	6	6	0.15	3.77	0.00	0.00
ch130	7	7	8	8	12.21	1813.38	12.50	12.50
ch150	8	8	9	8	19.35	161.09	11.11	0.00
eil76	8	8	8	8	1.08	1.48	0.00	0.00
gil262	11	11	12	12	83.45	1907.68	8.33	8.33
gr202	9	9	10	9	58.70	1456.22	10.00	0.00
kroA100	7	7	8	7	4.85	1177.36	12.50	0.00
kroA150	8	8	9	9	14.69	1819.08	11.11	11.11
kroA200	9	9	9	9	29.95	1154.45	0.00	0.00
kroB100	7	7	7	7	3.98	5.20	0.00	0.00
kroB150	8	8	9	9	19.81	1823.96	11.11	11.11
kroB200	9	9	10	10	45.91	1858.87	10.00	10.00
kroC100	7	7	7	7	4.21	46.09	0.00	0.00
kroD100	7	7	7	7	3.27	4.40	0.00	0.00
kroE100	7	7	7	7	2.67	3.91	0.00	0.00
lin318	16	16	18	18	36.84	1860.34	11.11	11.11
pcb442	34	33	34	34	56.02	1915.33	0.00	2.94
pr124	24	24	24	24	22.47	26.06	0.00	0.00
pr136	17	17	18	17	2.75	87.52	5.56	0.00
pr144	21	21	21	21	0.50	1292.64	0.00	0.00
pr152	11	11	12	11	6.88	536.45	8.33	0.00
pr226	72	72	72	72	4.43	16.54	0.00	0.00
pr264	23	23	29	29	13.93	1821.02	20.69	20.69
rd100	7	7	8	7	4.98	247.18	12.50	0.00
rd400	11	‡	13	13	661.39	‡	15.38	‡

Table 3.5: Results for MSST grid instances.

Instance	**LB**		**UB**		**Time**		**GAP%**	
	LR	B&C	LR	B&C	LR	B&C	LR	B&C
grid5a	7	7	7	7	0.01	0.10	0.00	0.00
grid5b	7	7	7	7	0.01	0.10	0.00	0.00
grid5c	7	7	7	7	0.01	0.09	0.00	0.00
grid5d	7	7	7	7	0.01	0.09	0.00	0.00
grid5e	7	7	7	7	0.01	0.09	0.00	0.00
grid8a	10	10	10	10	0.04	1.57	0.00	0.00
grid8b	10	10	10	10	0.03	0.19	0.00	0.00
grid8c	10	10	10	10	0.07	0.22	0.00	0.00
grid8d	11	11	13	11	0.15	1.10	15.38	0.00
grid8e	11	11	11	11	0.08	0.24	0.00	0.00
grid10a	13	13	14	13	0.44	4.31	7.14	0.00
grid10b	12	12	12	12	0.17	0.44	0.00	0.00
grid10c	13	13	14	13	0.45	3.78	7.14	0.00
grid10d	13	13	13	13	0.18	0.48	0.00	0.00
grid10e	13	13	14	13	0.47	9.17	7.14	0.00
grid15a	18	18	20	18	2.97	117.97	10.00	0.00
grid15b	20	20	23	20	3.17	368.78	13.04	0.00
grid15c	18	18	19	18	2.87	84.31	5.26	0.00
grid15d	19	19	21	19	2.35	125.61	9.52	0.00
grid15e	18	18	20	18	2.44	828.30	10.00	0.00
grid20a	24	24	27	27	15.48	1828.94	11.11	11.11
grid20b	24	24	27	27	11.16	1824.14	11.11	11.11
grid20c	25	25	28	25	11.06	1415.05	10.71	0.00
grid20d	25	25	29	29	9.98	1827.44	13.79	13.79
grid20e	25	25	31	25	11.95	1430.14	19.35	0.00

3.4.3 MSTR Experiments

The first stage of our testing comprised a comparison of the two alternative B&B algorithms that arise from the Edge and Triangle stabbing models discussed in Section 3.2. For the MWT, it was observed in [20] that the B&B algorithm performs better when it uses an IP model with variables defined on triangles than with variables associated to edges. Hence, a similar behavior was expected from the corresponding models when applied to the solution of the MSTR. Indeed, this was what happened and, thus, all the B&B results reported below were obtained using the Triangle Stabbing Model. More precisely, the results refer to a warm started exact algorithm (WSEA) using the mentioned formulation.

Regarding the LR algorithm, we implemented the subgradient method using both the Edge Stabbing Model and the Triangle Stabbing Model. Recall that, irrespective to which of the two models we consider, when the stabbing constraints are relaxed we are left with an IP formulation for the MWT problem (we use the term "relaxed" to refer to these models). However, in the subgradient procedure several such problems have to be solved at each iteration. This is done in two steps. The first step consists in the calculation of the LMT-skeleton while the second step actually solves the MWT problem in case the first step fails to do so.

Observe that the edge weights are the only differences between the instances of the MWT problems solved in two iterations of the subgradient method. The computation of the LMT-skeleton only depends on the edge costs. Therefore, for the first step, it is convenient from a computational point of view to have the problem defined in terms of the Edge Stabbing Model, as it allows for a quick recalculation of these costs. On the other hand, in the second step, when it comes to actually solve the MWT instance, we rely on the results reported in [20] where it was observed that the B&B algorithm for the MWT performs much better with the relaxed Triangle Stabbing Model than with the relaxed Edge Stabbing Model. Now, given two iterations of the subgradient method, the triangle costs are the only differences between the associated MWT instances. These costs can be easily computed after the LMT-skeleton has been found in the first step. Some additional details are given below.

As said in Section 3.3, to solve the Lagrangian primal problem, we used the LMT-skeleton code written by Beirouti and Snoeyink and downloadable at [18]. A few modifications were introduced in this program to make possible the usage of arbitrary edge weights instead of Euclidean ones. This included, for instance, the removal of the *diamond test*, a simple and effective way to determine whether an edge could be part of a triangulation of minimum (Euclidean) length. Such changes do not have significantly damaged the algorithm's performance, relative to Euclidean weights, confirming it as a viable option for general MWTs.

After running the LMT-skeleton, quite often we still do not have a triangulation. Hence, a B&B algorithm is used to solve the constrained MWT that remains, i.e., a MWT with sets of mandatory and forbidden edges. Since we use the (relaxed) Triangle Stabbing Model as the input for the B&B algorithm, these sets of edges have to be processed to identify the corresponding sets of triangles. Thus, if an empty triangle contains a forbidden edge, the associated variable is set to zero while, if all the edges forming its sides are mandatory, this variable is set to one.

Instances. The test suite used to analyze the performance of the MSTR algorithms was the same as in the MSST case. The reasons that support this choice are the same as before. Also, the time limit parameters inside the programs and in the timeout script remain unchanged, i.e., $1,800$ and $2,000$, respectively. Once again, the symbol ‡ in the tables with results signalizes that the process was killed by the timeout script and, thus, did not produced any output.

Results. As in the MSST case, we divide our analysis into two parts, one for the TSP instances and the other for the grid instances. Concerning the TSP instances, the B&B algorithm had its process killed in 12 out of the 25 instances and, when this was not the case, it proved optimality in all but three instances, where there is a 3.33% gap (the gap exists because of the $1,800$ seconds time limit). On the other hand, the Lagrangian SGM converged in all cases within the imposed time limit, with an average gap of 2.57%. The performance of the heuristic is remarkable. Optimality was proven for 7 instances, one of which could not be reached by the exact algorithm within the time limit (the inverse situation occurred four times). In 13 instances the difference between the upper and lower bounds was of just one unit. Using the WSEA we were able to improve the bounds provided by the LR algorithm in average by 0.97% while the time spent for this was 592.14 seconds in average. These results are summarized in Table 3.6.

The results for the grid instances can be seen in Table 3.7. For those instances, the Lagrangian subgradient method was able to solve to optimality every instance. The B&B algorithm was unable to solve 4 out of 25 grid instances. In fact, only one of the 20×20 grid instances was solved within the time limit (the processes were killed by the timeout script) and every other grid instance was solved to optimality. Regarding this set of instances, it is simply not worth executing a WSEA, since the LR is able to solve them relatively easy.

3.5 Conclusions and Future Directions

To our knowledge, this paper proposes the first exact approach to tackle the MSTR. Concerning the MSPM, our B&C algorithm is able to solve exactly all instance and runs in smaller computational times when compared to the results reported in [9]. As for the MSST, we developed an exact B&C algorithm based on a stronger formulation than the one introduced in [9, 10]. This algorithm obtained optimal solutions for several instances as well as high quality primal and dual bounds for many others in short computation times.

Moreover, we also devised Lagrangian heuristics for the three problems and conducted several computational experiments with them. These tests showed that they rapidly yield solutions with small costs, often proven optimal ones. It should be noticed that, we are

Table 3.6: Results for MSTR TSP instances.

Instance	**LB**		**UB**		**Time**		**GAP%**	
	LR	B&B	LR	B&B	LR	B&B	LR	B&B
berlin52	24	24	24	24	7.70	9.11	0.00	0.00
ch130	32	‡	33	33	165.09	‡	3.03	‡
ch150	34	‡	35	35	268.69	‡	2.86	‡
eil76	32	32	33	32	112.64	178.18	3.03	0.00
gil262	49	‡	50	50	1779.50	‡	2.00	‡
gr202	42	‡	42	42	615.63	‡	0.00	‡
kroA100	29	29	30	30	107.21	1967.38	3.33	3.33
kroA150	35	‡	36	36	330.66	‡	2.78	‡
kroA200	40	‡	41	41	736.80	‡	2.44	‡
kroB100	29	29	30	30	119.87	1976.12	3.33	3.33
kroB150	34	‡	35	35	408.44	‡	2.86	‡
kroB200	39	‡	40	40	705.75	‡	2.50	‡
kroC100	29	29	29	29	96.18	161.44	0.00	0.00
kroD100	29	29	29	29	30.45	86.90	0.00	0.00
kroE100	29	29	30	30	98.93	1962.76	3.33	3.33
lin318	69	‡	71	71	1803.40	‡	2.82	‡
pcb442	157	‡	180	180	1827.53	‡	12.78	‡
pr124	48	49	49	49	405.61	463.30	2.04	0.00
pr136	66	66	67	66	589.67	658.60	1.49	0.00
pr144	74	74	74	74	675.39	848.44	0.00	0.00
pr152	45	45	45	45	420.93	1015.55	0.00	0.00
pr226	141	150	150	150	1884.99	2855.06	6.00	0.00
pr264	90	‡	92	92	1811.44	‡	2.17	‡
rd100	29	29	29	29	17.45	82.05	0.00	0.00
rd400	52	‡	55	55	1803.73	‡	5.45	‡

Table 3.7: Results for MSTR grid instances.

Instance	**LB**		**UB**		**Time**		**GAP%**	
	LR	B&B	LR	B&B	LR	B&B	LR	B&B
grid5a	22	22	22	22	0.17	0.17	0.00	0.00
grid5b	21	21	21	21	0.27	0.36	0.00	0.00
grid5c	21	21	21	21	0.17	0.17	0.00	0.00
grid5d	21	21	21	21	23.14	23.21	0.00	0.00
grid5e	20	20	20	20	0.18	0.18	0.00	0.00
grid8a	34	34	34	34	2.20	2.36	0.00	0.00
grid8b	34	34	34	34	3.48	3.71	0.00	0.00
grid8c	34	34	34	34	1.61	1.81	0.00	0.00
grid8d	35	35	35	35	1.07	1.26	0.00	0.00
grid8e	35	35	35	35	1.11	1.35	0.00	0.00
grid10a	44	44	44	44	8.01	9.03	0.00	0.00
grid10b	42	42	42	42	3.31	3.93	0.00	0.00
grid10c	47	47	47	47	9.52	10.48	0.00	0.00
grid10d	46	46	46	46	2.61	3.43	0.00	0.00
grid10e	46	46	46	46	7.05	8.10	0.00	0.00
grid15a	66	66	66	66	75.13	127.64	0.00	0.00
grid15b	68	68	68	68	13.65	70.36	0.00	0.00
grid15c	64	64	64	64	20.70	67.39	0.00	0.00
grid15d	66	66	66	66	39.24	86.21	0.00	0.00
grid15e	67	67	67	67	79.53	141.38	0.00	0.00
grid20a	89	89	89	89	500.78	2491.35	0.00	0.00
grid20b	86	‡	86	86	73.09	‡	0.00	‡
grid20c	90	‡	90	90	1781.70	‡	0.00	‡
grid20d	87	‡	87	87	204.77	‡	0.00	‡
grid20e	90	‡	90	90	1213.83	‡	0.00	‡

not aware of another work in the literature which reports on computational results for the MSTR.

Future directions in this research are currently being considered. This includes improving the performance of our heuristics by adding new features to it, such as, a procedure for variable fixing in the traditional Lagrangian fashion and a fast local search to reduce primal bounds.

This research was partially supported by *Conselho Nacional de Desenvolvimento Científico e Tecnológico* – grants #301732/2007-8, #473867/2010-9, #147619/2010-6; *Fundação de Amparo à Pesquisa do Estado de São Paulo* – grant #07/52015-0, and a grant from FAEPEX/UNICAMP. The authors are also grateful to Prof. M. Lübbecke for making available some of the instances used in the experiments.

Bibliography

[1] P. Agarwal, B. Aronov, and S. Suri. Stabbing triangulations by lines in 3d. In *Proceedings of the Eleventh Annual Symposium on Computational Geometry*, SCG '95, pages 267–276, New York, NY, USA, 1995. ACM.

[2] J. Beasley. Lagrangean relaxation. In *Modern Heuristic Techniques for Combinatorial Problems*, pages 243–303. McGraw-Hill, 1993.

[3] R. Beirouti and J. Snoeyink. Implementations of the LMT heuristic for minimum weight triangulation. In *Proceedings of the Fourteenth Annual Symposium on Computational Geometry*, SCG '98, pages 96–105, New York, NY, USA, 1998. ACM.

[4] M. Berg and M. Kreveld. Rectilinear decompositions with low stabbing number. *Information Processing Letters*, 52(4):215 – 221, 1994.

[5] J. A. de Loera, S. Hosten, F. Santos, and B. Sturmfels. The polytope of all triangulations of a point configuration. *Documenta Mathematica*, 1:103–119, 1996.

[6] E. Demaine, J. Mitchell, and J. O'Rourke. The open problems project. Available online (accessed January 2010). `http://maven.smith.edu/∼orourke/TOPP/`.

[7] M.T. Dickerson and M.H. Montague. A (usually) connected subgraph of the minimum weight triangulation. In *Proc. of the 12th Annual ACM Symp. on Comp. Geom.*, pages 204–213, 1996.

[8] J. Edmonds. Maximum matching and a polyhedron with 0,1-vertices. *Journal of Research of the National Bureau of Standards*, 69B:125–130, 1965.

[9] S. Fekete, M. Lübbecke, and H. Meijer. Minimizing the stabbing number of matchings, trees, and triangulations. In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '04, pages 437–446, Philadelphia, PA, USA, 2004. Society for Industrial and Applied Mathematics.

[10] S. Fekete, M. Lübbecke, and H. Meijer. Minimizing the stabbing number of matchings, trees, and triangulations. *Discrete & Computational Geometry*, 40(4):595–621, 2008.

[11] M. Fischetti, J. Gonzalez, and P. Toth. Solving the orienteering problem through branch-and-cut. *INFORMS Journal on Computing*, 10:133–148, 1998.

[12] M. Grötschel and O. Holland. Solving matching problems with linear programming. *Mathematical Programming*, 33:243–259, 1985.

[13] T. Koch and A. Martin. Solving Steiner tree problems in graphs to optimality. *Networks*, 33:207 – 232, 1998.

[14] V. Kolmogorov. Blossom v: a new implementation of a minimum cost perfect matching algorithm. *Mathematical Programming Computation*, 2009.

[15] T.L. Magnanti and L.A. Wolsey. Optimal trees. *Handbooks in operations research and management science*, 7:503–615, 1995.

[16] J. Mitchell and J. O'Rourke. Computational geometry. *SIGACT News*, 32(3):63–72, 2001.

[17] J. Mitchell and E. Packer. Computing geometric structures of low stabbing number in the plane. In *Proc. 17th Annual Fall Workshop on Computational Geometry and Visualization.* IBM Watson, 2007.

[18] W. Mulzer. Index of /∼mulzer/pubs/mwt_software/old/ipelets. Available online (accessed March 2011). `http://page.mi.fu-berlin.de/mulzer/pubs/mwt_software/old/ipelets/LMTSkeleton.tar.gz`.

[19] W. Mulzer and G. Rote. Minimum-weight triangulation is NP-hard. *CoRR*, abs/cs/0601002, 2006.

[20] A. P. Nunes. Uma abordagem de programação inteira para o problema da triangulação de custo mínimo. Master's thesis, Institute of Computing, University of Campinas, Campinas, 1997. In Portuguese.

[21] M. W. Padberg and M. R. Rao. Odd minimum cut-sets and b-matchings. *Mathematics of Operations Research*, 7:67–80, 1982.

[22] B. Piva and C. de Souza. The minimum stabbing triangulation problem: Ip models and computational evaluation. In A. Ridha Mahjoub, Vangelis Markakis, Ioannis Milis, and Vangelis Th. Paschos, editors, *Combinatorial Optimization*, volume 7422 of *Lecture Notes in Computer Science*, pages 36–47. Springer Berlin Heidelberg, 2012.

[23] G. Reinelt. TSPLIB. Available online (accessed March 2011). `http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/`.

[24] J. R. Shewchuk. Stabbing Delaunay tetrahedralizations. *Discrete & Computational Geometry*, 32:343, 2002.

[25] M. Solomon. VRPTW benchmark problems. Available online (accessed August 2011). `http://w.cba.neu.edu/∼msolomon/problems.htm`.

[26] C. D. Tóth. Orthogonal subdivisions with low stabbing numbers. volume 3608 of *Lecture Notes in Computer Science*, pages 256–268. Springer Berlin / Heidelberg, 2005.

[27] L. A. Wolsey. *Integer Programming.* John Wiley and Sons, 1998.

Chapter 4

On Triangulations with Minimum Stabbing or Minimum Crossing Number

---

The text presented in this chapter is co-authored with Sándor Fekete and Cid C. de Souza and corresponds to a preprint of an article submitted for consideration in International Journal of Computational Geometry & Applications © 2015 copyright World Scientific Publishing Company http://www.worldscientific.com/loi/ijcga. In this paper, we show the complexity of finding a triangulation with minimum stabbing number (MSTR) both in axis-parallel and general cases, and finding a triangulation with minimum crossing number (MCTR) in the general case. Moreover, computational experiments with an iterative rounding algorithm for the MSTR, using axis-parallel instances, is presented and the results support the conjecture that it provides an approximation for the stabbing problem.

In this paper we consider the computational complexity of the Minimum Stabbing Triangulation Problem (MSTR), both in the axis-parallel and general cases, and the computational complexity of the Minimum Crossing Triangulation Problem (MCTR) in the general case. The complexity class of these problems were left as open questions in [9, 10]. Here we prove that the three problems are $\mathcal{NP}$-hard, thus answering those open questions. In addition, we perform a computational study based on two different polynomial-time heuristic approaches, one based on Lagrangian relaxation, the other on iterated rounding. With respect to the practical objective of finding good solutions in reasonable time, we demonstrate that both of these algorithms yield feasible solutions that are within a few percentage points of the optimal solutions. With respect to the theoretical objective of establishing a polynomial-time algorithm that gets within a constant factor of the optimum even in the worst case, we provide evidence supporting the conjecture that iterated rounding may be such an approximation algorithm.

4.1 Introduction

Triangulating a set of points is one of the basic problems of Computational Geometry: given a set P of n points in the plane, connect them by a maximal set of non-crossing line segments. This implies that all bounded faces of the resulting planar arrangement are triangles, while the exterior face is the complement of the convex hull of P.

Triangulations are computed and used in a large variety of contexts, e.g., in mesh generation, but also as a stepping stone for other tasks. While it is not hard to compute some triangulation, most of these tasks require triangulations with special properties that should be optimized. Examples include maximizing the minimum angle, minimizing the total edge weight or the longest edge length.

When dealing with structural or algorithmic properties, a relevant objective function is the *stabbing number*: for a given set of line segments, this is the maximum number of segments that are encountered (in their interior or at an endpoint) by any line. If we consider only axis-parallel lines, we get the *axis-parallel stabbing number*. A closely related measure defined by Matoušek [14] is the *crossing number*, which is the number of connected components of the intersection of a line with the union of line segments[1]. When considering structures like triangulations, the crossing number is precisely one more than the maximum number of triangles intersected by any one line.

Stabbing problems have been considered for several years. The complexity of many algorithms in computational geometry is directly dependent on the complexity of ray shooting; as described by Agarwal [1], the latter can be improved by making use of spanning trees of low stabbing number. A majority of previous work on stabbing and crossing problems has focused on extremal properties. Settling the complexity of Minimum Stabbing Number for spanning trees was one of the original 30 outstanding open problems of computational geometry on the list by Mitchell and O'Rourke [15]. (An up-to-date list is maintained online by Demaine, Mitchell, and O'Rourke [8].) In particular, problems in the context of triangulation are highly relevant. One of the theoretically best performing data structures for ray tracing in two dimensions is based on a triangulation of the polygonal scene; see Hershberger and Suri [12]: in their "pedestrian" approach to ray shooting, the complexity of a query is simply the number of triangles visited, i.e., corresponds precisely to the stabbing number. Held, Klosowski, and Mitchell [11] investigate collision detection in a virtual reality environment, again, based on "pedestrian" ray shooting. In other related work, Aronov et al. [5] have performed an experimental study of the complexity of ray tracing algorithms and run-time predictors, which include average number of intersection points for a transversal line, and depth complexity. Agarwal, Aronov, and Suri [2] investigate extremal properties of the stabbing number of triangulations in three dimensions, where the stabbed objects are simplices; see also Aronov and Fortune [6] for this problem. Shewchuk [19] shows that in d dimensions, a line can stab the interiors of $\Theta(n^{\lceil d/2 \rceil})$ Delaunay d-simplices. This implies, in particular, that a Delaunay triangulation in the plane may have linear stabbing number. Another closely related variant is studied by de Berg and van Kreveld [7]: the stabbing number of a decomposition of a rectilin-

[1]This should not be confused with the crossing number in graph drawing, which is the total number of crossing line segments.

ear polygon P into rectangles is the maximum number of rectangles intersected by any axis-parallel segment that lies completely inside of P; they prove that any simple rectilinear polygon with n vertices admits a decomposition with stabbing number $O(\log n)$, and they give an example of a simple rectilinear polygon for which any decomposition has stabbing number $\Omega(\log n)$. They generalize their results to rectilinear polygons with rectilinear holes. Furthermore, Tóth [20] showed that for any subdivision of d-dimensional Euclidean space, $d \geq 2$, by n axis-aligned boxes, there is an axis-parallel line that stabs at least $\Omega(\log^{1/(d-1)} n)$ boxes, which is the best possible lower bound. A concept similar to the crossing number was introducd by Aichholzer et al. [3] under a different name. They call a polygon *k-convex*, if every line intersects it in at most k connected components. In the followup paper [4], Aurenhammer et al. studied the concept of k-convex point sets: does a given set P of n planar points allow a polygon that is k-convex? Clearly, this is closely related to deciding whether P allows a simple polygon of crossing number at most $2k$.

All this makes it clear that computing a triangulation of low stabbing or low crossing number (for general or axis-parallel stabbing lines) are highly important problems. Three of the four variants have been left open for many years. In [9, 18] it was proved that the problem of finding a triangulation with minimum crossing number (MCTR) is $\mathcal{NP}$-hard in the axis-parallel case. However, the more interesting case of general orientation has remained an open problem. Furthermore, for either version of the stabbing problem (for axis-parallel lines or those of arbitrary orientation), no complexity result have been established so far. (As it turns out, [10] contains an erroneous statement in the introduction that results for the stabbing number are established in the paper. This is not the case, the only hardness result contained is for the axis-parallel crossing number.)

In this paper we to show that the Minimum Stabbing Triangulation Problem (MSTR) is $\mathcal{NP}$-hard both in the axis-parallel and general cases. We then present a proof that the MCTR, in the general case, is also $\mathcal{NP}$-hard. This closes all remaining gaps in the complexity analysis of optimal stabbing and crossing numbers for triangulations. In addition, we perform a computational study that supports the conjecture that a heuristic based on iterated rounding applied to an LP relaxation may provide a constant-factor approximation algorithm.

The paper is organized as follows: in Section 4.2 some basic concepts are defined and the problems are stated, Section 4.3 presents an $\mathcal{NP}$-hardness proof for the MSTR in the axis-parallel case, Section 4.4 shows a proof that MSTR is $\mathcal{NP}$-hard for general orientation, while Section 4.5 contains a proof of $\mathcal{NP}$-hardness of MCTR in the general case. Section 4.6 provides our computational study, with some concludig thoughts in Section 4.7.

4.2 Preliminaries

Given a set of points P in the plane, the **geometric graph** $G(P) = (V, E)$ induced by P is the complete graph such that the vertices of V are in one-to-one correspondence with the points in P and E is composed of the set of all straight line segments having one point of V at each end. Now, let l be a line in the plane and $G'(P) = (V, E')$ be a subgraph of $G(P)$. The **stabbing number of line** l relative to $G'(P)$ is the number of edges in E' intersected by l. Moreover, given a set of lines L, the **stabbing number of graph** $G'(P)$ relative to L is the maximum stabbing number among all lines in L.

Regarding the set of lines L, two choices were considered in [9, 18]. The first comprises the set of all axis-parallel lines in the plane. The second is formed by all lines in the plane, independent of their directions. From now on, the first choice will be referred to as **the axis-parallel case** and the latter as **the general case**.

Given a set of points P and a choice of L, the minimum stabbing triangulation problem asks for a subgraph $G'(P) = (V, E')$ of $G(P) = (V, E)$ that corresponds to a triangulation and has the minimum stabbing number among all possible triangulations.

A different but related quantity is the **crossing number**. The **crossing number of a line** l in the plane relative to a subgraph $G'(P) = (V, E')$ of $G(P) = (V, E)$ is the number of connected components in the intersection of l and $G'(P)$. Given a set of lines L in the plane, the **crossing number of graph** $G'(P)$ relative to L is the maximum crossing number among all lines in L.

From the above definitions, we obtain the minimum crossing triangulation problem, in which one seeks a subgraph $G'(P) = (V, E')$ of $G(P) = (V, E)$ that corresponds to a triangulation and has the minimum crossing number among all possible triangulations.

Figure 4.1 shows a triangulation and a set of stabbing lines for the general case. Line l in this drawing has stabbing number 14 and crossing number 2, while line r has both stabbing and crossing numbers equal to 8. On the other hand, Figure 4.2 shows a triangulation and a set of stabbing lines in the axis-parallel case. Line s has stabbing and crossing numbers equal to 8, while line t has stabbing number 11 and crossing number 6.

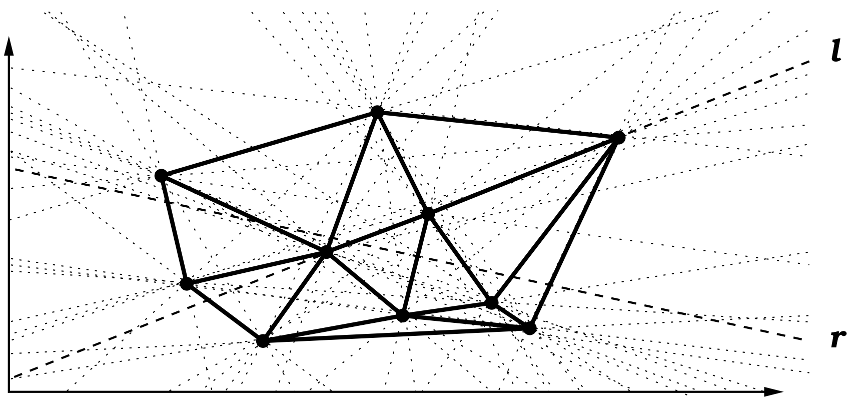

Figure 4.1: A triangulation with a general set of lines.

4.3 The Complexity of Finding a Triangulation with Minimum Axis-Parallel Stabbing Number

We use a terminology similar to the one presented in [9, 18], which is explained below. We consider a set P of points (vertices) in the plane and the corresponding geometric graph $G(P)$ as defined in the previous section.

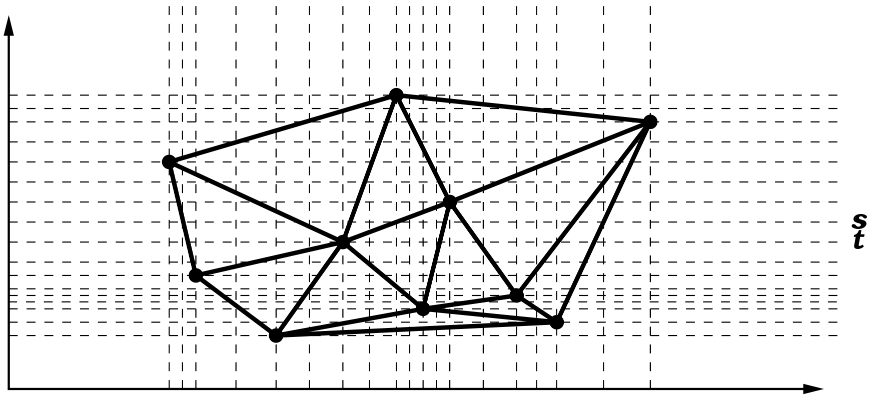

Figure 4.2: A triangulation with an axis-parallel set of lines.

Thus, given P, a *horizontal line* is a maximal set of vertices that are collinear in horizontal direction. A *vertical line* is a maximal set of vertices which are collinear in vertical direction. A *row* is composed by two horizontal lines (with no other horizontal line in the middle) and the space between them. A *column* is the vertical equivalent of a row. An *st-row* consists of three consecutive horizontal lines and the spaces between them. Finally, an *st-column* is formed by three consecutive vertical lines and the spaces between them.

The idea of the hardness proof for the axis-parallel case of MSTR is based on the observation that in this problem, the critical stabbers, i.e., those that have the greatest stabbing number, are those on horizontal or vertical lines, while in the MCTR, the critical stabbers, i.e., those that have the greatest crossing number, are the ones between horizontal or vertical lines. This observation allows us to adapt the structure of the proof in [9, 18] to the MSTR.

Next we present three lemmas that define properties that are useful for the proof of Theorem 4.1.

Lemma 4.1. *Let T be a triangulation in $G(P)$. Consider an st-row formed by three horizontal lines, l_a, l_b and l_c in P, having a, b and c vertices, respectively, with l_b being the middle line. If the number of edges of T in l_a, l_b and l_c are, respectively, $a - i_a - 1$, $b - i_b - 1$ and $c - i_c - 1$, then a horizontal stabber on l_b has stabbing number at least $a + 3b + c + i_a + i_c - 3$.*

Proof. It is easy to see that a horizontal stabber on l_b stabs all the edges having some point in the space between l_a and l_b, which is equal to the crossing number of a stabber between these lines. Moreover, as stated in [10], Section 1.1, § 2, the latter is equal to the number of triangles plus one. From Lemma 4 in [10], this crossing number is at least $a + b + i_a + i_b - 1$. Again, one can easily see that a horizontal stabber on l_b also stabs all the edges having some point between l_b and l_c. Hence, following the same reasoning as before, we can conclude that this contributes $b + c + i_b + i_c - 1$ units to the stabbing number. Clearly, such a horizontal stabber also stabs the edges on l_b, which contributes $b - i_b - 1$ units to the stabbing number. There is, however, an intersection between these sets of edges whenever $i_b \neq 0$. When this happens, for each two neighboring vertices u and v in l_b for which there is no edge (u, v), exactly one edge is counted both in the set between l_a and l_b and in the set between l_b and l_c; in our count of stabbing numbers, each missing edge in l_b contributes one unit to the set between l_a and l_b and one unit to the set between l_b and l_c. Therefore, we must subtract i_b from the stabbing counter. Hence, we can conclude that the stabbing number of a horizontal stabber on l_b is at least $a + b + i_a + i_b - 1 + b + c + i_b + i_c - 1 + b - i_b - 1 - i_b = a + 3b + c + i_a + i_c - 3$. □

Similar arguments can be used to show that the stabbing number of a vertical line l_b, which is the middle line in an st-column composed by l_a, l_b and l_c, is at least $a + 3b + c + i_a + i_c - 3$, with a, b, c, i_a, i_b and i_c defined as before.

The next lemma helps determining the stabbing number of lines crossing a structure that is later used as a variable gadget in the proof of Theorem 4.1.

Lemma 4.2. *Let T be a triangulation in $G(P)$. Consider an st-column formed by three vertical lines, l_a, l_b and l_c in P, having a, b and c vertices, respectively, with l_b being the middle line. Let the number of edges of T in l_a, l_b and l_c be, respectively, $a - i_a - 1$, $b - i_b - 1$ and $c - i_c - 1$. Moreover, for each of these lines, consider the pairs of consecutive vertices with no edges of T connecting them, say, $\{u_a, v_a\}$ in l_a, $\{u_b, v_b\}$ in l_b and $\{u_c, v_c\}$ in l_c. Let j_a, j_b and j_c be the number of horizontal edges in T between* **the three pairs** *$\{u_a, v_a\}$, $\{u_b, v_b\}$ and $\{u_c, v_c\}$, respectively. Suppose that $j_a = j_c > j_b$ and every horizontal edge crossing the space between u_b and v_b also crosses the ones between u_a and v_a and between u_c and v_c. Then a vertical stabber on l_b has stabbing number of at least $a + 3b + c + i_a + i_c + j_a + j_c - 5$.*

Proof. This lemma is very similar to Lemma 4.1, except that now we have horizontal edges crossing the space between specific pairs of vertices, a situation that is illustrated in Figure 4.3, in which arrows point to lines l_a, l_b and l_c. Thus, we start by making some changes in the calculations of the number of triangles between l_a and l_b in order to consider the j_a horizontal edges crossing the space between u_a and v_a.

Notice that there is one triangle intersecting the space between l_a and l_b for each edge in l_a. We denote this set of triangles by A. Besides, for each missing edge in l_a, there are at least two triangles intersecting the space between l_a and l_b. Let I_a denote this set of triangles. Finally, for each horizontal edge e_a between u_a and v_a , there is one triangle above e_a and one triangle below e_a; we let J_a denote this set of triangles. Similarly, we define sets B, I_b and J_b in l_b.

Thus, the number of triangles in the space between l_a and l_b is given by the sum of the cardinalities of the sets A, B, I_a, I_b, J_a and J_b minus the cardinality of their intersections. There are only four intersections to consider: the ones between J_a and B, between J_a and J_b, between J_a and I_a and, finally, between J_b and I_b. It is easy to see that $|J_a \cap B| \leq j_a - j_b$, $|J_a \cap J_b| = 2j_b$, $|J_a \cap I_a| \leq 2$ and $|J_b \cap I_b| = 2$.

The exact cardinality of $J_a \cap B$ and $J_a \cap I_a$ depends on the choice of triangulations that occur with the extreme horizontal edges between u_a and v_a, with, say, (x_{u1}, x_{u2}) being the closest to u_a and (x_{v1}, x_{v2}) being the closest to v_a. If, for instance, (x_{u1}, x_{u2}) forms a triangle with u_a, then $|J_a \cap B| \leq j_a - j_b - 1$ and $|J_a \cap I_a| \geq 1$. However, note that $|J_a \cap B| + |J_a \cap I_a| = j_a - j_b$.

As a consequence, the number of triangles in the space between l_a and l_b is equal to $|A|+|B|+|I_a|+|I_b|+|J_a|+|J_b|-|J_a \cap B|-|J_a \cap I_a|-|J_a \cap J_b|-|J_b \cap I_b| \geq (a-i_a-1)+(b-i_b-1)+(2i_a)+(2i_b)+(2j_a)+(2j_b)-(j_a-j_b)-(2j_b)-2 = a+b+i_a+i_b+j_a+j_b-4$.

We can calculate the number of triangles in the space between l_b and l_c in the same way we did for l_a and l_b and conclude that it must be at least $b+c+i_b+i_c+j_b+j_c-4$.

The stabbing number of a line on l_b is equal to the sum of the number of triangles in the space between l_a and l_b, the number of triangles in the space between l_b and l_c plus two (on each side, a line on l_b stabs one edge more than the number of triangles), plus the number of edges in l_b which is equal to $b-i_b-1$, minus the number of triangles that are in both spaces. The triangles which are in both spaces are those in sets I_b and J_b, subtracting the intersection between them. Therefore, the stabbing number of a line on l_b is at least $(a+b+i_a+i_b+j_a+j_b-4)+(b+c+i_b+i_c+j_b+j_c-4)+2+(b-i_b-1)-(2i_b)-(2j_b)+2$ $= a+3b+c+i_a+i_c+j_a+j_c-5$. □

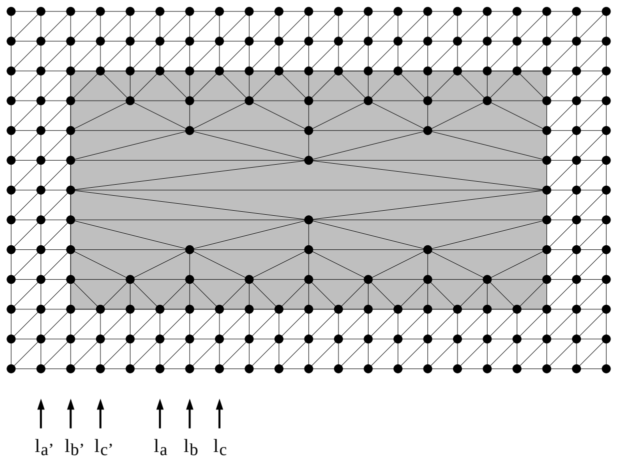

Figure 4.3: Extended rectangle of a variable and lines of the situations described in Lemma 4.2 and Lemma 4.3.

Next we state a lemma that helps us to determine the stabbing number of a line on the border of a variable gadget, which will be useful for proving Theorem 4.1.

Lemma 4.3. *Let T be a triangulation in $G(P)$. Consider an st-column formed by three vertical lines, l_a, l_b and l_c in P, having a, b and c vertices, respectively, with l_b being the middle line. Let the number of edges of T in l_a, l_b and l_c be, respectively, $a-i_a-1$, $b-i_b-1$ and $c-i_c-1$. Moreover, let j_c denote the number of horizontal edges in T between a pair of vertices $\{u_c, v_c\}$ in l_c that have no edges connecting them. Then a vertical stabber on l_b has stabbing number of at least $(a+b+i_a+i_b-1)+(b+c+i_b+i_c+j_c-4)+(b-i_b-1)-i_b = a+3b+c+i_a+i_c+j_c-6$.*

Proof. The situation considered in this lemma is shown in Figure 4.3, in which the vertical lines indicated by the arrows labelled $l_{a'}$, $l_{b'}$ and $l_{c'}$ play the roles of lines l_a, l_b and l_c in this lemma, respectively. Using the same reasoning as in the proof of Lemma 4.2, we find that the stabbing number of a line on l_b is equal to the number of triangles between l_a and l_b plus one, plus the number of edges on l_b, plus the number of triangles between l_b and l_c plus one minus the cardinality of the intersection of these sets.

The only set that differs from those calculated in the proof of Lemma 4.2 is the set of triangles between l_b and l_c. The size of it is given by the formula $|B| + |C| + |I_b| + |I_c| + |J_c| - |J_c \cap B| - |J_c \cap I_c|$, where all items are defined as in the proof of Lemma 4.2.

The cardinalities of the sets $J_c \cap B$ and $J_c \cap I_c$ depend on the same choices of triangulations for the extreme horizontal edges as in the case of sets $J_a \cap B$ and $J_a \cap I_a$ in the proof of Lemma 4.2. We conclude that in this case we have $|J_c \cap B| + |J_c \cap I_c| = j_c + 1$.

Therefore, the number of triangles between l_b and l_c is at least $(b - i_b - 1) + (c - i_c - 1) + 2i_b + (2i_c - 2) + 2j_c - (j_c + 1) = b + c + i_b + i_c + j_c - 5$. So the stabbing number of a line on l_b is at least $(a+b+i_a+i_b-2)+1+(b+c+i_b+i_c+j_c-5)+1+(b-i_b-1)-i_b = a + 3b + c + i_a + i_c + j_c - 6$. □

We are now ready to provide the main result of this section. We present a reduction from 3-SAT to the MSTR in the axis-parallel case, thus proving that the latter is $\mathcal{NP}$-hard.

Theorem 4.1. *Finding a triangulation with minimum axis-parallel stabbing number is $\mathcal{NP}$-hard.*

Proof. As stated earlier, the proof goes along the same lines as the one given in [9, 18] for the $\mathcal{NP}$-hardness of the problem of finding a triangulation with minimum axis-parallel crossing number. It is based on a reduction from 3-SAT and, to facilitate the understanding, our explanation uses the same example as in the cited paper. Thus, Figure 4.6 gives an idea of the MSTR instance obtained from the 3-SAT instance $B(x_0, x_1, x_2) = (x_0 \vee x_1 \vee \overline{x}_2) \wedge (x_0 \vee \overline{x}_1 \vee x_2) \wedge (\overline{x}_0 \vee \overline{x}_1 \vee \overline{x}_2)$.

In this proof we show that an instance of 3-SAT is satisfiable if and only if the corresponding MSTR instance has an axis-parallel stabbing number of at most $5K - 3$ for some value K which, in our construction, is the maximum number of vertices in any horizontal or vertical line. The next definition is used in the arguments that follow.

We say that an st-row (st-column) is *full* if it is composed by three horizontal (vertical) lines having K vertices each. Similarly, we say that a row (column) is full, whenever its composing horizontal (vertical) lines have K vertices each.

The construction. The idea behind the construction is very similar to the one for the $\mathcal{NP}$-hardness of the axis-parallel crossing number. In the next paragraphs we describe the essential components of the construction, such as variable gadgets, literal gadgets, clause gadgets and how the surroundings of these gadgets should be constructed in order to achieve the desired stabbing number.

Variable gadget. A variable gadget is composed of two sets of eight vertices forming rectangles with three vertices on each side. The two rectangles of a variable are horizontally aligned; together they represent a variable x_i. The strip induced by the left rectangle is called the *x_i-column*, while the strip induced by the right rectangle is the *$\overline{x_i}$-column* of the variable. Figure 4.4 shows the variable gadget (shaded), which is analogous to the one in [9, 18].

The triangulations of both rectangles are identical, except for the middle horizontal edge, which is present in one of the rectangles and missing in the other. The strip induced

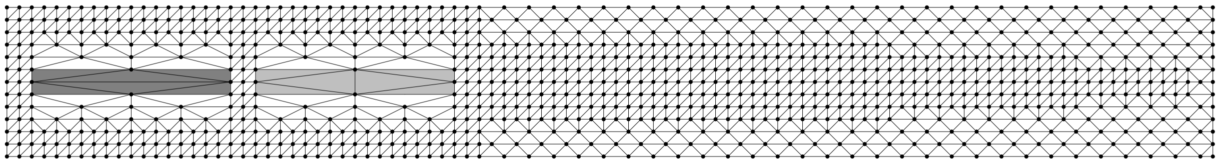

Figure 4.4: A variable gadget with vertices added to its right. Here we assume that the number of occurrences of the most frequent literal is $t = 2$.

by the rectangle with the missing horizontal edge is called the *true-column* of the variable and the strip induced by the other rectangle is the *false-column*. A setting in which the left column (x_i-column) is the false-column, is the *false setting*, while the other possible setting is the *true setting*.

The width of each rectangle must be the smallest power of two greater or equal to four times the number of occurrences of the most frequent literal. Therefore, the width of a rectangle is at most eight times the number of occurrences of the most frequent literal. This is the necessary width to accommodate all the literals and full columns between them as we shall see later.

Around the variables. The st-rows and st-columns neighboring the rectangles of a variable gadget are full. Therefore, from Lemma 4.1, the edges in the convex hull of these rectangles must be present in any triangulation with minimum stabbing number. Suppose an edge is missing in the upper boundary of a variable's rectangle. Then we apply Lemma 4.1 and conclude that the stabbing number of the middle horizontal line of the st-row is greater than $5K - 3$, because the st-row composed by the horizontal line containing the upper boundary and the two horizontal lines above it is full. A similar reasoning can be used for the st-row in the lower boundary and the st-columns surrounding them. Hence, all the edges must be present in the convex hull of the rectangles in any triangulation with minimum stabbing number.

At the rows above and below the rectangles, the horizontal distance between the vertices within the vertical strip defined by the rectangles is halved at each horizontal line farther to the horizontal lines in the boundary of the gadget. Thus, if the number of occurrences of the most common literal is t, the vertices within the vertical strip will have a horizontal distance of one unit after $\Theta(\log t)$ rows. The rectangle of a variable plus the vertices located above and below it within its vertical strip, until and including the lines where the horizontal distance between the vertices is of one unit, compose what we call an *extended rectangle* of a variable. An example of an extended rectangle of a variable is shown shaded in Figure 4.3.

In order to ensure that one of the rectangles of a variable will have a missing horizontal edge and the other will have it present, the horizontal line containing these vertices must have $K - 1$ vertices, while the two horizontal lines above it and the two horizontal lines below it must have K vertices. Thus, by Lemma 4.1, there can be only one edge missing in this middle horizontal line, otherwise the stabbing number would be greater than $5K - 3$.

The correct horizontal vertex count must be guaranteed by placing the proper number of vertices to the right of the variable gadgets and in the right side of the horizontal lines above and below the gadget. Let h be the height of an extended rectangle of a variable. Thus, the first line above the central horizontal line will have $2^0 = 1$ vertex missing at the right end, the second line above the central horizontal line will have $2^0 + 2 \times 2^1 = 5$ vertices missing at the right end and the yth line above the horizontal central line will have $1 + 2\sum_{i=1}^{y-1} 2^i$ vertices missing at the right end, where $y \leq h/2$. If y is odd, the first missing vertex will be at the second position from the right. If y is even, the first missing vertex will be at the third position from the right. After line $(h/2)$, every horizontal line has the same number of missing vertices. These missing vertices appear at the right end side of the construction alternating a present and a missing vertex, i. e., there is a present vertex, than a missing vertex etc., until the number of missing vertices is reached. The same number of vertices are missing at the right end of the lines below the central horizontal line. Figure 4.4 shows how this can be done.

Variable position. The gadget for a variable x_j is positioned above and to the right of the gadget for a variable x_i where $i < j$. As before, a variable gadget is adjacent to full st-rows and st-columns, so variable gadgets are horizontally separated by st-columns and vertically separated by st-rows.

Literal gadget. A literal gadget is composed by a 3×3 grid with the central vertex missing, i.e., a square composed of eight vertices. At each side of the square, the vertices are one unit of distance apart from each other.

The setting of a literal gadget in which the middle horizontal edge is missing is called the *false* setting of a literal gadget, while the setting in which the middle horizontal line is present is called the *true* setting of a literal gadget.

Clause gadget. As we start with an instance of 3-SAT, each clause has exactly three literals. In our construction the three literal gadgets of a clause are horizontally aligned. Above and below every clause are full st-rows.

If a literal x_i appears in clause c_j, we place a literal gadget in the x_i-column. If, however, a literal $\overline{x_i}$ appears in clause c_j, we place a literal gadget in the $\overline{x_i}$-column. If a literal x_i appears both in clauses c_j and c_k, where $j < k$, we place the literal x_i in clause c_k below and to the right of the literal x_i in clause c_j. This guarantees that a vertical line never stabs more than one literal gadget. Also, a clause c_k lies below a clause c_j for $k > j$.

To the right of clauses. We want the stabbing number to be less than or equal to $5K - 3$ if and only if the formula is satisfiable. Therefore, if a clause is not satisfied, i.e., its three literals have false settings, it must produce a stabbing number greater than $5K-3$. Because in a false setting of a literal gadget, the middle horizontal edge is missing, an unsatisfied clause implies three edges missing in the middle horizontal line. Thus, if the middle horizontal line of a clause has $K - 2$ vertices and the rows directly above and below it are full, an unsatisfied clause produces a stabbing number greater than $5K - 3$ for the top and bottom horizontal lines of the clause. Conversely, if at least one literal has a true setting, the stabbing number is less or equal to $5K - 3$ for those lines.

Hence, we must add vertices to the right of clauses making the middle horizontal line have $K-2$ vertices and the rows above and below them be full rows.

Below variables and literals. We add vertices below variables and literals in order to guarantee that the st-columns around variables are full and that variables and literals have the correct setting of missing and present middle edges.

Recall that we want to make a literal gadget have a true setting whenever its corresponding column is the true-column, e.g., we want literal $\overline{x_i}$ to have a true setting if the $\overline{x_i}$-column of variable x_i is the true-column, i.e., the one with the middle horizontal line missing. Therefore, we want a literal gadget to have the middle horizontal edge present (literal gadget true setting) if its corresponding column have a true setting.

Let l_b be the vertical line containing the left side of a literal gadget x_i, let l_c be the vertical line passing through the center of the same literal gadget, i.e., the line immediately to the right of l_b and let l_a be the line immediately to the left of l_b. Let a be the number of vertices in l_a and b be the number of vertices in l_b. Let i_a and i_c be the number of missing edges connecting neighboring vertices, respectively, in lines l_a and l_c. Moreover, let j_a and j_c be the number of horizontal edges crossing, respectively, lines l_a and l_c inside the extended x_i rectangle. Because all rows containing points in the extended rectangle of some variable are full (except for the two central rows), all the horizontal edges must be present inside the extended rectangle (except, possibly, the central edge). Therefore, according to Lemma 4.2 and Lemma 4.3, the stabbing number of l_b is at least $a+3b+c+i_a+i_c+j_c-6$ if the literal is the first of its kind, i.e., the leftmost literal in the x_i strip, and is at least $a+3b+c+i_a+i_c+j_a+j_c-5$ otherwise.

Whenever the central horizontal edge is present in the x_i variable rectangle and missing in the corresponding literal gadget, the expression for the stabbing number of a line on l_b has a strictly smaller value than when that edge is present in the gadget. Thus, we add vertices in the inferior portion of those three lines making the stabbing number equal to $5K-3$ whenever the central edge is missing in the variable and present in the literal. Observe that this does not prevent both middle edges from missing simultanously; the presence of the middle horizontal edges are enforced by the vertices we added to the right of clauses and variables.

If the literal under consideration is not the leftmost literal of its strip, Lemma 4.2 allows us to calculate the number of vertices we have to add at the bottom of the construction in order to guarantee the correct stabbing number. Let h be the height of the extended rectangle in question and let y_a, y_b and y_c be the number of vertices missing in lines l_a, l_b and l_c inside the extended rectangle. Thus, we have $y_a = h-1$, $y_c = h-1$, $a = K-(h-1)$, $b = K-y_b$, $c = K-(h-1)-1$, $i_a = 1$, $i_b = 1$, $i_c = 1$, $j_a = h-1$, $j_b = y_b$ and $j_c = h-1$, and we know that the stabbing number of line on l_b is at least $a+3b+c+i_a+i_c+j_a+j_c-5 = K-h+1+3K-3y_b+K-h+1+1+h-1+h-1-5 = 5K-3y_b-4$. Therefore, in order to guarantee that a line on l_b will have a stabbing number of $5K-3$ in a minimum triangulation, we must have $5k-3y_b-4+z = 5K-3$, so, $z = 3y_b+1$ and this is how much we must increase the stabbing number of l_b by adding vertices at the bottom of the construction.

To this end, we leave a distance of two units between the last $h+1$ lines and for each line l_b, add y_b vertices in its bottom and one vertex at the bottom of the vertical line to its right. Each vertex added to l_b increases the stabbing number of a line on it by three units, and each vertex added to the vertical line to its right adds one unit to the stabbing number of that line, thus achieving the desired stabbing number. Figure 4.5 gives an idea of how to obtain such a construction. Observe that as the number of vertices in l_a, l_c and the vertical line to the right of l_c are always smaller than in l_b, the stabbing numbers of stabbers on these vertical lines are not greater than $5K - 3$.

In the case of the leftmost literal in the x_i strip, we can use Lemma 4.3 to conclude that the stabbing number of a line on l_b has a stabbing number of $5K - 6$ in a minimum triangulation and, therefore, we add one vertex to l_b to guarantee a stabbing number of $5K - 3$.

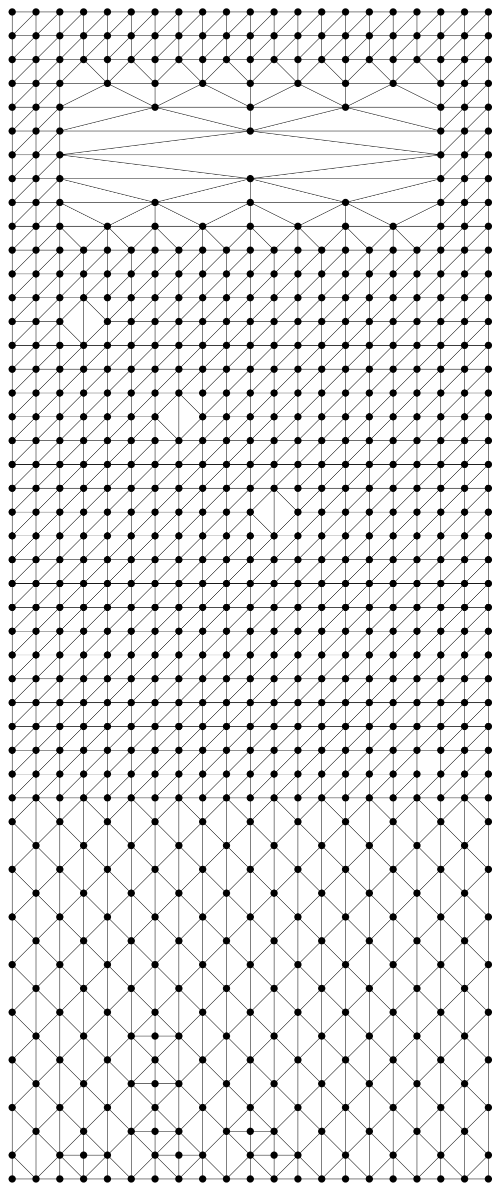

Figure 4.5: One rectangle of a variable gadget, three literal gadgets and the vertices added below them.

Satisfiability implies a stabbing number of $5K-3$. If the reduction we described is valid, a satisfiable formula must produce an MSTR instance with an optimal solution of value no more than $5K-3$, where K is the maximum number of vertices in any horizontal or vertical line.

The only parts of the construction that could have a stabbing number greater than $5K-3$ are the clause gadgets, which have horizontal stabbing number of $5K-2$ if the three literals in the clause have false settings. However, if the formula is satisfiable, there is a setting in which at least one literal has a true setting in every clause. By construction, this implies that the stabbing number cannot be greater than $5K-3$. Thus, satisfiability implies a stabbing number of at most $5K-3$.

Unsatisfiability implies a stabbing number greater than $5K-3$. For the converse direction, assume that the the formula is not satisfiable; we establish that the resulting MSTR instance must have an optimal solution with value greater than $5K-3$, where K is the maximum number of vertices in any horizontal or vertical lines.

If a formula is unsatisfiable, there is no setting of variables that satisfies every clause. Thus, for every setting of variables, there is always at least one clause that has a false setting for all three literals. By construction, this implies that for every setting of variable gadgets there is always a clause in which all three literal gadgets have the middle horizontal edge missing. Therefore, the stabbing number of the horizontal lines containing the top and bottom lines of this clause gadget is equal to $5K-2$. Hence, unsatisfiability implies a stabbing number greater than $5K-3$.

Polynomial size of the construction. It remains to be shown that the construction has polynomial size. As the construction is very similar to the one presented in [9, 18], the arguments are basically the same as used in that proof, except that a rectangle representing a variable has width of at most $8t$ instead of $4t$.

Let B be an instance of 3-SAT, let n be the number of variables, c the number of clauses and t the number of occurrences of the most common literal. The size of a rectangle representing a variable is at most $8t$. The number of vertices we have to add to the right of a variable gadget is $\Theta(t)$. Thus, the horizontal size of the construction is $\Theta(nt)$.

Each rectangle representing a variable has a height of $\Theta(\log t)$. The height of clauses is constant and equal to 2, plus that of the full *st*-rows between them, giving a total of 4 per clause. The number of vertices we have to add at the bottom of the construction to achieve the desired vertical stabbing number is $\Theta(\log t)$. Hence, the vertical size of the construction is $\Theta(c + n\log t)$. Therefore, the total size of the construction is polynomial on c, n and t.

□

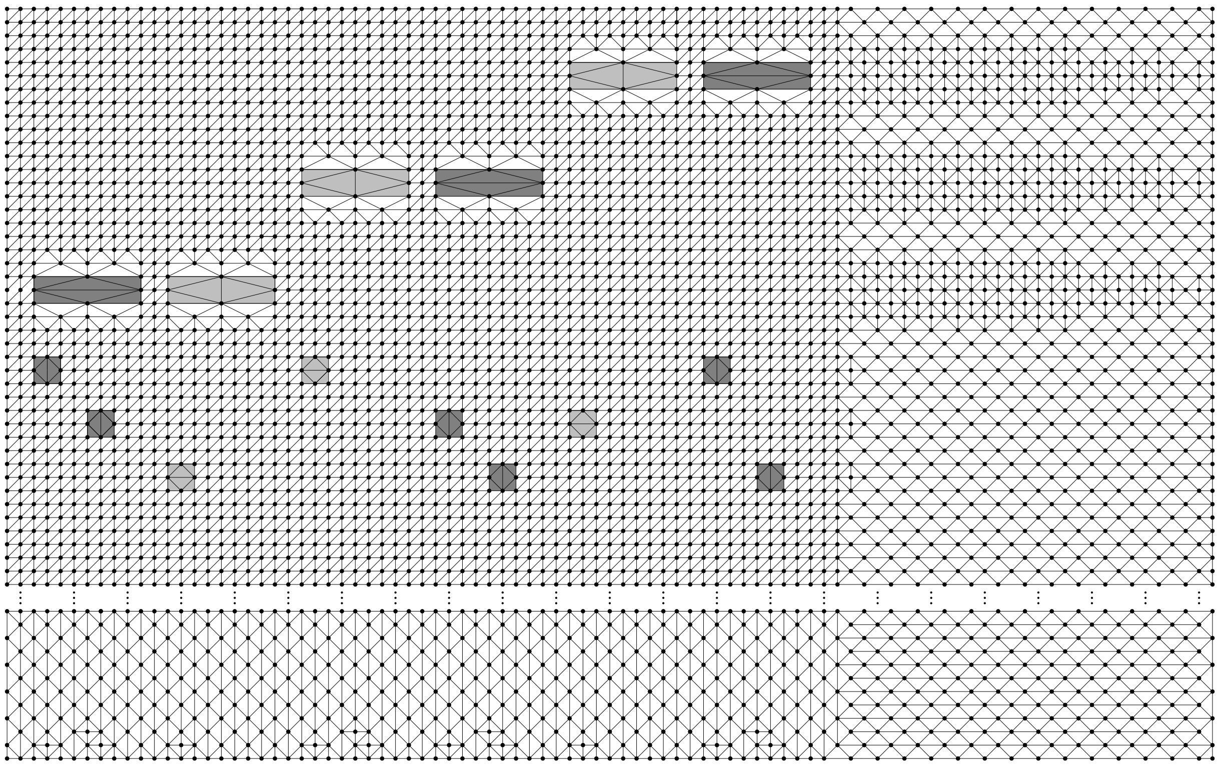

Figure 4.6: The construction for the formula $(x_0 \vee x_1 \vee \overline{x_2}) \wedge (x_0 \vee \overline{x_1} \vee x_2) \wedge (\overline{x_0} \vee \overline{x_1} \vee \overline{x_2})$ with assignments $x_0 = false$, $x_1 = true$ and $x_2 = true$. A dark shading indicates a false setting, while a light shading indicates a true setting.

4.4 The Complexity of Finding a Triangulation with Minimum General Stabbing Number

We now turn our attention to the problem of computing a triangulation with minimum general stabbing number. To this end, consider a slightly changed version of the construction given in the proof of Theorem 4.1, where the second vertical line is at distance one unit from the first vertical line, the third is at distance two units from the second vertical line, the fourth is at distance three units from the third vertical line and, in general, the $n+1$-th vertical line is at distance n units from the n-th vertical line. We will refer to this new construction as the *modified* one.

Before we proceed, we introduce some additional terminology. A diagonal stabbing line, or simply a *diagonal line*, is any stabbing line that is not vertical or horizontal. Consider a grid of $Q \times Q$ vertices with horizontal spaces, as in the modified construction, i.e., the distance of vertical line $n+1$ to vertical line n is n units. We call a grid with this spacing rule, a *modified grid*. Figure 4.7 shows an example of such a modified grid.

Consider a diagonal line l that stabs two vertices y and z, such that the segment $\overline{yz}$ (of l) contains no other vertex. Denote by h the number of horizontal lines containing vertices of the grid and intersected by l between y and z (or, more precisely, by the segment $\overline{yz}$). Analogously, define v to be the number of vertical lines containing vertices of the grid and intersected by the segment $\overline{yz}$ of l. We say that the *stabbing distance relative to l* between y and z is equal to $max[h, v]$; if the context is clear, we may omit the line.

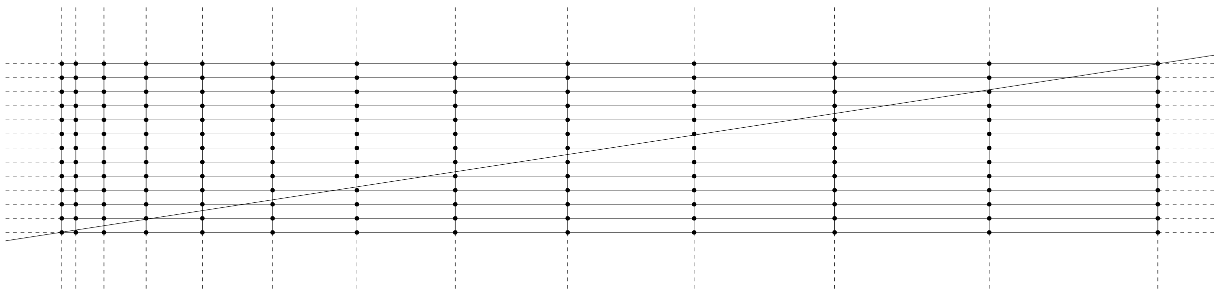

Figure 4.7: A 13 × 13 modified grid with a diagonal stabbing line. The dotted lines represent the extensions of the vertical/horizontal lines outside the grid.

Notice that just like in the axis-parallel case of the MSTR, the critical stabbing lines are the ones stabbing many vertices, as opposed to stabbing a lot of space between the vertices. Therefore, the main idea used in Theorem 4.2 is to show that diagonal stabbing lines stab more spaces between vertices, while orthogonal stabbing lines stab more vertices.

The following lemma gives us a property regarding the number of pairs of vertices with a given distance that can be stabbed by a given diagonal line. This property will later be used to establish the number of vertices that can be stabbed by a diagonal stabbing line in a $Q \times Q$ modified grid.

Lemma 4.4. *In a modified grid there are at most $2x + 1$ pairs of vertices with distance x that can be stabbed by a given diagonal line l.*

Proof. Let l be a diagonal stabbing line and a its angular coefficient, where we assume $a > 0$. Let y and z be two vertices stabbed by l with a distance of x between them. Moreover, let h and v be, respectively, the number of horizontal and vertical lines containing vertices of the grid and intersected by l between y and z. By definition, at least one of $h = x$ or $v = x$ must be true.

Consider the case in which $v = x$ and $h \leq x$. As each column has a unique width and every row has height one, we can say that the Euclidean distance between the horizontal coordinates of y and z is equal to $b + (b + 1) + ... + (b + x)$, where b is the width of the first column after y. Hence, this distance is equal to $b(x+1)+x(x+1)/2$. The Euclidean distance between the vertical coordinates of y and z is equal to $h + 1$. As the angular coefficient is a, we have $h + 1 = a(b(x + 1) + x(x + 1)/2)$. For each $h \in \{0, 1, 2, ..., x\}$, the previous equation has a unique solution for the given x. Therefore, there are at most $x+1$ pairs of vertices (one for each possible value of h) that can be stabbed with distance x by a diagonal line when $v = x$ and $h \leq x$.

The other possibility is that $h = x$ and $v < x$. As each column has a unique width and every row has height one, we can say that the Euclidean distance between the horizontal coordinates of y and z is equal to $b + (b + 1) + ... + (b + v)$, where b is the width of the first column after y. Hence, this distance is equal to $b(v+1)+v(v+1)/2$. The Euclidean distance between the vertical coordinates of y and z is equal to $x + 1$. As the angular coefficient is a, we have $x + 1 = a(b(v + 1) + v(v + 1)/2)$. For each $v \in \{0, 1, 2, ..., x - 1\}$, the latter equation has a unique solution for the given x. Therefore, there are also at most x pairs of vertices (one for each possible value of v) that can be stabbed with distance x by a diagonal line for $h = x$ and $v < x$.

Adding the values obtained in the two cases, we have $2x+1$ possible pairs with distance x. ☐

The proof of Lemma 4.4 for $a < 0$ is analogous and therefore omitted.

The purpose of the next lemma is to show that the number of vertices stabbed by any diagonal line is less than $Q/4$ if Q is big enough. This result is obtained by showing that for $Q \geq 535$, the function that yields the number of extended horizontal and vertical lines stabbed in points that do not contain vertices grows faster than the function that yields the number of vertices stabbed. Both of these functions are obtained from the relation in Lemma 4.4. This allows it to establish the central idea of the proof of Theorem 4.2 by showing that diagonal lines stab less vertices than orthogonal lines and, for this reason, have smaller stabbing numbers.

Lemma 4.5. *Any diagonal line stabs less than $Q/4$ vertices in the modified construction for any $Q \geq 535$.*

Proof. As the modified construction has some well-defined holes, i.e., missing vertices in comparison with a modified grid, the stabbing number of a diagonal line in a minimum stabbing triangulation of a $Q \times Q$ modified grid is greater than or equal to that in a minimum stabbing triangulation of the modified construction with a maximum number of vertical or horizontal lines equal to Q.

We call an extended horizontal/vertical line, a horizontal/vertical line plus its extension outside the grid (see Figure 4.7). The modified grid has $Q \times Q$ vertices, so every diagonal stabbing line intersects exactly Q extended horizontal lines and Q extended vertical lines.

As vertices only exist in the intersections of horizontal and vertical lines, stabbing Q vertices (which is the maximum possible number of vertices stabbed) is only possible if a diagonal line does not stab an extended horizontal/vertical line in any point other than a vertex. This means that whenever a stabbing line intersects an extended horizontal and an extended vertical line in points not containing vertices, one less vertex is stabbed by that line.

Now let l be a diagonal line and v and h be, respectively, the number of extended vertical and horizontal lines in the grid intersected by l in some point not containing a vertex. Because v and h have the same value, the number of vertices intersected by l is equal to $Q - (v + h)/2$ or $Q - v$ or $Q - h$.

Let p be the number of vertices stabbed by l and let d be the sum of the stabbing distances of each pair of consecutive vertices in l. As the stabbing distance of two vertices is the maximum of either the number of horizontal lines or the number of vertical lines stabbed between these two vertices, then $d < v + h$. Therefore, as $Q = p + (v + h)/2$, we have $2(Q - p) = v + h \Rightarrow d < 2(Q - p) \Rightarrow Q > p + d/2$.

From Lemma 4.4, there are at most $2x+1$ pairs of vertices with distance x stabbed by a diagonal line. Remember that whenever a pair of vertices is stabbed at distance x, there are x horizontal or vertical lines stabbed in points not containing vertices. Therefore, to maximize the number of stabbed vertices, the pairs with smallest distance should be stabbed. Let y be the greatest distance between two consecutive vertices (with no other vertex between them) stabbed by l. In this situation we have $p = 1 + \sum_{x=0}^{y} 2x + 1$ (two consecutive pairs of vertices share a vertex) and $d = \sum_{x=0}^{y} x(2x+1)$. Therefore, we have $p = 1 + ((1 + (2y+1))(y+1)/2) = y^2 + 2y + 2$ and $d = 2y^3/3 + 3y^2/2 + 5y/6$.

Notice that if $Q \geq 535$, for any $y = 0..10$ we have $p < Q/4$ and for any $y > 10$, the value of d as defined above grows faster than the value of p. Therefore, for any $Q \geq 535$, any diagonal line in a $Q \times Q$ modified grid stabs less than $Q/4$ vertices. □

We can now state the main result of this section. Knowing that a diagonal line in a $Q \times Q$ modified grid stabs less than $Q/4$ vertices, we conclude that the orthogonal stabbing lines are the critical ones in a set of points with that spacing rule. Moreover, we can use this information to show that a modified construction provides a reduction from 3-SAT to MSTR in the general case. The details are given below.

Theorem 4.2. *Finding a Triangulation with Minimum Stabbing Number is $\mathcal{NP}$-hard.*

Proof. The main idea is to modify the construction from Theorem 4.1 in order to obtain a new one in which every diagonal line has a stabbing number less than or equal to some constant defined a priori, i.e., the minimum axis-parallel stabbing number of a triangulation in the modified construction. Therefore, the reduction from 3-SAT to MSTR in the axis-parallel case is also valid for the general case.

It is easy to see that the modified construction has the same properties as our original construction and has polynomial size. Notice that the original construction has horizontal size of $\Theta(nt)$, where n is the number of variables of the 3-SAT instance and t is the number of occurrences of the most common literal. Thus, the new construction has a horizontal size of $\Theta(n^2t^2)$ and the same vertical size as the original construction. Therefore, it is still polynomial.

Note that our constructions allows a triangulation in which every vertex has degree at most seven and the axis-parallel stabbing number is $5K - 3$. Hence, when a line intersects a vertex, its stabbing number increases by at most seven (this value is overestimated in general). On the other hand, when a vertex is not intersected, i.e., when the line stabs horizontal and vertical lines of the grid in points having no vertex, the stabbing number is increased by at most four. One such triangulation in the original construction can be seem in Figure 4.6. It is possible to obtain other triangulations with different stabbing numbers for diagonal lines; however, for our purpose, it is enough to show that there is a triangulation with the desired properties.

From Lemma 4.5 we conclude that if $Q \geq 535$, then at most $Q/4$ vertices can be stabbed by any diagonal line. Thus, the stabbing number of any diagonal line is at most $7(Q/4) + 4(3Q/4)$. We want this expression to be smaller than or equal to $5K - 3$, which is the stabbing number of the axis-parallel version. Because by construction, $Q \leq K + 16t - 1$, we want the inequality $7/4(K + 16t - 1) + 3(K + 16t - 1) \leq 5K - 3$ to be true, implying that $K \geq 304t - 7$ should be true. As we can increase the value of K by an appropriate amount simply by adding vertical and horizontal lines to the right and bottom of the construction without altering its properties, this relation can be satisfied for any value of t. Therefore, there is a polynomial reduction from 3-SAT to MSTR, so the latter is $\mathcal{NP}$-hard. □

4.5 The Complexity of Finding a Triangulation with Minimum General Crossing Number

In this section we use a different approach to prove that the problem of computing a triangulation with minimum general crossing number is $\mathcal{NP}$-hard. The construction in the proof of Theorem 4.2 guarantees that the stabbing numbers of horizontal and vertical lines are greater than or equal to those of diagonal lines, thus making horizontal and vertical lines critical for determining lines with the greatest stabbing or crossing numbers. In the present case, in which we are interested in the general crossing number, the construction in the proof ensures that the *almost* horizontal and vertical lines, i.e., those with very big/small angular coefficients, are the critical ones.

As before the reduction is from 3-SAT. More precisely, we show that an instance of 3-SAT is satisfiable if and only if the corresponding MCTR instance has crossing number of at most $2K - 1$ for some value K, which is the maximum number of vertices in any horizontal or vertical line.

The $\mathcal{NP}$-hardness proof uses a terminology similar to the one used in the previous proofs. Accordingly, a *horizontal line* is a maximal set of vertices that are collinear in horizontal direction. A *vertical line* is a maximal set of vertices that are collinear in vertical direction. A *row* is composed by two horizontal lines (with no other horizontal line in the middle) and the space between them. A *column* is the vertical equivalent of a row. A *cr-row* consists of three consecutive horizontal lines and the spaces between them, where each horizontal line contains at least $K-3$ vertices. Finally, a *cr-column* is formed by three consecutive vertical lines and the spaces between them, where each vertical line contains at least $K - 3$ vertices.

Theorem 4.3. *Finding a triangulation with Minimum Crossing Number is $\mathcal{NP}$-hard.*

Proof. As before, we start by describing the gadgets and gving an explanation of how these gadgets interact for an overall reduction from 3-SAT. This is followed by an argument for the correctness of the reduction. The proof is completed by showing that the construction is polynomial.

The construction containing the gadgets has the form of a lattice (see Figure 4.8), with lines composed by cr-rows or cr-columns. Between these lines are spacer gadgets. Gadgets corresponding to variables, literals and clauses lie on the lines of the lattice.

Spacer gadget. A *spacer* is a set of points as the one depicted in Figure 4.9. The triangulation in that figure has crossing number 27, whihc is significantly less than the crossing number of a cr-row or cr-column, which have crossing number 34, as shown in Figure 4.10. The purpose of this difference is to enforce that lines intersecting spacers (with the possible exception at extreme positions) have smaller crossing numbers than the ones not intersecting them.

Variable switch gadget. A *variable switch* is composed by two sets of eight vertices that form squares with three vertices at each side. Each of these squares is called a *half-variable switch*. The two squares of a switch are horizontally aligned, i.e., they are in the

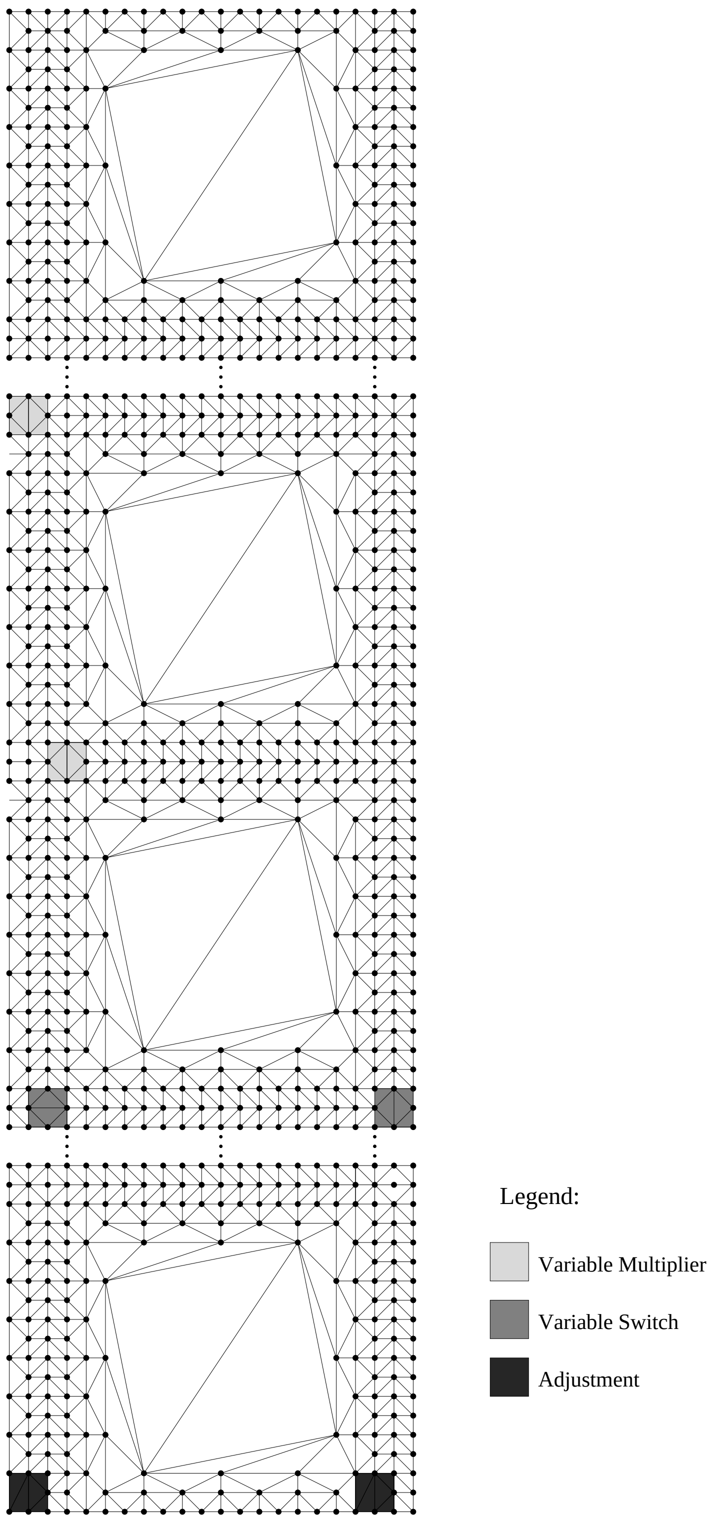

Figure 4.8: Part of the lattice containing variable switch, variable multiplier and spacers. At the bottom, the shaded areas indicate missing vertices to guarantee the correct crossing number.

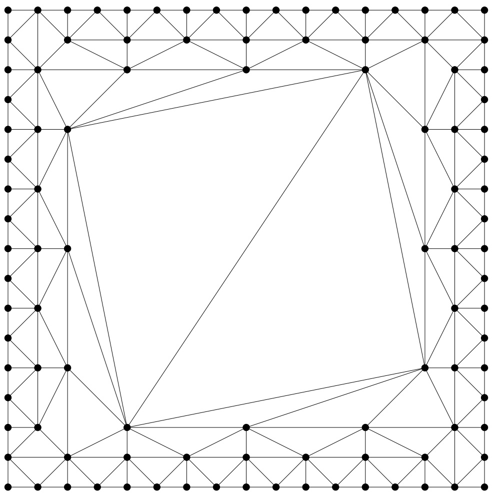

Figure 4.9: A spacer gadget and one of its possible triangulations.

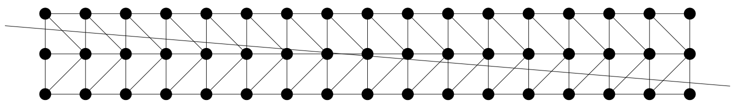

Figure 4.10: Part of a cr-row with a crossing line.

same horizontal line of the lattice; they also lie on neighboring vertical lines of the lattice. The left square is called the x_i-switch, while the right one is the $\overline{x_i}$-switch of the variable. Figure 4.8 shows the variable switch gadget shadowed according to the legend.

The triangulation of both squares is identical, except for the middle horizontal edge, which is present in one of the rectangles and missing in the other. A setting in which the left column (x_i-switch) has the horizontal edge present, is the *false* setting, while the other possible one is the *true* setting.

Variable multiplier gadget. A *variable switch gadget* of a variable x_i allows us to use at most two literals x_i and two literals $\overline{x_i}$ (described below). Thus, whenever we have some literal appearing in more than two clauses, we must use a *variable multiplier gadget.*

Such a gadget is composed by two sets of eight vertices forming a square with three vertices at each side. These squares are located above a variable switch gadget or a *variable gate gadget* (described below). A variable multiplier gadget has one of its top squares shifted by one unit to the right, while the other has it shifted one unit to the left relative to a variable switch gadget or a variable gate gadget's position. See Figure 4.8 for a variable multiplier gadget above a variable switch gadget. Note that this type of gadget is never vertically aligned with any other gadget.

Variable gate gadget. A *variable gate gadget* is the gadget that connects literals to a variable (literal gadgets are described below). The variable switch gadgets can also function as variable gate gadgets. These gadgets have the same form as the other variable gadgets: eight vertices forming a square with three vertices at each side. A variable gate gadget is located to the right of one of the variable multiplier gadget's square and horizontally aligned with it. See Figure 4.11 for a representation of an instance containing a variable gate gadget.

Variable gadget. A *variable gadget* is the composition of variable switches, multipliers and gates that emulates the behavior of a variable. All these gadgets are called generic *variable gadgets*, except when referring to a specific type of gadget.

The two lines containing the top and bottom of any variable gadget contain exactly K vertices, while the central line contain $K-2$ vertices.

Notice that when a variable multiplier m_{i_1} is above a x_i-switch with a true setting (middle horizontal edge missing), then m_{i_1} must have its middle horizontal edge present to ensure a minimum crossing number (i.e., $2K-1$) and a variable gate g_{i_1} to the right of m_{i_1} must have its middle horizontal edge missing, i.e., g_{i_1} must have the same setting as the x_i-switch.

This setting of switches, multipliers and gates generates a chain reaction guaranteeing that every gate related to a particular half switch has the same setting. Therefore, we call the x_i-columns the column of the x_i-switch and the column of every gate related to this particular half switch.

Around the variables. Above a half-variable switch or a variable gate there can be at most one variable multiplier, while below a half switch or a gate there can be at most two literal gadgets. However, a variable multiplier and a literal cannot be present at the same time.

Variable position. The gadget for a variable x_j is positioned above and to the right of the gadget for a variable x_i, where $i < j$. In the following, the gadget for a variable refers to the set of all gadgets composing a variable, i.e., variable switch, variable multipliers and variable gates.

Literal gadget. A literal gadget is composed of a set of eight vertices forming a square with three vertices at each side. This square is located below a variable gate gadget (or a variable switch gadget that plays the role of a variable gate). If the variable gate has only one literal, then it must be vertically aligned to that gadget. If, however, the variable gate has two literals, then the top one is shifted one unit to the left, while the other is shifted one unit to the right relative to the horizontal position of that gadget.

A literal gadget in which the middle horizontal edge is missing is called the *false setting* of the respective variable. Analogously, the setting in which the middle horizontal line is present is called the *true setting* of the variable.

Clause gadget. Because our reduction proceeds from is 3-SAT, each clause has exactly three literals. In our construction the three literals forming a clause are horizontally aligned. The clause is the cr-row containing the literal gadgets.

If a literal x_i appears in clause c_j, we place a literal gadget in one of the x_i-columns with room for a literal gadget. If, however, a literal $\overline{x_i}$ appears in clause c_j, we place a literal gadget in one of the $\overline{x_i}$-columns with room for a literal gadget. If a literal x_i appears both in clauses c_j and c_k, where $j < k$, we place the literal x_i in clause c_k below and to the right of the literal x_i in clause c_j. This guarantees that a vertical or almost vertical line never crosses more than one literal gadget. Moreover, a clause c_k lies below a clause c_j for $k > j$.

Adjustments to the crossing number. In order to guarantee the desired crossing number, some adjustments must be made to cr-rows and cr-columns. Adjustment to a cr-row is only necessary when there is no gadget in the cr-row (this may happen to obtain the same number of cr-rows and cr-columns). This is done in a region close to its extremities by removing one vertex from its central horizontal line. The removed vertex must not belong to a cr-column. Adjustment to a cr-column is necessary whenever there is no gadget in the cr-column. In this case, the adjustment is done in the same way as for the cr-rows. Another situation requiring the adjustment of a cr-column is when there are displaced gadgets (like variable multipliers or literals). In this case, the second (from the bottom) vertex is removed from the left vertical line of the cr-column (Figure 4.8).

Satisfiability implies a crossing number of $2K-1$. If the reduction we described is valid, then a satisfiable formula must produce a MCTR instance with an optimal solution of value no more than $2K-1$, where K is the maximum number of vertices in any horizontal or vertical lines.

A crossing line strictly contained in a cr-row or cr-column has a crossing number that is less than or equal to the crossing number of any line crossing a spacer gadget, as can be seen from Figure 4.9 and Figure 4.10. Therefore, the only parts of the construction that could have a crossing number greater than $2K-1$ are the clause gadgets, which have crossing number $2K$ if the three literals in the clause have false settings. However, if the formula is satisfiable, there is a setting in which at least one literal has a true setting in every clause. By construction, this implies that the crossing number cannot be greater than $2K-1$. Thus, satisfiability implies a crossing number of at most $2K-1$.

Unsatisfiability implies crossing number greater than $2K-1$. If the reduction is valid, the other direction of the proof must also be valid, i.e., if a formula is not satisfiable, the resulting MCTR instance must have an optimal solution with value greater than $2K-1$, where K is the maximum number of vertices in any horizontal or vertical lines.

If a formula is unsatisfiable, there is no setting of variables that satisfies every clause. Thus, for every setting of variables, there is always at least one clause that has a false setting for all three literals. By construction, this implies that for every setting of variable gadgets, there is always a clause in which all three literal gadgets have the middle horizontal edge missing. Therefore, the crossing number of the corresponding cr-row is equal to $2K$. Hence, unsatisfiability implies a stabbing number greater than $2K-1$.

Polynomial size of the construction. Let B be an instance of 3-SAT, let n be the number of variables, c the number of clauses and t the number of occurrences of the most

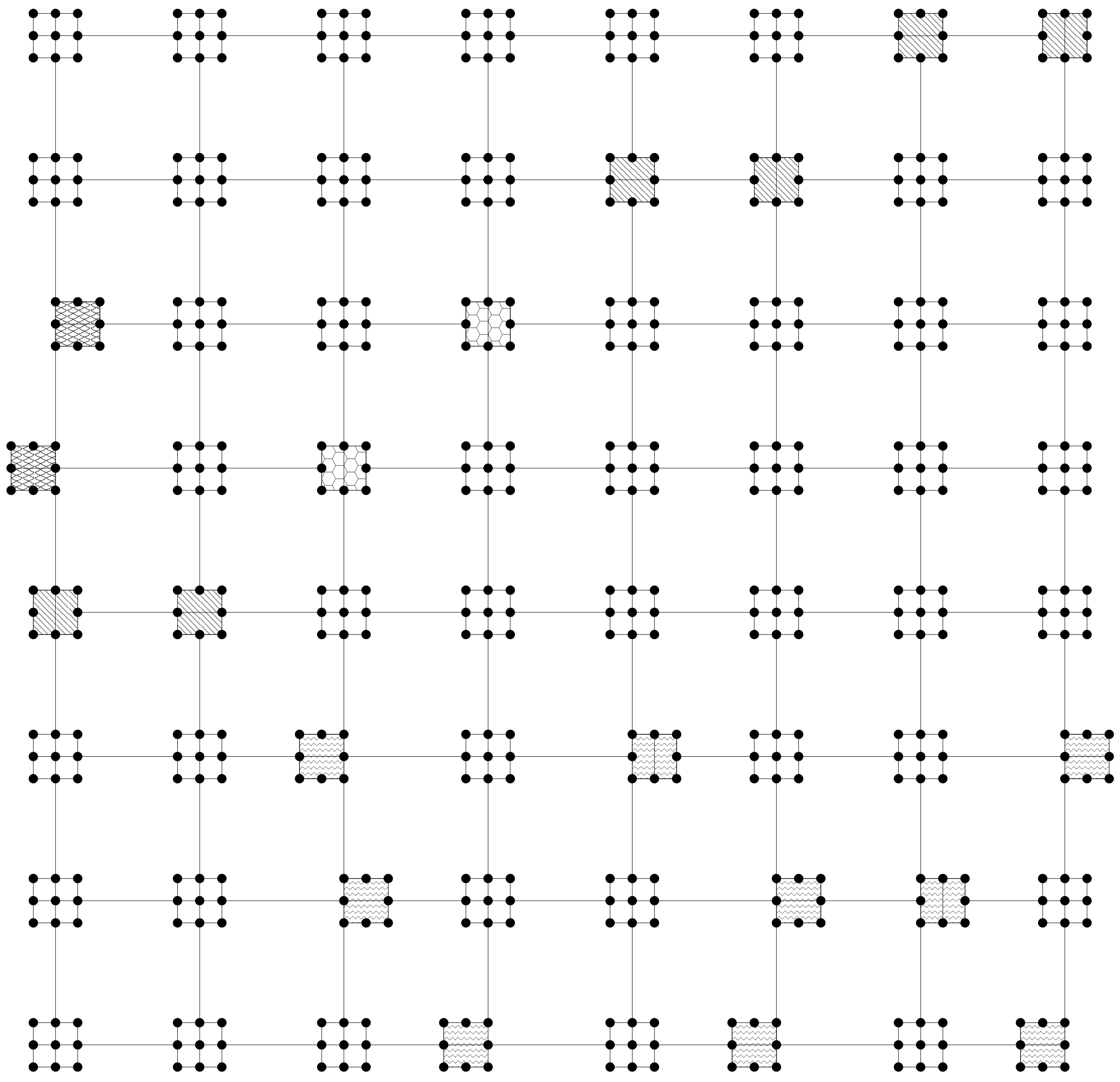

Figure 4.11: Representation of construction for the formula $(x_0 \vee x_1 \vee \overline{x_2}) \wedge (x_0 \vee \overline{x_1} \vee x_2) \wedge (x_0 \vee \overline{x_1} \vee \overline{x_2})$, and values $x_0 = true$ and $x_1 = x_2 = false$. The long edges represent pieces of cr-rows and cr-columns and the empty spaces between them represent spacer gadgets.

common literal. The number of cr-columns necessary to accommodate t occurrences of a literal is at most equal to t. The width of any variable gadget is fixed and equal to 3 and the width of a spacer gadget is equal to 17. Therefore, the horizontal size of the construction is less than or equal to $2 \times 20 \times n \times t + 17 = \Theta(nt)$.

The number of cr-rows necessary to accommodate the variable gadgets connected to t occurrences of a literal is at most t. The height of any variable gadget is equal to 3 and the height of a spacer is equal to 17. Each clause uses exactly one cr-row. Therefore, the height of the construction is less than or equal to $2 \times 20 \times n \times t + c + 17 = \Theta(nt + c)$.

As the construction must have the same almost horizontal/vertical crossing number, we must include new extra cr-columns or cr-rows so that their number is indeed equal. Therefore, the width and height of the construction is the maximum of the two values obtained for these parameters, so it is polynomial in c, n and t. □

4.6 Iterated Rounding

Following our proofs, we know that all variants triangulation with small stabbing or crossing number are $\mathcal{NP}$-hard, making it unlikely that there is a polynomial-time algorithm that can handle them. In the following, we study a heuristic approach for computing solutions in polynomial time, with the hope that the resulting objective values are within a constant factor of the optimal values.

An iterated rounding algorithm (**IRA**), as described in [13], proceeds by solving the linear relaxation of a given problem, finding a variable with high fractional value, fixing this variable to 1 and repeating the process until an integral solution is found. In [9] it was conjectured that an **IRA** yields an approximation algorithm for the problem of finding a perfect matching with minimum stabbing number (MSPM). That conjecture regarding the worst-case performance is still open; in any case, such an algorithm provides a heuristic for the MSPM.

As the MSTR is closely related to the MSPM, we may consider if the same **IRA** approach applied to the MSTR yields an approximation for our problem. As discussed in [9], one of the prerequisites for obtaining an approximation using an **IRA** is a guarantee that there is a “heavy” variable at each iteration, i.e., a variable with high fractional value: If we can guarantee the existence of a variable with value at least $1/k$ at each iteration, the hope is to get a k-approximation.

Different from the MSPM case, no proof is known for the MSTR that a heavy variable exists at each iteration. However, we can provide evidence for the existence by experimentally determining the smallest value of all heaviest variables in all instances, say $1/p$. After that, using the lower and upper bounds obtained by the **IRA** and results from other algorithms, we can check if the results are consistent with a p-approximation algorithm.

The integer programming model used in the algorithm is the triangle-based model described in [16] and reproduced below. Here $\Delta(P)$ denotes the set of empty triangles over a set of points P, $L^+(ij)$ and $L^-(ij)$ represent the two half-planes defined by the line containing (ij), while E_H is the set of line segments in the convex hull of P.

$$
\begin{aligned}
(MSTT) \quad & z = \min k && && (4.1)\\
\text{subject to} \quad & \sum_{\substack{ijl \in \Delta(P)\,:\\ ijl \subset L^{+}(ij)}} x_{ijl} = \sum_{\substack{ijl \in \Delta(P)\,:\\ ijl \subset L^{-}(ij)}} x_{ijl}, && \forall ij \in E \setminus E_H, && (4.2)\\
& \sum_{ijl \in \Delta(P)} x_{ijl} = 1, && \forall\, ij \in E_H, && (4.3)\\
& \sum_{ijl \in \Delta(P): ijl \bigcap s \neq \emptyset} c^{s}_{ijl} x_{ijl} \leq k, && \forall\, s \in S. && (4.4)\\
& k \in \mathbb{Z}, x_{ijl} \in \mathbb{B} && \forall\, ijl \in \Delta(P). && (4.5)
\end{aligned}
$$

In the model above, for every triangle $ijl \in \Delta(P)$, $x_{ijl} = 1$ if and only if the triangle ijl is in the triangulation. The variable k represents the stabbing number of the triangulation. Constraint (4.2) states that the number of triangles containing an edge ij (which is not in E_H) must be the same in both half-planes defined by the line containing ij. As the edges in E_H are present in every planar triangulation, constraint (4.3) ensures that a triangle containing one such edge is in the triangulation. Constraint (4.4) states that the sum of the coefficients c^{s}_{ijl} of the triangles ijl intersecting a line s of S cannot be larger than the stabbing number. A triangle ijl intersecting a line s has coefficient $c^{s}_{ijl} = \beta^{s}_{ij} + \beta^{s}_{il} + \beta^{s}_{jl}$, where $\beta^{s}_{ij} = 1$ if ij intersects s and is on the convex hull, $\beta^{s}_{ij} = 0.5$ if ij intersects s, but is not on the convex hull and $\beta^{s}_{ij} = 0$ if ij does not intersect s.

The experiments described in the following consider only the axis-parallel version of the MSTR, because this allows a comparison with previous computational results described by [16]. We focus on the instances described in that paper.

The heuristic method developed in [16] is based on Lagrangian Relaxation (LR). For fair comparison, we used the same computational environment for both.

Computational environment. We used a computer with an Intel Core 2 Quad 1.60GHz, 4096 KB cache, 4GB of RAM memory and an Ubuntu 10.04.4 OS. The programming language used was `C/C++` with `gcc` 4.4.3 compiler. Every program was compiled with the `-O5` optimization flag. We also used the `XPRESS-Optimizer 64-bit v22.01.09` IP solver. The default cuts, heuristics and preprocessing were turned off. In addition, the optimizer was set to use a single processor core.

Instances. As a test suite we used 25 instances from TSPLIB [17, 18] and the 25 regular grid instances used in [9] for the Minimum Stabbing Perfect Matching Problem. The choice of these instances is based on the fact that the TSPLIB is a well-known test library for geometric problems. Moreover, TSPLIB and all grid instances were also used in [9] for the MSPM. The choice of the instance sizes was made seeking tests that were hard enough to provide meaningful computation times, allowing a more precise comparison of the algorithms.

A time limit of 3, 600 seconds was set for the execution of the algorithms. As the time is verified at specific checkpoints in the code and the time spent between two such checkpoints may not be negligible, some of the times displayed in the tables are slightly over 3, 600 seconds.

Duality gaps were computed through the formula $100 \times (ub - \lceil lb \rceil)/ub$, where ub and lb denote, respectively, the upper and lower bounds yielded by the algorithm. Whenever a value is unkown because the algorithm was interrupted, the respective value is marked with the symbol ‡ in the table.

Results. The first observation is that the smallest large fractional value of all instances (that produced an output) is greater than or equal to 0.5. This means that at every iteration of the **IRA** and for every instance, there was always a variable with value at least 0.5. According to the approximation conjecture, that should give us a 2-approximation algorithm. The results obtained are consistent with this hypothesis, because no upper bound value is more than twice a known lower bound value (including the cases in which the upper bounds coincided with the optimal value).

In the following we divide our analysis into two parts, one for the TSP instances and the other for the grid instances.

For the TSP instances, the **IRA** had its process killed in 8 out of the 25 instances, while optimality was achieved in 10 cases. The remaining tests resulted in gaps of only 3.87% on average. The Lagrangian algorithm converged in all cases within the imposed time limit, with an average gap of 2.30% and proven optimality in 7 cases.

Considering only the instances for which **IRA** was not killed, the LR algorithm was faster in 11 situations while the **IRA** was faster in 6 cases. The total time spent with these instances was 108.77 seconds bigger with the **IRA**. These results are shown in Table 4.1.

For the grid instances, 23 out of 25 instances were solved to optimality by the **IRA**, while the LR algorithm solved all problems to optimality. The total running times for the instances solved by both of them was practically identical, except for the grid20 instances, for which the LR was significantly faster.

4.7 Conclusions

We have resolved a number of long-standing open problems on the problem of finding triangulations of small stabbing or crossing numbers, by proving them to be $\mathcal{NP}$-hard.

Naturally, this raises the need for the development of constant-factor approximation algorithms. We have supplied experimental evidence that an approach based on iterated rounding may be able to provide such an approximation algorithm. In particular, we were able to show that the performance is comparable to the best known heuristic based on Lagrangian relaxation, with no instance yielding an optimality gap larger than 6%.

Conjecture 1. *Iterated rounding provides a constant-factor approximation algorithm for* MSTR.

Table 4.1: Comparison of **IRA** and LR algorithm with TSP instances.

Instance	**min. Var.**	**# iters**	**LB**		**UB**		**Time**		**GAP%**	
	IRA	**IRA**	LR	**IRA**	LR	**IRA**	LR	**IRA**	LR	**IRA**
berlin52	0.50	34	23.670	23.700	24	24	7.70	2.25	0.00	0.00
eil76	0.50	54	31.561	31.564	33	33	112.58	21.19	3.03	3.03
kroD100	0.50	95	28.002	28.043	29	29	30.60	220.92	0.00	0.00
kroA100	0.50	100	28.518	28.529	30	30	107.25	205.42	3.33	3.33
kroE100	0.50	89	28.221	28.220	30	29	99.17	199.91	3.33	0.00
kroC100	0.50	83	28.123	28.141	29	29	96.56	186.51	0.00	0.00
kroB100	0.50	98	28.593	28.599	30	30	119.63	239.20	3.33	3.33
rd100	0.50	83	28.050	28.165	29	29	17.45	213.96	0.00	0.00
pr124	0.50	40	47.612	48.122	49	52	406.34	229.14	2.04	5.77
pr136	0.67	9	65.667	65.667	67	66	589.72	67.24	1.49	0.00
ch130	0.50	132	31.904	31.920	33	34	165.06	1015.82	3.03	5.88
pr144	0.50	13	73.084	74.000	74	74	673.28	187.63	0.00	0.00
pr152	0.50	55	44.012	45.000	45	45	420.05	795.93	0.00	0.00
kroA150	0.50	131	34.411	34.405	36	35	333.77	1525.61	2.78	0.00
kroB150	0.67	163	33.632	33.645	35	35	412.90	2153.08	2.86	2.86
ch150	0.67	163	33.292	33.307	35	35	272.60	2034.03	2.86	2.86
kroB200	‡	1	38.285	37.868	40	‡	705.74	3607.45	2.50	‡
kroA200	‡	1	39.578	39.246	41	‡	737.41	3607.57	2.44	‡
gr202	‡	1	41.059	39.004	42	‡	614.27	3607.65	0.00	‡
pr226	0.50	56	144.239	150.000	150	150	3690.80	3005.09	3.33	0.00
pr264	‡	1	89.761	91.000	92	‡	3600.70	3609.20	2.17	‡
gil262	‡	1	48.819	34.272	50	‡	1769.88	3611.15	2.00	‡
lin318	‡	1	68.538	49.000	70	‡	3602.31	3619.89	1.43	‡
pcb442	‡	1	161.246	147.000	180	‡	6017.10	3665.40	10.00	‡
rd400	‡	1	51.848	13.925	55	‡	3604.68	3656.82	5.45	‡

Table 4.2: Comparison of **IRA** and LR for grid instances.

Instance	**min. Var.**	**# iters**	**LB**		**UB**		**Time**		**GAP%**	
	IRA	**IRA**	LR	**IRA**	LR	**IRA**	LR	**IRA**	LR	**IRA**
grid5a	1.00	0	21.432	22.000	22	22	0.17	0.07	0.00	0.00
grid5b	0.50	1	20.029	20.500	21	21	0.27	0.07	0.00	0.00
grid5c	1.00	0	20.031	21.000	21	21	0.17	0.08	0.00	0.00
grid5d	1.00	0	21.000	21.000	21	21	23.14	0.07	0.00	0.00
grid5e	0.50	1	19.054	20.000	20	20	0.18	0.07	0.00	0.00
grid8a	0.50	4	33.004	34.000	34	34	2.2	0.16	0.00	0.00
grid8b	0.80	1	33.275	34.000	34	34	3.48	0.23	0.00	0.00
grid8c	1.00	0	33.038	34.000	34	34	1.61	0.19	0.00	0.00
grid8d	1.00	0	34.009	35.000	35	35	1.07	0.2	0.00	0.00
grid8e	0.50	3	34.071	34.500	35	35	1.11	0.24	0.00	0.00
grid10a	1.00	0	43.123	44.000	44	44	8.01	1.02	0.00	0.00
grid10b	1.00	0	41.764	42.000	42	42	3.31	0.62	0.00	0.00
grid10c	0.50	3	46.023	47.000	47	47	9.52	0.96	0.00	0.00
grid10d	1.00	0	45.002	46.000	46	46	2.61	0.82	0.00	0.00
grid10e	1.00	0	45.003	46.000	46	46	7.05	1.05	0.00	0.00
grid15a	0.67	2	65.166	66.000	66	66	75.13	52.3	0.00	0.00
grid15b	0.50	3	67.153	68.000	68	68	13.65	55.75	0.00	0.00
grid15c	0.50	8	63.043	64.000	64	64	20.7	46.53	0.00	0.00
grid15d	0.67	15	65.071	65.200	66	66	39.24	51.54	0.00	0.00
grid15e	0.80	3	66.081	67.000	67	67	79.53	60.51	0.00	0.00
grid20a	0.50	17	88.020	89.000	89	89	500.78	2357.88	0.00	0.00
grid20b	‡	1	85.174	85.000	86	‡	73.09	3615.74	0.00	‡
grid20c	0.50	13	89.016	90.000	90	90	2222.62	2517.32	1.11	0.00
grid20d	‡	1	86.112	87.000	87	‡	204.77	3616.62	0.00	‡
grid20e	0.50	13	89.078	90.000	90	90	1213.83	2015.84	0.00	0.00

Given that there is a variety of different IP formulations, and thus different LP relaxations for our problems, the actual worst-case performance may depend on a specific IP version. Given that the time for solving the involved linear programs grows very rapidly with instance size, studying different formulations is also of practical importance.

Bibliography

[1] P. K. Agarwal. Ray shooting and other applications of spanning trees with low stabbing number. *SIAM J. Computing*, 21(3):540–570, 1992.

[2] P. K. Agarwal, B. Aronov, and S. Suri. Stabbing triangulations by lines in 3D. In *Proceedings of the 11th ACM Symposium on Computational Geometry*, pages 267–276, 1995.

[3] O. Aichholzer, F. Aurenhammer, E. Demaine, F. Hurtado, P. Ramos, and J. Urrutia. On k-convex polygons. *Computational Geometry*, 45(3):73–87, 2012.

[4] O. Aichholzer, F. Aurenhammer, T. Hackl, F. Hurtado, A. Pilz, P. Ramos, J. Urrutia, P. Valtr, and B. Vogtenhuber. On k-convex point sets. *Computational Geometry*, 47(8):809–832, 2014.

[5] B. Aronov, H. Brönnimann, A. Y. Chang, and Y.-J. Chiang. Cost-driven octree construction schemes: an experimental study. *Computational Geometry: Theory and Applications*, 31:127–148, 2005.

[6] B. Aronov and S. Fortune. Approximating minimum-weight triangulations in three dimensions. *Discrete & Computational Geometry*, 21(4):527–549, 1999.

[7] M. de Berg and M. van Kreveld. Rectilinear decompositions with low stabbing number. *Inform. Process. Lett.*, 52(4):215–221, 1994.

[8] E. Demaine, J. Mitchell, and J. O'Rourke. The open problems project. Available online (accessed January 2010). `http://maven.smith.edu/∼orourke/TOPP/`.

[9] S. Fekete, M. Lübbecke, and H. Meijer. Minimizing the stabbing number of matchings, trees, and triangulations. In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '04, pages 437–446, Philadelphia, PA, USA, 2004. Society for Industrial and Applied Mathematics.

[10] S. Fekete, M. Lübbecke, and H. Meijer. Minimizing the stabbing number of matchings, trees, and triangulations. *Discrete & Computational Geometry*, 40(4):595–621, 2008.

[11] M. Held, J. T. Klosowski, and J. S. B. Mitchell. Evaluation of collision detection methods for virtual reality fly-throughs. In *Proceedings of the 7th Canadian Conference on Computational Geometry*, pages 205–210, 1995.

[12] J. Hershberger and S. Suri. A pedestrian approach to ray shooting: Shoot a ray, take a walk. *J. Algorithms*, 18:403–431, 1995.

[13] K. Jain. A factor 2 approximation algorithm for the generalized steiner network problem. *Combinatorica*, 21(1):39–60, 2001.

[14] J. Matoušek. Spanning trees with low crossing number. *Theoretical Informatics and Applications.*, 25:102–123, 1991.

[15] J. S. B. Mitchell and J. O'Rourke. Computational geometry column 42. *International Journal of Computational Geometry & Applications*, 11(5):573–582, 2001.

[16] B. Piva, C. de Souza, Y. Frota, and L. Simonetti. Integer programming approaches for minimum stabbing problems. *RAIRO - Operations Research*, 48:211–233, 4 2014.

[17] G. Reinelt. TSPLIB. Available online (accessed March 2011). `http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/`.

[18] G. Reinelt. TSPLIB – A traveling salesman problem library. *ORSA Journal on Computing*, 3(4):376–384, 1991.

[19] J. R. Shewchuk. Stabbing Delaunay tetrahedralizations. *Discrete & Computational Geometry*, 32(3):339–343, 2004.

[20] C. Tóth. Orthogonal subdivisions with low stabbing numbers. In *Proceedings of the 9th International Workshop on Algorithms and Data Structures (WADS 2005)*, volume 3608 of *Springer LNCS*, pages 256–268, 2005.

Chapter 5

Minimum Stabbing Rectangular Partitions of Rectilinear Polygons

The current chapter presents the complete version of a work presented at VIII Latin-American Algorithms, Graphs and Optimization Symposium (LAGOS 2015) as an extended abstract co-authored with Cid C. de Souza [10]. This is the text of the extended version of that work and that was submitted for publication to a scientific journal. This work studied rectangular partitions of rectilinear polygons with minimum stabbing number, presenting two integer programming formulations for the problem including a polyhedral study for one of them. Computational experiments were performed to compare the different formulations.

We study integer programming (IP) models for the problem of finding a rectangular partition of a rectilinear polygon with minimum stabbing number. Strong valid inequalities are introduced for an existing formulation and a new model is proposed. We compare the dual bounds yielded by the relaxations of the two models and prove that the new one is stronger than the old one. Computational experiments with the problem are reported for the first time in which polygons with thousands of vertices are solved to optimality. The (IP) branch-and-bound algorithm based on the new model is faster and more robust than those relying on the previous formulation.

5.1 Introduction

Let P be a rectilinear polygon, and π be a rectangular partition of P, i.e., a partition of the interior of P, $\text{int}(P)$, into rectangles. Define the set L of all maximal line segments that are axis-parallel and belong to $\text{int}(P)$. Given a segment s of L, the stabbing number of s relative to π is the number of rectangles of the partition whose interior is intersected by s. The stabbing number of π is then the maximum stabbing number among all lines in L. The Rectilinear Partition with Minimum Stabbing Number Problem (RPST) is: given a rectilinear simple polygon, find a rectangular partition having minimum stabbing number among all possible partitions. Figure 5.1 shows an RPST instance and a possible rectangular partition. If an edge e of a rectangle in a rectangular partition of P has both of its endpoints on the boundary of P, $\delta(P)$, e is said to be *fully anchored.* A rectangular partition of P is called *conforming* if all edges of its rectangles are fully anchored.

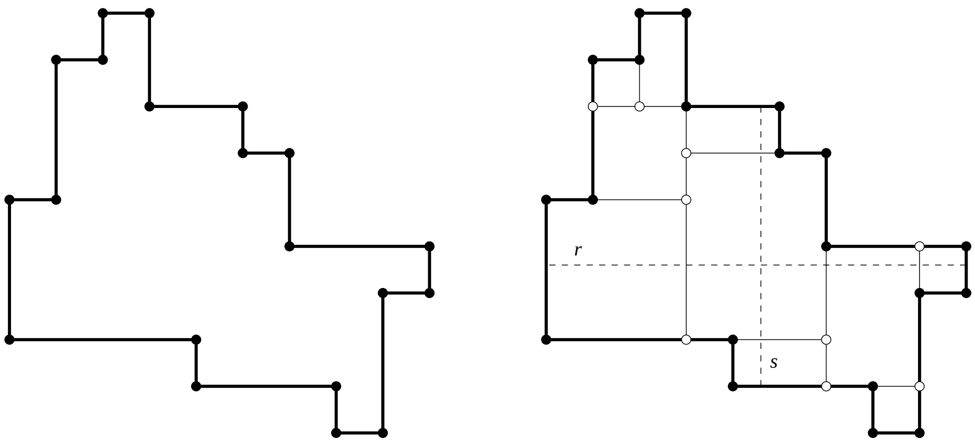

Figure 5.1: An instance of RPST (to the left) and a feasible solution (to the right). The segments r and s have stabbing numbers, respectively, 4 and 3.

Problems requiring the decomposition of rectilinear polygons have applications, for example, in VLSI layout design and image processing (cf. [7]). On the other hand, obtaining sets of objects satisfying some properties and having the lowest stabbing number is a recurring problem in Computational Geometry. In [11] a wide variety of applications of that sort are mentioned including the design of efficient algorithms for simplex range searching, ray shooting, motion planning and collision detection among others. Clearly, the RPST merges these two types of problems and, that is probably why it attracted the attention of many researchers.

The RPST was studied in [5], [1] and [7]. In [5], the authors show that any rectilinear polygon with n vertices have a rectangular partition with stabbing number $O(\log n)$ for a hole-free polygon and $O(\sqrt{k}\log n)$ for a polygon with $k \geq 1$ rectilinear holes. Abam et al. [1] present a 3-approximation polynomial time algorithm for the problem, based on the partition of histograms. Finally, Durocher and Mehrabi [7] prove that the problem of finding a conforming rectangular partition in a polygon with holes is $\mathcal{NP}$-hard. They also present an integer programming (IP) formulation for the problem and develop a 2-approximation algorithm for the conforming case.

Our Contribution The first contribution of this work is a polyhedral investigation on the model proposed in [7]. There the authors did not investigated the strength of their formulation nor carried out any computational experiments with it. The inequalities obtained in our polyhedral study were used tested computationally. These experiments show that the new inequalities allow us to solve more instances to optimality in a reasonable time.

A key aspect of our work is the establishment of a relation between the RPST and the Minimum Length Rectangular Partition (RGP) previously studied in [6] and [3]. In the RGP, we are given a rectangle R and a set T of points in its interior, called *terminals.* The goal is to use axis-parallel segments to partition R into rectangles so that every terminal is intersected by at least one of these segments and the sum of their lengths is minimized. For given R and T, a feasible solution for the RGP is called a *rectangular partition of R constrained by T*. Figure 5.2 depicts an instance of RGP and a feasible solution.

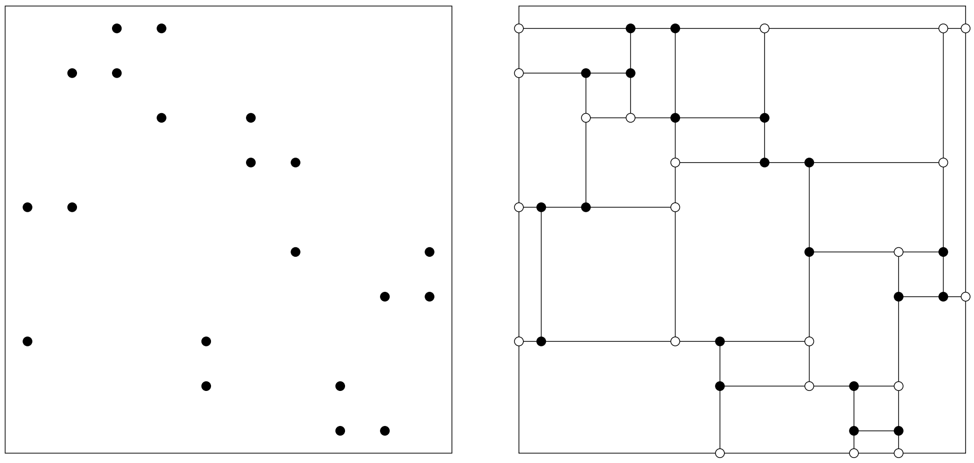
Figure 5.2: An instance of RGP (to the left) and a feasible solution (to the right). The black points indicate the terminals.

Another contribution of this work is the specification of a new IP formulation for RPST. The new model describes the problem through variables that indicate if a rectangle (instead of a segment) is in the solution. For reasons that will become clear later, we call it the *set partition model.* This formulation is then proved to be stronger than the one given in [7].

We further investigate the set partition model and establish conditions for fixing some variables of the IP formulation in an optimal solutions, reducing its quantity. These properties allow to eliminate variables. Because this model is a restriction on the original set partition formulation, it is no weaker than that model. This new formulation led to the best running times for large instances and the results suggest that as the polygon size increases, becomes not only faster than the competitors but also more robust.

Organization of the text The paper is organized as follows. The next section describes IP models for the RPST and the RGP where the variables are related to segments of the rectangular partition. Section 5.3 shows the relation between these models. In Section 5.4, we show how the RPST can be modeled as a set partition problem and some properties of this model, while experiments are discussed in Section 5.5. Finally, Section 5.6 presents some conclusions and directions for future work.

5.2 Segment Based IP Models

In this section we present an IP model for the RPST. In this model the variables are related to segments of the rectangular partition. Later we show a formulation for the RGP that is closely related to the one for RPST.

Given a rectilinear polygon P, input of RPST, let V_r^P (V_c^P) be the set of *reflex (convex) vertices* of P. The grid induced by P, grid(P), is the set of all vertical and horizontal maximal line segments in the interior of P having (at least) one vertex in V_r^P as one of its endpoints. Let V_s^P be the set of points of int(P) located at the intersections of two line segments in grid(P), which are called *Steiner vertices.* The points in $\delta(P)$ that are endpoints of segments in grid(P) and are not in V_r^P are called *border vertices* and form the set V_b^P. The set of all vertices is defined by $V^P = V_c^P \cup V_r^P \cup V_s^P \cup V_b^P$. We now turn our attention to the edge set.

Suppose we traverse $\delta(P)$, the boundary of P, say, clockwisely. The segments between two consecutive vertices of V^P form the set E_h^P. Now, if we traverse any horizontal (vertical) segment of grid(P) from left to right (from bottom to up), the segments between two consecutive vertices of V^P form the set E_g^P. These are called the *grid segments* and, together with the segments in E_h^P, they compose the set E^P, i.e., $E^P = E_h^P \cup E_g^P$. A *canonical rectangle* in grid(P) is a rectangle where each side is a unique segment of E^P. Figure 5.3 depicts the grid for the example in Figure 5.1. Steiner and border vertices are represented by gray and white vertices, respectively. From the formulation in [7] one can deduce that there exists an optimal solution to RPST such that all rectangles in the partition have sides lying on grid(P).

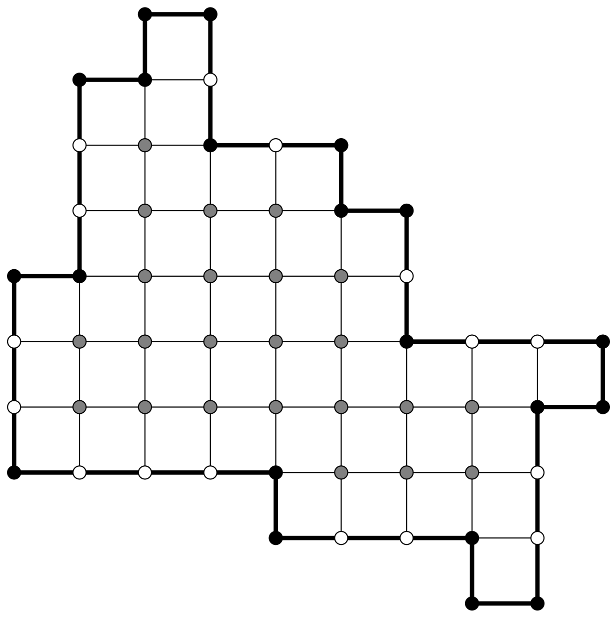

Figure 5.3: Grid for the example in Figure 5.1 containing 40 canonical rectangles.

Two configurations are relevant for the description of a feasible RPST solution. A subset E'^P of E^P defines a *knee* in a vertex $u \in V^P$ if there are exactly two edges in E'^P incident to u and they are orthogonal. On the other hand, if only one edge in E'^P is incident to u, we say that E'^P defines an *island* at u. Clearly, if E'^P induces a rectangular partition of P, it can not define a knee or an island at any point.

Now, denote by $\theta(ua, ub)$ the angle between two edges ua and ub in E^P that are incident to a point $u \in V^P$. With these definitions, the RPST can be modeled as [7]:

$$(M^{RPST}_{sum}) \qquad z = \min \max_{s \in L} \sum_{\substack{uv \,\in\, E^P_g \,:\\ uv \bigcap s \,\neq\, \emptyset}} x_{uv} + 1 \tag{5.1}$$

$$\text{s.t.} \qquad x_{ua} + x_{ub} \geq 1, \qquad \forall\ u \in V^P_r, ua, ub \in E^P_g, \tag{5.2}$$

$$x_{ua} + x_{ub} \geq x_{uc}, \qquad \forall\ u \in V^P_s, ua, ub, uc \in E^P_g : \theta(ua, ub) = \frac{\pi}{2}, \tag{5.3}$$

where the binary variable x_{uv} is set to one if and only the edge $uv \in E^P_g$ belongs to the solution. The set L comprises all horizontal and vertical maximal line segments fully contained in P. Thus, the objective function minimizes the maximum of a set of $|L|$ sums, each corresponding to the stabbing number of a segment. Notice that, in principle, L is infinite. However, as stated in [7], for every $w \in V^P_r$, we only need to consider the two axis-parallel segments containing a point along the bisector of the internal angle in w. This point is chosen so that its distance from w is smaller than the distance between any two vertices. By doing that, we have $|L| = 2|V^P_r|$ and the model size becomes polynomial in the size of P.

Inequalities (5.2) guarantee that a solution does not define a knee or island in a reflex vertex. Meanwhile, inequalities (5.3) enforce that a solution can not contain a knee or an island in a Steiner vertex. Durocher and Mehrabi [7] argue the correctness of the formulation with these two sets of constraints. They also mention that there exists an optimal solution where at most three grid segments meeting at a Steiner vertex are present. This property is expressed by the linear inequalities

$$\sum_{uv \in E^P_g} x_{uv} \leq 3,\ \forall\ u \in V^P_s. \tag{5.4}$$

Due to the objective function, the model M^{RPST}_{sum} is not linear. Using standard techniques, it can be linearized through the introduction of an auxiliary integer variable k to represent the stabbing number. For each element s of L, we add a constraint requiring that k is at least as large as the summation corresponding to s in (5.1). With the x variables defined as before, the new model reads:

$$(M^{RPST}) \min \left\{ k \in \mathbb{R} : x \in \mathbb{B}^{E^P_g}, (5.2) - (5.3), \sum_{\substack{uv \in E^P_g \\ uv \bigcap s \neq \emptyset}} x_{uv} + 1 \leq k, \forall s \in L \right\}. \tag{5.5}$$

This model is similar to those discussed in [8] for other stabbing problems.

As stated before, the RPST model is closely related to a RGP model. Their relationship will become clearer in Section 5.3. For now, we restrict ourselves to present an IP formulation for the RGP. Prior to that, some more notation is necessary.

Given an instance $I = (R, T)$ of the RGP, where R is a rectangle and T is a set of *terminal points* in R, let grid(R, T) be the set of vertical and horizontal maximal line segments in the interior of R intersecting a point of T. Let T_s be the set of points in the intersection of segments in grid(R, T) but not in T. Let T_b be the set of points on the boundary of R intersected by segments in grid(R, T) and let $T_t = T \cup T_s \cup T_b$. Define S to be the set of fractions of segments in grid(R, T) containing exactly two points in T_t, both located at its extremities. The elements of S are referred to as *grid segments*. The set of all grid segments induces a planar subdivision of the surface of R. Each inner face of this subdivision is called a *canonical rectangle* of grid(R, T).

As in the RPST case, some properties of feasible and optimal solutions of the RGP are useful to model the problem as an IP. First, the knee and an island configurations are defined as before, but this time for points in T_t and segments in S. Both formations are obviously forbidden in any feasible solution of the RGP. Besides, in [9] it was stated that there is always an optimal solution for the RGP whose rectangles have sides lying on grid(R, T).

From these definitions and recalling that $\theta(ua, ub)$ is the angle between segments ua and ub, we obtain the following model for the RGP:

$$
\begin{aligned}
(M^{RGP}) \qquad & z = \min \sum_{uv \in S} d_{uv} x_{uv} && && (5.6)\\
\text{s.t.} \qquad & x_{ua} + x_{ub} \geq 1, && \forall\ u \in T, ua, ub \in S : \theta(ua, ub) = \frac{\pi}{2} && (5.7)\\
& x_{ua} + x_{ub} \geq x_{uc}, && \forall\ u \in T_s, ua, ub, uc \in S : \theta(ua, ub) = \frac{\pi}{2}, && (5.8)
\end{aligned}
$$

where, for every $uv \in S$, the binary variable x_{uv} is set to one if and only if the segment uv is in the solution. The objective function is given by the sum of the lengths of the segments that belong to the solution. Inequalities (5.7) and (5.8) guarantee that the solution does not define knees and islands in points in T or T_s, respectively. Meneses and de Souza [6] showed that the latter constraints describe all feasible rectangular partitions. Constraints (5.9) below enforce that at most three of the four grid segments incident to a Steiner vertex can be in the solution:

$$\sum_{uv \in S} x_{uv} \leq 3,\ \forall\ u \in T_s. \qquad (5.9)$$

This property also holds for optimal RGP solutions, so the addition of these constraints to the model causes no harm, while it may be quite helpful in computation.

Looking at models (M^{RGP}) and (M^{RPST}) it is possible to see that although the problems statements are rather different, their formulations have several similarities. In the next section we establish the relationship between the polyhedra defined by these models.

5.3 Polyhedral Study of the Segment Based Model

In this section we show how the IP models given before are related. The goal is to utilize previous findings about the M^{RGP} to improve the models for the RPST. To facilitate the understanding on how this is done, we first give some basic results on the projection of polyhedra and then explain how an RPST instance can be transformed into an RGP instance. We finally combine these ideas to derive facet defining inequalities for the M^{RPST}.

5.3.1 Projection of Polyhedra

We briefly review some relevant findings of Balas and Oosten [2] relative to the projection of polyhedra. Consider a non empty polyhedron $Q = \{(u, y) \in \mathbb{R}^p \times \mathbb{R}^q : Au + By \leq b\}$, where A, B and b have m rows. The projection of Q onto the subspace defined by $u = 0$, called the y-space, is defined as

$$Proj_y(Q) = \{y \in \mathbb{R}^q : \exists u \in \mathbb{R}^p \text{ with } (u, y) \in Q\}.$$

Let us partition the rows of (A, B, b) into $(A^=, B^=, b^=)$ and $(A^<, B^<, b^<)$, where $A^=u+B^=y = b^=$ is the equality subsystem of Q, i.e. the set of equations corresponding to the inequalities satisfied at equality by every $(u, y) \in Q$. Assume that the *equality subsystem* has no redundant rows and that no equality is implied by the *inequality subsystem*. Let $r = rank(A^=, B^=) = rank(A^=, B^=, b^=)$, where the last equality follows from $Q \neq \emptyset$. Moreover, let $\dim(X)$ denote the dimension of a set X. It is well known that $\dim(Q) = p + q - r$, and that Q is full-dimensional, i.e. $\dim(Q) = p + q$, if and only if the equality subsystem is vacuous. The first results states that, if Q is full-dimensional so is its projection onto the y-space.

Proposition 5.1 ([2], Prop. 2.1). *If* $\dim(Q) = p + q$, *then* $\dim(Proj_y(Q)) = q$.

The next result establishes necessary and sufficient conditions for an inequality defining a facet of Q to define a facet of $Proj_y(Q)$. Let $\alpha u + \beta y \leq \pi_0$ be a valid inequality for Q, and suppose $F = \{(u, y) \in Q : \alpha u+\beta y = \alpha_0\}$ is a facet of Q. Let $\begin{pmatrix} \alpha \\ A^= \end{pmatrix} u + \begin{pmatrix} \beta \\ B^= \end{pmatrix} y = \begin{pmatrix} \alpha_0 \\ b^= \end{pmatrix}$ be the equality subsystem defining the polyhedron F and, let $r_F = rank\left(\begin{pmatrix} \alpha \\ A^= \end{pmatrix}, \begin{pmatrix} \beta \\ B^= \end{pmatrix}\right)$. Notice that $r_F - r = 1$, since $\dim(F) = \dim(Q) - 1$. Further, denote $r_F^* = rank\left(\begin{pmatrix} \alpha \\ A^= \end{pmatrix}\right)$ and $r^* = rank(A^=)$. The next statement relates the facets of Q and those of $Proj_y(Q)$.

Proposition 5.2 ([[2], Cor. 3.6). *Let F be a facet of Q. Then $Proj_y(F)$ is a facet of $Proj_y(Q)$ if and only $r_F^* = r^*$.*

5.3.2 Transforming RPST into RGP

We now explain how to transform an instance of the RPST into an instance of the RGP. To this, we start with the following definition.

For a given set of points S in the plane, let $x_{\min}$ ($x_{\max}$) be the minimum (maximum) x-coordinate of a point in S. Define the values of $y_{\min}$ and $y_{\max}$ analogously. The *enlarged bounding box* of S is the rectangle with vertices at $(x_{\min} - 1, y_{\min} - 1)$ and $(x_{\max} + 1, y_{\max} + 1)$ and sides parallel to the axes.

Now, given the rectilinear polygon P in the RPST instance, define the external rectangle R in the RGP instance as the *enlarged bounding box* of P. Also, in the later, include in the set T of *terminal points* all the vertices of P. Clearly, any rectangular partition π of P can be extended to a rectangular partition of R with terminals in T. It suffices to add to P all the segments in grid(R, T) that are not in int(P). On the other hand, let ϕ be a rectangular partition of R constrained to T. Consider the set S of grid segments of ϕ which are in int(P). We claim that the subdivision induced by S in P is a feasible solution for the RPST. If not, at least one of the faces of the subdivision defined by S in int(P), say f, is not a rectangle. So, f has a reflex vertex u that is also a vertex of P since, otherwise, ϕ would form a knee at some point of grid(R, T) in int(P), and consequently would not be feasible for the RGP. However, as f is the intersection of some rectangle R' induced by ϕ and P, this implies that u is in the interior of this rectangle. But, as u is a terminal, ϕ could not be a solution of the RGP either.

5.3.3 Polyhedral results for the RPST

Given the RPST and the transformed RGP instance described above, denote by Q the convex hull of feasible solutions of M^{RGP} called the RGP polytope. Similarly, let Q^x be the RPST polytope given by the convex hull of the integer solutions of the linear system (5.2)-(5.3). Let $s = p + q$ be the total number of grid segments in the RGP instance, such that $Q \subset \mathbb{R}^s$. In the sequel, for a vector $w \in \mathbb{R}^s$, assume that the first p components correspond to the segments of grid(R,T) that are not in int(P) and the last q elements are associated to the remaining grid segments. Denote the first p (last q) components of w by u (y). Suppose that $Q = \{(u,y) \in \mathbb{R}^p \times \mathbb{R}^q : Au + By \leq b\} \neq \emptyset$ and notice that $Q^x \subset \mathbb{R}^q$. From the previous subsection, it is clear that $Q^x = \{y \in \mathbb{R}^q : \exists u \in \mathbb{R}^p : (u,y) \in Q\} = Proj_y(Q)$, i.e. Q^x is the orthogonal projection of Q onto $\mathbb{R}^q$. Since the Q was proven to be full-dimensional in [6], the results from Section 5.3.1 can be used to find the dimension of Q^x.

Proposition 5.3. *The polytope Q^x is full dimensional, i.e.,* $\dim(Q^x) = q$.

Proof. Immediate from Proposition 5.1. □

Besides, known facet defining inequalities for Q can also be facet defining for Q^x. The next proposition gives necessary conditions for this to hold.

Proposition 5.4. *Let $\pi w = \alpha u + \beta y \leq \alpha_0$ be a facet defining inequality for Q for which α is the null vector and $F = \{(u,y) \in Q : \alpha u + \beta y = \alpha_0\}$. Then, for $y \in \mathbb{R}^q$, $\beta y \leq \alpha_0$ is facet defining for Q^x.*

Proof. From the definition of Q, let $(A^=, B^=, b^=), (A^{\leq}, B^{\leq}, b^{\leq})$ be a partition of (A, B, b) where $(A^=, B^=, b^=)$ is the equality subsystem of Q, let $r^* = rank(A^=)$ and $r^*_F = rank\binom{\alpha}{A^=}$. Since Q is full dimensional, $A^=$ is empty and $r^* = 0$. Moreover, since $\alpha = 0$, $r^*_F = 0$. Then, the result follows from Proposition 5.2. □

Now, let Q^x_k be the convex hull of the feasible solutions of M^{RPST}, i.e., the linearized model of the RPST with the stabbing variable k given by (5.5). Renaming the x variables in this model by y, it is easy to see that $Proj_y(Q^x_k) = Q^x$. Notice that if $\{y^1, y^2, \dots, y^r\}$ is an affinely independent set of vectors of Q^x representing r rectangular partitions of P and $k_{\max}$ is the largest stabbing number among these partitions, the $r+1$ vectors $\left\{\binom{y^1}{k_{\max}}, \binom{y^2}{k_{\max}}, \dots, \binom{y^r}{k_{\max}}, \binom{y^1}{k_{\max}+1}\right\}$ belong to Q^x_k and are affinely independent. As a consequence, Q^x_k is full-dimensional and any facet defining inequality of Q^x also defines a facet of Q^x_k.

Consider then a facet defining inequality for the Q whose support vector does not contain elements associated to segments that are not in int(P). From the results seen in this section, this inequality also defines a facet of Q^x and of Q^x_k. Next we see how to use this idea to tighten the M^{RPST} model.

We begin describing three families of inequalities proposed in [6] that are facet-defining for Q and which satisfy the conditions of Proposition 5.4. These inequalities are characterized by geometric configurations related to the location of terminal and Steiner vertices in grid(R,T). The configurations of interest are shown in Figure 5.4 and correspond to the so-called Classes III, IV and VI of inequalities, as defined by Meneses and de Souza in their paper. The form of the constraints in Classes III, IV and VI are given in equations (5.10), (5.11) and (5.12), respectively.

$$x_{e1} + x_{e2} + x_{e3} + x_{e4} \geq 2 \tag{5.10}$$

$$x_{e1} + x_{e2} + x_{e3} + x_{e4} \geq 1 \tag{5.11}$$

$$x_{e1} + x_{e2} + x_{e3} + x_{e4} + x_{e5} + x_{e6} + x_{e7} + x_{e8} \geq 2 \tag{5.12}$$

Notice that, as inequalities (5.7) and (5.8) define facets for Q [6], from Proposition 5.4, their counterparts, inequalities (5.2) and (5.3), also define facets for Q^x.

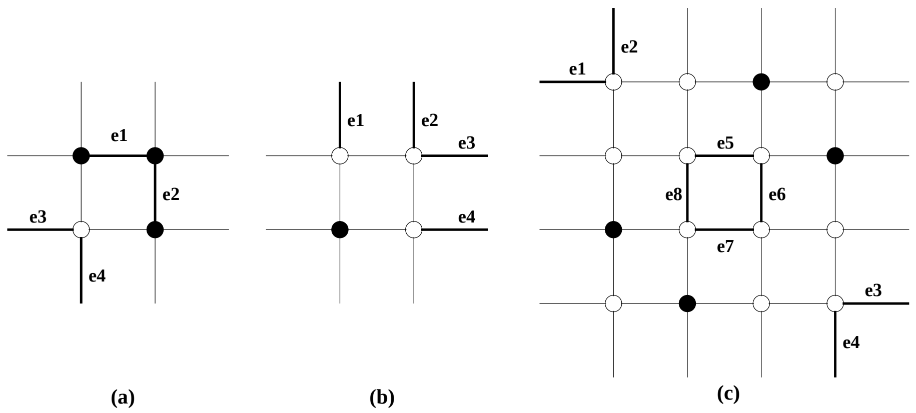

Figure 5.4: Point configurations for inequalities Classes III (a), IV (b) and VI (c) of an RGP instance. Filled (empty) points are terminal (Steiner) vertices.

5.4 Set Partition Models

Besides the (M^{RGP}) model, a formulation where the variables are related to rectangles of the rectangular partition was also studied for the RGP in [6] and [3]. With the use of these variables, the RGP translates into a set partition problem (SPP). As we have seen in Sections 5.2 and 5.3, RGP and RPST are closely related. Hence, it is natural to formulate RPST as an SPP too with, of course, the additional stabbing variable and constraints. The current section shows how this can be done and also presents some properties of the new model.

Let $H = \{1, \ldots, p\}$ be a finite set and $K = \{K_1, K_2, \ldots, K_q\}$ be a family of subsets of H. Then, $K' \subseteq K$ forms a *partition of H* if $K_i \cap K_j = \emptyset$ for every pair of distinct elements K_i and K_j of K', and $\bigcup_{K_j \in K'} K_j = H$. If a cost c_j is associated to each set K_j in K, then a partition K' have total cost $\sum_{K_j \in K'} c_j$. The set partition problem consists in finding a partition of H with minimum cost and it can be formulated as an IP problem as follows:

$$(M^{SPP}) \qquad z = \min \sum_{j=1}^{q} c_j \lambda_j \tag{5.13}$$

$$\text{s.t.} \qquad \sum_{j=1}^{q} a_{ij} \lambda_j = 1, \qquad i = 1, \ldots, p\ , \tag{5.14}$$

$$\lambda_j \in \mathbb{B}, \qquad j = 1, \ldots, q\ , \tag{5.15}$$

where the binary variable λ_j is set to 1 if and only if K_j is in the partition. The coefficient a_{ij} is equal to 1 if $i \in K_j$ and 0 otherwise. Therefore, constraints (5.14) ensure that every element in H is covered by exactly one set K_j.

In order to model a given problem as set partition problem we must first define the sets H and K. In [6] this was done for the RGP. Given an instance $I = (R, T)$, H was defined as the set of canonical rectangles of grid(R, T) (as defined in Section 5.2) and K as the set of rectangles whose sides are composed by grid segments of I and having no terminal points in their interior. With H and K defined in that way, a_{ij} is set to one if and only if the j-th rectangle contains the canonical rectangle i. Also, the variable λ_j takes value one if and only if rectangle j is part of the optimal rectangular partition.

To model the objective function, appropriate costs have to be assigned to each rectangle of K. This is accomplished by assigning the cost of a rectangle to its *weighted perimeter*. Given a rectangle K_j with sides composed of segments of grid(R, T), the weight of a segment is zero if the segment lies on the border of R and 1/2 otherwise. De Meneses and de Souza proved that with costs computed in this way, the optimum of the set partition model for the RGP is equal to that of the M^{RGP}. From now on, the resulting model for the RGP is denoted by M^{SPP}_{rgp}.

Using a similar reasoning, given the polygon P at the input of the RPST, H can be defined as the set of canonical rectangles in grid(P) (as defined in Section 5.2) and K as the set of rectangles having its sides composed by segments of E^P. As before, the coefficients a_{ij} are set to one if and only if the j-th rectangle contains the canonical rectangle i. The variables λ_j are defined as for the RGP case.

Because in the RPST the objective function is not expressed by a summation, the problem can not be casted directly as a set partition problem. However, as we did for M^{RPST}, the stabbing variable k can also be used to get a linear formulation. To this, it is enough to add the following constraints to the model:

$$\sum_{R_j \in K : R_j \cap s \neq \emptyset} \lambda_j \leq k, \tag{5.16}$$

where R_j denotes the rectangle associated to variable λ_j. Obviously, the objective function asks for the minimization of k. Although this is not a *pure* set partition formulation of the RPST, we will name the resulting model *the set partition model* of the problem and denote it by M^{SPP}_{rpst}.

5.4.1 Properties of the Set Partition Model for the RPST

Whenever there are two IP formulations for a problem, it is interesting to know if one of them dominates the other or, in other words, if the dual bound produced by the linear relaxation of one of them is always at least as good as the one computed by the relaxation of the other. For the RGP, it was shown in [6] that (M^{SPP}_{rgp}) dominates (M^{RGP}). Based on that, we show below that (M^{SPP}_{rpst}) dominates (M^{RPST}), i.e., the set partition model is also stronger than the segment model for the RPST.

Proposition 5.5. *Given an instance of* RPST*, let W be the optimal value for the linear relaxation of the (M^{SPP}_{rpst}) and let Z be the optimal linear relaxation value of (M^{RPST}). Then, $W \geq Z$ and the formulations are not equivalent.*

Proof. Initially, for each segment $s \in E^P_g$, let Γ_s be the set of the rectangles having one side containing s. Notice that if s belongs to a feasible solution, there are exactly two rectangles of this partition that have s on their boundaries. Now the variables λ and x in the M^{SPP}_{rpst} and M^{RPST} models, respectively, can be related such that $x_s = (1/2)\sum_{k\in\Gamma_s}\lambda_k$. From the previous observation, it is clear that this equality holds for any integral solution of the RPST.

Suppose we add all these equalities as constraints to M^{SPP}_{rpst} together with the x variables for all $s \in E^P_g$. Of course, the set of feasible (λ, k) vectors in this extended model is the same as in the original one. However, denote by Q' the set of (λ, k, x) vectors that are feasible for the extended model and by Q the set of all (k, x) vectors satisfying the M^{RPST}. We show below that $Proj_x(Q') \subseteq Q$, which proves that $W \geq Z$. To this, we must show that the x vector of any feasible solution of the extended M^{SPP}_{rpst} model satisfies the constraints of M^{RPST}.

First, notice that as $x_s = \frac{1}{2}\sum_{k\in\Gamma_s}\lambda_k$ and every $\lambda_k \geq 0$, then $x_s \geq 0 \; \forall \; s \in E^P_i$. Also, as each segment s is the side of two canonical rectangles R^1_s and R^2_s and, from (M^{SPP}), $\sum_{j=1}^q a_{R^1_s,j}\lambda_j = 1$ and $\sum_{j=1}^q a_{R^2_s,j}\lambda_j = 1$. Hence, $x_s = \frac{1}{2}\sum_{k\in\Gamma_s}\lambda_k \leq \frac{1}{2}(\sum_{j=1}^q a_{R^1_s,j}\lambda_j + \sum_{j=1}^q a_{R^2_s,j}\lambda_j) \leq 1$. Ergo, $x_s \leq 1$ for all $s \in E^P_i$.

Now, to show that a vector x defined as indicated above satisfies constraint (5.2), consider Figure 5.5 depicting a reflex vertex with its incident segments and the three canonical rectangles surrounding it.

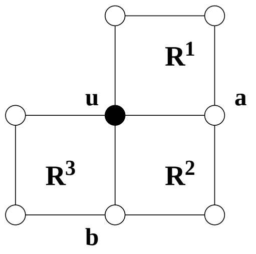

Figure 5.5: A reflex vertex u with its incident segments and the three surrounding canonical rectangles R^1, R^2 and R^3.

In the remaining of the proof we use the following notation. For a point u in grid(P), let $X = \{1, \dots, p\}$, where p is the number of canonical rectangles in the grid having u as one of its vertices. Let $R^1, \dots, R^p$ be these canonical rectangles. Notice that for a reflex vertex $p = 3$, whereas for a Steiner vertex we have $p = 4$. For $X' \subseteq X$, we denote by $\Sigma_{X'}$ the sum of the λ variables in M^{SPP} corresponding to rectangles containing all R^j for $j \in X'$ and not containing R^j for $j \in X \setminus X'$.

From the definition of vector x, $x_{ua} = (1/2)(\sum_1 + \sum_{2,3} + \sum_{2,2})$ and $x_{ub} = (1/2)(\sum_{2,2} + \sum_{1,2} + \sum_{3,3})$, implying that $x_{ua} + x_{ub} = \sum_{2,2} + (1/2)(\sum_1 + \sum_{1,2}) + (1/2)(\sum_{2,3} + \sum_{3,3})$. Because $\sum_1 + \sum_{1,2} = 1$ and $\sum_{2,3} + \sum_{3,3} = 1$ are constraints from (M^{SPP}), we end up with $x_{ua} + x_{ub} = \sum_{2,2} + 1 \geq 1$.

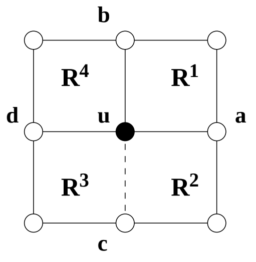

Figure 5.6: A Steiner vertex u with its incident segments ub, ua, uc and ud, and its four surrounding canonical rectangles R^1, R^2, R^3 and R^4.

It remains to show that x satisfies constraint (5.3), i.e., for a given Steiner vertex u as shown in Figure 5.6, we have $x_{ub} + x_{ua} - x_{uc} \geq 0$. From the definition of x:

$$\begin{aligned} x_{ub} &= (1/2)(\sum_{1} + \sum_{1,2} + \sum_{3,4} + \sum_{4}) \\ x_{ua} &= (1/2)(\sum_{1} + \sum_{1,4} + \sum_{2} + \sum_{2,3}) \\ x_{uc} &= (1/2)(\sum_{1,2} + \sum_{2} + \sum_{3} + \sum_{3,4}) \end{aligned}$$

from (M^{SPP}) constraints relative to R^3 and R^4 we have:

$$\sum_{1,2,3,4}+\sum_{2,3}+\sum_{3}+\sum_{3,4}=1 \Rightarrow 1-\sum_{1,2,3,4}-\sum_{2,3}-\sum_{3,4}=\sum_{3}$$

and

$$\sum_{1,4}+\sum_{1,2,3,4}+\sum_{3,4}+\sum_{4}=1$$

Therefore,

$$\begin{aligned}
x_{ub}+x_{ua}-x_{uc} &= (1/2)(\sum_{1}+\sum_{1,2}+\sum_{3,4}+\sum_{4}+\sum_{1}+\\
&\quad +\sum_{1,4}+\sum_{2}+\sum_{2,3}-\sum_{1,2}-\sum_{2}-\sum_{3}-\sum_{3,4}) \Rightarrow\\
x_{ub}+x_{ua}-x_{uc} &= (1/2)(2\sum_{1}+\sum_{1,4}+\sum_{2,3}+\sum_{4}-\sum_{3}) \Rightarrow\\
x_{ub}+x_{ua}-x_{uc} &= (1/2)(2\sum_{1}+\sum_{1,4}+\sum_{2,3}+\sum_{4}-1+\sum_{1,2,3,4}+\sum_{2,3}+\sum_{3,4}) \Rightarrow\\
x_{ub}+x_{ua}-x_{uc} &= (1/2)(2\sum_{1}+\sum_{2,3}-1+\sum_{2,3}+1) \Rightarrow\\
x_{ub}+x_{ua}-x_{uc} &= \sum_{1}+\sum_{2,3}\geq 0.
\end{aligned}$$

So far we proved that $Proj_x(Q')$ is contained in Q. It remains to show that (M^{RPST}) and (M^{SPP}) are not equivalent formulations. To this, it is sufficient to present an instance where $W > Z$. Our computational experiments show that this inequality is true for the majority of the instances tested. $\square$

If we analyze the number of variables and constraints in (M^{RPST}) and (M^{SPP}) we conclude that (M^{RPST}) have $O(n^2)$ variables and $O(n^2)$ restrictions, resulting in a constraint matrix of size $O(n^4)$, where n is the number of vertices in the polygon. Meanwhile, (M^{SPP}_{rpst}) have $O(n^4)$ variables and $O(n^2)$ constraints, resulting in a $O(n^6)$ sized matrix. So, the size of (M^{SPP}_{rpst}) could pose an algorithmic disadvantage when compared to (M^{RPST}).

In order to mitigate this disadvantage, we can try to reduce the number of variables in $(M^{SPP}_{M^{RPST}})$ by identifying sets of variables that are unnecessary for obtaining an optimal solution. This idea was explored in [6] to decrease the size of the (M^{SPP}) model of RGP. The sliding operation defined in the next paragraph is at the heart of the reduction procedures applied to the RPST.

Let π a rectangular partition of P and e be a segment of grid(P) that belongs to π. Suppose without loss of generality that e is horizontal and that it can be slided in at least one vertical direction, either upwards or downwards, by a small positive amount such that the resulting partition is still feasible. If the displacement is possible both upwards and downwards, assume that e is slided in the direction such that the number of maximal vertical segments of π with endpoints in the interior of e is maximum (see Figure 5.7). Suppose that the sliding is done until one of the extremities of e becomes a reflex vertex of P or part of e coincides with another segment of the partition or the border of P. We call this operation the *maximal sliding* of e. When this sliding is performed, it is clear that the stabbing number of no horizontal line can increase. On the other hand, the stabbing number of a vertical line can only decrease, which occurs for all those lines intersecting the interior of e. As a consequence, the rectangular partition obtained from π after the maximal sliding of e has stabbing number no larger than that of π.

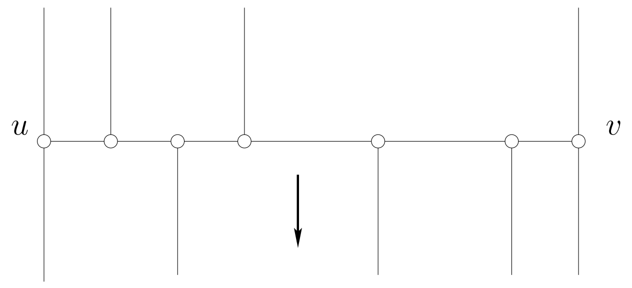

Figure 5.7: Sliding operation on the horizontal segment e. The number of vertical segments with endpoint in int(e) above e (2) is smaller than those below e (3). The sliding is done downwards.

The next result is instrumental for eliminating rectangles that are not needed to compute an optimal solution for the M_{rpst}^{SPP}.

Lemma 5.1. *Any rectilinear polygon P has an optimal rectangular partition π in which every maximal segment of π has at least one reflex vertex of P as an endpoint.*

Proof. Suppose that e is a maximal segment of grid(P) in an optimal partition π of P having no reflex vertex of P as an endpoint. Without loss of generality, assume that e is horizontal. As the endpoints of e can only be border or Steiner vertices of grid(P), e admits a maximal sliding. If the sliding is interrupted because e hits a portion of $\delta(P)$ of dimension one, the operation is equivalent to erase e and all the vertical segments of π that collapse as e moves. Thus, the new partition has at least one less maximal segment having no reflex vertex as one of its extremities. The same happens when the sliding stops because one of the endpoints of e becomes a reflex vertex of P. Therefore, if we keep repeating this operation, we must end up with a partition for which all maximal segments have at least one of its extremes in a reflex vertex of P. ☐

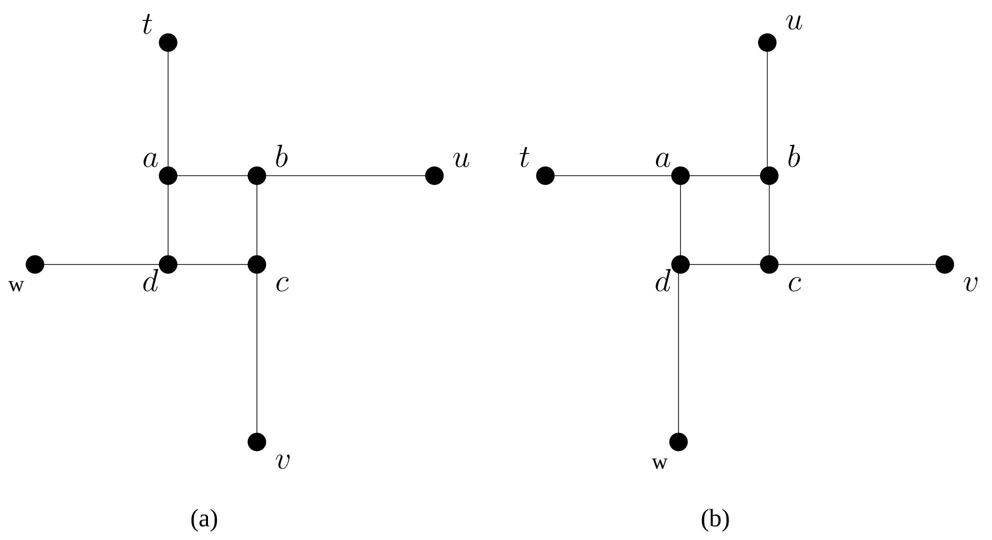

Figure 5.8: A windmill (a) and a reverse windmill (b) with its adjacent maximal segments and reflex vertices.

Definition 5.1. *Let abcd be a rectangle in a rectangular partition of a polygon where a is the left upper vertex and its four vertices are Steiner vertices as shown in Figure 5.8. If there are four segments at, bu, cv and dw contained in the polygon (except for its endpoints) where $t, u, v, w \in V_r^P$ and at is above a, bu is to the right of b, cv is below c and dw is to the left of d. Then, abcd is a* ***windmill****.*

If, however, there are four segments at, bu, cv and dw contained in the polygon (except for its endpoints) where $t, u, v, w \in V_r^P$ and at is to the left of a, bu is above b, cv is to the right of c and dw is below d. Then, abcd is a ***reverse windmill (rev-windmill*** *for short).*

Notice that a rectangle with four Steiner points as vertices can be simultaneously a windmill and a rev-windmill, a windmill and not a rev-windmill (or the converse), or neither of them as in Figure 5.9.

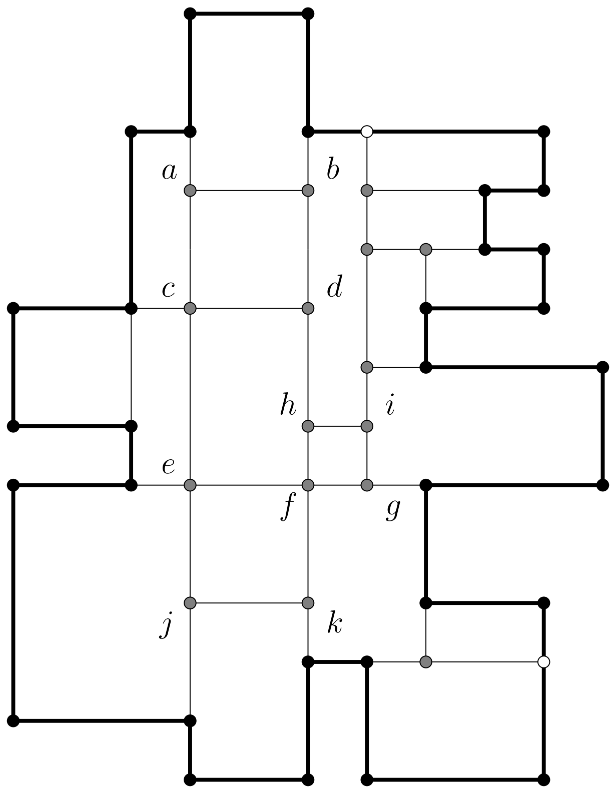

Figure 5.9: Rectangle with vertices at the Steiner points c, d, e and f that is both a windmill and a rev-windmill. Rectangle with vertices at points a, b, c and d is a windmill but not a rev-windmill. Rectangle with vertices at points e, f, j and k is a rev-windmill but not a windmill. Rectangle with vertices at points f, g, h and i is neither a windmill nor a rev-windmill.

Definition 5.2. *Let R be a rectangle with vertices in V^P. A vertex v of R is called* corner reflex *relative to R if $v \in V_r^P$ and the bisector of the internal angle of v contains one of the diagonals of R. If, on the other hand, $v \in V_r^P$ but its bisector does not contain a diagonal of R, v is called* non-corner reflex *relative to R. Figure 5.10 depicts these situations.*

Let $V_{cr}^P(R)$ denote the set of corner reflex vertices relative to rectangle R and let $V_{\overline{cr}}^P(R)$ denote the set of non-corner reflex vertices relative to rectangle R.

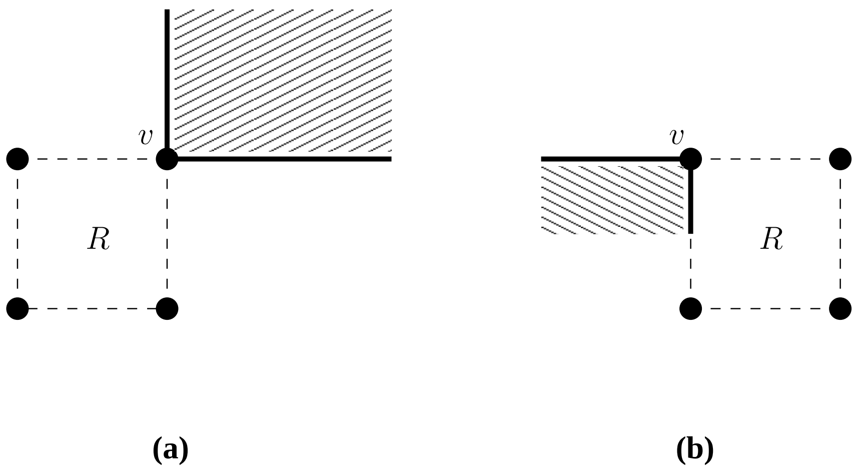

Figure 5.10: (a) a rectangle R with a corner reflex vertex v and (b) a rectangle R with a non-corner reflex vertex v. The hatched area indicates the exterior face of the polygon.

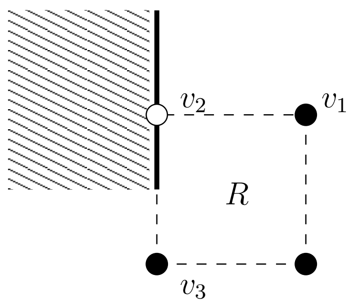

Figure 5.11: Vertex v_2 is perpendicular border relative to v_1 and R and non perpendicular border relative to v_3 and R. The hatched area indicates the exterior face of the polygon.

Definition 5.3. *Let R be a rectangle with vertices in V^P. Let $v_1 \in V^P$ and $v_2 \in V_b^P$ where v_1v_2 is a side of R. The vertex v_2 is called* perpendicular border *relative to R and v_1 if the border edge containing v_2 is perpendicular to v_1v_2. If, however, the border edge containing v_2 is not perpendicular to v_1v_2, v_2 is called* non-perpendicular border *relative to R and v_1.*

These definitions are illustrated in Figure 5.11. Below we denote by $V_{eb}^P(R, v)$ ($V_{ib}^P(R, v)$) the set of (non) perpendicular border vertices relative to R and v.

We are now ready to characterize a subset of variables that is sufficient to describe a polytope containing optimal solutions for (M_{rpst}^{SPP}).

Proposition 5.6. *For every instance of* RPST, *there is always an optimal solution for (M^{SPP}) where each rectangle in the solution is a windmill, a reverse windmill or has a point in $V_c^P \cup V_r^P \cup V_b^P$ as a vertex.*

Proof. Let us consider the possibilities for an optimal solution containing a rectangle *abcd* where all four vertices are Steiner vertices. As the solution is a rectangular partition, there is no knee at any vertex in the solution. Therefore, there are two possibilities for the configuration of the edges incident to a, b, c and d.

The first possibility is that there is a pair of parallel edges incident to a pair of adjacent vertices in the rectangle, as shown in Figure 5.12 (a). Suppose without loss of generality that b and c are the adjacent vertices and bu and cv are the parallel edges. However, from Lemma 5.1,

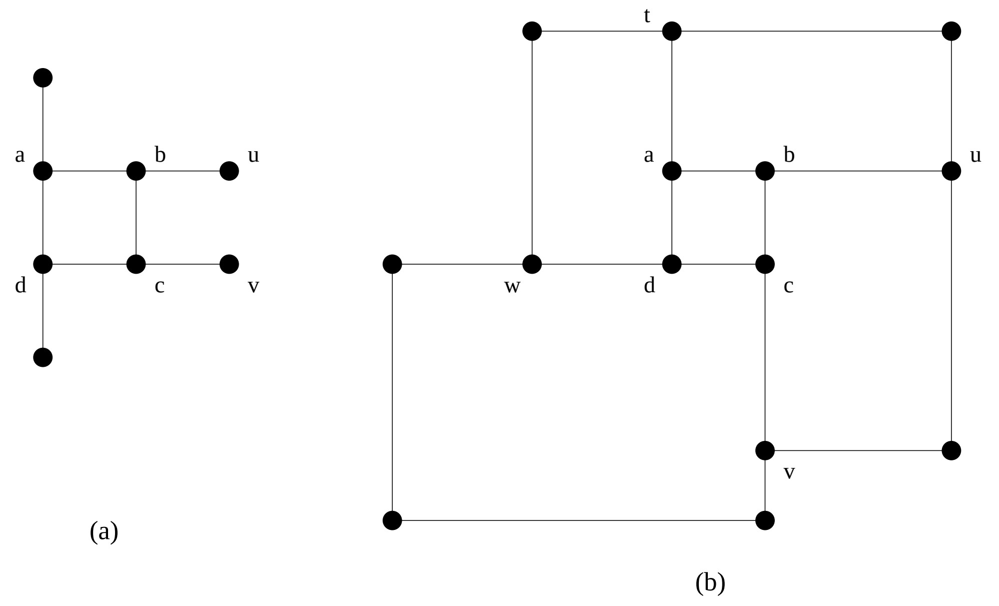

Figure 5.12: The two possibilities of a rectangle composed by Steiner vertices only.

bc must be part of a maximal segment with a reflex vertex as an endpoint. Since b and c are both Steiner vertices, then at least one of them must have degree four. But as stated in the definition of the RPST model, there is always an optimal solution where no Steiner vertex have degree four. Hence, this situation can not happen.

The second possibility is that there is no pair of parallel edges incident to a pair of adjacent vertices in the rectangle, as shown in Figure 5.12 (b). Notice that from Lemma 5.1, every maximal segment in the solution have a reflex vertex as an endpoint, so segment at must be contained in a segment having a reflex vertex as an endpoint. This is also true for segments bu, cv and dw. Hence, from definition 5.1, we conclude that rectangle $abcd$ must be either a windmill or a rev-windmill. □

Definition 5.4. *Let R be a rectangle with vertices in V^P. If u and v are adjacent vertices of R, the segment $\overline{uv}$ is said to be* slidable *if $int(\overline{uv}) \cap \delta(P)$ is empty.*

Proposition 5.7. *Let R be a rectangle in π having vertices v_1, v_2, v_3 and $v_4 \in V^P$ (in clockwise order). Consider the following conditions:*

- $F_1 = (v_1 \in V^P_{cr}(R)) \wedge (v_2 \in V^P_{cr}(R) \cup V^P_{eb}(R, v_1)) \wedge (\overline{v_1v_2}$ *is slidable*$)$,
- $F_2 = (v_1 \in V^P_{cr}(R)) \wedge (v_3 \in V^P_{cr}(R)) \wedge (v_2 \in V^P_S) \wedge (\overline{v_1v_2} \wedge \overline{v_2v_3}$ *are slidable*$)$,
- $F_3 = (v_1 \in V^P_{cr}(R)) \wedge (\{v_2, v_3, v_4\} \subseteq V^P_S) \wedge ($*all sides of R are slidable*$)$,
- $F_4 = (v_1 \in V^P_{cr}(R)) \wedge (\{v_3, v_4\} \subseteq V^P_S) \wedge (v_2 \in V_{ib}(R, v_1)) \wedge$
  $($*all sides of $R \setminus \{\overline{v_1v_2}\}$ are slidable*$)$,
- $F_5 = (v_1 \in V^P_{cr}(R)) \wedge (\{v_2, v_4\} \subseteq V^P_{ib}(R, v_1)) \wedge (v_3 \in V^P_S \cup V^P_b) \wedge$
  $(\overline{v_2v_3}$ *and* $\overline{v_3v_4}$ *are slidable*$)$,
- $F_6 = (v_1 \in V^P_{cr}(R)) \wedge (v_3 \in V^P_{eb}(R, v_2)) \wedge \{$
  $[(v_2 \in V^P_S) \wedge ($*both sides of R incident to v_2 are slidable*$)] \vee$
  $[(v_2 \in V_{ib}(R, v_1)) \wedge (\overline{v_2v_3}$ *is slidable*$)]\ \}$.

If R satisfies one of the conditions above, there is an optimal solution that does not contain R.

Proof. The proof is divided into six cases, one for each condition F_k, for $k \in \{1...6\}$. In each case we assume that we start with an optimal partition π that contains the rectangle R. Another partition is obtained from π by sliding one of the sides of R which can be easily verified to not increase the stabbing number. In other words, the new partition is also optimal and does not contain R. Without loss of generality, we assume that v_1 is the left-upper vertex of R. Besides, for $i \in \{1, 2, 3, 4\}$, we denote by a_i (b_i) the horizontal (vertical) segment of E^P incident to v_i that is external to R if it exists (see Figure 5.13).

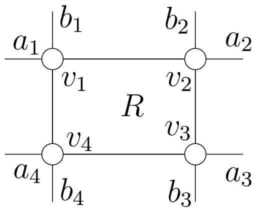

Figure 5.13: Proof of Proposition 5.7: basic notation.

The situation treated in each of the six cases is illustrated in Figure 5.14.

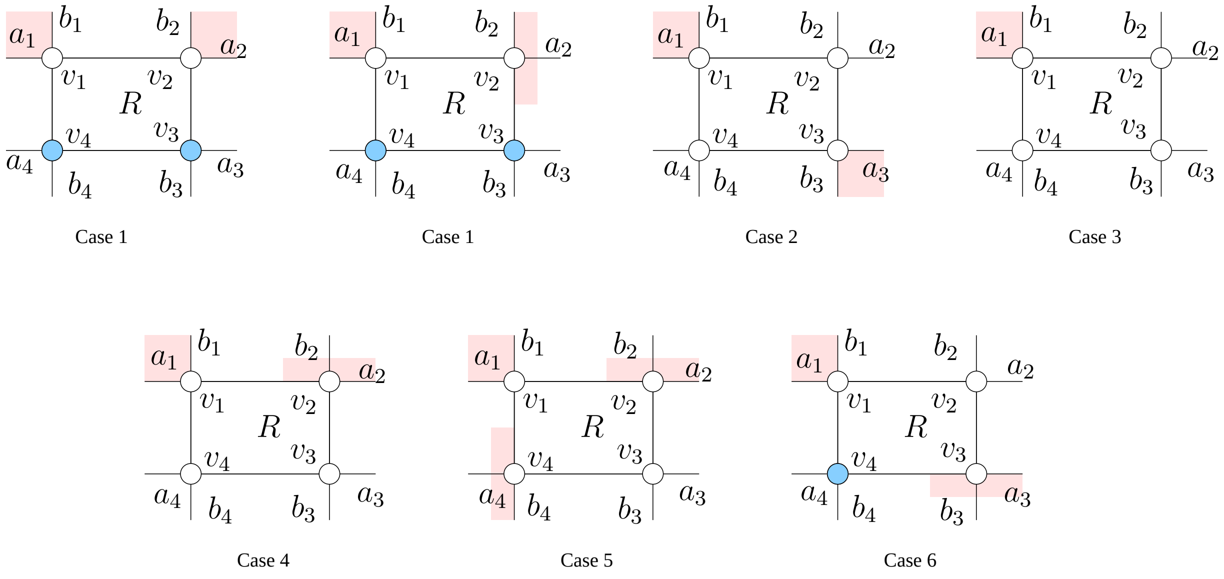

Figure 5.14: Proof of Proposition 5.7: cases 1 to 6. Shaded regions are external to P and shaded points are grid vertices of no prespecified type.

Case 1, condition F_1 is satisfied. Another optimal partition without R can be obtained from π by sliding the segment $\overline{v_1v_2}$ upwards. The sliding is possible since, in this case, b_1 and b_2 are necessarily in π as they are part of $\delta(P)$.

In essence, by symmetry, F_1 shows that an optimal solution for RPST exists that has no rectangle R with two adjacent corner reflex vertices or with a corner reflex vertex u that is adjacent to a perpendicular vertex v with respect to R and u. The next cases consider the situation where R has just one corner reflex vertex.

Case 2, condition F_2 is satisfied. Necessarily a_1, b_1, a_3 and b_3 are in π. One of the segments a_2 or b_2 must belong to π otherwise there would be a knee in v_2. Therefore, it is possible to obtain a new partition without R by applying the sliding operation to either $\overline{v_1v_2}$ (upwards) or $\overline{v_2v_3}$ (rightwards). Notice that, the same arguments hold if $v_2 \notin V_S^P$ but $v_4 \in V_S^P$. This is easily seen applying reflection symmetry to the straight line containing the diagonal $\overline{v_1v_3}$ of R.

Case 3, condition F_3 is satisfied. We have that a_1 and b_1 are in π and all sides of R are slidable. If b_2 is in π, sliding $\overline{v_1v_2}$ upwards removes R from the solution. The same holds if a_4 is in π and $\overline{v_1v_4}$ is slided leftwards. On the other hand if π contains neither b_2 nor a_4, it must contain a_2 and b_4 simultaneously (to avoid knees in v_2 and v_4). Since there can not be a knee in v_3, either a_3 or b_3 is in π and we can slide either $\overline{v_2v_3}$ or $\overline{v_3v_4}$ to get the new partition without R.
Case 4, condition F_4 is satisfied. Since a_1 (a_2) is necessarily in π then, if a_4 (a_3) is also in the partition, a new one not containing R is obtained by sliding $\overline{v_1v_4}$ ($\overline{v_2v_3}$). However, if both a_3 and a_4 are not in π, b_3 and b_4 must be present in the partition (to avoid knees in v_3 and v_4). In this case, sliding $\overline{v_3v_4}$ gives rise to a new partition not containing R.
Case 5, condition F_5 is satisfied. In this case we have that a_1, b_1, a_2 and b_4 belong to π. To avoid a knee in v_3, a_3 or b_3 must be in π. In the first situation, the slide of $\overline{v_2v_3}$ rightwards leads to a partition without R. An analogous situation occurs if b_3 is in π and we slide $\overline{v_3v_4}$ downwards.
Case 6, condition F_6 is satisfied. Necessarily a_1, b_1 and a_3 belong to π. Consider first the situation where $v_2 \in V_S^P$. Then either a_2 or b_2 is in π, otherwise there would be a knee in v_2. So, sliding $\overline{v_1v_2}$ (upwards) or $\overline{v_2v_3}$ (rightwards) produces a new partition not having R.

Now, suppose that $v_2 \in V_{ib}(R, v_1)$ (the case where $v_2 \in V_{eb}(R, v_1)$ was treated in F_1). This forces a_2 to be in π. But, since a_3 is also in π, the new partition is obtained by sliding $\overline{v_2v_3}$ rightwards. The proof is complete. □

The previous proposition treated the rectangles with at least one corner reflex vertex while the next one considers those without such vertices.

Proposition 5.8. *Let R be a rectangle having vertices v_1, v_2, v_3 and $v_4 \in V^P$ (in clockwise order). Consider the following conditions:*

- $F_1 = v_1 \in V_{eb}^P(R, v_2) \wedge (v_2 \in V_{eb}^P(R, v_1)) \wedge (\overline{v_1v_2}$ *slidable*$)$,
- $F_2 = (v_1 \in V_{eb}^P(R, v_2)) \wedge (v_3 \in V_{eb}^P(R, v_2)) \wedge (v_2 \in V_S^P) \wedge (\overline{v_1v_2}$ *and* $\overline{v_2v_3}$ *are slidable*$)$,
- $F_3 = (v_1 \in V_{eb}^P(R, v_2)) \wedge (\{v_2, v_3\} \subseteq V_S^P) \wedge (v_4 \in V_{eb}^P(R, v_3)) \wedge$
  (*all sides of* $R \setminus \{\overline{v_1v_4}\}$ *are slidable*),

If R satisfies one of the conditions above, there is an optimal solution that does not contain R.

Proof. The proof is divided into six cases, one for each condition F_k, for $k \in \{1...3\}$. The assumptions and the notation used are the same as the one in the proof of Proposition 5.7.
Case 1, F_1 is satisfied. As v_1 is in $V_{eb}^P(R, v_2)$ and v_2 is in $(v_2 \in V_{eb}^P(R, v_1)$, b_1 and b_2 are both in π. Hence, a new optimal partition not containing R is obtained by sliding $\overline{v_1v_2}$ upwards, a feasible operation since this segment is slidable.
Case 2, F_2 is satisfied. In this case b_1 and a_3 are in π by definition. To avoid a knee in v_2, a_2 or b_2 must be in π. In the first situation the new optimal solution not containing R can be obtained by sliding $\overline{v_2v_3}$ rightwards while, in the second, this can be done by sliding $\overline{v_1v_2}$ upwards.
Case 3, F_3 is satisfied. In this case b_1 and b_3 are in π by definition. Hence, if b_2 (b_3) also belongs to the current partition, a new optimal one is generated if $\overline{v_1v_2}$ ($\overline{v_3v_4}$) is slided upwards (downwards). On the other hand, if neither b_2 nor b_3 is in π, both a_2 and a_3 belong to the partition otherwise there would be knees in v_2 and v_3. But, then, sliding $\overline{v_2v_3}$ rightwards produces the desired partition.

As in the previous proof, in all cases the sliding operation yields a new partition with stabbing number no greater than the original one containing R, hence optimal. The proof is complete. □

Notice that based on Propositions 5.6, 5.7, 5.8 we can formulate the RPST as a set partition problem using a reduced set of variables and still have a valid formulation. Two things should be noticed concerning this new formulation. The first is that this formulation is a restriction of the original set partition formulation. Therefore, the linear relaxation of the former is at least as strong as the linear relaxation of the latter. Actually, the computational experiments show that the relaxation of the reduced model yields lower bounds that are often strictly larger than those computed by the original model. Second, despite our efforts, the number of variables in the reduced model remains $O(n^4)$. Ideally this quantity should become asymptotically smaller, however, we could neither find ways to do this nor prove that it can not be done.

5.5 Computational Results

We now discuss the results obtained from the computational experiments we performed to compare four (integer programming) branch-and-bound (B&B) algorithms that resulted from the models introduced in the previous sections. The first B&B algorithm is denoted by SEG and is based on the (M^{RPST}) model. The second algorithm is a B&B that implements the stronger model arising from adding the inequalities (5.10), (5.11) and (5.12) to M^{RPST}. This algorithm is denoted by SSEG. The third algorithm is a B&B algorithm which uses the M^{SPP}_{rpst} model and is named REC. At last, the B&B algorithm denoted by RREC employs the reduced M^{SPP}_{rpst} model obtained by applying Propositions 5.7 and 5.8.

The experiments were performed using a computer equipped with an Intel Xeon E3-1230 v2 3.30 GHz, 8MB cache, 32GB of RAM memory and operating system Ubuntu 12.04 OS. The programming language used was `C/C++` with `gcc` 4.6.3 compiler and every program was compiled with `-O5` optimization flag. `XPRESS-Optimizer 64-bit v27.01.02` was used as the IP solver. The default cuts, heuristics and preprocessing were turned off as we primarily intended to verify the strength of the formulations.

In order to compare the algorithms we execute them with random simple polygon instances from [4], specifically from the AGP2009a set. This set contains 600 instances with polygons varying from 20 to 2,500 vertices, 30 instances for each size. Since presenting all the results here would be very tedious and not so useful, we restrict ourselves to display the tables relative to the biggest instances with 2,500 vertices. However, the analysis considers the results for the complete benchmark.

Every test was performed with a time limit of 1,800 seconds for computations. Notice, however, that the elapsed time is checked at certain points in the program and the time between two checks may not be negligible. For this reason, the times reported here are, sometimes, slightly larger than 1,800 seconds.

The data gathered from the computational experiments are displayed in four tables, one for each algorithm. In these tables, the columns with **nVars** and **nRows** headers contain, respectively, the number of variables and constraints of each instance for the corresponding formulation. Columns with **Root LP** exhibit the value of the optimal solution of the linear relaxation at the root node of the enumeration tree. Headers **LB** and **UB** identify the columns containing, respectively, the best lower and upper bounds found. Columns with **tSetup** headers comprise the times spent in initializing and creating the integer programming problem, **tRoot** indicates the time for solving the linear relaxation at the root node of the B&B tree. Finally, **tTotal** headers identify the total execution time for each instance and the corresponding IP model. All running times are given in seconds.

Table 5.1 presents some of the data obtained from the experiments performed with SEG using the 30 instances from the set mentioned above as input. One can see that, although all the polygons have the same number of vertices, the number of variables in the IP model vary from 7,279 up to 8,159 and the number of constraints is directly proportional to the number of variables. From the 30 instances in the table, only 3 were not solved, leaving an absolute gap of just one unit. The average execution time of the 27 instances solved to optimality was 39.14 seconds.

Concerning the whole set of 600 instances, SEG was unable to solve 62 of them to optimality and whenever optimality remained unproven, the gap was of only of one unit. The average solving time for the remaining 528 instances was 17.43 seconds.

The results for SSEG with 2,500 vertices instances can be seen in Table 5.2. The number of variables and constraints in this model varies as in the previous model. This algorithm was able to solve 28 out of the 30 biggest instances with an average time of 97.47 seconds. Considering the complete set of 600 instances, for 574 of them the algorithm achieved optimality with an average of 23.14 seconds spent for instance solved. For the unsolved instances, the gap left was always of a single unit. With respect to the additional constraints used in the model, Class IV inequalities appear in 599 of the instances, Class VI in only 33 and Class III inequalities are not present in any of the instances tested. Although the point formation associated to the latter inequalities is not forbidden in RPST instances, apparently it is rare. The average increase in the number of constraints from SEG to SSEG is 2.11%.

It is worth noting that the results presented in Tables 5.1 and 5.2 are inconsistent with the ones we reported in [10]. This is because an implementation error was found in the code used in the tests of that previous work which is now fixed.

Table 5.3 displays the results obtained by running REC on the 30 biggest instances of the test set. This algorithm left a unitary duality gap in only 2 of the 30 instances with an average execution time of 73.19 seconds for the instances solved to optimality. If we consider the whole benchmark, 567 instances were solved to optimality and, once again, the ones not solved had unitary duality gaps. The average running time for the optimally solved instances was 27.19 seconds.

Finally, Table 5.4 shows some of the data produced by RREC when executed on the set of 30 largest instances. The algorithm solved 29 of these 30 instances to optimality with an average running time of 36.90 seconds. Turning to the complete instance set, the algorithm was able to solve 570 instances to optimality with an average execution time of 20.90 seconds and, once more, a gap of one unit persisted for the remaining 30 instances.

Table 5.5 summarizes the main statistics of the B&B algorithms discussed above. The meaning of the row headers are: *Solved ($n = 2,500$)*: number of instances of size $2,500$ that were solved; *Unsolved (all)*: number of unsolved instances in the entire benchmark; *Avg. Time ($n = 2,500$)*: average time in seconds computing optimal solutions for instances of size $2,500$; *Avg. Time (all)*: average time in seconds computing optimal solutions in all instances of the benchmark; and *Avg. Time (solved by all)*: average running time considering only those instances solved to optimality by the four B&B algorithms, 513 in total (see Table 5.8 for the totals per instance size); The rationale behind the computation of statistics for the group of instances *solved by all* algorithms is to avoid distorting some analyses. For example, suppose that algorithm A solves just one instance more than algorithm B. It may happen that A and B take about a hundred seconds to compute the instances they both solved to optimality but, say, A is always 10% faster in these cases. However, suppose that the additional instance that A can handle consumes all the $1,800$ seconds of computing time. In this extreme situation, if this extra instance is considered in the calculation of A's average computing time, we could reach the wrong conclusion that A is "slower" than B.

Table 5.1: Results for SEG and instances with $2,500$ vertices.

Instance	nVars	nRows	Root LP	LB	UB	tSetup	tRoot	tTotal
random-2500-1	7,835	27,765	2.88	4	4	0.22	0.88	25.84
random-2500-2	8,001	28,512	2.93	4	4	0.22	0.94	49.20
random-2500-3	7,519	26,343	2.84	4	4	0.21	0.66	15.63
random-2500-4	8,115	29,025	2.89	4	4	0.23	0.98	27.76
random-2500-5	7,701	27,162	2.78	4	4	0.22	0.89	57.84
random-2500-6	7,645	26,910	2.86	4	4	0.21	0.79	18.59
random-2500-7	7,547	26,469	2.92	4	4	0.22	0.90	27.20
random-2500-8	7,307	25,389	2.80	4	4	0.20	0.88	30.83
random-2500-9	7,905	28,080	3.01	4	4	0.22	0.94	86.68
random-2500-10	7,579	26,613	2.88	4	4	0.20	0.83	23.28
random-2500-11	7,421	25,902	2.93	4	4	0.22	0.64	15.39
random-2500-12	7,691	27,117	2.80	4	4	0.22	0.84	45.72
random-2500-13	7,397	25,794	2.91	4	4	0.21	0.74	22.31
random-2500-14	7,869	27,918	2.84	3	4	0.22	1.01	1,798.05
random-2500-15	7,411	25,857	2.85	4	4	0.21	0.76	23.03
random-2500-16	7,797	27,594	2.87	4	4	0.23	0.78	51.80
random-2500-17	8,073	28,836	2.92	4	4	0.23	1.05	81.01
random-2500-18	7,577	26,604	3.00	4	4	0.21	0.82	31.97
random-2500-19	8,129	29,088	2.95	4	4	0.22	1.12	79.17
random-2500-20	8,159	29,223	2.83	4	4	0.23	1.08	48.90
random-2500-21	7,735	27,315	2.83	4	4	0.23	0.97	28.62
random-2500-22	7,501	26,262	2.98	4	4	0.22	0.67	18.15
random-2500-23	8,137	29,124	2.83	4	4	0.23	1.22	143.71
random-2500-24	7,489	26,208	2.89	4	4	0.22	0.73	18.90
random-2500-25	7,663	26,991	2.89	4	4	0.23	0.77	24.48
random-2500-26	7,739	27,333	2.90	4	4	0.21	0.73	22.96
random-2500-27	7,895	28,035	2.81	3	4	0.22	0.82	1,798.22
random-2500-28	7,709	27,198	2.92	4	4	0.22	0.90	23.63
random-2500-29	7,279	25,263	2.83	3	4	0.22	0.54	1,797.54
random-2500-30	7,485	26,190	2.97	4	4	0.22	0.60	14.30

Table 5.2: Results for SSEG and instances with $2,500$ vertices.

Instance	nVars	nRows	Root LP	LB	UB	tSetup	tRoot	tTotal
random-2500-1	7,835	28,334	2.92	4	4	0.26	0.87	72.11
random-2500-2	8,001	29,086	2.94	4	4	0.26	1.03	88.36
random-2500-3	7,519	26,918	2.82	4	4	0.26	0.86	26.14
random-2500-4	8,115	29,653	3.00	4	4	0.26	1.21	30.31
random-2500-5	7,701	27,750	2.79	4	4	0.26	1.09	62.62
random-2500-6	7,645	27,474	2.89	4	4	0.26	0.91	41.11
random-2500-7	7,547	27,020	2.95	4	4	0.25	1.06	59.51
random-2500-8	7,307	25,913	2.88	4	4	0.26	0.90	33.05
random-2500-9	7,905	28,671	3.09	4	4	0.26	0.92	27.92
random-2500-10	7,579	27,152	2.98	4	4	0.24	0.81	49.28
random-2500-11	7,421	26,450	2.94	4	4	0.25	0.75	24.93
random-2500-12	7,691	27,673	2.82	3	4	0.26	0.87	1,798.48
random-2500-13	7,397	26,336	3.02	4	4	0.25	0.97	14.79
random-2500-14	7,869	28,500	2.91	4	4	0.26	0.95	47.74
random-2500-15	7,411	26,398	2.89	4	4	0.25	1.00	33.60
random-2500-16	7,797	28,171	2.91	4	4	0.26	0.92	52.23
random-2500-17	8,073	29,434	2.90	4	4	0.26	1.11	77.38
random-2500-18	7,577	27,179	3.01	4	4	0.26	1.08	44.75
random-2500-19	8,129	29,673	2.97	4	4	0.26	1.46	1,405.86
random-2500-20	8,159	29,831	2.93	4	4	0.27	1.16	78.64
random-2500-21	7,735	27,865	2.85	4	4	0.26	1.17	53.56
random-2500-22	7,501	26,818	3.00	4	4	0.26	1.01	26.32
random-2500-23	8,137	29,682	2.92	4	5	0.26	1.39	1,798.83
random-2500-24	7,489	26,749	2.93	4	4	0.25	0.80	26.89
random-2500-25	7,663	27,541	2.98	4	4	0.26	0.95	63.81
random-2500-26	7,739	27,887	2.94	4	4	0.25	0.91	42.22
random-2500-27	7,895	28,601	2.90	4	4	0.26	1.09	164.57
random-2500-28	7,709	27,755	3.09	4	4	0.25	1.12	21.62
random-2500-29	7,279	25,792	2.83	4	4	0.25	0.66	21.71
random-2500-30	7,485	26,732	3.00	4	4	0.26	0.80	38.18

Table 5.3: Results for REC and instances with 2,500 vertices.

Instance	nVars	nRows	Root LP	LB	UB	tSetup	tRoot	tTotal
random-2500-1	50,964	7,662	2.99	4	4	27.97	4.96	69.37
random-2500-2	52,192	7,745	3.04	4	4	28.52	6.16	51.82
random-2500-3	43,838	7,504	2.96	4	4	23.47	3.57	85.51
random-2500-4	52,745	7,802	3.01	4	4	29.17	5.69	90.72
random-2500-5	48,322	7,595	2.94	4	4	26.19	5.03	149.00
random-2500-6	47,064	7,567	3.00	4	4	25.46	4.43	106.58
random-2500-7	47,734	7,518	3.10	4	4	25.58	5.50	66.59
random-2500-8	44,771	7,398	2.96	4	4	23.82	3.97	99.13
random-2500-9	51,545	7,697	3.18	4	4	28.08	5.94	72.31
random-2500-10	47,225	7,534	2.92	4	4	25.54	3.99	56.10
random-2500-11	44,397	7,455	2.99	4	4	23.95	3.65	88.33
random-2500-12	48,752	7,590	2.99	4	4	26.23	4.17	105.01
random-2500-13	46,826	7,443	3.07	4	4	24.96	5.34	50.24
random-2500-14	50,420	7,679	2.97	4	4	27.51	4.65	55.31
random-2500-15	44,872	7,450	2.96	4	4	24.08	3.63	94.45
random-2500-16	50,343	7,643	2.95	3	4	27.19	4.94	1,827.53
random-2500-17	55,026	7,781	3.06	4	4	30.32	6.33	50.89
random-2500-18	46,446	7,533	3.10	4	4	24.84	4.91	51.54
random-2500-19	59,821	7,809	3.11	4	4	32.86	7.59	75.38
random-2500-20	58,305	7,824	3.03	4	4	31.86	6.83	57.29
random-2500-21	50,196	7,612	3.01	4	4	27.17	6.39	53.92
random-2500-22	45,395	7,495	3.05	4	4	24.30	4.14	58.42
random-2500-23	61,346	7,813	3.15	4	4	33.64	8.18	64.18
random-2500-24	46,485	7,489	2.98	4	4	25.05	4.17	65.51
random-2500-25	49,785	7,576	3.12	4	4	26.58	5.32	80.23
random-2500-26	49,821	7,614	3.02	4	4	26.98	5.88	57.84
random-2500-27	52,494	7,692	2.98	4	4	28.76	5.39	85.45
random-2500-28	48,326	7,599	3.11	4	4	26.13	5.55	64.79
random-2500-29	41,645	7,384	2.87	3	4	22.27	2.88	1,821.79
random-2500-30	44,653	7,487	3.09	4	4	24.15	4.51	43.46

Table 5.4: Results for RREC and instances with 2,500 vertices.

Instance	nVars	nRows	Root LP	LB	UB	tSetup	tRoot	tTotal
random-2500-1	29,559	7,662	2.99	4	4	17.27	3.10	36.30
random-2500-2	29,794	7,745	3.04	4	4	17.46	3.82	33.92
random-2500-3	25,930	7,504	2.96	4	4	14.93	2.14	27.84
random-2500-4	31,382	7,802	3.01	4	4	18.54	3.76	36.89
random-2500-5	27,328	7,595	2.94	4	4	16.34	2.67	46.73
random-2500-6	27,126	7,567	3.00	4	4	15.76	3.18	29.65
random-2500-7	27,522	7,518	3.10	4	4	16.19	3.18	29.61
random-2500-8	25,871	7,398	2.96	4	4	14.76	2.24	33.17
random-2500-9	29,351	7,697	3.18	4	4	17.13	3.50	37.27
random-2500-10	27,697	7,534	2.92	4	4	16.08	2.69	39.71
random-2500-11	25,759	7,455	2.99	4	4	14.92	2.06	30.08
random-2500-12	27,782	7,590	2.99	4	4	16.06	2.58	41.75
random-2500-13	26,803	7,443	3.09	4	4	15.32	3.24	30.64
random-2500-14	29,395	7,679	2.97	4	4	17.12	2.79	33.64
random-2500-15	26,223	7,450	2.96	4	4	15.10	2.51	33.52
random-2500-16	28,477	7,643	2.95	4	4	16.61	2.54	103.63
random-2500-17	31,010	7,781	3.06	4	4	18.35	3.63	35.97
random-2500-18	27,254	7,533	3.10	4	4	15.62	2.96	25.36
random-2500-19	32,473	7,809	3.12	4	4	19.11	4.58	39.41
random-2500-20	32,076	7,824	3.03	4	4	18.75	4.17	36.25
random-2500-21	28,507	7,612	3.01	4	4	16.54	3.87	32.24
random-2500-22	26,543	7,495	3.05	4	4	15.21	2.82	27.26
random-2500-23	32,758	7,813	3.15	4	4	19.26	4.56	41.10
random-2500-24	27,140	7,489	2.98	4	4	16.09	2.75	37.64
random-2500-25	28,018	7,576	3.12	4	4	16.04	3.01	35.63
random-2500-26	28,400	7,614	3.02	4	4	16.53	3.34	30.73
random-2500-27	29,708	7,692	2.98	4	4	17.51	2.90	41.44
random-2500-28	27,654	7,599	3.11	4	4	16.04	3.10	36.27
random-2500-29	24,329	7,384	2.87	3	4	14.00	2.26	1,809.19
random-2500-30	26,762	7,487	3.09	4	4	15.47	2.55	26.55

Table 5.5: Summary of results for the B&B algorithms.

	SEG	SSEG	REC	RREC
Solved ($n = 2,500$)	27	28	28	29
Unsolved (all)	62	26	33	30
Avg. Time ($n = 2,500$)	39.14	97.47	73.19	36.90
Avg. Time (all)	17.43	23.14	27.19	20.90
Avg. Time (solved by all)	15.64	20.20	19.53	11.62

Comparing the results for SEG and SSEG, one can see that the strengthening of the segment formulation had a positive effect on the number of instances solved to optimality. On the other hand, the larger number of restrictions had a negative impact on the average time of solutions solved to optimality by both algorithms (see penultimate row of Table 5.5).

Now, REC uses a model theoretically stronger than the one in SEG, and the computational results show that more instances were solved to optimality by the former algorithm. However, the average running time for the instances solved to optimality by both algorithms was smaller in SEG. When compared to SSEG, REC performed worse both in terms of the number of instances solved to optimality and average time for the solution of the instances solved by both. But, remarkably, the average time of REC becomes about 25% smaller than the one of SSEG when it comes to find the optimum of 2,500-sized instances.

Taking advantage of the results in Propositions 5.7 and 5.8, RREC uses a model with, on average (considering all instances tested), only 58.63% of the variables used by REC. This reduction on the number of variables allowed RREC to augment the total of instances solved to optimality and to reduce the average computing time relative to REC. Despite these improvements RREC solved four instances less than SSEG, the most efficient of the four algorithms in this criterion, although it was faster than SSEG in the resolution of the instances computed to optimality by all algorithms. In this same subset of instances, when compared to SEG, the faster of the four algorithms, the average time of RREC was greater. But, notice that SEG was by far less effective than RREC leaving about twice as many instances unsolved.

From the discussion above, SSEG and RREC seem to emerge as the winners among the four B&B algorithms. In spite of that, we extend our analysis a little further for a better understanding of the situation. Initially we report in Tables 5.6 and 5.7 the average times of RREC for each instance size, considering only the ones solved to optimality by all four algorithms. Then, Figure 5.15 displays a graph of the standardized average times of SEG, SSEG and REC by instance sizes. The standardization of the average times was done taking those of RREC as the mean in the calculation of the standard deviation. Hence, a positive value means that the average time was greater than the one of RREC while a negative value means the opposite.

From Figure 5.15, one can see that for $n \geq 1,500$, only SEG was a true competitor for RREC. This observation and the fact that RREC solves more instances to optimality than any other algorithm but SSEG, suggest that RREC scales better than the other algorithms.

Another aspect we consider was the strength of the different formulations. Tables 5.8 and 5.9 display statistics concerning the number of nodes explored in the B&B search for the four algorithms. Each line in these tables contains the data for a group of instances with the same number of vertices, indicated by **n**. The number of instances of a given size considered for the statistics is shown in column #. Columns with headers **avg**, **med** and **stdev** contains, respectively the average, median and standard deviation for the number of explored nodes of the algorithm identified in the header. The smallest average value among the four algorithms for

Table 5.6: Average Time for Rrec 20 to 600 vertices.

#vertices	time
20	0.08
40	0.17
60	0.24
80	0.51
100	0.56
200	1.51
300	1.98
400	2.56
500	4.40
600	8.22

Table 5.7: Average Time for Rrec 700 to 2,500 vertices.

#vertices	time
700	76.85
800	6.51
900	7.79
1,000	8.69
1,250	12.25
1,500	16.49
1,750	19.10
2,000	24.80
2,250	31.94
2,500	33.70

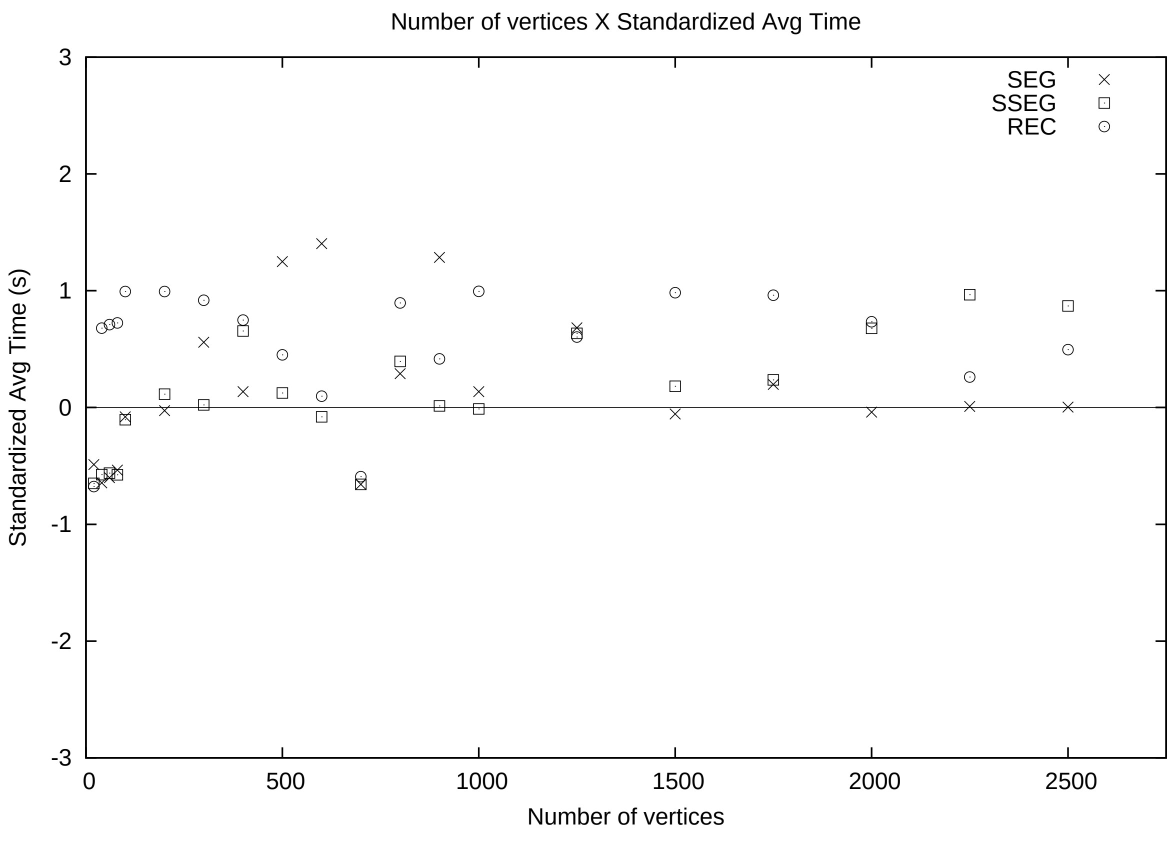

Figure 5.15: Standardized Average Time of the algorithms having Rrec as the mean for calculating standard deviation.

each **n** is presented in bold face.

The data in Tables 5.8 and 5.9 show that Rrec, on average, explores less nodes than the other algorithms for most polygon sizes. When this is not the case, its standard deviation is large which, together with the median, suggests that the high average is caused by few outliers. The smaller number of nodes explored evince the strength of the model used by Rrec when compared to the others.

As a final test, we decide to experiment with larger instances. The new set of instances contains polygons with $3,000$ up to $5,000$ vertices with increments of 500. Thirty polygons of each size were generated totalizing 150 new instances. In the analysis of Figure 5.15 we saw that seg and Rrec present the best average running times for instances with $1,500+$ vertices and these values are very close to each other. Hence, the two algorithms were executed for these large instances. The results of these experiments are summarized in Table 5.10. The row headers have the same meaning as in Table 5.5.

The average running time of Rrec considering all the large instances solved by the two algorithms is 22.87% smaller than that of seg. If we consider only the biggest instances ($n = 5,000$) this improvement goes up to 36.05%. This suggests that Rrec becomes much faster than seg as size increases. The absolute gap for the instances not solved to optimality was always of one unit for both algorithms. As before, more instances were solved to optimality by Rrec than by seg.

These computational results corroborate with the theoretical result in Proposition 5.5 relative to the strength of the M_{rpst}^{SPP} formulations. This fact is also noticeable through the **LP** values at the root nodes. Rrec had an average 3.54% improvement on this value compared to seg, considering the instances solved to optimality by both algorithms (in the original instance set). When compared to rec, the **LP** value of Rrec only presented an improvement in few cases. However the smaller number of variables of Rrec led to faster computations of the linear relaxation, as expected. Also, although the number of variables is potentially much greater than the one in the formulation used in seg, for the instances tested, this drawback was handily offset by the stronger bounds yielded by the M_{rpst}^{SPP} model.

5.6 Conclusions and Future Work

In this paper, we investigated the rpst from many different aspects. We performed the first polyhedral study about the M^{RPST} formulation presented by Durocher and Mehrabi [7]. New strong valid inequalities were obtained that effectively improve the lower bound of M^{RPST} in practice. We also proposed an alternative integer programming formulation for rpst based on the set partition problem, named M_{rpst}^{SPP}, whose relaxation was proved to yield better dual bounds than M^{RPST}. Through geometric arguments, we devised procedures that can substantially decrease the number of variables in M_{rpst}^{SPP}, making it a viable alternative to solve the rpst. As far as we know, we carried out the first computational experiments with the problem, where the different branch-and-bound algorithms arising from the ip formulations were compared. The experiments showed that it is possible to compute the optimum of polygons having thousands of vertices in a reasonable time. Besides, it was observed that the findings in this work lead to a faster and more robust algorithm.

However, we noticed that the instances that could not be solved to optimality are not the largest instances. This suggests that the hardness of an instance could be more dependent on some geometric characteristic than on its size. The identification of this characteristic is a

Table 5.8: Statistics for the number of explored nodes for SEG and SSEG algorithms.

		SEG			SSEG		
n	**#**	**avg**	**med**	**stdev**	**avg**	**med**	**stdev**
20	30	7.57	7.00	2.97	7.10	6.00	3.67
40	30	110.91	14.00	17.88	27.37	19.00	20.34
60	30	109.89	17.00	31.26	**22.97**	15.00	24.64
80	30	87.69	23.00	53.48	47.77	42.00	33.03
100	30	163.55	36.50	179.20	64.83	46.00	68.03
200	30	**121.98**	63.50	378.28	1,457.33	97.50	5,617.95
300	28	1,253.93	99.50	5,160.31	433.96	111.00	902.18
400	25	735.28	126.50	1,160.99	2,819.72	73.50	11,398.99
500	23	4,875.13	95.50	15,921.29	718.83	61.00	2,196.21
600	27	17,818.11	163.50	58,034.07	437.04	88.00	1,077.49
700	21	242.24	80.50	222.90	222.05	138.50	183.91
800	22	911.95	109.00	2,088.59	1,478.05	87.00	3,505.54
900	23	8,386.78	87.00	33,194.26	236.61	150.00	187.12
1,000	22	703.55	123.00	1,550.10	272.00	93.50	240.35
1,250	26	2,929.96	104.00	12,989.26	5,108.08	233.00	20,371.21
1,500	19	291.53	121.50	206.45	574.58	324.00	460.26
1,750	23	581.00	257.00	873.49	675.87	300.50	574.02
2,000	25	385.76	322.50	224.58	1,819.20	599.00	3,245.71
2,250	25	479.20	326.00	352.30	5,121.16	888.00	8,659.97
2,500	24	669.92	203.00	798.66	3,475.29	639.50	10,182.65

Table 5.9: Statistics for the number of explored nodes for REC and RREC algorithms.

		REC			RREC		
n	**#**	**avg**	**med**	**stdev**	**avg**	**med**	**stdev**
20	30	7.53	6.00	4.58	**5.13**	4.00	2.62
40	30	25.27	20.00	20.57	**18.07**	15.00	13.75
60	30	43.33	23.00	39.96	24.47	18.00	20.65
80	30	65.23	58.00	60.71	**45.90**	34.50	30.75
100	30	263.87	40.00	897.11	**47.80**	25.00	47.23
200	30	3,200.77	30.50	16,578.03	239.17	29.00	1,115.98
300	28	379.57	53.00	1,586.30	**99.93**	43.50	203.05
400	25	924.24	54.00	3,512.56	**69.20**	41.00	52.46
500	23	185.39	36.00	522.68	**93.96**	32.50	100.03
600	27	**384.37**	52.50	1,295.10	732.33	47.50	3,295.29
700	21	**182.38**	40.50	292.04	18,994.43	31.50	72,523.59
800	22	272.50	33.50	801.60	**109.73**	30.00	190.71
900	23	198.39	41.00	538.25	**54.09**	32.00	56.65
1000	22	497.86	33.50	1,595.90	**81.45**	35.00	97.41
1250	26	489.77	63.50	1,883.51	**53.85**	44.50	36.76
1500	19	92.42	54.00	61.57	**57.16**	36.50	34.31
1750	23	75.96	62.00	43.82	**51.26**	38.00	30.82
2000	25	72.72	56.00	39.72	**58.12**	42.50	36.93
2250	25	84.92	75.50	43.41	**66.52**	61.00	34.91
2500	24	96.79	68.00	68.15	**63.42**	42.00	46.59

Table 5.10: Summary of results for the B&B algorithms with big instances.

	SEG	RREC
Solved ($n = 5,000$)	30	30
Unsolved (all)	9	6
Avg. Time ($n = 5,000$)	170.17	108.82
Avg. Time (all)	105.56	81.37
Avg. Time (solved by all)	105.94	81.71

possible line of investigation to be pursued that may result in stronger IP models for RPST. But future research directions should also include the determination of the problem's complexity.

Bibliography

[1] M. Abam, B. Aronov, M. De Berg, and A. Khosravi. Approximation algorithms for computing partitions with minimum stabbing number of rectilinear and simple polygons. In *Proceedings of the Twenty-seventh Annual Symposium on Computational Geometry*, SoCG '11, pages 407–416, New York, NY, USA, 2011. ACM.

[2] E. Balas and M. Oosten. On the dimension of projected polyhedra. *Discrete Applied Mathematics*, 87(1-3):1–9, 1998.

[3] F. Calheiros, A. Lucena, and C. de Souza. Optimal rectangular partitions. *Networks*, 41(1):51–67, 2003.

[4] M. Couto, P. de Rezende, and C. de Souza. Instances for the Art Gallery Problem, 2009. (accessed in September, 2015).

[5] M. de Berg and M. van Kreveld. Rectilinear decompositions with low stabbing number. *Information Processing Letters*, 52(4):215 – 221, 1994.

[6] C. de Meneses and C. de Souza. Exact solutions of rectangular partitions via integer programming. *International Journal of Computational Geometry & Applications*, 10(05):477–522, 2000.

[7] S. Durocher and S. Mehrabi. Computing partitions of rectilinear polygons with minimum stabbing number. In Joachim Gudmundsson, Julián Mestre, and Taso Viglas, editors, *Computing and Combinatorics*, volume 7434 of *Lecture Notes in Computer Science*, pages 228–239. Springer Berlin Heidelberg, 2012.

[8] S. Fekete, M. Lübbecke, and H. Meijer. Minimizing the stabbing number of matchings, trees, and triangulations. *Discrete & Computational Geometry*, 40(4):595–621, 2008.

[9] A. Lingas, R. Pinter, R. Rivest, and A. Shamir. Minimum edge-length partitioning of rectilinear polygons. In H. V. Poor and W. K. Jenkins, editors, *Proceedings 20th Annual Allerton Conference on Communication, Control, and Computing*, pages 53–63. University of Illinois (Urbana, Illinois), Department of Electrical Engineering and Coordinated Science Laboratory, 1982.

[10] B. Piva and C. de Souza. Partitions of rectilinear polygons with minimum stabbing number. In *Proceedings of the VIII Latin-American Algorithms, Graphs and Optimization Symposium*, Lagos'15, pages 1–6, 2015. (to appear in Eletronic Notes in Discrete Mathematics).

[11] C. D. Tóth. Orthogonal subdivisions with low stabbing numbers. volume 3608 of *Lecture Notes in Computer Science*, pages 256–268. Springer Berlin / Heidelberg, 2005.

Chapter 6

Counterexample for the 2-approximation of finding partitions of rectilinear polygons with minimum stabbing number

---

Here a technical note made public on the `arXiv` website [3] is reproduced. This note co-authored with Cid C. de Souza exhibits a counterexample to the claim given in [2] that an algorithm proposed in that paper provides a 2-approximation for RPST. A similar result was published afterwards in [1].

This paper presents a counterexample to the approximation algorithm proposed by Durocher and Mehrabi [2] for the general problem of finding a rectangular partition of a rectilinear polygon with minimum stabbing number.

6.1 Introduction

Given a rectilinear polygon P and a rectangular partition R of P, a segment is said to be **rectilinear** relative to P if it is parallel to one of P's sides. Let s be a maximal rectilinear line segment inside P. The stabbing number of s relative to R is defined as the number of rectangles of R that s intersects. The stabbing number of R is the largest stabbing number of a maximal rectilinear line segment inside P. The Minimum Stabbing Rectangular Partition Problem (RPST) consists in finding a rectangular partition R of P having the smallest possible stabbing number. Figure 6.1 illustrates these definitions.

Variants of the problem arise from restricting the set of rectangular partitions that are considered to be valid. One of these variants is called the *conforming case*, in which every edge in the solution must be maximal, i.e., both of its endpoints must touch the border of the polygon. For this problem, in [2], Durocher et al. propose an integer programming model for the conforming case where there are exactly two edges (that can be in the solution) having each reflex vertex as endpoint. Thus, there are also precisely two variables associated to each reflex vertex.

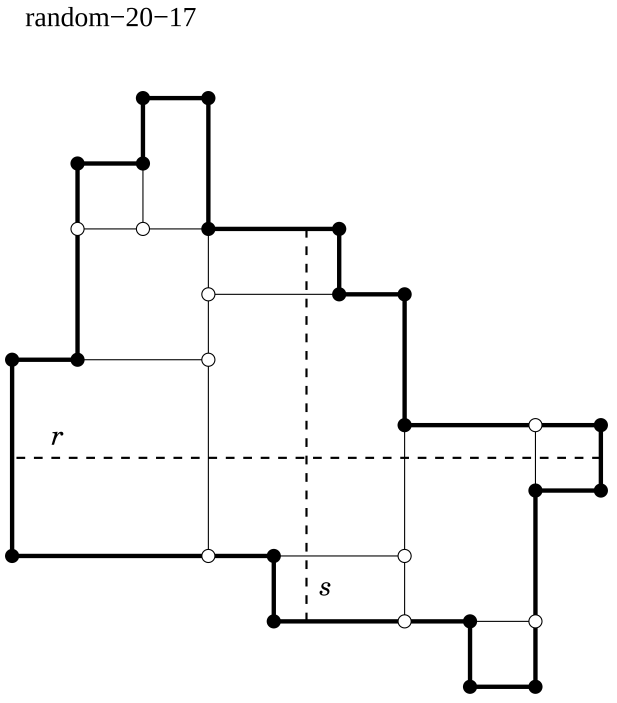

Figure 6.1: A rectilinear polygon with a rectangular partition of stabbing number 4. The dashed lines represent maximal rectilinear line segments inside the polygon. Segment r has stabbing number 4 while segment s has stabbing number 3.

In [2] a 2-approximation algorithm is presented for the conforming case of partitions of rectilinear polygons with minimum stabbing number. That approximation algorithm is based in a rounding of the variables. In the section named *Generalizing the Approximation Algorithm* of the article, it is stated that the algorithm could be extended for the general case using a formulation described informally and the same rounding rules used in the conforming case.

In this paper we show that the algorithm as described in [2] cannot give a 2-approximation for the general case of the (RPST). This is done by means of a counterexample to the referred algorithm.

6.2 IP Models

The RPST can be modelled via integer programming in a number of different ways. In this section we present two such models for the general case of RPST in an attempt to formalize the description given in [2]. But first, we need some definitions.

Let P be a rectilinear polygon, input of the RPST. Define as V_r^P the set of **reflex vertices** of P, i.e., those having internal angles equal to $3\pi/2$. Let V_c^P be the set of vertices of P that are not reflex. Denote by $grid(P)$, the set of all maximal rectilinear line segments in the interior of P having a vertex in V_r^P as one of its endpoints. Let V_s^P be the set of points in the intersection of two segments in $grid(P)$. We refer to these points as **Steiner Vertices**. The points that are not in V_r^P or V_c^P and are in the intersection of a segment in $grid(P)$ and the border of P compose the set V_b^P. Denote by V^P the set resulting from the union of all the point sets defined before, i.e., $V^P = V_r^P \cup V_c^P \cup V_s^P \cup V_b^P$.

Define E_h^P as the set of line segments in the border of P having only two points in V^P which are its extremities. Any fragment of a segment in $grid(P)$ containing exactly two vertices in V^P is called an **internal edge**. The set of all internal edges is E_i^P and the set of all edges in P is $E^P = E_h^P \cup E_i^P$. A subset E'^P of E^P defines a *knee* in a vertex $u \in V_s^P \cup V_r^P$ if exactly two edges in E'^P have u as an endpoint and these edges are orthogonal. A subset E'^P of E^P is said to define an *island* in a vertex $u \in V_r^P$ if only one edge of E'^P have u as an endpoint. At last, if ua and ub are two edges in E^P having a common endpoint u, we denote the angle between ua and ub by $\theta(ua, ub)$.

Now, we can formalize the model described in [2] as follows:

$$
\begin{array}{llll}
(RPST) & z = \min k & & (6.1)\\
\text{subject to} & x_{ua} + x_{ub} \geq 1, & \forall\, u \in V_r^P \wedge ua, ub \in E_i^P, & (6.2)\\
& x_{ua} + x_{ub} - x_{uc} \geq 0, & \forall\, u \in V_s^P, \forall\, ua, ub, uc \in E_i^P & \\
& & \text{with } \theta(ua, ub) = \pi/2, & (6.3)\\
& \displaystyle\sum_{\substack{uv \in E_i^P \\ uv \bigcap s \neq \emptyset}} x_{uv} \leq k - 1, & \forall\, s \in L, & (6.4)\\
& x_{uv} \in \mathbb{B} & \forall\, uv \in E_i^P, & (6.5)\\
& k \in \mathbb{Z}. & & (6.6)
\end{array}
$$

In the model above, we have one binary variable x_{uv} for each internal edge uv in P which is set to 1 if and only if the corresponding edge is in the rectangular partition of P. Constraints (6.2) ensure that the solution does not contain a knee in a reflex vertex. Inequalities (6.3) impose that the solution does not form a knee or an island in a Steiner vertex. Inequalities (6.4) relate the x variables with variable k, which represents the stabbing number of the solution. As a consequence, the objective function (6.1) is to minimize k. Finally, (6.5) and (6.6) are integrality restrictions for the variables. Figure 6.2 shows an instance of the RPST (called `random-20-17`) with 62 internal edges and their corresponding variables.

As stated before, the $(RPST)$ model above is not the only model for the problem and next we show another way of modelling it. However, to guarantee the correctness of the model we must first prove a property of optimal solutions for the RPST. The following proposition is a generalization of *Observation 1* in [2].

Proposition 6.1. *Any rectilinear polygon P has an optimal rectangular partition R in which every maximal segment of R has at least one reflex vertex of P as an endpoint.*

Proof. Let R be a rectangular partition of a rectilinear polygon P. Let e be a maximal segment in R having a and b as its endpoints. Suppose neither a nor b are reflex vertices. Since e is maximal and R is a rectangular partition, both endpoints of e must lie in segments perpendicular to e.

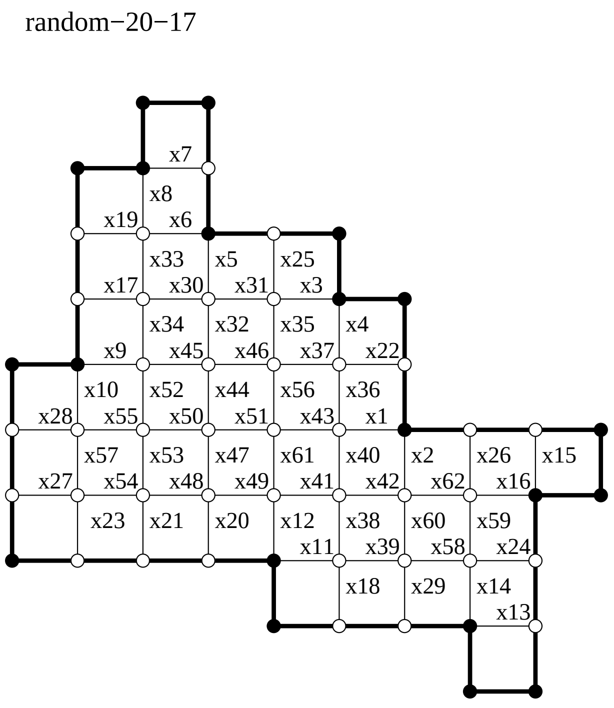

Figure 6.2: Instance `random-20-17` with 62 internal edges and the corresponding variables.

Now, since R is a rectangular partition, e define two minimal rectangles (each one possibly containing other rectangles) having e as one of its sides, let us denote them by r_1 and r_2. There are three cases to consider.

The first case consists of r_1 and r_2 been empty rectangles, i.e., neither r_1 nor r_2 contain other rectangles. Therefore, the removal of e unite these rectangles, composing a single rectangle. Therefore, $R \setminus e$ is still a rectangular partition. It is clear that removing a segment cannot increase the stabbing number of the solution. Thus, if R is an optimal solution, so is $R \setminus e$.

The second case to consider is when only one of r_1 or r_2 contains other rectangles. Suppose without loss of generality that r_1 is the one containing other rectangles. Now, we can drag e towards r_1, shrinking any segment with an endpoint in e, until e meets a reflex vertex or the border of P. In the latter case, e is merged to the border of P. It is easy to see that the result of this dragging operation is also a rectangular partition besides, the only stabbing segments affected by this operation are the ones parallel to e and their stabbing number cannot increase. Therefore, as R is optimal, so must be the new solution.

At last, we must consider the case where both r_1 and r_2 contain other rectangles. Suppose without loss of generality that the number of segments in r_1 having an endpoint in e (thus, perpendicular to it) is greater or equal than the number of segments with these characteristics in r_2. Then, again, we can drag e towards r_1, shrinking any segment with an endpoint in e, until e meets either a segment parallel to e or a reflex vertex or the border of P. If a parallel segment is met, e is merged to it and the process is repeated until a reflex vertex or the border of P is met. In case the border of P is met, e ceases to exist together with the segments in the space between e and the border. Once again, the dragging operation results in a rectangular partition of P and the only stabbing segments affected by this operation are parallel to e. But, as the number of segments in r_1 is greater or equal than the number of segments in r_2, one can see that the stabbing number of the new rectangular partition cannot be greater than that of R.

Ergo, there is always an optimal rectangular partition where every maximal segment has at least one reflex vertex of P as an endpoint. □

In the next model, given the same definitions as before, we consider the set E_e^P of rectilinear segments uv where $u \in V_r^P$ and $v \in V^P$. Notice that a segment of E_e^P can be comprised of several consecutive segments of E_i^P. Hence, we call E_e^P the **extended edge** set. In the formulation below, we have a variable x_{uv} for each edge in E_e^P and from Proposition 6.1 it is easy to notice that this set of variables is sufficient to provide optimal rectangular partitions.

$$(RPST2) \qquad z = \min k \tag{6.7}$$

subject to

$$\sum_{ua \in E_e^P} x_{ua} \geq 1, \qquad \forall\, u \in V_r^P \tag{6.8}$$

$$x_{ab} + x_{uv} \leq 1, \qquad \forall\, ab, uv : ab \cap uv \neq \emptyset \wedge \wedge\, ab \cap uv \neq a, b, u \text{ or } v \tag{6.9}$$

$$\sum_{\substack{\theta(uv,ab)=\pi/2 \ \wedge \\ \wedge\ b \in uv\ \wedge\ b \neq u\ \wedge\ b \neq v}} x_{uv} - x_{ab} \geq 0, \qquad \forall\, a \in V_r^P, b \in V_s^P \tag{6.10}$$

$$\sum_{uv \in E_e^P : uv \bigcap s \neq \emptyset} x_{uv} \leq k - 1, \qquad \forall\, s \in L \tag{6.11}$$

$$x_{uv} \in \mathbb{B} \qquad \forall\, uv \in E_e^P. \tag{6.12}$$

$$k \in \mathbb{Z} \tag{6.13}$$

In this model, inequalities (6.8) guarantee that the solution does not contain a knee in a reflex vertex. Constraints (6.9) enforce planarity (two segments of the partition can only intersect at their extremes). Constraints (6.10) prevent the existence of knees and islands in a Steiner vertex. Finally, (6.11) are the stabbing constraints and (6.12) and (6.13) are integrality constraints. Figure 6.3 shows instance `random-20-17` with 42 internal edges and the corresponding variables.

6.3 The Counterexample

Before discussing the counterexample, we first present the rounding scheme proposed in [2] for the conforming case. Once the optimum of the linear relaxation is computed, the rules for rounding variables in the conforming case are really simple: a variable corresponding to a horizontal segment is rounded down to zero if its value is smaller than or equal to 0.5 and is rounded up to one if its value is greater than 0.5. A variable corresponding to

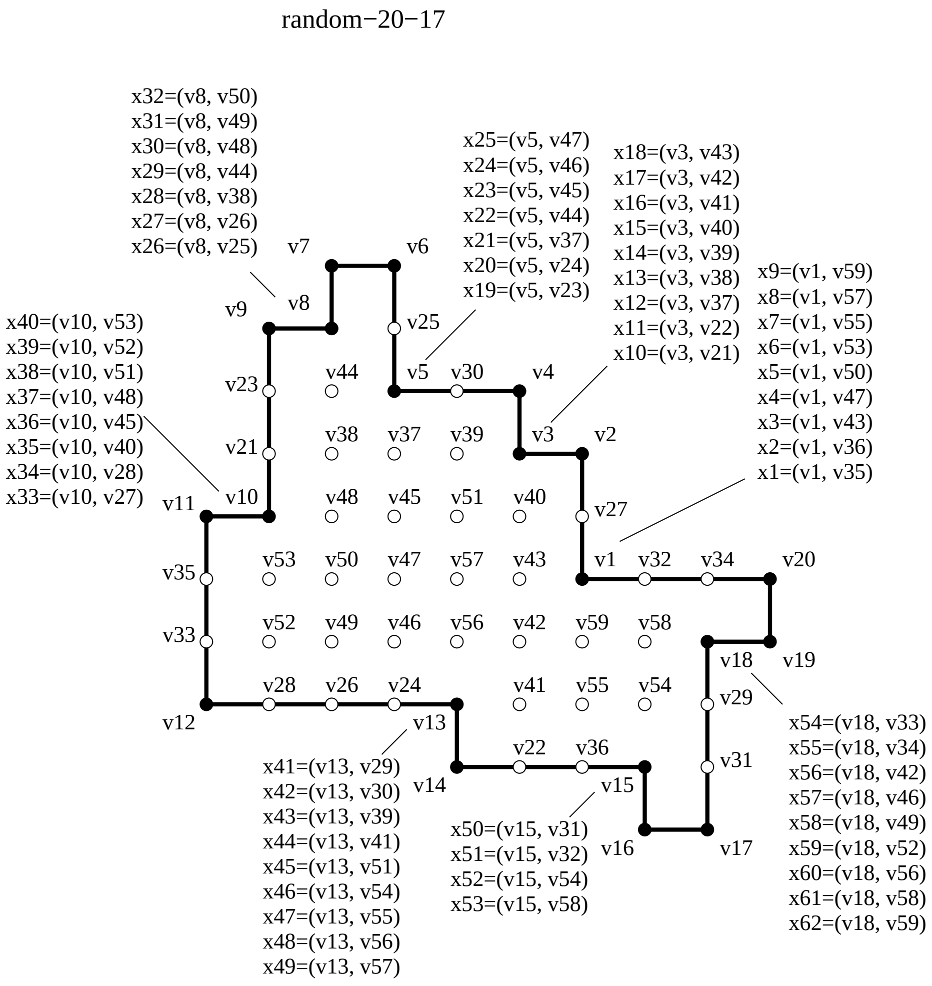

Figure 6.3: Instance `random-20-17` with its extended edges and corresponding variables.

a vertical segment is rounded down to zero if its value is smaller than 0.5 and is rounded up to one if its value is greater than or equal to 0.5.

In the *Generalizing the Approximation Algorithm* section of [2], a model for the general (non-conforming) case is described informally. From the discussion, apparently such model is equivalent to the $(RPST)$ formulation given in Section 6.2. According to the authors, the same rounding rules used in the conforming case provide a 2-approximation for the general case.

The rounding rules do not mention what should be done for Steiner vertices, and no guarantee is given that applying them directly in these situations will avoid the formation of a knee or an island. In fact, the instance displayed in Figure 6.4 shows that this cannot always be done without sacrificing feasibility. In this figure, the optimal values of the variables corresponding to edges incident to Steiner vertex `v38` (see Figure 6.3) after solving the linear relaxation associated to instance `random-20-17` are given. As only the variable corresponding to one vertical edge incident to that vertex has value greater than 0.5 and the other three are smaller than 0.5, rounding according to that rule would result in an island at `v38`. Therefore, the set of edges obtaining after rounding does not form a rectangular partition.

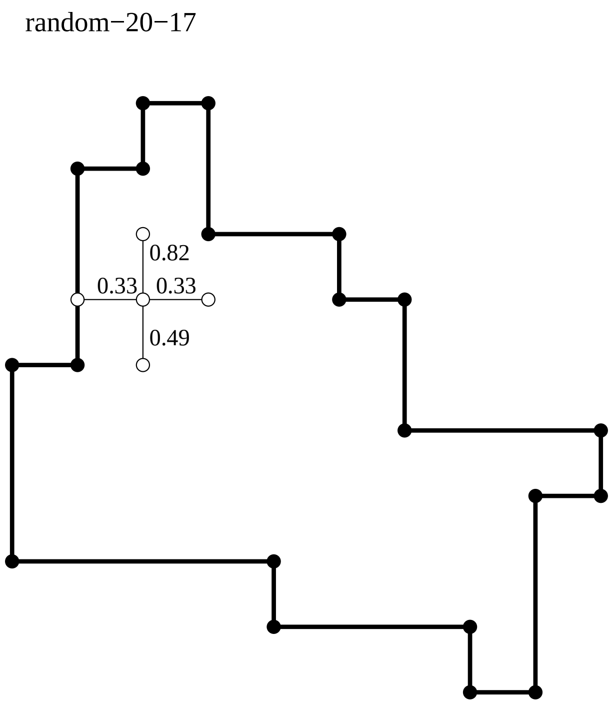

Figure 6.4: Values of variables corresponding to edges incident to a Steiner vertex after solving linear relaxation. The values are rounded with two digits after the decimal point.

It is however possible that we misinterpreted the model the authors were thinking of (although there is evidence in contrary) and the idea is actually to define variables corresponding to all edges having a reflex vertex as one of its endpoints. If so, the formulation would look like $(RPST2)$ model in the previous section. In this alternative formulation, rounding the variables using that rule does not cause the same problem as before since every variable correspond to an edge having a reflex vertex as endpoint.

Contrary to what happens in the conforming case, however, the reflex vertices here have more than two incident edges. Therefore, it is possible that the solution of the linear relaxation result in values smaller than 0.5 for all the variables corresponding to the edges incident to a certain reflex vertex. Thus, the rounding of such solution would result in a partition having a knee in a reflex vertex.

The situation described above occurs in practice with instance `random-20-17`, as shown in Figure 6.5. Consider the edges incident to vertex `v3`. All the associated variables incident to this vertex have value smaller than 0.5. As consequence, they will be rounded to zero, resulting in the formation of a knee at `v3` and, therefore, in an infeasible solution.

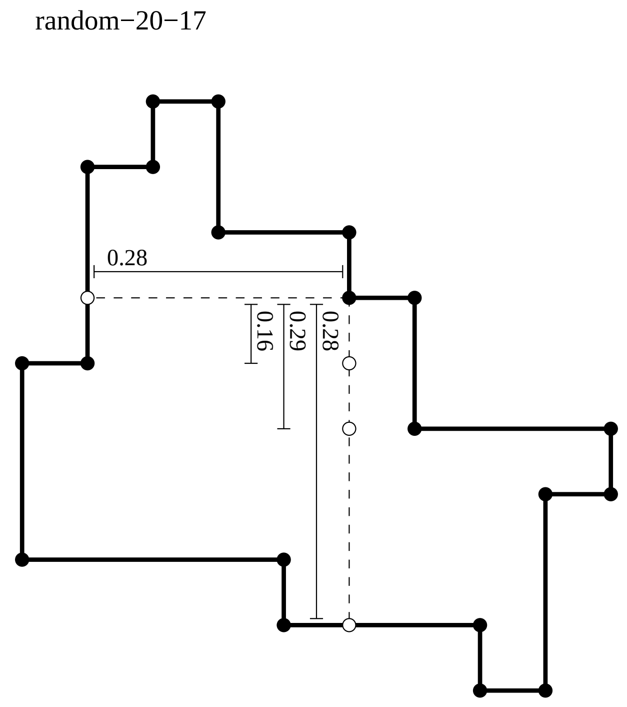

Figure 6.5: Values of variables corresponding to edges incident to a reflex vertex after solving linear relaxation. Variables with value zero are omitted. The values are rounded with two digits after the decimal point.

6.4 Conclusion

From the counterexample presented in Section 6.3, we conclude that it remains open whether a 2-approximation for the RPST in the general case exists. It is, however, noteworthy that many other contributions are presented in [2] and none of them are diminished by this counterexample.

Bibliography

[1] S. Durocher and S. Mehrabi. Erratum to: Computing partitions of rectilinear polygons with minimum stabbing number. Available online (accessed June 2015). http://http://www.cs.umanitoba.ca/∼durocher/research/pubs/cocoonErratum.pdf.

[2] Stephane Durocher and Saeed Mehrabi. Computing partitions of rectilinear polygons with minimum stabbing number. In J. Gudmundsson, J. Mestre, and T. Viglas, editors, *Computing and Combinatorics*, volume 7434 of *Lecture Notes in Computer Science*, pages 228–239. Springer Berlin Heidelberg, 2012.

[3] B. Piva and C. de Souza. Counterexample for the 2-approximation of finding partitions of rectilinear polygons with minimum stabbing number. *CoRR*, abs/1506.03865, 2015.

Appendix

```
File name: random-20-17.rect
Model: RPST
Vertex number: 59
Edge number: 62

Reading Problem stab
Problem Statistics
         231 (      0 spare) rows
          63 (      0 spare) structural columns
         752 (      0 spare) non-zero elements
Global Statistics
          63 entities        0 sets        0 set members
Minimizing MILP stab
Original problem has:
     231 rows          63 cols        752 elements      63 globals
Will try to keep branch and bound tree memory usage below 6.1Gb

   Its         Obj Value      S   Ninf  Nneg   Sum Dual Inf  Time
     0           .000000      D     24     0        .000000     0
    87          2.411765      D      0     0        .000000     0
Optimal solution found
```

```
 *** Search unfinished ***    Time:     0 Nodes:          0
Number of integer feasible solutions found is 0
Best bound is     2.411765

Solution:

x1  =  0.568627  x2  =  0.431373  x3  =  0.274510  x4  =  0.725490
x5  =  0.470588  x6  =  0.529412  x7  =  0.000000  x8  =  1.000000
x9  =  0.555556  x10 =  0.444444  x11 =  0.686275  x12 =  0.313725
x13 =  0.705882  x14 =  0.294118  x15 =  0.294118  x16 =  0.705882
x17 =  0.326797  x18 =  0.686275  x19 =  0.183007  x20 = -0.000000
x21 = -0.000000  x22 =  0.156863  x23 =  0.098039  x24 = -0.000000
x25 =  0.124183  x26 = -0.000000  x27 =  0.000000  x28 =  0.346405
x29 =  0.431373  x30 =  0.326797  x31 =  0.326797  x32 =  0.143791
x33 =  0.816993  x34 =  0.490196  x35 =  0.052288  x36 =  0.568627
x37 =  0.156863  x38 =  0.274510  x39 =  0.411765  x40 =  0.274510
x41 =  0.000000  x42 =  0.274510  x43 =  0.294118  x44 =  0.000000
x45 =  0.209150  x46 =  0.209150  x47 =  0.000000  x48 = -0.000000
x49 =  0.000000  x50 =  0.346405  x51 =  0.346405  x52 =  0.346405
x53 =  0.000000  x54 =  0.000000  x55 =  0.346405  x56 =  0.052288
x57 =  0.098039  x58 = -0.000000  x59 =  0.294118  x60 =  0.431373
x61 =  0.313725  x62 =  0.705882  x63 =  2.411765


*****************************************************************

File name: random-20-17.rect
Model: RPST2
Vertex number: 59
Edge number: 62
Reading Problem stab
Problem Statistics
         336 (      0 spare) rows
          63 (      0 spare) structural columns
         996 (      0 spare) non-zero elements
Global Statistics
          63 entities        0 sets        0 set members
Minimizing MILP stab
Original problem has:
     336 rows         63 cols        996 elements      63 globals
Crash basis containing 13 structural columns created

   Its         Obj Value      S   Ninf  Nneg        Sum Inf  Time
     0           .000000      D      1     0      24.000000     0
```

```
    82          2.413793      D      0     0        .000000     0
Optimal solution found
 *** Search unfinished ***    Time: 0
Number of integer feasible solutions found is 0
Best bound is     2.413793

Solution:

x1  =  0.293103  x2  =  0.431034  x3  =  0.275862  x4  = -0.000000
x5  = -0.000000  x6  = -0.000000  x7  = -0.000000  x8  = -0.000000
x9  = -0.000000  x10 =  0.275862  x11 =  0.275862  x12 = -0.000000
x13 = -0.000000  x14 = -0.000000  x15 =  0.155172  x16 = -0.000000
x17 = -0.000000  x18 =  0.293103  x19 =  0.241379  x20 = -0.000000
x21 =  0.275862  x22 =  0.293103  x23 =  0.051724  x24 = -0.000000
x25 =  0.137931  x26 = -0.000000  x27 = -0.000000  x28 =  0.275862
x29 =  0.241379  x30 =  0.189655  x31 = -0.000000  x32 =  0.293103
x33 =  0.155172  x34 =  0.103448  x35 = -0.000000  x36 =  0.137931
x37 =  0.293103  x38 =  0.017241  x39 =  0.000000  x40 =  0.293103
x41 =  0.000000  x42 =  0.017241  x43 = -0.000000  x44 =  0.275862
x45 = -0.000000  x46 =  0.293103  x47 =  0.120690  x48 =  0.000000
x49 =  0.293103  x50 =  0.706897  x51 = -0.000000  x52 = -0.000000
x53 =  0.293103  x54 = -0.000000  x55 =  0.293103  x56 =  0.275862
x57 =  0.000000  x58 = -0.000000  x59 = -0.000000  x60 = -0.000000
x61 =  0.000000  x62 =  0.431034  x63 =  2.413793
```

Chapter 7

Conclusions and Future Work

In this work we studied problems of finding geometric structures with minimum stabbing number. Integer programming techniques were used to create algorithms for all the problems and computational results were reported.

The complexity classes of MSTR and MCTR were proved and now we know that unless $\mathcal{P}$=$\mathcal{NP}$, there is no polynomial time algorithm to solve these problems. Moreover, besides the exact algorithm and computational results presented for MSTR, we also proposed a Lagrangian heuristic and reported experiments with an iterated rounding algorithm. The results show empirically that it is possible that **IRA** provides an approximation for MSTR.

For the RPST, a polyhedral study was also performed through a relationship with RGP, and we showed that the additional inequalities are useful in computation. A set partition formulation was also presented and compared with the segment based model both theoretically and computationally. We showed that the set partition model is stronger than a basic segment based model. Computationally, the segment based model with additional inequalities is comparable to the set partition model.

Moreover, we gave a counterexample to the claim in [16] regarding an approximation algorithm for RPST. Later, the authors of the paper also published an erratum confirming the mistake [15].

Obviously, there is still a lot of work to be done on this subject. From the integer programming perspective, we could consider different formulations for stabbing problems. For instance, in a formulation with one variable for each stabbing line it is possible to consider the relationship between the stabbing numbers of different lines.

A very interesting question still unanswered is whether the iterated rounding algorithms provide approximations for the stabbing problems if we can guarantee the existence of a highly valued fractional variable in the linear programming relaxation.

The complexity of RPST is still an open problem both for polygons with and without holes. If the problem turns out to be $\mathcal{NP}$-hard, an obvious question is if the existing approximation factor can be improved.

To conclude, in Table 7.1 we summarize the problems that were treated and the articles that originated in the thesis. The meaning of the headers are: *Problem*: name of the problem treated in the paper; *Article*: citation to the paper; *Status*: the status of the paper, i.e., *published* in a journal or conference proceedings, *submitted* to a journal or *released* on-line; *Type*: full paper/abstract/technical note; and *Contribution*: the type of contribution presented in the paper for that problem.

Table 7.1: Summary of problems approached and papers composing the thesis.

Problem	Article	Status	Type	Contribution
MSPM (axis parallel)	[37]	published	full paper	algorithms and experiments
MSST (axis parallel)	[37]	published	full paper	algorithms and experiments
MSTR (axis parallel)	[33, 37]	published	full paper	algorithms and experiments
MSTR (axis parallel)	[38]	submitted	full paper	$\mathcal{NP}$-hardness proof and experiments
MSTR (general)	[38]	submitted	full paper	$\mathcal{NP}$-hardness proof and experiments
MCTR (general)	[38]	submitted	full paper	$\mathcal{NP}$-hardness proof
RPST	[36]	published	extended abstract	IP model+algorithms and experiments
RPST	[34]	released	technical note	counterexample
RPST	[35]	submitted	full paper	IP models+algorithms and experiments

Bibliography

[1] arxiv.org e-print archive. Available online (accessed July 2015). `http://arxiv.org`.

[2] M. Abam, B. Aronov, M. De Berg, and A. Khosravi. Approximation algorithms for computing partitions with minimum stabbing number of rectilinear and simple polygons. In *Proceedings of the Twenty-seventh Annual Symposium on Computational Geometry*, SoCG '11, pages 407–416, New York, NY, USA, 2011. ACM.

[3] P. K. Agarwal. Ray shooting and other applications of spanning trees with low stabbing number. In *Proceedings of the Fifth Annual Symposium on Computational Geometry*, SCG '89, pages 315–325, New York, NY, USA, 1989. ACM.

[4] J. Beasley. Lagrangean relaxation. In *Modern Heuristic Techniques for Combinatorial Problems*, pages 243–303. McGraw-Hill, 1993.

[5] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag TELOS, Santa Clara, CA, USA, 3rd ed. edition, 2008.

[6] R. Bixby. A Brief History of Linear and Mixed-Integer Programming Computation. *Documenta Mathematica*, pages 107–121, 2012.

[7] J. A. Bondy and U.S.R. Murty. *Graph Theory With Applications*. Elsevier Science, 1976.

[8] F. Calheiros, A. Lucena, and C. de Souza. Optimal rectangular partitions. *Networks*, 41(1):51–67, 2003.

[9] S. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC '71, pages 151–158, New York, NY, USA, 1971. ACM.

[10] G. Dantzig. Linear programming. *Operations Research*, 50(1):42–47, 2002.

[11] G. Dantzig and P. Wolfe. Decomposition principle for linear programs. *Operations Research*, 8(1):101–111, 1960.

[12] M. de Berg and M. van Kreveld. Rectilinear decompositions with low stabbing number. *Information Processing Letters*, 52(4):215 – 221, 1994.

[13] C. de Meneses and C. de Souza. Exact solutions of rectangular partitions via integer programming. *International Journal of Computational Geometry & Applications*, 10(05):477–522, 2000.

[14] E. Demaine, J. Mitchell, and J. O'Rourke. The open problems project. `http://maven.smith.edu/~orourke/TOPP/`. Available online (accessed January 2010).

[15] S. Durocher and S. Mehrabi. Erratum to: Computing partitions of rectilinear polygons with minimum stabbing number. Available online (accessed June 2015). `http://http://www.cs.umanitoba.ca/~durocher/research/pubs/cocoonErratum.pdf`.

[16] S. Durocher and S. Mehrabi. Computing partitions of rectilinear polygons with minimum stabbing number. In Joachim Gudmundsson, Julián Mestre, and Taso Viglas, editors, *Computing and Combinatorics*, volume 7434 of *Lecture Notes in Computer Science*, pages 228–239. Springer Berlin Heidelberg, 2012.

[17] S. Fekete, M. Lübbecke, and H. Meijer. Minimizing the stabbing number of matchings, trees, and triangulations. In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '04, pages 437–446, Philadelphia, PA, USA, 2004. Society for Industrial and Applied Mathematics.

[18] S. Fekete, M. Lübbecke, and H. Meijer. Minimizing the stabbing number of matchings, trees, and triangulations. *Discrete & Computational Geometry*, 40(4):595–621, 2008.

[19] M. R. Garey and D. S. Johnson. *Computer and intractability: A guide to the theory of NP–completeness*. Freeman, San Francisco, 1979.

[20] P. C. Gilmore and R. E. Gomory. A linear programming approach to the cutting-stock problem. *Operations Research*, 9(6):849–859, 1961.

[21] P. C. Gilmore and R. E. Gomory. A linear programming approach to the cutting stock problem—part ii. *Operations Research*, 11(6):863–888, 1963.

[22] M. Grötschel, L. Lovász, and A. Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1(2):169–197, 1981.

[23] M. Held and R.. Karp. The traveling-salesman problem and minimum spanning trees. *Operations Research*, 18(6):1138–1162, 1970.

[24] M. Held and R. Karp. The traveling-salesman problem and minimum spanning trees: Part II. *Mathematical Programming*, 1(1):6–25, December 1971.

[25] M. Held, J. Klosowski, and J. Mitchell. Collision detection for fly-throughs in virtual environments. In *Proceedings of the Twelfth Annual Symposium on Computational Geometry*, SCG '96, pages 513–514, New York, NY, USA, 1996. ACM.

[26] J. Hershberger and S. Suri. A pedestrian approach to ray shooting: shoot a ray, take a walk. *Journal of Algorithms*, 18(3):403–431, 1995.

[27] H. Everett III. Generalized lagrange multiplier method for solving problems of optimum allocation of resources. *Operations Research*, 11(3):399–417, 1963.

[28] R. Karp. Reducibility among combinatorial problems. In *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York.*, pages 85–103, 1972.

[29] L. Levin. Universal sequential search problems. *Problemy Peredachi Informatsii*, 1973.

[30] J. Mitchell and J. O'Rourke. Computational geometry column 42. *International Journal Computational Geometry & Applications*, 11(5):573–582, 2001. Also in *SIGACT News* 32(3):63-72 (2001), Issue 120.

[31] G. L. Nemhauser and L.A. Wolsey. *Integer and Combinatorial Optimization.* John Wiley and Sons, New York, 1988.

[32] J. O'Rourke. *Computational Geometry in C.* Cambridge University Press, New York, NY, USA, 2nd edition, 1998.

[33] B. Piva and C. de Souza. The minimum stabbing triangulation problem: Ip models and computational evaluation. In A.Ridha Mahjoub, Vangelis Markakis, Ioannis Milis, and VangelisTh. Paschos, editors, *Combinatorial Optimization*, volume 7422 of *Lecture Notes in Computer Science*, pages 36–47. Springer Berlin Heidelberg, 2012.

[34] B. Piva and C. de Souza. Counterexample for the 2-approximation of finding partitions of rectilinear polygons with minimum stabbing number. *Computing Research Repository*, abs/1506.03865, 2015.

[35] B. Piva and C. de Souza. Minimum stabbing rectangular partitions of rectilinear polygons. Submitted, 2015.

[36] B. Piva and C. de Souza. Partitions of rectilinear polygons with minimum stabbing number. In *Proceedings of the VIII Latin-American Algorithms, Graphs and Optimization Symposium*, Lagos'15, pages 1–6, 2015. (to appear in Eletronic Notes in Discrete Mathematics).

[37] B. Piva, C. de Souza, Y. Frota, and L. Simonetti. Integer programming approaches for minimum stabbing problems. *RAIRO - Operations Research*, 48:211–233, 4 2014.

[38] B. Piva, S. Fekete, and C. de Souza. On triangulations with minimum stabbing or minimum crossing number. Submitted, 2015.

[39] W. Pulleyblank. Edmonds, matching and the birth of polyhedral combinatorics. *Doc. Math., J. DMV*, Extra Vol.:181–197, 2012.

[40] M. Sipser. *Introduction to the Theory of Computation.* Thomson Course Technology, Boston, MA, USA, 2nd edition, 2006.

[41] J. L. Szwarcfiter. *Grafos e Algoritmos Computacionais.* Campus, Rio de Janeiro, 1984.

[42] M. Todd. The many facets of linear programming. *Mathematical Programming*, 91(3):417–436, 2002.

[43] E. Welzl. Partition trees for triangle counting and other range searching problems. In *Proceedings of the Fourth Annual Symposium on Computational Geometry*, SCG '88, pages 23–33, New York, NY, USA, 1988. ACM.

[44] L. A. Wolsey. *Integer Programming.* John Wiley and Sons, 1998.