Universidade Estadual de Campinas
Instituto de Computação

Allan Sapucaia Barboza

Geometric Decomposition Problems

Problemas Geométricos de Decomposição

CAMPINAS
2022

Allan Sapucaia Barboza

Geometric Decomposition Problems

Problemas Geométricos de Decomposição

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.

Thesis 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. Pedro Jussieu de Rezende
Co-supervisor/Coorientador: Prof. Dr. Cid Carvalho de Souza

Este exemplar corresponde à versão final da Tese defendida por Allan Sapucaia Barboza e orientada pelo Prof. Dr. Pedro Jussieu de Rezende.

CAMPINAS
2022

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca do Instituto de Matemática, Estatística e Computação Científica
Ana Regina Machado - CRB 8/5467

B234g

Barboza, Allan Sapucaia, 1992-
Geometric decomposition problems / Allan Sapucaia Barboza. – Campinas, SP : [s.n.], 2022.

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

1. Programação linear inteira. 2. Geometria computacional. 3. Método de decomposição. I. Rezende, Pedro Jussieu de, 1955-. II. Souza, Cid Carvalho de, 1963-. III. Universidade Estadual de Campinas. Instituto de Computação. IV. Título.

Informações para Biblioteca Digital

Título em outro idioma: Problemas geométricos de decomposição
Palavras-chave em inglês:
Integer linear programming
Computational geometry
Decomposition method
Área de concentração: Ciência da Computação
Titulação: Doutor em Ciência da Computação
Banca examinadora:
Pedro Jussieu de Rezende [Orientador]
Alexandre Salles da Cunha
Yoshiko Wakabayashi
Rafael Crivellari Saliba Schouery
Lehilton Lelis Chaves Pedrosa
Data de defesa: 22-08-2022
Programa de Pós-Graduação: Ciência da Computação

Identificação e informações acadêmicas do(a) aluno(a)
- ORCID do autor: https://orcid.org/0000-0002-4763-9675
- Currículo Lattes do autor: http://lattes.cnpq.br/6901518758549730

Universidade Estadual de Campinas
Instituto de Computação

Allan Sapucaia Barboza

Geometric Decomposition Problems

Problemas Geométricos de Decomposição

Banca Examinadora:

- Prof. Dr. Pedro Jussieu de Rezende
  Instituto de Computação - UNICAMP
- Prof. Dr. Alexandre Salles da Cunha
  Instituto de Ciências Exatas - UFMG
- Profa. Dra. Yoshiko Wakabayashi
  Instituto de Matemática e Estatística - USP
- Prof. Dr. Rafael Crivellari Saliba Schouery
  Instituto de Computação - UNICAMP
- Prof. Dr. Lehilton Lelis Chaves Pedrosa
  Instituto de Computação - UNICAMP

A ata da defesa, assinada pelos membros da Comissão Examinadora, consta no SIGA/Sistema de Fluxo de Dissertação/Tese e na Secretaria do Programa da Unidade.

Campinas, 22 de agosto de 2022

À minha mãe.

Acknowledgements

Firstly, I want to thank my advisors, Pedro J. de Rezende and Cid C. de Souza. Their genuine interest in my growth as a researcher was very apparent from the very beginning of our work together, many years ago. Their patience and dedication allowed me to develop many skills, while not losing track of something very important: having fun.

I also want to thank Andre Cire, for collaborating with us. His comments and belief that there were more that could be done was crucial for shaping our work.

Without all my friends and colleagues, who had to endure so many bad drawings and half complete ideas, this journey would have been much less enjoyable.

I would like to express my gratitude to my wife, Olivia, and the rest of my family for their unconditional support.

This study was financed in part by the São Paulo Research Foundation (FAPESP), under grant 2018/14883-5, and by the Coordination for the Improvement of Higher Education Personnel – Brazil (CAPES) - Finance Code 001.

Resumo

Problemas de decomposição, que incluem particionamento, cobertura e empacotamento, constituem um assunto central em Pesquisa Operacional. Estudamos variações geométricas $\mathbb{NP}$-difíceis desses problemas no plano e apresentamos modelos de programação linear inteira (PLI) e heurísticas para eles. Lançamos também as bases para mais investigações com novos algoritmos e estruturas de dados, juntamente com benchmarks disponibilizados publicamente. Em primeiro lugar, estudamos o Problema da Partição Convexa Mínima de Conjuntos de Pontos, cujo objetivo é particionar a envoltória convexa de um conjunto de pontos P em um número mínimo de polígonos convexos vazios (sem pontos de P em seu interior) com vértices em P. Propomos um modelo PLI baseado em grafos e outro baseado em arranjos. Para o segundo modelo, fornecemos uma implementação eficiente utilizando geração de colunas, juntamente com heurísticas, pré-processamento e regras de ramificação geométricas. Identificamos um pequeno subconjunto de faces do arranjo, ou seja, restrições, suficientes para expressar o modelo, bem como uma estrutura de dados que permite consultas rápidas sobre somas de variáveis duais associadas a elas. Em segundo lugar, investigamos a Quadrangulação Convexa de Conjuntos de Pontos Bicromáticos com Inversões Mínimas. Nesse problema, dado um conjunto de pontos bicromático P, pede-se para encontrar o número mínimo de inversões de cores que permite que a envoltória convexa de P seja particionada em quadriláteros convexos vazios com vértices em P e cujas arestas têm extremidades de cores diferentes. Introduzimos um modelo PLI baseado em grafos e outro baseado em arranjos. O segundo modelo é uma nova abordagem que nos permite expressar coloração e particionamento do espaço como um modelo compacto de particionamento. Usamos esse modelo para derivar heurísticas análogas a abordagens de emparelhamento da literatura. Em terceiro lugar, estudamos o problema da Coesão em que, dado um conjunto de pontos bicromático P, busca-se particioná-lo usando polígonos convexos maximizando a diferença mínima no número de pontos de cada cor cobertos por cada polígono. Descrevemos um modelo PLI com número exponencial de variáveis que é eficientemente implementado usando geração de colunas. A combinação da relaxação desse modelo com uma heurística da literatura produz um algoritmo iterativo de pré-processamento polinomial. Esse algoritmo foi capaz de resolver otimamente grande parte do nosso benchmark. Finalmente, estudamos um problema de cobertura inspirado em redes sem fio, chamado de Cobertura Mínima 3-Colorível por Discos Unitários. Neste problema, dado um conjunto de pontos P e um conjunto D de discos de mesmo raio, deseja-se encontrar uma cobertura mínima de P selecionando um subconjunto de D que pode ser colorido com até 3 cores. Descrevemos um modelo PLI e mostramos como ele pode ser estendido e implementado iterativamente usando Decomposição de Benders Baseada em Lógica. Essa decomposição tem um problema mestre de cobertura e um subproblema de 3-coloração. Provamos que a solução da primeira iteração usa no máximo 18 vezes o menor número de cores dentre todas as coberturas de P por D. Também apresentamos algoritmos geométricos e baseados em resultados de teoria de grafos para obter cortes mais fortes, reduzindo significativamente o tempo de execução.

Abstract

Decomposition problems, which include set-partition, set-cover and set-packing, constitute a core subject in Operations Research. We study $\mathbb{NP}$-hard planar geometric variants of these problems and present integer linear programming (ILP) models and heuristics for them. We also lay the groundwork for further investigations with novel algorithms, data structures, and publicly available benchmarks. Firstly, we study the Minimum Convex Partition of Point Sets, where the goal is to partition the convex hull of a point set P into a minimum number of empty (with no points of P in their interior) convex polygons whose vertex set is P. We propose a graph-based and an arrangement-based ILP model for this problem. For the arrangement-based model, we provide an efficient column generation implementation, together with heuristics, preprocessing and geometry-based branching rules. We identify a small subset of faces of the arrangement, i.e., constraints, that suffice to express the model, as well as a data structure that enables fast queries on sums of dual variables associated to them. Secondly, we investigate the Convex Quadrangulation of Bichromatic Point Sets with Minimum Flips. In this problem, given a bichromatic point set P, one is asked to find the minimum number of color flips that allows the convex hull of P to be partitioned into empty convex quadrangles whose vertex set is P, and whose edges have endpoints of different colors. We introduce a graph-based and an arrangement-based ILP model for this problem. The second model is a novel approach that allows us to express coloring and space partitioning as a compact set-partition model. We use this model to derive heuristics analogue to matching approaches from the literature. Thirdly, we study the Coarseness problem where, given a bichromatic point set P, one seeks to partition P using convex polygons while maximizing the minimum difference between the number of points of each color covered by each polygon. We describe an ILP model with an exponential number of variables that is efficiently implemented using column generation. We combine the relaxation of this model with a heuristic from the literature leading to a polynomial-time iterative preprocessing algorithm. This algorithm solved to proven optimality a large portion of our benchmark. Lastly, we investigated a wireless network inspired set cover problem, called Minimum 3-Colorable Discrete Unit Disk Cover, where, given a point set P and a set D of disks of the same radius, one is asked to find a minimum cover for the points of P using a subset of D that can be colored with at most 3 colors. We describe an ILP model and show how it can be extended and implemented iteratively using Logic-based Benders Decomposition. This decomposition has a set-cover master problem and a 3-coloring subproblem. We prove that the solution of its first iteration uses at most 18 times the minimum number of colors from among all covers of P that use disks in D. We also present graph-theoretical and geometric algorithms to derive stronger cuts that significantly reduce the running time.

List of Figures

1.1 Example of a convex polygon with vertices p_0, p_1, p_2, p_3, p_4 and p_5 in blue and edges e_0, e_1, e_2, e_3, e_4 and e_5 in black. The set of edges determines its boundary, while its interior is shown in gray. 16
1.2 Convex Hull Partitioning 17
1.3 Minimum Convex Partition of Point Sets 17
1.4 Convex Quadrangulation of Bichromatic Point Sets with Minimum Flips. Bold points indicate color flips. 18
1.5 Example of a point set $P = \{A, B, C, D\}$ and an island $Q = \{A, C, D\}$. Dashed shapes are some of the infinitely many convex shapes that determine Q. Except for $\{A, B, C\}$, all non-empty subsets of P are islands. . . . 18
1.6 Point set P and two partitions of its points into disjoint islands. Discrepancy of the islands and the resulting partitions are shown. 19
1.7 Example of an instance of the Minimum 3-Colorable Discrete Unit Disk Cover Problem and an optimal solution. 20
1.8 Branch-and-Cut Flowchart 21
1.9 Branch-and-Cut Flowchart including Row Generation 24
1.10 Branch-Cut-and-Price Flowchart 29
1.11 Illustration of an arrangement of circles $\{d_0, d_1, d_2\}$, which has faces $\{f_0, f_1, f_2, f_3, f_4, f_5, f_6, f_7\}$ and its dual. 30

2.1 A suboptimal and an optimal solution for an instance of the MCPP. 34
2.2 An instance of the MCPP whose solution of the the RELAX model without constraints (2.1e) has a vertex p with fractional degree 2.5. Dashed edges have value 0.5 and solid edges have value 1. 37
2.3 Evaluation of the support of $x^\star$. 39
2.4 Evaluation of the support of $x^\star$. 40
2.5 Summary graphs: solution quality and runtime. 44

3.1 $P = \{i, 0, 1, \ldots, 8\}$, $q \notin P$, $\mathrm{CCW}(i, q) = (0, 1, \ldots, 8)$, $\mathrm{CCW}^+(i, q) = (0, 1, 2, 3, 4)$ and $\mathrm{CCW}^-(i, q) = (5, 6, 7, 8)$ 52
3.2 Example of a point set P and its arrangement. Red faces are the only ones that are **not** in P-wedges. 53
3.3 Example of a convex partition superimposed on the arrangement of Figure 3.2. Edges that belong to the polygons in the partition are heavier. . . 54
3.4 To illustrate Theorem 2.2, consider an uncovered region R supported by edge $e = \{i, k\}$. This edge also supports face b of the arrangement, which is already covered by polygon h. 54
3.5 Illustration of a Branch-and-Price algorithm. Adapted from [13]. 56
3.6 Example of a polygon whose last triangle is (k, l, m). Dashed lines show how it can be decomposed into triangles with k as their leftmost vertex, emphasizing the edges of (k, l, m). 57

3.7 An example of the circular order of i-faces around a given vertex i. Each pair of consecutive points in a CCW ordering of $P \setminus \{i\}$ defines an i-wedge, highlighted in red. 58
3.8 To illustrate the execution of the Column Pricing Algorithm 1. consider how P_k is divided into the lists P_{lk}^- and P_{lk}^+ by the oriented line $\overrightarrow{lk}$, as well as the movement of pointers l and o. Points in the gray region are candidates for second-to-last point for polygons whose last triangle is (k, l, m). 59
3.9 Example of an instance with four points inducing four triangles and one quadrilateral. Arc $(1, 3) \in A(P)$ splits those polygons into four sets $\mathcal{S}_{13}^+ = \{134\}$, $\mathcal{S}_{13}^- = \{123\}$, $\mathcal{S}_{13}^{over} = \{124, 1234\}$ and $\mathcal{S}_{13}^{disj} = \{234\}$. 62
3.10 Example of two distinct polygons p and h that share edges. Edge ik belongs only to p, but point i belongs to both polygons. Polygon h covers both the $(k-1)$-st and the k-th i-wedge while p only covers the k-th i-wedge. . . . 63
3.11 Lower bound of the relaxations given by the Compact Model and the Full Set Partition Model for instances of size 55 compared with the optimal solution. 68
3.12 Total solving time for the FullSP with different configurations. 69
3.13 Number of nodes of the search tree explored during for different configurations of FullSP. 70
3.14 Lower bound of the relaxations given by the FullSP with and without Degree Constraints for instances of size 90 compared with the optimal value. 70
3.15 Total solving time for the CGSP for different set stabilization methods. . . 71
3.16 Peak memory consumption for the two variations of the Set Partition Model: full and column generation using the barrier method. 72

4.1 Illustration of the arrangement induced by $L(P)$ for the set P of points indicated by the larger disks. The smaller disks show vertices of the arrangement that are crossings of line segments. Faces of the arrangement that are not i-wedges are colored red. 83
4.2 Solving time per instance size for the NATURAL Model. 88
4.3 Solving time per instance size for the SET-PARTITION Model. 89
4.4 Relative duality gap per instance size for the NATURAL Model. 90
4.5 Relative duality gap per instance size for the SET-PARTITION Model. . . . 91
4.6 Solving time per instance size for the relaxation of the SET-PARTITION Model. Green stars indicate the median solving time for the exact model. . 93

5.1 Point set P and two partitions of its points into disjoint islands. 98
5.2 Coarseness, UB and value of the relaxation given by the base model for instances with 20 points. 101
5.3 An island whose last triangle is (k, ℓ, m). Dashed lines show how it is decomposed into triangles with k as the leftmost vertex. The edges of (k, ℓ, m) are emphasized. 104
5.4 Illustration of the angular sweep technique. 106
5.5 Relative gap for different configurations of the Base model. 108
5.6 Total solving time for the relaxation of different configurations of the model. 109
5.7 Total time for Base +heuLB +cut with and without itUB. 110
5.8 Total solving time and peak memory usage for Base +heuLB +itUB +cut using Column Generation. Only instances that were solved to optimality are included. 110

6.1 Example of an instance of the M3CDUDC with 80 points and 40 disks and an optimal solution with 11 disks. Dashed circles correspond to disks in the input that are not part of the solution. 115
6.2 Example of an arrangement of four disks. Face f_0 is the unbounded face. Faces f_1 and f_8 are convex. The remaining faces are non-convex. 118
6.3 Instance with five points and six disks and its optimal solution. 123
6.4 Solution to a hard instance constructed from 10 gadgets. The displayed 3-colorable cover contains 50 disks, while the minimum cover containing 40 disks has chromatic number four. 124
6.5 Instance with seven points and eight disks and its optimal solution. 125
6.6 Solution to a hard instance constructed from 10 gadgets. The displayed 3-colorable cover contains 70 disks, while the minimum cover containing 60 disks has chromatic number four. 125
6.7 Remaining cases for lifting proof based on the number of vertices of a subgraph G' that belong to clique K. 126
6.8 Illustration of the lifting algorithm. All disks here have radius 2. A maximum clique of the vertices corresponding to disk centers that lie in the face containing k leads to the strongest lifting. 126
6.9 Geometric construction used to establish an upper bound on the chromatic number of a minimum cover. 130
6.10 An instance (a) showing a minimum cover (b) requiring 12 colors when a cover that minimizes colors (c) requires only one. 131
6.11 Number of instances solved as a function of time by the LAZY model, in combination with the heuristics REPAIR, RELAX and GREEDY, and the ITERATIVE_BENDERS algorithm. For a clearer presentation, we start with # Instances solved at 300. 135
6.12 Number of instances solved as a function of time for the LAZY_BENDERS and ITERATIVE_BENDERS. For a clearer presentation, we start Instances solved at 700. 136
6.13 Example of an instance sampled from a population density distribution with 1000 points and 250 disks. An optimal solution with 115 disks is also shown, and dashed circles represent disks in the input that are not part of the solution. 137
6.14 Number of instances solved, from the population benchmark, as a function of time for ITERATIVE_BENDERS and LAZY_BENDERS. The exclusion of odd-wheels separation is denoted by *no wheels* and the exclusion of block-cut tree decomposition by *no decomp*. 138

List of Tables

4.1 Analysis of the instances in the quadrangulable benchmark, averaged for each instance size, where '# quads' indicate the number of empty convex quadrangles, '# i-wedges' the number of i-wedges and '# faces' the number of faces of the arrangement. 87
4.2 Number of solved instances and of tight relaxations for the NATURAL and SET-PARTITION Model per size. 88
4.3 Description of the support of optimal linear relaxations found by the NATURAL Model as a function of instance size. The first ten lines indicate the percentage of edge values within the given interval. The last two lines show the number of edge variables in the support and the total number of edges. 92
4.4 Description of the support of optimal linear relaxations found by the SET-PARTITION Model as a function of instance size. The first ten lines indicate the percentage of projected edge values within the given interval. The last two lines show the number of projected edges variables in the support and the total number of edges. 92
4.5 Number of quadrangles in each heuristic, and the number of quadrangles in an unrestricted model per instance size. 92
4.6 Number of feasible solutions found by each heuristic (out of 60). 93
4.7 Solving time (in seconds) for each heuristic by instance size. For the ones based on the relaxation, (+) indicates that the average time spend solving the relaxation is shown separately. 93
4.8 Relative primal gap for each heuristic per instance size. Heuristics that did not find solutions when the relaxation was not integral are indicated by *. 94

5.1 Number of instances solved by different methods using column generation for different instance sizes. Solved in preprocessing indicates instances where $UB = LB$. 110

6.1 Results for the arrangement-based model: all instances from the configurations shown were solved to optimality. 132
6.2 Instances solved by the LAZY model alone and in combination with primal heuristics. For each pair (r, n), best performing algorithm has its **Max** m value in bold. 134
6.3 Instances solved by the LAZY_BENDERS model and ITERATIVE_BENDERS. For each pair (r, n), the best performing algorithm in indicated by its **Max** m value being in bold. The number of lazy cuts found by the LAZY_BENDERS implementation is denoted by "# Cuts" while the number of iterations performed by the ITERATIVE_BENDERS implementation is denoted by "# Iterations". 136
6.4 Instances from the population benchmark solved by the ITERATIVE_BENDERS. Instances not solved have been proven to be infeasible. . 138

Contents

1 Introduction **15**
1.1 Background . . . 20
1.1.1 Integer Linear Programming . . . 20
1.1.2 Basics of Computational Geometry . . . 29
1.2 Contributions . . . 30
1.3 Text Organization . . . 31

2 Minimum Convex Partition of Point Sets **33**
2.1 Introduction . . . 33
2.2 Experimental Environment . . . 35
2.3 A Mathematical Model for the MCPP . . . 35
2.4 Heuristics . . . 40
2.5 Computational Results . . . 42
2.6 Final remarks . . . 44

Bibliography **46**

3 Solving the Minimum Convex Partition of Point Sets with Integer Programming **47**
3.1 Introduction . . . 48
3.1.1 Our Contribution . . . 48
3.1.2 Literature Review . . . 49
3.1.3 Organization of the text . . . 49
3.1.4 Basic Notation . . . 50
3.2 A Set Partition Model for the MCPP . . . 51
3.3 Column Generation Algorithm for SPM . . . 55
3.4 Branch-and-Price . . . 60
3.4.1 Introducing Edges . . . 60
3.4.2 Primal Heuristic . . . 64
3.4.3 Initial Primal Solution . . . 65
3.4.4 Lower Bound and Early Stopping . . . 65
3.4.5 Stabilization . . . 65
3.4.6 Degree Constraints . . . 67
3.5 Experimental Results . . . 67
3.6 Conclusions . . . 72

Bibliography **74**
3.7 Additional Comments . . . 77

4 Convex Bichromatic Quadrangulation of Point Sets **78**
4.1 Introduction . . . 79
4.2 Integer Linear Programming Model . . . 80
4.3 Pure Set-Partition Formulation . . . 82
4.4 Linear Relaxation Based Heuristics . . . 84
4.4.1 Quadrangulation-based . . . 85
4.4.2 Edge-based . . . 85
4.4.3 Triangulation-based . . . 85
4.4.4 Matchings and Edge-Flips . . . 86
4.5 Experiments . . . 86
4.6 Concluding Remarks . . . 90

Bibliography **95**

5 Solving the Coarseness Problem by ILP Using Column Generation **97**
5.1 Introduction . . . 98
5.2 ILP Model for the Coarseness Problem . . . 99
5.3 Improving the Base Model . . . 100
5.3.1 Lower Bound (LB) . . . 101
5.3.2 Upper Bound (UB) . . . 102
5.3.3 Least Discrepancy Constraints . . . 102
5.4 A Column Generation Algorithm . . . 102
5.5 Experimental Results . . . 106

Bibliography **111**

6 Covering Points with Unit Disks under Color Constraints **113**
6.1 Introduction . . . 114
6.2 Solving the M3CDUDC using ILP . . . 117
6.2.1 Notation and Concepts . . . 117
6.2.2 Arrangement-based Model . . . 118
6.2.3 Symmetry Breaking Constraints . . . 118
6.2.4 Convex Faces of the Arrangement . . . 119
6.2.5 Lazy Coloring Constraints . . . 119
6.3 Extended Model . . . 120
6.3.1 Solving the Subproblems . . . 122
6.4 Heuristics . . . 126
6.4.1 Violation Repair . . . 127
6.4.2 Relaxation-based reduction . . . 127
6.4.3 Greedy Heuristic . . . 127
6.5 Analysing the decomposition . . . 128
6.5.1 Implementing Benders Decomposition . . . 128
6.5.2 Coloring Minimum Covers . . . 129
6.6 Empirical Evaluation . . . 131

Bibliography **141**

7 Conclusions **144**

Bibliography **146**

Chapter 1

Introduction

The interplay between Computational Geometry (CG) and Operations Research (OR) has been gaining traction from both communities in recent years. The quest for fast algorithms for hard geometric problems using OR techniques has led to significant advances. In particular, this cooperation between both communities resulted in the CGSHOP [7], an annual competition focused on solving $\mathbb{NP}$-hard geometric problems.

In the context of OR, several important problems are formulated, or at least illustrated, in geometric settings. Although very general, problems such as the Traveling Salesman Problem [2] and the Facility Location Problem [8], were first inspired by geometric instances, usually involving geographical maps. Naturally, many geometric variants of core problems in OR have also been investigated.

In this dissertation, we study geometric decomposition problems, which are geometric variations of set-partition, set-cover and set-packing. Decomposition problems are central to the field of OR. Not only are they employed to model industrial applications [22], but they also appear as a substructure that can be leveraged in more complex settings.

When it comes to the CG literature, spatial decomposition appears as a key step in the development of efficient algorithms, either exact, approximation or heuristics [16]. Decomposing a region into smaller, simpler regions before solving a problem can lead to algorithms that are easier to implement, debug and usually have better performance. Notably, problems that partition regions into triangles, known as triangulation problems, are among the most researched topics in the field [6].

In this work, we focus on decomposition into convex regions, which can be regarded as a superset of triangles. Moreover, convex partitions induce fewer parts when compared to triangulations. Although triangles are very simple regions, the strong properties that emanate from convexity can still be leveraged when applying algorithms to each part [5].

In particular, practical applications of spatial decomposition in the context of assigned regions for human supervision, such as in airspace sectorization, can benefit from the simplicity of convexity [9].

Before stating the problems studied in this work, we provide some fundamental definitions and concepts from computational geometry.

Herein, we consider all objects to be embedded in the plane, and coordinates of points to be given as two rational numbers.

A cyclic sequence of distinct points in the plane suffice to characterize a *polygon* p.

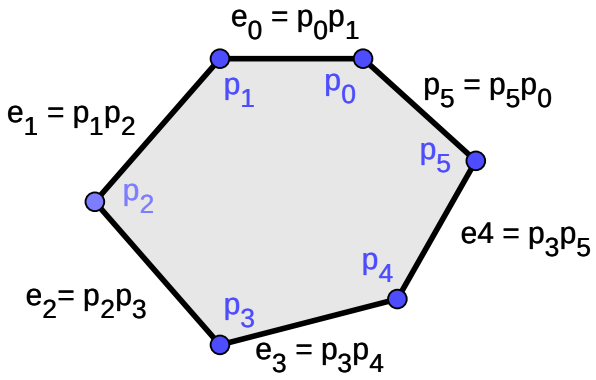


Figure 1.1: Example of a convex polygon with vertices p_0, p_1, p_2, p_3, p_4 and p_5 in blue and edges e_0, e_1, e_2, e_3, e_4 and e_5 in black. The set of edges determines its boundary, while its interior is shown in gray.

These points are the *vertices* of p, and the segments between consecutive points are the *edges* of p. In this dissertation, we only consider *simple polygons*, i.e., polygons whose edges do not intersect unless they are consecutive, in which case they must only share a vertex. Figure 1.1 illustrates a convex polygon.

The closed cyclic sequence of edges of a simple polygon p forms a Jordan (polygonal) curve, called the *boundary* of p, that divides the plane into two regions: a bounded one, called the *interior* of p and an unbounded region, called the *exterior* of p.

Whenever no confusion arises, we also refer to the union of the boundary of p with its interior simply as p.

A polygon p is *convex* if the line segment between any two points in the interior of p is entirely contained in the union of the interior and the boundary of p.

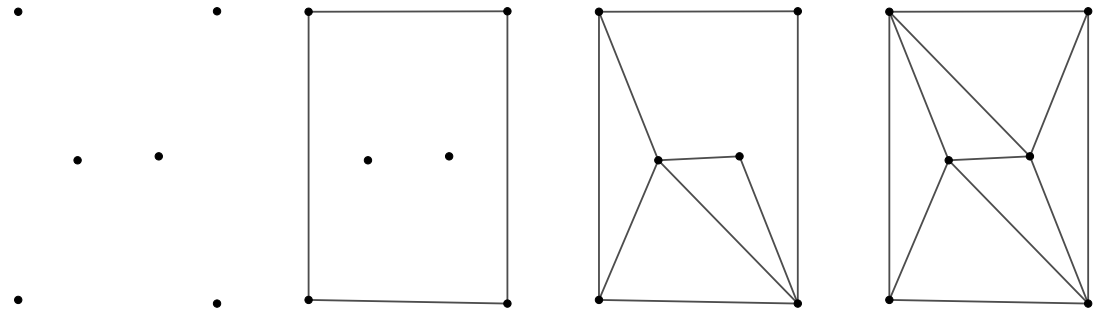
Given a point set P, its *convex hull*, denoted by $\mathrm{CH}(P)$, is the smallest convex polygon that contains P.

A polygon is said to be *empty* with respect to a point set P if there are no points of P in its interior. When the point set P is made clear from context, we only say that the polygon is empty.

A *partition* of $\mathrm{CH}(P)$ is a set of interior-disjoint empty polygons with vertices in P whose union covers $\mathrm{CH}(P)$. In other words, a partition of $\mathrm{CH}(P)$ can be seen as a planar subdivision obtained by connecting pairs of points of P with non intersecting segments such that no point is left disconnected. If all polygons are convex, we say that the partition is *convex*, otherwise, the partition is *non-convex*. Figure 1.2 shows a point set, its convex hull and two partitions of its convex hull.

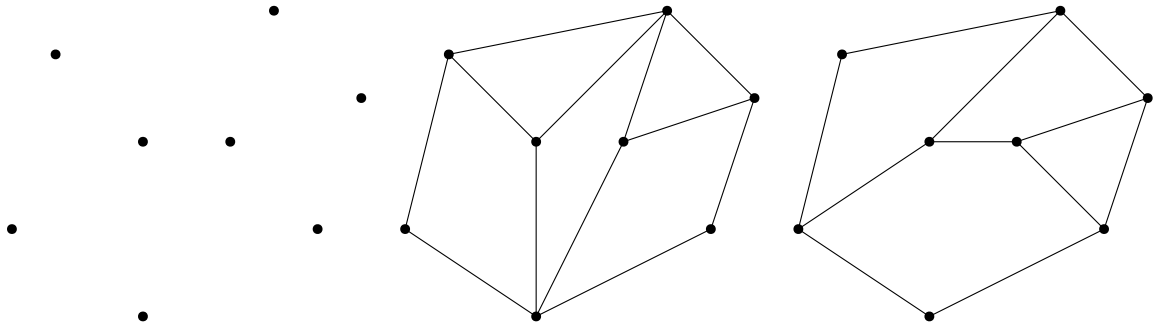
We say that a point set P is *bichromatic* if each of its points is colored either blue or red. This color assignment partitions the point set into two subsets. To distinguish between the two colors, we denote such set as $P = R \cup B$, with $R \cap B = \emptyset$. In a bichromatic setting, a *color flip* is the operation of inverting the color of a given point.

Next, we define two problems studied here in which the goal is to partition the convex hull of a point set.



(a) Point set P. (b) $\mathrm{CH}(P)$. (c) A non-convex partition of $\mathrm{CH}(P)$. (d) A convex partition of $\mathrm{CH}(P)$.

Figure 1.2: Convex Hull Partitioning



(a) Point set P. (b) Suboptimal partition of $\mathrm{CH}(P)$. (c) Minimum partition of $\mathrm{CH}(P)$.

Figure 1.3: Minimum Convex Partition of Point Sets

Minimum Convex Partition of Point Sets In the Minimum Convex Partition Problem (MCPP), given a point set P, one wants to find a partition of $\mathrm{CH}(P)$ into a minimum number of convex polygons that are empty with respect to P. Figure 1.3 shows a point set and two partitions of its convex hull.

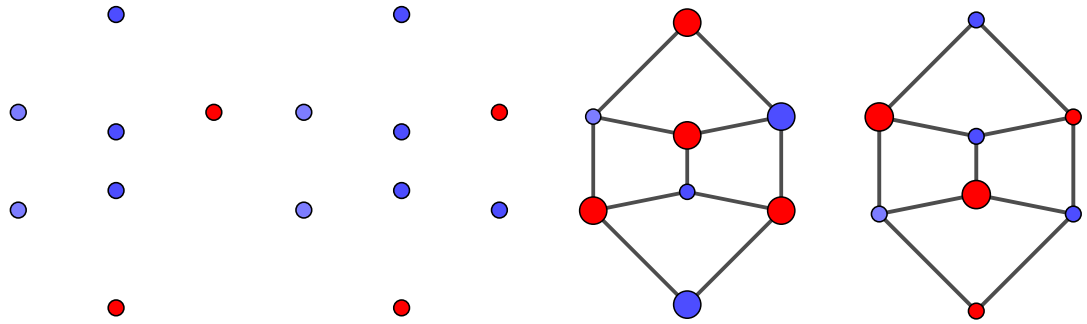
Convex Quadrangulation of Bichromatic Point Sets with Minimum Flips In the Convex Quadrangulation of Bichromatic Point Sets with Minimum Flips (CQBPS), given a bichromatic point set $P = R \cup B$, with $R \cap B = \emptyset$, the goal is to find the minimum number of color flips, such that the convex hull of the resulting bichromatic point set can be partitioned into bichromatic quadrangles that are empty with respect to P or decide that this is not possible. Figure 1.4 shows a bichromatic point set and a bichromatic quadrangulation that requires the minimum number of flips.

Note that all quadrangulations have the same number of faces.

To present the next problem, we need to define new relations between polygons and bichromatic point sets.

Given a bichromatic point set $P = R \cup B$, with $R \cap B = \emptyset$, a set of points $Q \subset P$ is called an *island* if there is a convex region S such that $Q = P \cap S$. Alternatively, $Q \subset P$ is an island if, and only if, $P \cap \mathrm{CH}(Q) = Q$. Figure 1.5 illustrates the concept of islands.

We denote the set of all islands of P by $\mathcal{I}(P)$ and say that two islands are *disjoint*



(a) A non-quadrangulable point set.

(b) Quadrangulable point set P.

(c) Bichromatic convex quadrangulation of P.

(d) Bichromatic convex quadrangulation of P with minimum flips.

Figure 1.4: Convex Quadrangulation of Bichromatic Point Sets with Minimum Flips. Bold points indicate color flips.

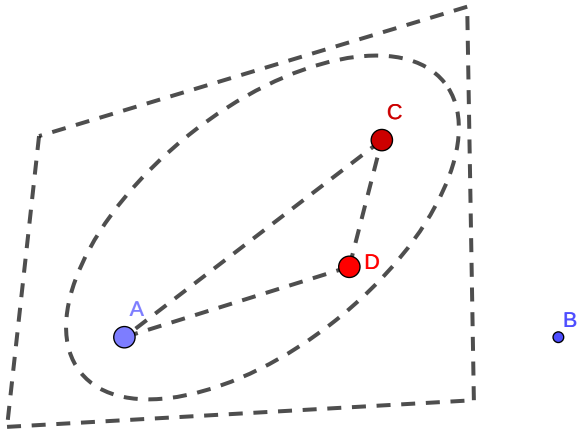
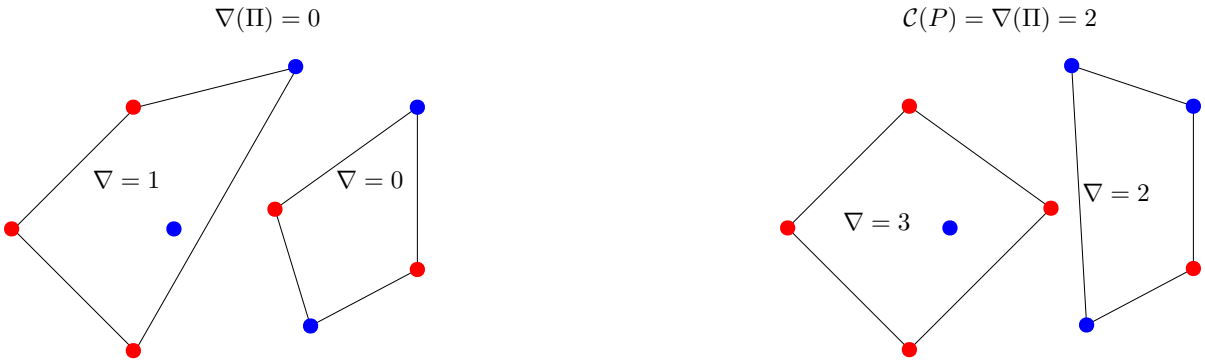


Figure 1.5: Example of a point set $P = \{A, B, C, D\}$ and an island $Q = \{A, C, D\}$. Dashed shapes are some of the infinitely many convex shapes that determine Q. Except for $\{A, B, C\}$, all non-empty subsets of P are islands.



(a) Partition of P with discrepancy 0

(b) Partition of P with discrepancy 2

Figure 1.6: Point set P and two partitions of its points into disjoint islands. Discrepancy of the islands and the resulting partitions are shown.

if their convex hulls are disjoint. Thus, it might be helpful to think of an island as its convex hull, i.e., a (not necessarily empty) convex polygon with vertices in P.

The *discrepancy* of an island Q, denoted by ∇_Q, is the absolute difference between the number of red and blue points in it, and is given by $\nabla_Q = ||R \cap S| - |B \cap S||$.

Given a partition $\Pi = \{Q_1, Q_2, \cdots, Q_k\} \subseteq \mathcal{I}(P)$ of P, its discrepancy, denoted by ∇_Π is the minimum discrepancy among its islands, and it is given by $\nabla_\Pi = \min_{Q \in \Pi} \nabla_Q$.

Figure 1.6 illustrates two partitions and their discrepancies.

With this, we can define the following problem.

Coarseness In the Coarseness problem, given a bichromatic point set $P = R \cup B$, with $R \cap B = \emptyset$, one wants to find a partition of P into disjoint islands with maximum discrepancy.

Between the two partitions shown in Figure 1.6, the one in Figure 1.6a not only has a larger discrepancy, but it is an optimal partition of P.

To define the last problem, we introduce some definitions relating disks and points.

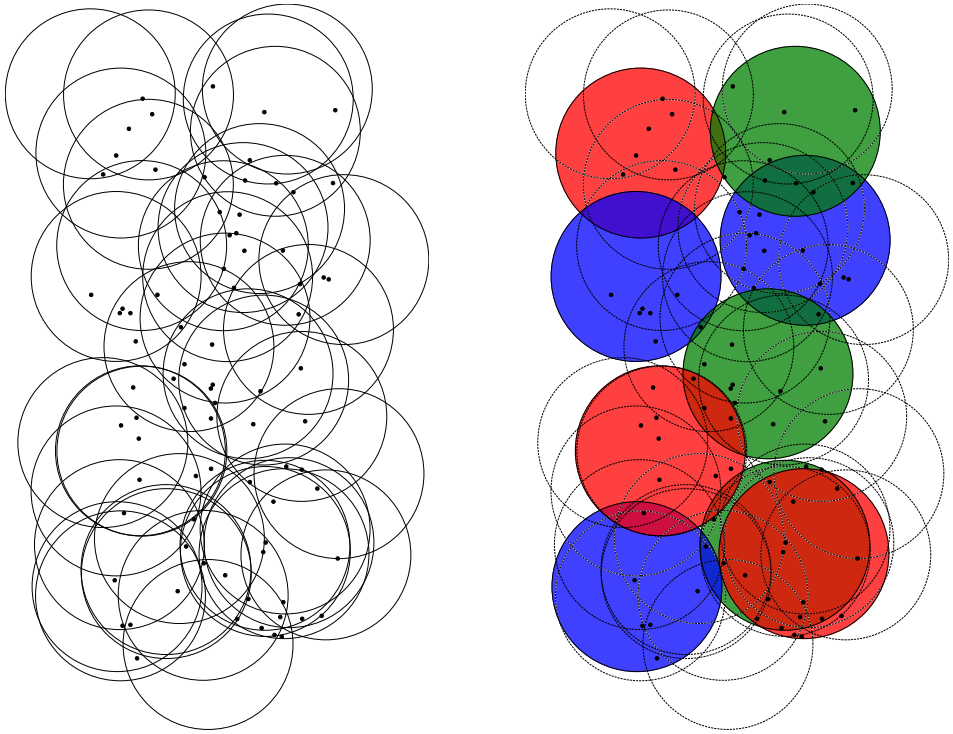
Given a point set P and a disk set D, we say that D *covers* P if for each point in P there is at least one disk in D that contains, or covers, it. A disk set D is said to be *k-colorable* if there is a assignment of at most k colors to the disks of D such that no two disks of the same color overlap. Lastly, a *unit disk* is a disk of radius one.

Minimum 3-Colorable Discrete Unit Disk Cover In the Minimum 3-Colorable Discrete Unit Disk Cover, given a set D of unit disks and a point set P, the goal is to find a minimum cardinality 3-colorable subset of D that covers P.

Figure 1.7 shows an instance and an optimal cover.

Observe that given a disk set and a point set, if all disks in a disk set have the same radius, we can change the scale so that all disks have radius one.

To conclude the presentation of the problems, we note that they are all known to be $\mathbb{NP}$-hard.



(a) Instance (b) Solution

Figure 1.7: Example of an instance of the Minimum 3-Colorable Discrete Unit Disk Cover Problem and an optimal solution.

1.1 Background

Each of the following chapters corresponds to a self-contained published or submitted paper. Each paper introduces the necessary concepts from both subjects, Combinatorial Optimization and Computational Geometry. To guide the unfamiliar reader, we provide below brief introductions and references for some recurring topics in this dissertation.

1.1.1 Integer Linear Programming

Integer Linear Programming is a core technique used throughout this dissertation. Next, we will define some fundamental concepts of linear and integer linear programs that are necessary for understanding the remainder of this work. For more comprehensive texts on integer linear programming, in particular the branch-and-cut framework, refer to [15, 24].

A Mixed-Integer Linear Program (MILP) is an optimization problem of the form

$$\min \quad cx + dy \tag{1.1a}$$

$$\text{s.t.} \quad Ax + By \geq b \tag{1.1b}$$

$$x \in \mathbb{Z}_+^{n_x}, y \in \mathbb{R}_+^{n_y}, \tag{1.1c}$$

where c is a $1 \times n_x$ vector, d is a $1 \times n_y$ vector, A is an $m \times n_x$ matrix, D is an $m \times n_y$ matrix and b is an $m \times 1$ vector. This problem has n_x integer variables x and n_y continuous variables y, totalling $n = n_x + n_y$ variables. Function 1.1a is the *objective function* of the

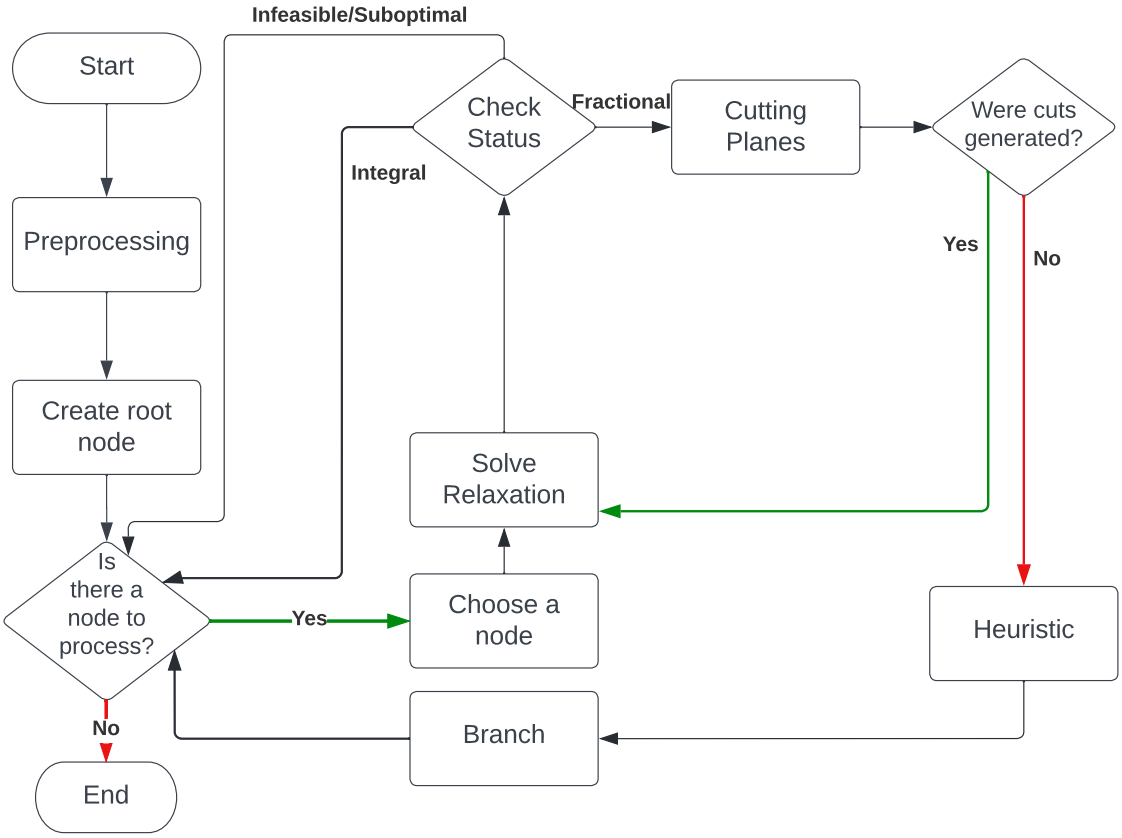


Figure 1.8: Branch-and-Cut Flowchart

program, or model, and inequalities 1.1b define its constraints. Matrices A and B are called *coefficient matrices* and vector b is simply called the *right-hand side*. Inspired by the matricial notation, we might refer to variables as columns and constraints as rows.

When there are only integer variables, the problem is called an *Integer Linear Program* (ILP). Conversely, when there are only continuous variables, the problem is called a *Linear Program* (LP).

While linear programs are solvable in polynomial time, the requirement that at least some variables be integral makes both ILP and MILP $\mathbb{NP}$-hard problems in general.

W.l.o.g, for the remainder of this chapter, we consider all (M)ILPs to be minimization problems.

Given an (M)ILP, a (linear) relaxation is obtained by making all integer variables continuous. This relaxation is very useful since it gives, in polynomial time on the size of the model, a lower bound for the (M)ILP.

The most popular technique to solve MILPs is the *Branch-and-Cut* (B&C) algorithm based on linear relaxations. This approach consists of implicitly enumerating a search tree, recursively. A simplied version of this algorithm is illustrated in Figure 1.8 and described below.

The first step of the B&C consists of preprocessing. In the *preprocessing phase*, the solver attempts to remove redundant variables and constraints to reduce the size of the problems. In some cases, based on the structure of the ILP, the solver might add

constraints that are known to increase the lower bound given by linear relaxation of the model.

When all preprocessing is done, the first node of the tree, known as the root node, is added to the pool of nodes. This node corresponds to the preprocessed ILP. Then, the node processing loop is started.

At each iteration, the solver selects a node in the pool to be processed. The B&C algorithm terminates when there are no nodes left to process.

A node is *pruned* if we are guarantee not to improve our best known solution by further exploring the subtree rooted at that node.

The first step when processing a node is to solve its relaxation. Next, we check the status of the solution. If the relaxation is not feasible, the node is pruned by *infeasibility*. If all relaxed integer variables were assigned integer values, a solution to the original problem was found, and the node is pruned by *integrality*. Finally, if the lower bound given by the relaxation of a node is larger than the best known integer solution, the node is pruned by *suboptimality*.

If the node was not yet pruned, at least one of its relaxed integer variables was assigned a fractional value. We then call the Cutting Plane algorithms. The goal of a *Cutting Plane* algorithm is to produce constraints, also known as cuts, that are valid for all integer solutions, but are violated by the current fractional solution. If such cut is found, the relaxation is solved again. This process is repeated until no cutting plane is found.

The addition of cutting planes might increase the objective value of the relaxation, or even make it infeasible, leading to the node being pruned. Modern solvers have several classes of general cutting planes implemented, however, problem-specific cuts can be added with the goal of levering the combinatorial properties of the problem.

Then the solver might call heuristics, passing the value of the current relaxation as input. The goal of a *heuristic* is to construct good feasible solutions for the original problem using the information derived from the relaxation of a given node. The upper bound provided by feasible solutions helps prune nodes by suboptimality, even when the solutions are not optimal. As with cutting planes, even though modern solvers are equipped with several general heuristics, problem-specific heuristics can be added.

To concluded the node processing, a branching rule is called. A *branching rule* selects one of the relaxed integer variables, say x, assigned a fractional value x^* by the relaxation and creates two new child nodes. Each child node contains all constraints in the current node, both inherited from its parent, as well as its on cuts, plus additional *branch constraints*. For one node the branch constraint is $x \leq \lfloor x^* \rfloor$, while for the other it is $x \geq \lceil x^* \rceil$. These branch constraints ensure that the two nodes correspond to two disjoint subproblems, and neither contains the solution to the relaxation of its parent. A good branching rule leads to child nodes with search subtrees of similar size, while also having its cuts increase the lower bound for both child nodes with respect to the parent. Once again, modern solver are equipped with powerful branching rules, but a problem-specific one can be added, leveraging the combinatorial properties of integral solutions.

When processing a node has finished, be it by being pruned or by branching, a new one is selected from the pool of nodes, and the process repeats until the entire tree is implicitly searched.

To summarize, when it comes to improving solver performance, some of the key components of the B&C are: preprocessing, heuristics, cutting plane algorithms and branching rules.

Next, we discuss more advance techniques for solving MILPs.

Row Generation

In order to solve a MILP model, it can be useful to initially omit some of its constraints and work with a reduced MILP. This may occur when there are too many constraints to be included or when they (even if few) significantly increase the solving time.

Consider the following ILP.

$$\begin{aligned} \min \quad & cx && \text{(1.2a)} \\ \text{s.t.} \quad & Ax \geq b && \text{(1.2b)} \\ & A'x \geq b' && \text{(1.2c)} \\ & x \in \mathbb{Z}^+ && \text{(1.2d)} \end{aligned}$$

Let Constraints (1.2c) be the complicating constraints to be omitted. Then, instead of solving Model (1.2), we solve

$$\begin{aligned} \min \quad & cx && \text{(1.3a)} \\ \text{s.t.} \quad & Ax \geq b && \text{(1.3b)} \\ & x \in \mathbb{Z}^+ && \text{(1.3c)} \end{aligned}$$

However, to ensure that the solution found is correct, Constraints (1.2c) must still be enforced. To this end, a separating procedure is called when an integer feasible solution is found for the reduced MILP and it either certifies that the solution is feasible for the original model or provides a set of constraints that were missing and are violated by said solution. This is called the *lazy separation*, and the constraints found are called *lazy constraints*.

Notably, since we are only omitting constraints, a relaxation of the reduced problem still gives a valid, although likely weaker, dual bound, allowing the problem to be solved using traditional B&C, as long as integral feasible solutions found during the search are properly separated.

In Figure 1.9, we show an updated version of the B&C flowchart, including where row generation routines are called.

It is important to highlight the difference between Cutting Planes and Lazy Constraint Separation. While the former is used to improve dual bounds, the later is necessary to ensure that the model provides valid solutions.

In order to improve performance, one might adapt the Lazy Constraint Separation routine to also separate fractional solutions, adding a Cutting Plane routine to find the violated omitted constraints. This ensures that the dual bound provided by the reduced MILP is equivalent to the one of the original MILP. However, the effectiveness of separating fractional solutions is dependent on the MILP being solved.

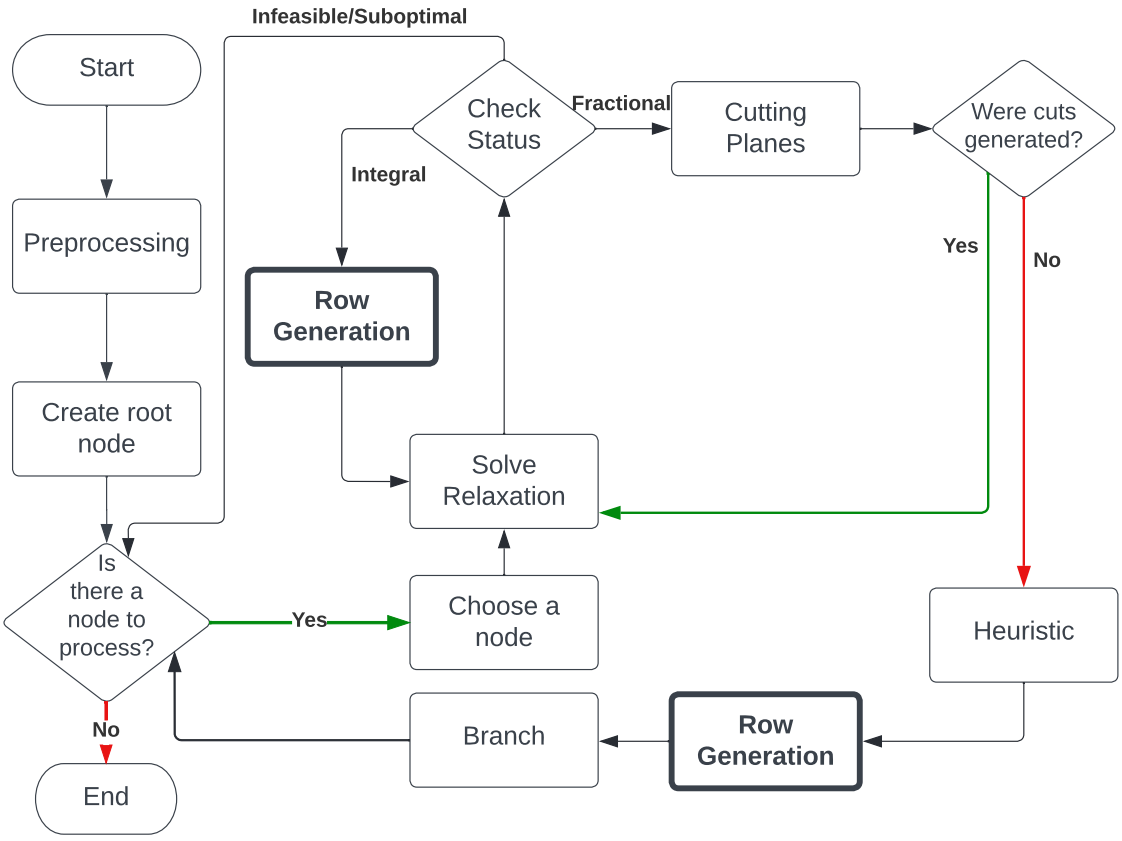


Figure 1.9: Branch-and-Cut Flowchart including Row Generation

Given a solution x^* to Model (1.3), we can naively check for each constraint $a_i'x \geq b_i'$ whether $a_i'x^* \geq b_i'$ is violated.

Alternatively, one can instead solve an optimization problem that implicitly decides whether an omitted constraint is violated. The most well-known application of implicitly lazy constraint separation appears in formulation of routing problems and is known as *Sub-cycle Elimination*[24].

The most notable result in Row Generation applies when lazy constraints are implicitly separated as cutting planes, and can be expressed as follows [23].

Theorem 1. *Given a MILP M, if we omit a set of constraints R such that the remaining of M is polynomial in size and there is a polynomial-time separation routine for relaxation solutions with respect to R, then the relaxation of M can be solved in polynomial time.*

The strength of this result comes from the fact that R can be comprised of exponentially many constraints.

Next, we discuss a very important usage of row generation called *Benders Decomposition*. This decomposition allows us to leverage problem structure to split the variables of the problem.

Let x be a set of integer variables and y a set of continuous variables. A special case of Benders Decomposition, where y is not part of the objective function, applies to the following model.

$$\begin{aligned}
\min \quad & cx & & & & (1.4a)\\
\text{s.t.} \quad & Ax & + By & \geq b & & (1.4b)\\
& Dx & & \geq d & & (1.4c)\\
& & D'y & \geq d' & & (1.4d)\\
& x \in \mathbb{Z}^+, & y \in \mathbb{R}^+ & & & (1.4e)
\end{aligned}$$

This model presents a *block structure* as its constraints are separated into three sets. Constraints 1.4b are the linking constraints, and have both x and y variables, while Constraints 1.4c and 1.4d contain only x and y variables, respectively.

Instead of solving the entire model, we can solve a reduced MILP, called the *master problem*, and rely on an auxiliary problem, simply called *subproblem*, to validate solutions to the master problem.

The master problem is restricted to only x variables and is as follows.

$$\begin{aligned}
\min \quad & cx & (1.5a)\\
s.t. \quad & Dx \geq d & (1.5b)\\
& x \in \mathbb{Z}^+ & (1.5c)
\end{aligned}$$

The subproblem is constructed for a given solution x^* of the master problem as follows.

$$\min \ 0 \tag{1.6a}$$
$$s.t. \ By \geq d - Ax^* \tag{1.6b}$$
$$D'y \geq d' \tag{1.6c}$$
$$y \in \mathbb{R}^+ \tag{1.6d}$$

Since, in the special case that we are interested in, the y variables are not part of the objective function, the subproblem is only responsible for checking feasibility.

By solving the subproblem, we either obtain a valid assignment y^* of y, such that (x^*, y^*) is a solution to Model 1.4 or find a constraint that can be added to it so that x^* is no longer feasible.

There are two standard ways of implementing this decomposition. The first one uses lazy constraints, in such a way that one search tree is used and the subproblem is solved when an integer solution is found. This is known as the *lazy implementation*. One drawback of this approach is that, since the model being solved is not completely known at the start, most preprocessing algorithms must be disabled.

Alternatively, one can solve Model 1.5 to optimality as a separate ILP and then separate the optimal solution. If it does not lead to a feasible solution, a new ILP that includes the violations found by the subproblem is solved almost from scratch. This is done iteratively until either a solution is found or the master problem becomes infeasible. This approach enables preprocessing to be used at each iteration, but, since each iteration requires an ILP to be solved from scratch, a large number of iterations might make the procedure prohibitively time-consuming.

When the variables in the subproblem are indeed continuous, linear programming duality gives us infeasibility cuts for infeasible solutions to the master problem as a byproduct of solving this subproblem as an LP.

However, if one wants to apply the decomposition when the variables in the subproblem are integral, infeasibility cuts must be derived for the specific problem. In this case, the decomposition is called *Logic-Based Benders Decomposition*.

For more details and more general cases of Logic-Based Benders Decomposition, the reader may refer to [11].

Next, we discuss how to handle complicating variables.

Column Generation

In some cases, one might want to omit some of the variables of a model in order to solve it, but it does not have the block structure necessary to apply Benders Decomposition. In this case, a different approach, known as *Column Generation*, works on variables similarly to how Row Generation works on constraints.

Consider the following ILP

$$\begin{aligned} \min \ & cx + c'y && \text{(1.7a)} \\ \text{s.t. } & Ax + \ By \geq b && \text{(1.7b)} \\ & x \in \mathbb{Z}^+, \ y \in \mathbb{Z}^+, && \text{(1.7c)} \end{aligned}$$

and let the y variables be the complicating variables to be omitted.

In this case, we start with a reduced MILP, called the *Restricted Master Problem* (RMP) comprised only of the x variables, as follows.

$$\begin{aligned} \min \ & cx && \text{(1.8a)} \\ \text{s.t. } & Ax \geq b && \text{(1.8b)} \\ & x \in \mathbb{Z}^+, && \text{(1.8c)} \end{aligned}$$

In contrast to Row Generation, any integer solution restricted to the x variables can be extended to the original problem by assigning zero to all y variables. However, given a solution to the relaxation of Model (1.8) it is not guaranteed to be a lower bound for the original problem. In fact, the removal of variables might lead to an infeasible reduced MILP, while the original problem was feasible.

Thus, to obtain a lower bound, an auxiliary problem, known as the *Pricing Problem*, is called when a relaxation is solved in a node. If there are any variables that could improve the value of the relaxation but are missing from the RMP, the Pricing Problem must provide at least one of them. This process is repeated until there are no variables to be added, only then the relaxation is considered solved and its objective value is a valid lower bound for the node problem. Due to the numerical nature of solving the RMP, it is usually said that it *converged* when no improving variable is found.

The pricing uses a solution to the dual of the master problem to construct variables that could improve the objective value, if there are any.

Before continuing, some matrix notation must be introduced. Given an m by n matrix M, let M^t denote its transpose and $M_{:,j}$ denote the n by 1 vector corresponding to its j-th column.

Associating dual variables α to Constraints (1.7b), the dual of (the relaxation of) Model (1.7) is as follows.

$$\begin{aligned} \min \ & \alpha b && \text{(1.9a)} \\ \text{s.t. } & \alpha A^t \leq c && \text{(1.9b)} \\ & \alpha B^t \leq d && \text{(1.9c)} \\ & \alpha \in \mathbb{R}^+, && \text{(1.9d)} \end{aligned}$$

Let $\overline{\alpha}$ be an optimal solution to the dual of the master problem, i.e., Model (1.9) without Constraints (1.9c). The *reduced cost* of a variable y_i w.r.t. $\overline{\alpha}$, denoted by $\overline{c'}_i$, is

given by

$$\overline{c'}_i = c'_i - \alpha B_{:,i}. \tag{1.10}$$

To improve the solution to the current relaxation, one must find variables with negative reduced cost. Since we solved the relaxation of RMP, all x variables have non-negative reduced costs.

For the y variables, let Y be the set of combinatorial objects associated to each of them. A variable with the minimum reduced cost can be found by solving the following presentation of the Pricing Problem.

$$\overline{c'}_{min} = \min_{i:y_i \in Y} c'_i - \alpha B_{:,i} \tag{1.11}$$

Similarly to Row Generation, this problem can be solved by inspection. However, since our goal is to avoid enumerating the set of variables, this problem is usually solved implicitly.

If c'_{min} is negative, we add the corresponding variable, say y_i, to the master problem, which includes adding column $B_{:,i}$ to A. Otherwise, the relaxation of the master problem already has the same value as the relaxation of the original problem.

In the case where the master problem is infeasible, the pricing problem has to be slightly modified, replacing the c' side by 0, as follows [1].

$$\overline{f'}_{min} = \min_{i:y_i \in Y} 0 - \alpha B_{:,i} \tag{1.12}$$

This problem is known as the *Farkas Pricing Problem*, as it is a consequence of the well-known Farkas Lemma. If its solution is non-negative, then the relaxation is indeed infeasible. Otherwise, we add the variable found to the master problem. Feasibility checks become particularly useful when enumerating the search tree, as branching decisions naturally lead to nodes corresponding to infeasible subproblems.

To conclude the presentation of Column Generation, we discuss some difficulties that may arise when using it and how to minimize their impact.

Firstly, the subproblem is called at each node, possibly several times until the relaxation is solved and a valid dual bound is obtained. For problems for which the Pricing Problem is time-consuming to be solved to optimality, it is possible to to use heuristics to find variables with a negative, but not necessary minimum, reduced cost. It might also be beneficial to add multiple variables at once to speed up convergence.

Secondly, any constraints added to the master problem, be them from cutting planes or even from branching, incur a new dual variable. If some of the omitted variables have non-zero coefficients in these newly added constraints, a new dual variable is introduced into the pricing problem. The introduction of new dual variables modifies the pricing problem and has to be handled carefully.

Thus, to include column generation into a B&C framework, one needs to ensure that the algorithm used to solve the subproblem can handle the branching rule and the cutting planes included.

Alternatively, the study of cuts that do not involve the omitted variables, known as *Robust Cuts*, removes this difficulty. An example of the usage Robust Cuts is presented

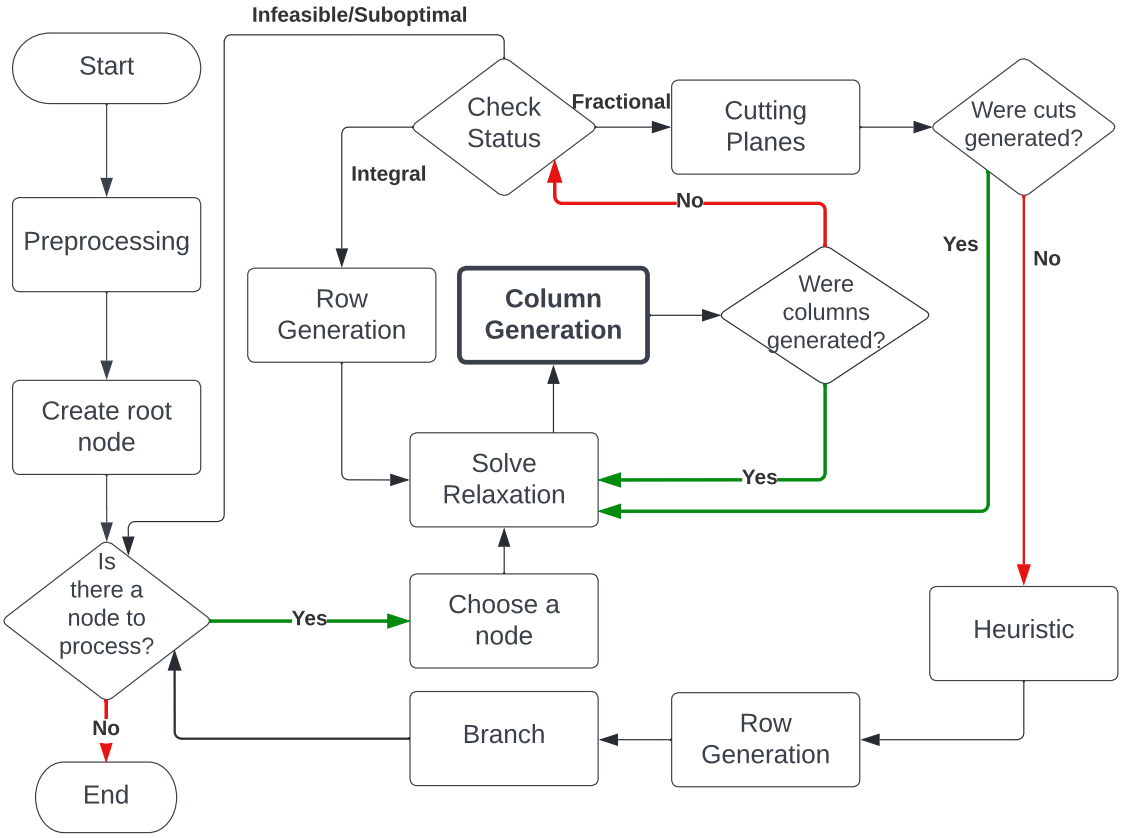


Figure 1.10: Branch-Cut-and-Price Flowchart

in Chapter 3.

The resulting algorithm, which may even contain row generation, is called *Branch-Cut-and-Price* (BC&P). Its execution is illustrated in Figure 1.10.

Similarly to Row Generation, Column Generation disables most preprocessing, as the model is not entirely known at the start.

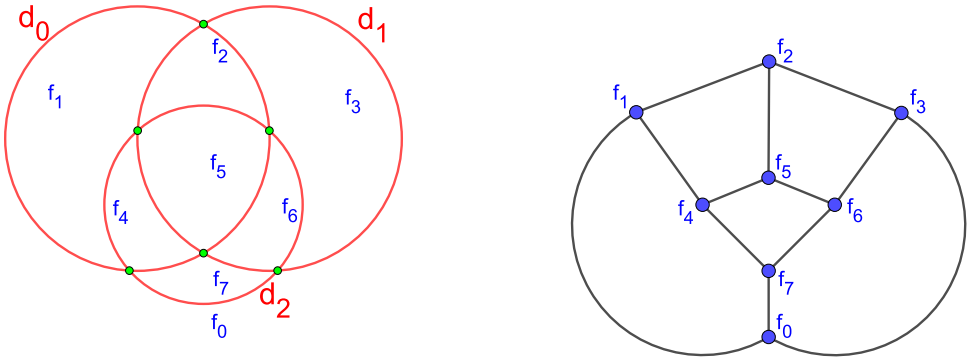
The reader may refer to [4] for a comprehensive introduction to the subject. Alternatively, a gentler introduction can be found in [24].

1.1.2 Basics of Computational Geometry

Fundamentals of computational geometry can be found in [6]. Most of the necessary concepts were already presented in the beginning of this chapter as the problems were first introduced. We are only missing one key geometric structure used to write ILP models for geometric problems: arrangements.

Arrangements

A finite set of curves in the plane, such as lines, line segments or circles, induces a planar subdivision which partitions the plane into an *arrangement*. The zero-, one- and two-dimensional objects of an arrangement are its vertices, edges and faces, respectively, and



(a) Arrangement of circles. Its vertices are shown in green, its edges in red and its faces in white.

(b) Dual of an arrangement of circles. Its vertices are shown in blue and its edges in black.

Figure 1.11: Illustration of an arrangement of circles $\{d_0, d_1, d_2\}$, which has faces $\{f_0, f_1, f_2, f_3, f_4, f_5, f_6, f_7\}$ and its dual.

are called the *elements* of the arrangement. An example of a arrangement of circles is shown in Figure 1.11a.

The dual of an arrangement is a graph with a vertex for each face in the arrangement and an edge connecting pairs of vertices that represent faces that share an edge in the arrangement. The dual of the arrangement in Figure 1.11a is shown in Figure 1.11b.

Both the arrangement and its dual can be traversed very efficiently. For the problems studied in this dissertation, the number of elements in the arrangements is quadratic on the number curves used to construct it.

This discretization of the plane into elements of an arrangement enables us to express problems using constraints and even variables associated to said elements. For example, in Figure 1.11a, each point in the interior of a given face is covered by the same set of disks.

Theoretical aspects of geometric arrangements can be found in [10], while more practical discussions, including implementations, are presented in the documentation of the CGAL Library [21]. This library provides implementation not only of arrangements but also of several algorithms and data-structures commonly used in CG.

1.2 Contributions

A goal of this dissertation is to develop tools to tackle geometric decomposition problems that can be used for solving other similar hard geometric problems. To this end, we investigated ILP models for the problems studied, together with theoretical and empirical results obtained by solving these models, as listed below.

Minimum Convex Partition of Point Sets (MCPP) Our earliest paper presented the first ILP model for the MCPP problem, which was graph-based, and derived heuristics based on its relaxation. Our second paper presented an arrangement-based model, which was able to solve instances with twice as many points as the previous model when run on

similar hardware. This model included a data structure and a pricing algorithm that can be generalized to other problems involving convex hull partitioning. We also incorporated the graph-based variables from the first model, adapting cuts and heuristics proposed for it. This combination allowed us to draw connections between geometric aspects of the problem and classical results in set-partitioning. The final result is a framework that can be adapted to solve additional convex hull partitioning problems.

Convex Quadrangulation of Bichromatic Point Sets with Minimum Flips (CQBPS) We introduced this new optimization variation of the quadrangulation problem where we want to minimize color flips. We presented two ILP models for it; the best one of which handles both convex hull partitioning and color assignment as a pure set-partition problem. We used the convex hull partitioning constraints introduced for the the MCPP and also presented several heuristics, based on the ones introduced for the MCPP in our previous papers. These new heuristics draw connections with matching approaches from the quadrangulation literature.

Coarseness We presented a novel ILP model to compute the coarseness of a bicolored point set and proposed several improvements to it. This model was implemented using column generation and its relaxation could be solved in polynomial time. This relaxation was used in an iterative preprocessing algorithm to find successively tighter lower bounds and used these bounds to remove variables. When combined with a heuristic from the literature, this polynomial-time iterative procedure was find optimal solutions of large portion of our benchmark.

Minimum 3-Colorable Discrete Unit Disk Cover (M3CDUDC) We introduced the first ILP model and the first known exact algorithm for the M3CDUDC problem. Our best implementation uses row generation in the form of Logic-Based Benders Decomposition. We provided logic-based feasibility cuts, and presented several improvements to make them stronger. The resulting algorithm solved instances with thousands of disks and points, which compares very favorably with the rather impractical approximation algorithms from the literature. We also proved that the first iteration of our decomposition, which corresponds to a pure set cover problem, produces solutions with a number of colors bounded by a constant times the minimum number of colors among all covers.

Overall, we show in this dissertation how geometric decomposition problems can be solved efficiently in practice by either row or column generation, with a classical decomposition problem as a master problem and a subproblem that can better take advantage of the geometry. We also show that using the geometry of the problem in other components of the branch-and-cut, or branch-cut-and-price, algorithms, such as heuristics, branching rule or preprocessing, is key for the success of these techniques.

1.3 Text Organization

This dissertation is presented in an alternative format introduced by the University of Campinas some years back. In this format, the Introduction is followed by one chapter for each paper either published or submitted during the doctorate program and its respective

bibliography, while the last chapter presents concluding remarks. Chapter 2 and Chapter 3 correspond to our studies of the MCPP. Chapter 4 presents our work on the CQBPS. Next, Chapter 5 shows our results for the Coarseness Problem. Finally, Chapter 6 presents our results for the M3CDUDC. A final discussion is presented in Chapter 7, followed by a bibliography containing the works cited in the introduction and conclusion.

Chapter 2

Minimum Convex Partition of Point Sets

In the following, we present the first empirical study of the Minimum Convex Partition of Point Sets. We present an ILP model based on constructing a planar subdivision and derive heuristic from it. This chapter corresponds to the work [20] presented at the *11th International Conference on Algorithms and Complexity (CIAC 2019)* [20]. It was co-authored with Pedro J. de Rezende[1] and Cid C. de Souza.[1]

Abstract A convex partition of a point set P in the plane is a planar subdivision of the convex hull of P whose edges are segments with both endpoints in P and such that all internal faces are empty convex polygons. In the Minimum Convex Partition Problem (MCPP) one seeks to find a convex partition with the least number of faces. The complexity of the problem is still open and so far no computational tests have been reported. In this paper, we formulate the MCPP as an integer program that is used both to solve the problem exactly and to design heuristics. Thorough experiments are conducted to compare these algorithms in terms of solution quality and runtime, showing that the duality gap is decidedly small and grows quite slowly with the instance size.

2.1 Introduction

Let P be a set of n points in the plane in general position, i.e., with no three points being collinear. We say that a simple polygon is *empty*, w.r.t. P, if it contains no points of P in its interior. Denote by $H(P)$ the convex hull of P. A convex *partition* (or *decomposition*) of P is a planar subdivision of $H(P)$ into non overlapping empty convex polygons whose vertices are the points of P. The Minimum Convex Partition Problem (MCPP) asks to find a convex partition of P minimizing the number of faces. These concepts are illustrated in Fig. 2.1.

A practical application of the MCPP in the area of network design is described in [7]. The goal is to form a communication network connecting the points of P. When the edges

[1]Institute of Computing, University of Campinas, Campinas, Brazil

Figure 2.1: (a) Point set P; (b) $H(P)$; (c) minimal and (d) optimal partition.

used as links in the network form a convex partition of P, a simple randomized algorithm can be used for routing packages [2]. Hence, one way to build a low-cost network, i.e., one with few links that still enables the application of that routing algorithm, is to solve the MCPP for P.

Besides the actual application outlined above, the MCPP lies in the broader context of polygon decomposition, an important topic of study in computational geometry [6]. A frequently used approach for designing divide-and-conquer algorithms for geometric problems for general polygons is to decompose these into simpler component parts, solve the original problem on each component using a specialized algorithm and then combine the partial solutions. Convex polygons are often the best choice for the role of the smaller components. This is, in fact, a reason why triangulations of sets of points, and of polygons in general, have been so extensively studied and became a central problem in computational geometry. Since triangulations are special cases of convex partitions, a deeper understanding of the MCPP gains importance.

Literature overview. To the best of our knowledge, the complexity of the MCPP remains unknown. Moreover, even though some articles describe non-polynomial algorithms to solve the problem, no implementations or results were found that make an empirical assessment of the efficiency of those algorithms.

Fevens et al. [3] proposed an exact algorithm for the MCPP using dynamic programming. If h denotes the onion depth of P, i.e. the number of nested convex hulls that need to be removed before P becomes empty [8], the complexity of this algorithm can be expressed as $O(n^{3h+3})$ and is, therefore, exponential in h, which can actually be as large as $\Theta(n)$. Spillner [10] designed another exact algorithm whose complexity is $O(2^k k^4 n^3 + n \log n)$, where $k = |I(P)|$ is the number of points of P in the interior of $H(P)$. Moreover, Spillner et al. [7] present a $\frac{30}{11}$-factor approximation algorithm for the MCPP.

Other papers investigate the MCPP in an attempt to find theoretical bounds for the optimal value. Let $Inst(n)$ be the set of all MCPP instances of size n. Denoting the optimal value of an instance $i \in Inst(n)$ by $\mathrm{OPT}(i)$, define $F(n) = \max_{i \in Inst(n)} \mathrm{OPT}(i)$, that is, $F(n)$ is the maximum among all optima for instances of size n. The best known lower bound for $F(n)$ is $\frac{12}{11}n - 2$ as shown in [4], for $n \geq 4$, whereas the best upper bound proven to date is $\lceil \frac{7(n-3)}{5} \rceil$, see [5]. It is shown in [9] that any minimal convex partition has at most $\frac{3(n-2)}{2}$ faces, which also serves as an upper bound on $F(n)$.

Our contribution. In this paper, we introduce the first known integer linear programming formulation of the MCPP. Through extensive experimentation, we show that the solutions of the linear relaxation of this model provide invaluable information on which segments are likely to be in an optimal solution of the problem. From this observation, we derive a powerful heuristic for the MCPP that is capable of producing high quality

solutions for instances with up to one hundred points in no more than a few minutes of computation on a currently standard machine. We hope that the present work will spearhead efforts of researchers in the field to publish computational evaluation of algorithms for the MCPP. To this end, all instances we used and their solutions, are made available [1].

Organization of the text. The next sections of the paper are organized as follows. Section 2.2 describes the computational environment in which our tests were carried out. Section 2.3 presents an integer programming formulation for the MCPP and reports on experiments with this model. Section 2.4 is devoted to the discussion of the heuristics developed in this work, whereas in section 2.5 we analyze the results yielded by these algorithms. Section 2.6 contains a few conclusions and points out future research directions.

2.2 Experimental Environment

In the following sections, we propose exact and heuristic algorithms for the MCPP and report on their experimental evaluation. Since observations made from initial empirical results had an impact on the design of these algorithms, it is necessary to first introduce the computational environment where tests were carried out and to describe how the benchmark instances were created.

Software and Hardware. All experiments were run on an Intel Xeon E5-2420 at 1.9GHz, and 32GB of RAM running Ubuntu 14.04. The algorithms were implement in C++ 11 and compiled with GCC 7.2.0. Geometrical structures and procedure were implement using CGAL 4.2-5, library Gmpq was used to represent rational numbers exactly. To solve integer linear programs and relaxations, we used CPLEX 12.8.0, in single-thread mode, with default configurations, except for a time limit of 1200 seconds.

Instances. Instances consist of points whose x and y coordinates were randomly generated in the interval $[0, 1]$ according to a uniform distribution. For an instance of size n, points were created in sequence until n points in general position were included. To that end, when a newly spawned point resulted collinear with a previously generated pair the former was rejected, otherwise it was accepted.

Two sets of instances were generated, with different instance sizes. For each size, 30 instances were created. The set $T1$ is comprised of instances of sizes 30, 32, 34, ..., 50. For each size, we chose the first 30 found to be optimally solvable within 20 minutes. The second set, $T2$, simply contains instances of sizes 55, 60, 65, ..., 110 for which optimal solutions are still not known.

2.3 A Mathematical Model for the MCPP

In this section, we propose an Integer Linear Programming (ILP) formulation for the MCPP and discuss its correctness and performance on the randomly generated instances described in Section 2.2.

Model Description. Before describing the model itself, we need to introduce some terminology and notation. Recall that P denotes a set of n points in general position. The set of *internal points* of P, denoted $I(P)$, is the subset of P formed by the points

that are not vertices of the convex hull of P, $H(P)$. Let S denote the set of $\Theta(n^2)$ line segments whose endpoints belong to P. Given a pair of segments $\overline{ij}, \overline{k\ell} \in S$, we say that $\overline{ij}$ and $\overline{k\ell}$ *cross* when $\overline{ij} \cap \overline{k\ell} \setminus \{i, j, k, \ell\} \neq \emptyset$.

Consider the complete (geometric) undirected graph $G = (P, E(P))$, induced by P, where $E(P) = \{\{i, j\} \mid \overline{ij} \in S\}$. We will refer to an edge $\{i, j\}$ of $E(P)$ and its corresponding segment $\overline{ij}$ in S interchangeably. We will also need to allude to the set of pairs of crossing edges (segments), denoted S^C. These pairs, $(\overline{ij}, \overline{k\ell}) \mid \overline{ij}$ and $\overline{k\ell}$ cross, may easily be identified in $O(n^4)$ time using simple geometric procedures. Similarly, a complete directed graph $\overrightarrow{G} = (P, A(P))$ can be defined, whose arcs correspond to the segments of S with either one of the two possible orientations: $(i, j) \in A(P)$ iff $\overline{ij} \in S$.

A few additional terminologies are needed before we can present the model and argue about its correctness. Given two points a, b in the plane, we denote by CCW(ab) (CW(ab)) the set of points c in the plane such that the triple abc is positively (negatively) oriented, i.e., $0° < \sphericalangle abc < 180°$ ($-180° < \sphericalangle abc < 0°$). Let a be a point in the plane and L be a list of non collinear segments, all sharing a as one of their endpoints. Assume that $|L| \geq 2$ and that its segments are given in a *clockwise circular order* around a. Point a is called *reflex with respect to* L if there are two consecutive segments in L, say ba and ca, so that $c \in$ CW(ba). Notice that this is equivalent to say that CCW(ac) contains no endpoints of segments in L. With respect to L, when a is not reflex, we call it *convex*.

Lastly, to each edge $\{i, j\} \in E(P)$, we associate a binary variable x_{ij} whose value is one iff $\{i, j\}$ is in the solution. In the formulation below, the unique variable associated to the segment $\overline{ij}$ is naturally referred to as x_{ij} and as x_{ji}. Accordingly, the proposed ILP model, referred to as BASIC, reads:

$$z = \min \sum_{\{i,j\} \in E(P)} x_{ij} \tag{2.1a}$$

$$\text{s.t.} \quad x_{ij} + x_{k\ell} \leq 1 \qquad \forall \{\{i, j\}, \{k, \ell\}\} \in S^c \tag{2.1b}$$

$$x_{ij} = 1 \qquad \forall \{i, j\} \in H(P) \tag{2.1c}$$

$$\sum_{k \in \text{CCW}(ij) \cap P} x_{ik} \geq 1 \qquad \forall (i, j) \in A(P), i \in I(P) \tag{2.1d}$$

$$\sum_{j \in P} x_{ij} \geq 3 \qquad \forall i \in I(P) \tag{2.1e}$$

$$x_{ij} \in \{0, 1\} \qquad \forall \{i, j\} \in E(P) \tag{2.1f}$$

The objective function (2.1a) can be expressed in terms of the number of edges in a solution because Euler's formula implies that minimizing the number of edges in a connected planar graph (subdivision) is equivalent to minimizing the number of faces of any planar embedding of that graph. That is, if f, e and v denote, respectively, the number of faces, edges and vertices of an embedding of a planar graph, then $f = e - v + 2$. Since we seek to build a planar connected subdivision and $v = |P|$ is given, minimizing e is equivalent to minimizing f.

Constraints (2.1b) guarantee planarity since they prevent that both edges of a crossing pair be included in a solution. Constraints (2.1c) establish that the edges of the convex

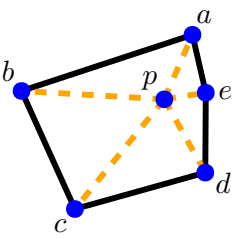


Figure 2.2: An instance of the MCPP whose solution of the the RELAX model without constraints (2.1e) has a vertex p with fractional degree 2.5. Dashed edges have value 0.5 and solid edges have value 1.

hull of P are included in the solution. Constraints (2.1d) ensure that any point i in $I(P)$ is convex with respect to the set of segments that are in the solution. As shown in Proposition 1, the degree constraints (2.1e) are redundant when we consider integer solutions and constraints (2.1d). However, we opted to keep them in the formulation because we verified that they improve the value of the relaxation. Figure 2.2 depicts an instance whose solution of the relaxation of the BASIC model without constraints (2.1e) has an internal point with *fractional* degree less than three. Finally, constraints (2.1f) require all variables to be binary.

Proposition 1. *Let x^L be the incidence vector of a subset L of segments in S satisfying all constraints* (2.1d)*. Then, x^L also satisfies constraints* (2.1e)*.*

Proof. Denote by $G(L)$ the subgraph induced by L in G. Due to constraints (2.1d), the degree in $G(L)$ of any point $i \in I(P)$ is at least one, meaning that, at the minimum, there is one segment $\overline{ia}$ in L for some $a \in P$. Now, taking $j = a$ in constraint (2.1d), we obtain that there must be another point b in $\text{CCW}(ia) \cap P$ for which $\overline{ib}$ is also in L. Hence, i has degree at least two. By construction, a must be in $\text{CW}(ib)$. Together with constraint (2.1d) for (i, b), this implies that there must be a third point c in $\text{CCW}(ib) \cap P$ so that the segment $\overline{ic}$ is in L. □ □

Theorem 2. *The* BASIC *model is a correct formulation for the* MCPP*.*

Proof. First, we show that the incidence vector x^L of any set L of segments in S that corresponds to a feasible solution of the MCPP satisfies all the constraints of the BASIC model. By definition, x^L is a binary vector and hence, constraint (2.1f) holds. The feasibility of L implies planarity, so x^L satisfies all constraints (2.1b). Besides, as the segments in the convex hull of P are all present in any feasible solution, constraints (2.1c) are also verified by x^L. Proposition 1 implies the fulfillment of constraints (2.1e). It remains to prove that x^L satisfies constraints (2.1d) as well. Let L_i be the clockwise ordered subset of segments of L that are incident to $i \in I(P)$. The feasibility of L requires i to be convex with respect to L_i, or else i would be a reflex vertex of a face in the planar subdivision corresponding to L, contradicting the hypothesis that the set is a feasible solution for the MCPP. As a consequence, for any direction $\overrightarrow{u}$, the set $\text{CCW}(i(i + \overrightarrow{u}))$[2] contains at least one endpoint of a segment in L_i. Hence, for any $(i, j) \in A(P)$, at least one variable x^L_{ik} in the summation on the left of constraint (2.1d) has value one, ensuring that the inequality is satisfied by x^L.

[2]Here, $(i + \overrightarrow{u})$ denotes any point obtained by a translation of i in the direction $\overrightarrow{u}$.

We now focus on the proof that the set L associated to any incidence vector x^L satisfying the linear system (2.1b)–(2.1f) is a convex partition of P. Let $G(L)$ denote the subgraph of G induced by L. Because of (2.1b), L is planar and, due to (2.1c), it contains all the segments in $H(P)$. From Proposition 1, the degree of any point $i \in I(P)$ in $G(L)$ is at least three. For L_i defined as before, we claim that i is convex with respect to L_i. To see this, suppose by contradiction that i is reflex. So, w.l.o.g, there is a segment $\overline{ia}$ in L_i such that, for any b with $\overline{ib} \in L_i$, b is in $\mathrm{CW}(ia)$. This implies that there is no $k \in \mathrm{CCW}(ia) \cap P$ so that $\overline{ik}$ is in L. Consequently, x^L does not satisfy constraint (2.1d) for (i, a), contradicting the feasibility of x^L. We next show that $G(L)$ is connected.

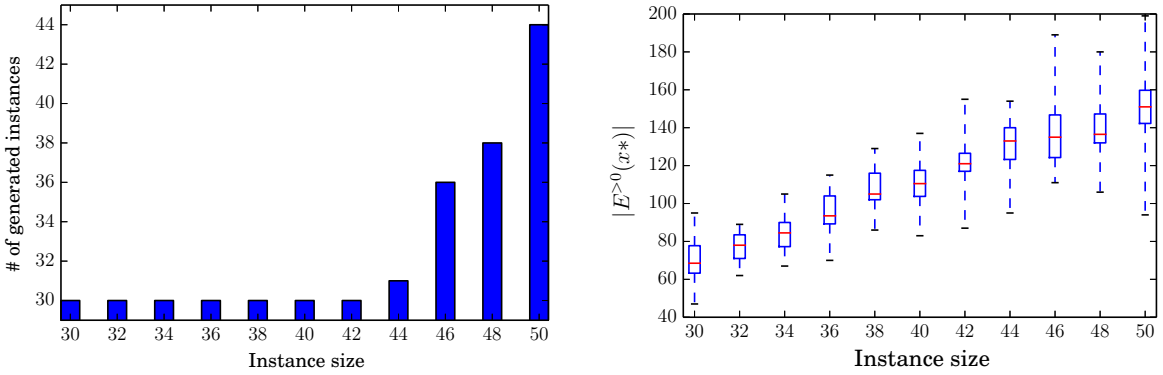
By contradiction, suppose that $G(L)$ has two or more connected components. Since L contains the segments in $H(P)$, the points in at least one of these connected components must all be internal. Let i be the rightmost point in this component. Since i is not connected to any point to its right and has degree at least three, i must be reflex relative to L. But then, constraint (2.1d) for (i, j) is violated by x^L, where $\overline{ij}$ is the first segment visited when L_i is traversed starting from a vertical line that goes through i.

On the other hand, we can also prove that $G(L)$ contains no articulation points (and hence, no bridges) by applying arguments similar to those employed above. It then follows that each segment in $L \setminus H(P)$ is incident to exactly two (distinct) faces of the planar subdivision defined by L and that all faces in this subdivision are empty polygons. Moreover, as all points in $I(P)$ are convex w.r.t. L, these polygons are convex. The minimization condition is ensured by (2.1a). □ □

Empirical Evaluation. We now describe the empirical evaluation of the BASIC model for the benchmark set $T1$. Recall that to construct the 330 instances in $T1$, we sought to have only instances for which an optimal solution could be found. To achieve that, a sequence of instances was generated for each of the 11 intended sizes and we attempted to solve them to optimality within the time limit of 20 minutes of computation. Those instances for which this process was successful were kept and the failed ones were discarded and replaced.

As a form of probing the space of possible instances in regard to how much harder it gets, as sizes increase, for finding instances that are solvable within our time limit, consider Fig. 2.3a. It shows how many instances had to be generated until a set of 30 instances could be found that satisfied our criteria. One can see that the first 30 instances of sizes between 30 and 42 were all solvable. Differently, from size 44 onwards, some instances timed out and had to be replaced. Clearly, solvable instances within our time frame become rarer as the size increases.

One of the major drawbacks for the performance of the BASIC model is the huge number of crossing constraints (2.1b), which is $O(n^4)$. For instances with 100 points, the formulation is too big even to be loaded into memory. A conceivable approach in this case is to implement a branch-and-cut algorithm to compute the BASIC model where the crossing constraints are added as they are needed. Initially, no constraints (2.1b) are included in the model. The processing of a node in the enumeration tree involves the execution a cutting-plane algorithm that is composed of the following steps. In the first one, the current linear relaxation is computed. Then, a separation routine is run that



(a) (b)

Figure 2.3: (a) Number of instances generated until 30 were solved exactly, and (b) cardinality of the support of $x^\star$, as functions of the instance size.

checks for crossing constraints that are violated by the optimal solution. The latter can be done by inspection in $O(n^4)$ time and, if violated constraints are found, they are added to the current relaxation and the previous steps are repeated; if not, the processing of the node halts and branching takes place provided that the current solution is not integral.

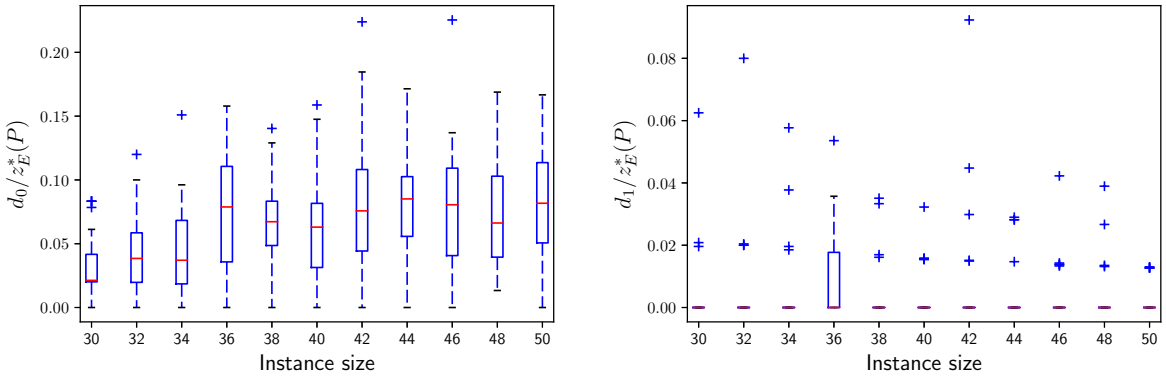
Having implemented and tested this algorithm, we noticed that the computation of the linear relaxation via the cutting-plane algorithm was faster than solving the complete relaxed model. This proved helpful for developing our heuristics. However, considering the imposed time limit and the size of the tested instances, the branch-and-cut algorithm as a whole was slower than the standard branch-and-bound algorithm used by CPLEX when applied to the full model.

The difficulty in computing optimal solutions motivated us to design heuristics for the MCPP. The starting point for the development of such algorithms came from observations on the solutions of the relaxation of the BASIC model.

Given a vector $y \in \mathbb{R}^{|E(P)|}$, the *support* of y is the set of edges of $E(P)$ for which the corresponding variables have positive values in y. Now, let RELAX be the linear relaxation of the BASIC model, $x^\star$ an optimal solution of RELAX and $E^{>0}(x^\star)$ the support of $x^\star$. Similarly, let $E^{=1}(x^\star)$ be the subset of edges of $E(P)$ corresponding to the variables of value one in $x^\star$.

With these definitions, we are ready to analyze the quality of optimal solutions of the linear relaxation of the BASIC model computed by CPLEX. We consider the support of $x^\star$ to be *good* if it is small-sized and there exists an optimal (integer) solution of BASIC whose support intersects $E^{>0}(x^\star)$ for a large number of edges. In that vein, Fig. 2.3b shows the average number of edges in $E^{>0}(x^\star)$ for instances in the set $T1$. Notice that this value grows linearly in n, even though there are $\Theta(n^2)$ edges in $E(P)$. Since any convex partition of P has $\Omega(n)$ edges, $E^{>0}(x^\star)$ is indeed small-sized.

To estimate the quality of the support of the solutions found by CPLEX, we modify the BASIC model to look for an optimal solution with the largest intersection with $E^{>0}(x^\star)$. To achieve this, the weights of the variables in the objective function are changed. A null weight is assigned to edges in $E^{>0}(x^\star)$ while a weight of one is assigned to all remaining variables. To ascertain that the solution found is optimal for the MCPP, we add the



(a) (b)

Figure 2.4: Ratio between the minimum number of (a) zeros in $x^\star$ that changed to ones in an optimal solution of value d_0; (b) ones in $x^\star$ that changed to zeros in an optimal solution of value d_1, and the optimum ($z_E^*(P)$) of the MCPP.

following constraint: $\sum_{e \in E(P)} x_e = z_E^*(P)$, where $z_E^*(P)$ is the number of edges in a minimum convex partition of P. Objectively, we seek an optimal solution having as few edges not in $E^{>0}(x^\star)$ as possible. Let $d_0(P)$ be the optimum of the modified model.

Figure 2.4a shows the ratio between $d_0(P)$ and $z_E^*(P)$ obtained for each size for the instances in $T1$. Note that this value is quite small, with an average of less than 10%. This suggests that the support of $x^\star$ contains most of the edges present in some optimal solution.

Another important observation for the development of heuristics for the MCPP concerns the number of edges in $E^{=1}(x^\star)$ that do not appear in an optimal solution. To assess this quantity, we modify the BASIC model again but this time to find a minimum convex partition of P that uses as many ones from $x^\star$ as possible. As before, we add the constraint $\sum_{e \in E(P)} x_e = z_E^*(P)$ to ensure that the solution found is an optimal solution for the MCPP. Also, we alter the objective function by replacing it with $|E^{=1}(x^\star)| - \sum_{e \in E^{=1}(x^\star)} x_e$. This function computes the number of variables that have value one in $x^\star$ but not in the solution of the new model. Hence, minimizing its value is tantamount to obtaining an optimal convex partition with as many edges in $E^{=1}(x^\star)$ as possible.

Now, let $d_1(P)$ be the optimal value of the latter model. Figure 2.4b displays the ratio between $d_1(P)$ and $z_E^*(P)$ obtained for each size for the instances in $T1$. As can be seen, the ratios are minute, meaning that a very small fraction of the edges in $E^{=1}(x^\star)$ are not present in some minimum convex partition.

Inspired by the two previous remarks, we decided to develop heuristics for the MCPP based on the optimal solutions of the RELAX model. Accordingly, we explored such solutions prioritizing the use of edges in their support, noticeably those at value one. The next section explains how this is done.

2.4 Heuristics

We now describe our proposed heuristics for the MCPP. The general framework of the heuristics is summarized by the following steps:

Step 1: Solve the RELAX model and let $x^\star$ be the optimal solution found;

Step 2: Using $x^\star$, build a subset B of $E(P)$ that induces at least one convex partition of P and, for the sake of efficiency, having $O(n)$ crossings;

Step 3: Find a smallest convex partition S^h of P in B and return this solution.

From the discussion in the previous section, in **Step 2** we try to use the information conveyed by $x^\star$ to select edges that have a high probability of belonging to a minimum convex partition. We devise different procedures to construct the set B containing these edges, which results in the heuristics that are detailed below. The main strategy here is to pack B with more edges than are really needed to obtain a convex partition of P and then, in **Step 3** to select among them a subset of minimum size that is a feasible solution for the MCPP.

Steps 1 and **3** are addressed in the same way in all heuristics developed in this work. **Step 1** corresponds to solving the RELAX model, in our case using CPLEX, to obtain $x^\star$. As for **Step 3**, S^h is computed by solving a restricted version of the BASIC model where the variables x_{ij} for (i, j) not in B are removed from the model (equivalently, one could set them to zero *a priori*). It only remains to decide how the set B is to be built in **Step 2**. Alternatives are proposed below to accomplish this task giving rise to four distinct heuristics.

The GREEDY Heuristic. In this heuristic, the set B in **Step 2** is constructed as follows. Initially, B is empty and the edges in $E(P)$ are organized in a sorted list σ in non increasing order of their corresponding value in $x^\star$. Ties are broken by the number of times the edge crosses the support of the relaxation; edges with less crossings appear earlier in σ. Then, a triangulation of P is constructed iteratively in a greedy fashion. At each iteration, the next edge e in σ is considered. If it does not cross any of the edges already in B, the set is updated to $B \cup \{e\}$; otherwise, B remains unchanged. Clearly, the greedy strategy prioritizes the edges in the support of $x^\star$, which is in consonance with the results seen in Section 2.3. In the end, since all edges in $E(P)$ have been considered, B must determine a triangulation of P. Thus, the BASIC model relative to B computed in **Step 3** has no constraints of the form (2.1b). In the experiments reported at the end of this section, it is shown that the computation of this restricted version of the BASIC model does not compromise the efficiency of the GREEDY heuristic. In fact, the algorithm turned out to be remarkably fast.

The MAXSUP Heuristic. The previous heuristic constructs B as the set of edges of a greedy triangulation of P by favoring edges in the support of $x^\star$ which, as seen, are more likely to be in a minimum convex partition. In a similar mode, the MAXSUP heuristic provides an alternative way for obtaining B. The set is initialized with a convex partition of P having the largest possible intersection with the support of $x^\star$. Next, edges are added to B in a greedy fashion until a triangulation is formed whose associated BASIC model is computed in **Step 3**.

The difficulty with this approach lies on obtaining a convex partition of P that maximizes the number of edges that are in the support of $x^\star$. This can be accomplished by modifying the costs of the variables in the objective function of the BASIC model and changing the problem into one of maximization. In the new formulation, the costs of the variables associated to edges in the support of $x^\star$ are set to one and all others to zero. Besides, variables related to edges in $E^{=1}(x^\star)$ are fixed to one. Since this ILP has

fewer variables with positive costs, in practice, it can be computed faster than the original model.

Adding Flips to a Triangulation. In an attempt to improve the results from the heuristics even further, we decided to forgo the planarity requirements for B in **Step 2**. By this strategy, we expect to increase the search space in **Step 3**, potentially leading to better solutions. Nonetheless, some of the constraints (2.1b) will have to be brought back into the model in **Step 3**, which could severely impact its runtime. To avert this, we strive to keep its size linear in n.

With that intent in mind, assume that initially B determines a triangulation of P. We say that an edge $e \in B \setminus H(P)$ *admits a flip* if the two triangles incident to e form a convex quadrilateral. If e admits a flip, its *flip edge* f_e, which is not in B, corresponds to the other diagonal of this quadrilateral. Clearly, each edge in B has at most one flip edge and, hence, there are $O(n)$ flip edges in total. Let $F(B)$ be the set of flip edges of the edges in B (that admit one). Evidently, $B \cup F(B)$ has $O(n)$ edges and the total number of pairs of edges in this set that cross each other is also $O(n)$. Thus, in **Step 2**, if B is initialized as a triangulation of P and is later extended with its flip edges, in **Step 3** we have to solve a shorter version of the BASIC model with $O(n)$ variables and constraints.

Notice that interchanging an edge of a triangulation and its flip edge gives us an alternative triangulation. Therefore, one can think of the effect of **Step 3** when B is replaced with $B \cup F(B)$ as the computation of the smallest convex partition that can be obtained from a triangulation generated from B by a sequence of flips involving the edges of $B \cup F(B)$. Observe that this amounts to a considerable growth in the search space when compared to the use of a single triangulation. Of course, this is only worthwhile when the computing time does not increase too much while the solution quality improves. As we shall see later, the expected benefit is confirmed in practice.

Motivated by the preceding discussion, we created one new heuristic from each of the previous ones by aggregating flip edges. The enhanced version of GREEDY (MAXSUP) using flip edges is called GREEDYF (MAXSUPF).

2.5 Computational Results

We now assess the results from the four heuristics discussed above comparing them in regard to solution quality and running time. Firstly, we analyze the performance of the heuristics for the instances with known optimum ($T1$). The first row of the table below displays statistics on the mean values, per instance size, of the average relative gap given by $100(z_E^h - z_E^*)/z_E^*$, where z_E^* and z_E^h are the cost of an optimal and of the heuristic solution, respectively. The minimum, average, standard deviation and maximum values are given for each heuristic.

	GREEDY	MAXSUP	GREEDYF	MAXSUPF
mean gap (min/avg±std/max)	0.0/4.2±2.4/12.5	0.0/3.0±1.9/9.0	0.0/1.9±1.6/7.5	0.0/1.1±1.2/6.0
% of instances solved to optimality	7.58	12.12	27.58	43.33

The improvements caused by considering flips are evident. The version of each heuristic including flips reduces the gap by more than half when compared to their original counterparts. When flips are taken into consideration, no gap exceeds 8%. As expected, MAXSUP has a better overall performance relative to GREEDY. The second row of the

table above shows the percentage of instances for which the optimum was found in each case. Again, the benefit of including flips and the superiority of MaxSup over Greedy are clear. In fact, there was no instance where any other heuristic found a better solution than MaxSupF.

The mean values of the average ratio, per instance size, between the time spent by each heuristic and the basic model is given in the next table. On average, the additional computational cost incurred by considering flips is quite small both for Greedy and for MaxSup. Together with the enhancement in solution quality this favors even more the latter strategy. Although both heuristics took just a fraction of the time spent by the basic model, one can see that the Greedy versions used about half the time needed by their MaxSup equivalents. Also, as revealed by the maximum values, the time required by the MaxSup heuristics can surpass that of the basic model. This occurs because these heuristics compute two ilps, and the instances in $T1$ are small.

time ratio relative to basic	Greedy	MaxSup	GreedyF	MaxSupF
mean (min/avg±std/max)	0.3/13.2±13.4/60.6	0.5/28.7±28.7/146.8	0.3/15.5±15.9/82.2	0.5/31.1±31.3/169.3

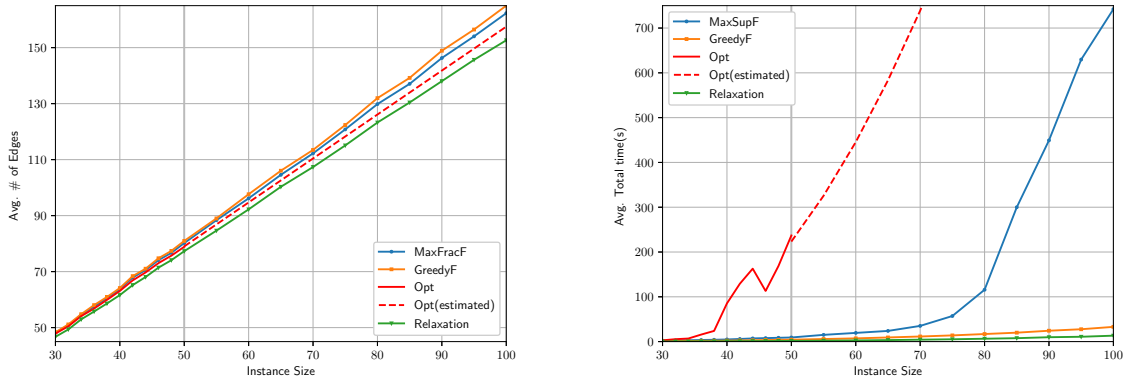
Next, the heuristics are evaluated for the instances in $T2$. While the optimum in this case is unknown, we observed that in all instances of $T2$ the best heuristic solution was again found by MaxSupF. Therefore, for comparison purposes, MaxSupF is used as reference, playing a role similar to that of the basic model in the analysis of the instances in $T1$. The table below exhibits, per instance size, the mean of the average gap between each heuristic and MaxSupF. As before, the minimum, average, standard deviation and maximum values are given.

	Greedy	MaxSup	GreedyF	MaxSupF
mean gap (min/avg±std/max)	0.0/6.0±2.3/14.5	0.0/3.6±1.5/9.0	0.0/1.4±1.4/6.4	0.0/0.0±0.0/0.0

The same pattern perceived for $T1$ is observed here with GreedyF performing slightly worse than MaxSupF, while the versions with flips outstrip by far their original counterparts. Mean solving times relative to MaxSupF for all heuristics are shown in the next table. Once again, we observe the insubstantial impact of the inclusion of flips in the running time of the algorithms. The fact that the MaxSup versions are almost four times slower on average than their Greedy equivalents is remarkable. This is due to the fact that the computation of the additional ilp in MaxSup to find the triangulation with the largest intersection with $x^\star$ consumes too much time as the instance size grows.

time ratio relative to MaxSupF	Greedy	MaxSup	GreedyF	MaxSupF
mean (min/avg±std/max)	1.7/25.3±16.2/55.2	44.0/98.4±14.1/264.5	1.7/27.0±17.5/59.4	100.0/100.0±0.0/100.0

From the previous discussions, we are left with two competitive heuristics: GreedyF and MaxSupF. In Fig. 2.5a, we compare the solutions found by MaxSupF and GreedyF with the dual bound given by the relax model and the optimal solution, when available. We also extrapolate the partial average function corresponding to the optimal solution for the instances with sizes equal to those in $T2$ and, as expected, the extrapolation corresponds to a linear function since the optimum of the mcpp for a point set P is in $\Theta(|P|)$. The graph reveals that the quality of the heuristic solutions deteriorates quite slowly as the instance sizes increase. In particular, the distances between the values of the MaxSupF and GreedyF solutions and between the dual bound and the (estimated) optimum seem to grow at no more than a constant rate, as the instance sizes get larger. With this in mind, one can estimate the error incurred by the algorithm



(a) (b)

Figure 2.5: (a) Average number of edges in the solution; (b) Runtime (seconds)

when executed on an instance of a given size.

Similarly, we analyze the total time spend by MaxSupF and GreedyF in Fig. 2.5b, comparing them to the total time to solve the model and to find optimal solutions for the instances in $T1$. We notice an erratic behavior for the BASIC model at around size 45. This correlates to the difficulty of finding harder instances that are still solvable to optimality within our preset time limit above size 42, as shown in Fig. 2.3a. This suggests that the estimating curve should actually grow even faster if no instances had to be discarded.

The MaxSupF heuristic eventually becomes less efficient since it involves the solution of two ILPs. However, it still remains much faster than computing a true optimum. Thus, if more running time is allowed, and solution quality is a prime concern, MaxSup should be considered an effective alternative when solving the MCPP. On the other hand, we can see that, on average, running RELAX and GreedyF are both very fast. In fact, at the expense of a small loss in quality, the Greedy approach takes less than one minute, on average, to find solutions for random instances of up to one hundred points. Thus, it is also a viable option to obtain high quality solutions quickly. Furthermore, both heuristics endorse that using the support of linear relaxations to guide heuristics is a powerful strategy.

2.6 Final remarks

In this paper, we investigated the problem of finding a minimum convex partition of a set of points. An ILP model was developed that enabled the computation of exact solutions for small-sized instances. The linear relaxation of this formulation served as a guide for the design of heuristics that lead to demonstrably high quality solutions within reasonable runtimes. To the best of our knowledge, both the ILP modeling and the extensive computational experimentation with MCPP algorithms done here are novelties on the study of this problem and may lead to further practical developments. Moreover, research directions currently being pursued include the development of new mathematical models for this problem aiming at solving larger instances to optimality.

Acknowledgments. This research was financed in part by grants from Conse-

lho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) #311140/2014-9, #304727/2014-8, Coordenação de Aperfeiçoamento do Pessoal do Ensino Superior - Brasil (Capes) #001 and Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP) #2014/12236-1, #2017/12523-9, #2018/14883-5.

Bibliography

[1] A. S. Barboza, C. C. de Souza, and P. J. de Rezende. Minimum Convex Partition of Point Sets – Benchmark Instances and Solutions, 2018. `www.ic.unicamp.br/~cid/Problem-instances/Convex-Partition`.

[2] P. Bose, A. Brodnik, S. Carlsson, E. D. Demaine, R. Fleischer, A. López-Ortiz, P. Morin, and J. I. Munro. Online routing in convex subdivisions. *International Journal of Computational Geometry & Applications*, 12(4):283–296, 2002.

[3] T. Fevens, H. Meijer, and D. Rappaport. Minimum convex partition of a constrained point set. *Discrete Applied Mathematics*, 109(1-2):95–107, 2001.

[4] J. García-López and M. Nicolás. Planar point sets with large minimum convex partitions. In *Abstracts 22nd European Workshop on Computational Geometry*, pages 51–54, 2006.

[5] K. Hosono. On convex decompositions of a planar point set. *Discrete Mathematics*, 309(6):1714–1717, 2009.

[6] J. M. Keil. Polygon decomposition. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 491–518. North-Holland, Amsterdam, 2000.

[7] C. Knauer and A. Spillner. Approximation algorithms for the minimum convex partition problem. In *Proceedings of the 10th Scandinavian Workshop on Algorithm Theory (SWAT)*, volume 4059 of *Lecture Notes in Computer Science*, pages 232–241. 2006.

[8] M. Löffler and W. Mulzer. Unions of onions: Preprocessing imprecise points for fast onion decomposition. *Journal of Computational Geometry*, 5(1):1–13, 2014.

[9] V. Neumann-Lara, E. Rivera-Campo, and J. Urrutia. A note on convex decompositions of a set of points in the plane. *Graphs & Combinatorics*, 20(2):223–231, 2004.

[10] A. Spillner. A fixed parameter algorithm for optimal convex partitions. *Journal of Discrete Algorithms*, 6(4):561–569, 2008.

Chapter 3

Solving the Minimum Convex Partition of Point Sets with Integer Programming

In this work, we propose a new ILP model for the Minimum Convex Partition of Point Sets. This novel model has variables associated to convex polygons and has much better dual bounds when compared to the previous one, which had edge variables. The final model combines elements from both approaches and has much better performance.

This chapter contains the article [19] published at the journal *Computational Geometry: Theory and Application, Volume 99*. It was co-authored with Pedro J. de Rezende[1] and Cid C. de Souza.[1]

Abstract The partition of a problem into smaller sub-problems satisfying certain properties is often a key ingredient in the design of divide-and-conquer algorithms. For questions related to location, the partition problem can be modeled, in geometric terms, as finding a subdivision of a planar map – which represents, say, a geographical area – into regions subject to certain conditions while optimizing some objective function. In this paper, we investigate one of these geometric problems known as the Minimum Convex Partition Problem (MCPP). A *convex partition* of a point set P in the plane is a subdivision of the convex hull of P whose edges are segments with both endpoints in P and such that all internal faces are empty convex polygons. The MCPP is an NP-hard problem where one seeks to find a convex partition with the least number of faces.

We present a novel polygon-based integer programming formulation for the MCPP, which leads to better dual bounds than the previously known edge-based model. Moreover, we introduce a primal heuristic, a branching rule and a pricing algorithm. The combination of these techniques leads to the ability to solve instances with twice as many points as previously possible while constrained to identical computational resources. A comprehensive experimental study is presented to show the impact of our design choices.

[1]Institute of Computing, University of Campinas, Campinas, Brazil

3.1 Introduction

Partitioning problems constitute a fundamental topic in Computational Geometry. One of the best studied among them is the Triangulation Problem where we are given a point set P in the plane, and the goal is to partition its convex hull into triangles using line segments with endpoints in P. There are many variations of this problem that optimize different metrics such as segment length and minimum angle [6]. The widespread applicability of triangulations stems from a core idea of the *divide-and-conquer* paradigm: if a problem is too complex to be solved at once, break it into smaller, more tractable, sub-problems. Geometric problems tend to benefit from spacial subdivisions generated from their input where sub-problems can more easily be dealt with by considering the faces of the resulting arrangement. In two dimensions, for instance, it is desirable that these faces be regions satisfying properties that can be explored to make the algorithms more efficient. Therefore, an often desired structure for the faces is that they are convex polygons. The smallest convex polygons being triangles, it is understandable why questions regarding triangulations are so vastly investigated. For an in-depth discussion on triangulations, including structural properties, algorithms and applications, we refer to the book of de Loera et al. [8]. However, all triangulations of a given point set have the same number of triangles and edges and they may be too much of a refinement among the possible convex subdivisions. Hence, from the perspective of divide-and-conquer algorithms, when it comes to the number of sub-problems to be dealt with, which is strongly related to the overall complexity, a triangulation might be just as good as another.

This is where the relevance of the Minimum Convex Partition Problem (MCPP) stands out. The MCPP is a generalization of the triangulation problem in that one seeks a partition of the convex hull of a point set into the minimum number of convex polygons – some of which might even be triangles.

3.1.1 Our Contribution

Besides our earlier work [29], there are only a few attempts to solve the MCPP exactly, and, to date, no other comprehensive experimentation has been reported in the literature. By casting the problem as an integer linear program (ILP) and designing several algorithmic strategies, we were able, in [29], to solve instances of up to 50 points.

In the present paper, we explore further the usage of ILP to compute optimal solutions for the MCPP. Our main contribution is a new integer programming formulation for the MCPP, whereby we solve to provable optimality instances of up to 105 points, while using the same amount of computational resources as in our previous work, thus more than doubling the size of the largest instances with known optimum. To achieve this, we devise a primal heuristic and a branching rule, and show their effectiveness through experimentation. Also, since the number of variables in the new proposed model is exponential in the cardinality of the input point set, we resort to the use of column generation, which leads to the development of a *branch-and-price* algorithm. Since ILP is often applied in Operations Research, but much less frequently in Computational Geometry, this article can be seen as a further contribution towards bridging these two communities [7, 9, 12, 13, 16, 27, 33].

3.1.2 Literature Review

The MCPP has been studied from different perspectives in the literature, including the development of exact and approximation algorithms, heuristics and theoretical bounds on the optimal value.

Fevens et al. [14] proposed a dynamic programming formulation for the problem. Let h be the depth of P, defined as the number nested convex hulls that need to be removed from P before it becomes empty. Their algorithm has time complexity of $O(n^{3h+3})$, therefore exponential in h, which can be as large as $\Theta(n)$. Spillner et al. [31] proposed another exact algorithm with complexity $O(2^k k^4 n^3 + n \log n)$, where k is the number of points of P in the interior of its convex hull.

A compact ILP formulation for the MCPP based on the construction of a planar subdivision representing a partition obtained by selecting line segments with end-points in P was proposed by Barboza et al. [29]. The authors presented empirical results showing that, when fed as input to a state-of-the-art ILP solver, it could solve instances with up to 50 points in general position to provable optimality. They also show how to use the linear relaxation of the formulation to find good heuristic solutions for instances with up to 105 points. Those instances were made publicly available [4].

The best approximation algorithm for the MCPP, with factor of $\frac{30}{11}$, was proposed by Spillner et al. [21].

Bounds on the value of optimal solution as a function of n were also studied. Let $F(n)$ denote the maximum cardinality of the minimum convex partition among all instances of size n in general position. The tightest known bounds for $F(n)$ are $\frac{12n}{11} - 2 \leq F(n) \leq \frac{10n-18}{7}$, where the lower bound was shown in [18] and the upper bound in [26].

The 2020 Computational Geometry Challenge (CGSHOP) [10] motivated the advancement of the state-of-the-art for the MCPP from both theoretical and heuristic points of view. When the challenge was announced in September 2019, the complexity of the MCPP was still open and the only known empirical study was by Barboza et al. [29]. The challenge consisted in finding good solutions for 346 instances, with sizes ranging from 10 to 1,000,000 points and with different sets of instances, including sets with a large number of collinear points. Those instances and the best solution found were made publicly available. Details about the competition and the teams' progresses are discussed in [10]. The top three competitors proposed heuristic solutions based on local search.

In November 2019, Grelier [19] announced a proof of NP-hardness for the case when the point set is not in general position. Their proof relies heavily on the construction of instances with a large number of points lying on the same straight line.

3.1.3 Organization of the text

This paper is organized as follows. Section 3.2 describes a polygon-based formulation for the MCPP with an exponential number of variables. In Section 3.3, we address the issue of the number of variables being exponential by describing a column generation approach to solve its linear relaxation. Section 3.4 explains how we incorporate column generation into a branch-and-pricing framework in order to solve the problem to integrality, which

includes a branching rule and implementation details. Computational experiments and their corresponding results are discussed in Section 3.5.

3.1.4 Basic Notation

A *polygon* p of size t can be defined as a cyclic sequence of t distinct points in the plane $p_0, p_1, \cdots, p_{t-1}$, called the *vertices* of p. Two consecutive points p_i and p_{i+1} of p define a line segment $\overline{p_i p_{i+1}}$, called an *edge of* p, with addition taken module t. We say that p_i and p_{i+1} are the *endpoints* or *extremes* of the edge $\overline{p_i p_{i+1}}$. Sometimes, for practicality, we consider polygons as given by their cyclic sequence of edges $(\overline{p_0 p_1}, \overline{p_1 p_2}, \ldots, \overline{p_{t-2} p_{t-1}}, \overline{p_{t-1} p_0})$.

A polygon is called *simple* if the intersection between two distinct edges is empty unless they are consecutive, in which case they only share an endpoint. This essentially means that no two edges have a proper crossing. We refer to the sequence of edges of a simple polygon as its *boundary*.

Since the boundary of a simple polygon p constitutes a closed planar curve, by the Jordan Curve Theorem, it divides the plane into an unbounded and a bounded region. The latter is called the *interior* of p, denoted $\text{INT}(p)$, while the former is the *exterior of* p. Two polygons are said to be *interior-disjoint* if their interiors do not overlap.

Given a sequence of three points (k, l, m) in the plane, we say that they are *collinear*, *positively oriented*, or *negatively oriented* depending on the value of the cross product $\overrightarrow{kl} \times \overrightarrow{lm}$ being 0, positive or negative. To simplify notation we write $\text{CVX}(k, l, m) = \text{true}$ (or simply $\text{CVX}(k, l, m)$) when (k, l, m) is positively oriented. A geometrical interpretation is that a sequence (k, l, m) is positively (negatively) oriented when we make a left (right) turn at l as we traverse segment kl followed by lm.

We say that a polygon p is given in *counterclockwise* (CCW) order if $\text{INT}(p)$ is always to the left as one traverses the edges of p in the given order. It can be proved that this may be checked by verifying that $\text{CVX}(p_{j-1}, p_j, p_{j+1})$ is true when p_j is the lowest vertex of smallest abscissa. We assume that all simple polygons are given in CCW order. A simple polygon p is *convex* if $\text{CVX}(p_{i-1}, p_i, p_{i+1})$ for all $0 \leq i < t$. For convenience, given a simple polygon p, we may often employ the term polygon to also refer to the union of the boundary and the interior of p.

Given a set P of n points in the plane and a polygon p, we say that p is *empty* with respect to (w.r.t.) P, if p contains no points of P in its interior. When P is understood from the context, we simply say that p is empty. Denote by $\mathcal{S}(P)$ the set of all convex polygons with vertices in P that are empty w.r.t. P, and by $CH(P)$ the convex hull of P.

In this paper, unless otherwise noted, we do not assume that the sets of points are in general position, i.e., multiple collinear points are allowed. Moreover, henceforth all polygons referred to will be convex, unless stated otherwise.

A set $U \subseteq \mathcal{S}(P)$ of interior-disjoints polygons is called a *convex partition* of P if $CH(P) = \bigcup_{p \in U} p$.

Given a set P of n points, let $L(P)$ denote the set of $\Theta(n^2)$ line segments whose endpoints belong to P. The *complete (geometric) graph* induced by P is $G(P) = (P, E(P))$, where $E(P) = \{\{i, j\} : \overline{ij} \in L\}$. In this text, we refer to a segment $\overline{ij} \in L$ and the corresponding edge in $\{i, j\} \in E(P)$ interchangeably. Similarly, we denote the

complete oriented graph induced by P as $\overrightarrow{G}(P) = (P, A(P))$, where the arcs in $A(P)$ correspond to the two orientations of the edges in $E(P)$. Notice that, when multiple collinear points are present, we need to restrict $L(P)$ to include only segments that do not contain points of P in their interior.

For each convex partition U of P, there is a unique planar graph $G_U = (P, E^U) \subseteq G$, where E^U denotes the set of edges whose line segments belong to polygons in U.

The set of line segments $L(P)$ determines a planar subdivision called the *arrangement* of P. Assuming that P is in general position and $n \geq 3$, this arrangement contains an unbounded face, corresponding to the exterior of $CH(P)$, while all other faces are bounded. We denote the set of bounded faces of the arrangement of P by $\mathcal{A}(P)$. It can be proved that each face f of the arrangement $\mathcal{A}(P)$ is a convex polygon and that if $f \cap p \neq \emptyset$ for some $p \in \mathcal{S}(P)$, then $f \subseteq p$. In this sense, we say that every face of an arrangement $\mathcal{A}(P)$ is *atomic*. Moreover, a polygon $p \in \mathcal{S}(P)$ is said to *contain a face* $f \in \mathcal{A}(P)$, denoted by $f \subset p$, if the interior of p contains the interior of f. Lastly, a line or line segment ℓ *supports* a face $f \in \mathcal{A}(P)$ if ℓ contains one of the edges of f.

3.2 A Set Partition Model for the MCPP

In this section, we present a new Integer Linear Programming (ILP) model for the MCPP from a standard set partition point of view. Again, let P be a set of n points in the plane.

In this model, we associate a binary variable u_p with each polygon $p \in \mathcal{S}(P)$ such that polygon p is used in the partition of $CH(P)$ if and only if $u_p = 1$. This polygon-based approach is different from the edge-based model presented in [29], which will be discussed in Section 3.4.1.

Recall that $\mathcal{A}(P)$ denotes the set of faces of the arrangement of segments induced by the edges in $E(P)$. We use $f \subset p$ to indicate that polygon $p \in \mathcal{S}(P)$ contains (or covers) face $f \in \mathcal{A}(P)$.

We now introduce the following Model **M1**:

$$\min \quad \sum_{p \in \mathcal{S}(P)} u_p \tag{1a}$$

$$\text{s.a.} \quad \sum_{p \in \mathcal{S}(P): f \subseteq p} u_p = 1 \qquad \forall f \in \mathcal{A}(P) \tag{1b}$$

$$u_p \in \{0, 1\} \qquad \forall p \in \mathcal{S}(P) \tag{1c}$$

This model is very straightforward as it has only one family of constraints, namely (1b), which ensure that each face of the arrangement is covered by a polygon in the solution exactly once. The objective function (1a) minimizes the number of polygons forming the partition.

The main issue with this model, which will be addressed in the next section, is that the number of polygons in $\mathcal{S}(P)$ is typically exponential in $n = |P|$, making it impractical to enumerate all variables for large instances. Besides, the number of constraints in the model, although polynomial in n, is also large since an arrangement of $\Theta(n^2)$ line segments can have $O(n^4)$ faces [2] and each of them corresponds to a constraint in (1b).

However, we show next that only a small fraction, $\Theta(n^2)$, of those faces are necessary to ensure that the area within $CH(P)$ is correctly partitioned.

We start by defining necessary concepts and notations. Let q be an arbitrary point in the plane. We denote by $\text{CCW}(i,q)$ the sequence of $n-1$ points in $P \setminus \{i\}$ sorted angularly w.r.t. the ray $\overset{\mapsto}{iq}$ that starts at i an passes through q.

Recall that we use $\text{CVX}(k,l,m)$ to denote that a sequence of points (k,l,m) is positively oriented.

Let $\overrightarrow{iq}$ denote the *oriented line* that passes through i and q in this order, which divides the plane into two half-planes, called *left* and *right*. A point k on the left (right) half-plane of $\overrightarrow{iq}$ satisfies $\text{CVX}(i,q,k)$ ($\text{CVX}(q,i,k)$). The remaining points are on $\overrightarrow{iq}$ itself. We define $\text{CCW}^+(i,q)$ as the prefix of $\text{CCW}(i,q)$ whose points are non-negatively oriented w.r.t. the oriented line $\overrightarrow{iq}$. Similarly, we use $\text{CCW}^-(i,q)$ to denote $\text{CCW}(i,q) - \text{CCW}^+(i,q)$; i.e., the suffix of $\text{CCW}(i,q)$ whose points are negatively oriented w.r.t. $\overrightarrow{iq}$. We illustrate these concepts in Figure 3.1.

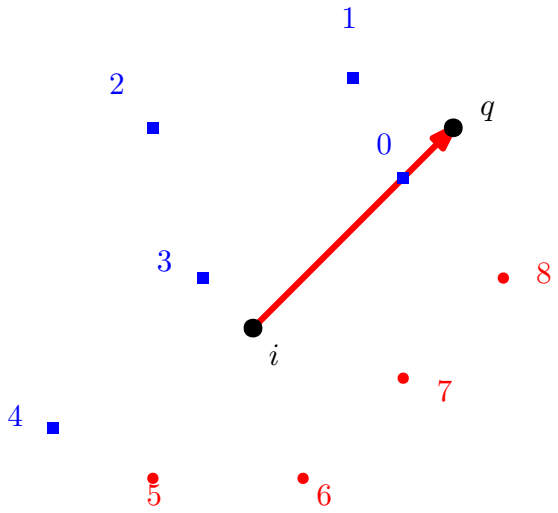


Figure 3.1: $P = \{i, 0, 1, \ldots, 8\}$, $q \notin P$, $\text{CCW}(i,q) = (0,1,\ldots,8)$, $\text{CCW}^+(i,q) = (0,1,2,3,4)$ and $\text{CCW}^-(i,q) = (5,6,7,8)$

Let $i \in P$. A face $f \in \mathcal{A}(P)$ that has i as one of its vertices is called an *i-wedge*. Let i_x be the x-coordinate of i and y_{max} be the maximum y coordinate among all points in P and consider the point $q = (i_x, y_{max}+1)$. Let $\text{CCW}(i,q) = (i_0, i_1, \cdots, i_{n-2})$. For each consecutive pair of points (i_k, i_{k+1}) for $0 \leq k < n-1$, addition being taken mod $n-1$, we have exactly one face of the arrangement $\mathcal{A}(P)$ that is incident to i and is supported by the edges $\{i, i_k\}$ and $\{i, i_{k+1}\}$. We call this face the *k-th i-wedge*. We denote by *P-wedges* the set of all i-wedges for $i \in P$. Figure 3.2 shows the arrangement of a point set P and highlights the faces that are not i-wedges.

Let $U \subset \mathcal{S}(P)$. We say that a face f of $\mathcal{A}(P)$ is *exactly-covered* by a polygon $p \in U$, if p is the only polygon in U that covers f. A face that is not exactly-covered by some (single) polygon in U is called *non-exactly-covered*. A set of faces is exactly-covered if all of its faces are exactly-covered.

In the next lemma, we show that if u is a solution of Model **M1** with constraint set (1b) limited to the equations associated to the faces of $\mathcal{A}(P)$ that are i-wedges and

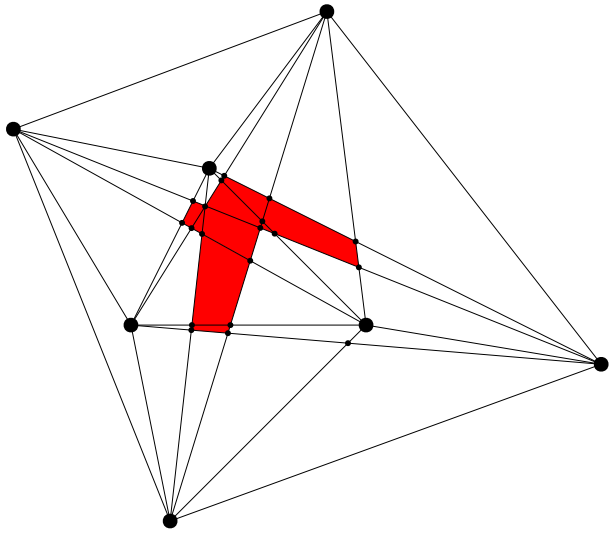


Figure 3.2: Example of a point set P and its arrangement. Red faces are the only ones that are **not** in P-wedges.

$U = \{p \in \mathcal{S}(P) : u_p = 1\}$, then U is a partition of $CH(P)$. Our goal is to show that there are no faces in $\mathcal{A}(P)$ that are non-exactly-covered by polygons in U.

Lemma 3. *Let u be a solution of Model* **M1** *and $U = \{p \in \mathcal{S}(P) : u_p = 1\}$ be the corresponding set of polygons. Then, if p is a polygon in U, each one of its edges that is not an edge of $CH(P)$ supports exactly one other polygon in U.*

Proof. Let $\{i, k\}$ be an edge of p that is not an edge of $CH(P)$, and $k-1$ be the predecessor of k in $\text{CCW}(i, (i_x, y_{max} + 1)))$. Suppose, w.l.o.g., that the $(k-1)$-st i-wedge is exactly-covered by p. As u is a solution of Model **M1**, there must be a polygon h in U that covers the k-th i-wedge. Recall that there is no other edge in $E(P)$ that corresponds to a segment that contains (the segment that corresponds to) $\{i, k\}$. Also, the interior of h cannot intersect the segment $\{i, k\}$, since, otherwise, both p and h would be covering the $(k-1)$-st i-wedge, violating (1b). Therefore, h shares the edge $\{i, k\}$ with p. The same holds if we exchange the roles of the k-th and the $(k-1)$-st i-wedges. Figure 3.3 illustrates the polygons in a solution and the i-wedges they cover. □

This result helps us to prove that constraints that do not correspond to i-wedges are, in fact, redundant, and can be removed from the model, as follows.

Theorem 4. *Let u be a solution of Model* **M1** *without the Constraints* (1b) *that do not correspond to i-wedges and $U = \{p \in \mathcal{S}(P) : y_p = 1\}$ be the corresponding set of polygons. Then, U covers each face of the arrangement $\mathcal{A}(P)$ exactly once.*

Proof. First, we show that given a solution u of Model **M1** without the Constraints (1b) that do not correspond to P-wedges, there are no uncovered regions in $CH(P)$.

Notice that the i-wedges are exactly-covered. Also, observe that non-exactly-covered regions are formed by unions and intersections of polygons in U. Thus, the boundary of each maximal connected non-exactly-covered region is comprised of of line segments.

Let R be a maximal connected region of $CH(P)$ comprised by the union of uncovered faces of $\mathcal{A}(P)$. Let e be an edge of R that does not lie on the boundary of $CH(P)$. Such

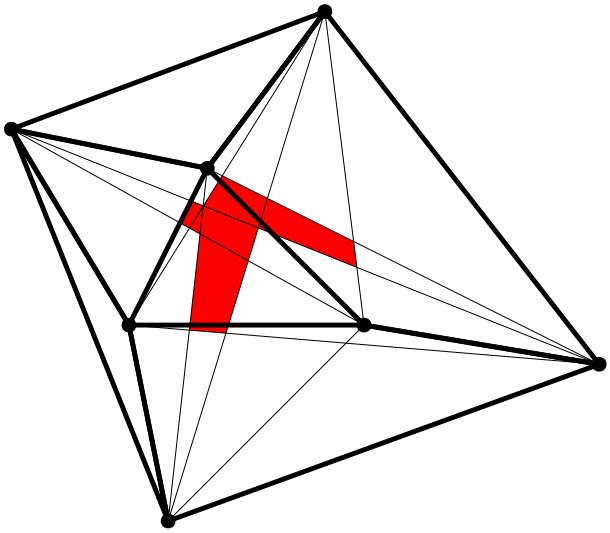


Figure 3.3: Example of a convex partition superimposed on the arrangement of Figure 3.2. Edges that belong to the polygons in the partition are heavier.

edge always exists since each i-wedge is exactly-covered. Thus, R cannot be equal to $CH(P)$. Notice that there is a single edge in $E(P)$, say $\{i, k\}$, that supports e. Since R is maximal, there is a face b of the arrangement $\mathcal{A}(P)$ that is adjacent to e on the other side of e relative to R and is covered at least once. Let $h \in U$ be one of the polygons covering b. As h cannot cover any part of R, it must include edge $\{i, k\}$ on its boundary. Figure 3.4 illustrates this configuration. By Lemma 3, there must be a polygon $g \in U$ that shares $\{i, k\}$ with h on the same side of e as R. Thus, $R \cap g$ is covered, which is a contradiction with the fact that R is entirely not covered.

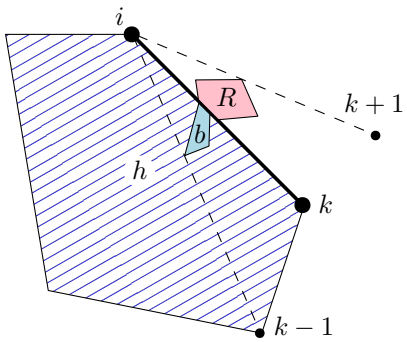


Figure 3.4: To illustrate Theorem 2.2, consider an uncovered region R supported by edge $e = \{i, k\}$. This edge also supports face b of the arrangement, which is already covered by polygon h.

Since U leaves no uncovered regions, we now focus on the proof that there is no region of $CH(P)$ that is covered by two or more polygons.

Let R be a maximal connected region of $CH(P)$ comprised by the union of faces of $\mathcal{A}(P)$ that are covered more than once by polygons in U. Let e be an edge of R that does not lie on the boundary $CH(P)$. Since R is maximal, one side of e is covered at least twice and the other is covered exactly once, as previously discussed, implying that there is a polygon g in U with an edge $\{i, k\}$ supported by e on the same side as R. Let

us assume, w.l.o.g., that the $(k-1)$-st i-wedge and R are on the same side of the line supported by $\{i,k\}$, otherwise, we change the roles played by i and k. Let b be a face of $\mathcal{A}(P)$ in R adjacent to e. Then, b is covered by g and there must be another polygon $h \in U$ that also covers b. As previously argued, there is no edge other than $\{i,k\}$ in $E(P)$ that is supported by e. However, h cannot have $\{i,k\}$ as one of its edges, since, otherwise, the $(k-1)$-st i-wedge would be covered by both g and h. Also, since h is empty (w.r.t. points in P), it cannot contain i or k in its interior. This means that polygon h cannot have e on its boundary, and must cover a face c adjacent to e on the opposite side of g relative to $\{i,k\}$. By Lemma 3, there is a polygon p that shares $\{i,k\}$ with g and covers c, implying that c is covered at least twice. We conclude that $R \cup c$ is covered more than once, contradicting our assumption that R is maximal. □

3.3 Column Generation Algorithm for SPM

In this section, we address the issue of having an exponential number of variables in Model **M1**. We solved this difficulty by the use of Column Generation.

Instead of enumerating all the variables, we start by solving the model with only a small subset of them and proceed by generating new ones when necessary. This approach, known as *Column Generation*, guarantees that, if our procedure for generating columns with negative reduced cost is polynomial, the linear relaxation of Model **M1** can be solved in polynomial time [20].

To solve the linear relaxation of an LP model using column generation, we iterate between solving the *restricted master problem* (RMP) and the *pricing problem*. The RMP is the original model restricted to a subset of variables. At each iteration, an RMP is solved to optimality and the pricing problem is used to find variables with negative (in the case of minimization problems) reduced cost that should be added to the RMP. This process stops when no variable has negative reduced cost. Figure 3.5 illustrates this process.

Preliminary studies with different sets of initial polygons based on an starting solution did not lead to a noticeable difference in performance compared to using all triangles. Thus, we start the RMP with the variables corresponding to all triangles. Next, we show how to solve the pricing problem for Model **M1**.

By associating a vector of variables α with Constraints (1b), we obtain the dual of the linear program corresponding to the relaxation of Model **M1**, shown in Model **M2** below.

$$\max \quad \sum_{f \in \mathcal{A}} \alpha_f \tag{2a}$$

$$\text{s.a.} \quad \sum_{f \in \mathcal{A}(P): f \subseteq p} \alpha_f \leq 1 \qquad \forall p \in \mathcal{S} \tag{2b}$$

$$\alpha_f \in \mathbb{R} \qquad \forall f \in \mathcal{A}(P) \tag{2c}$$

Let $\overline{\alpha}$ be an optimal dual solution of the RMP in a given iteration of the column generation procedure. We can express the reduced cost $\overline{c}_p$ of the variable corresponding to polygon $p \in \mathcal{S}(P)$ in Model **M1** as:

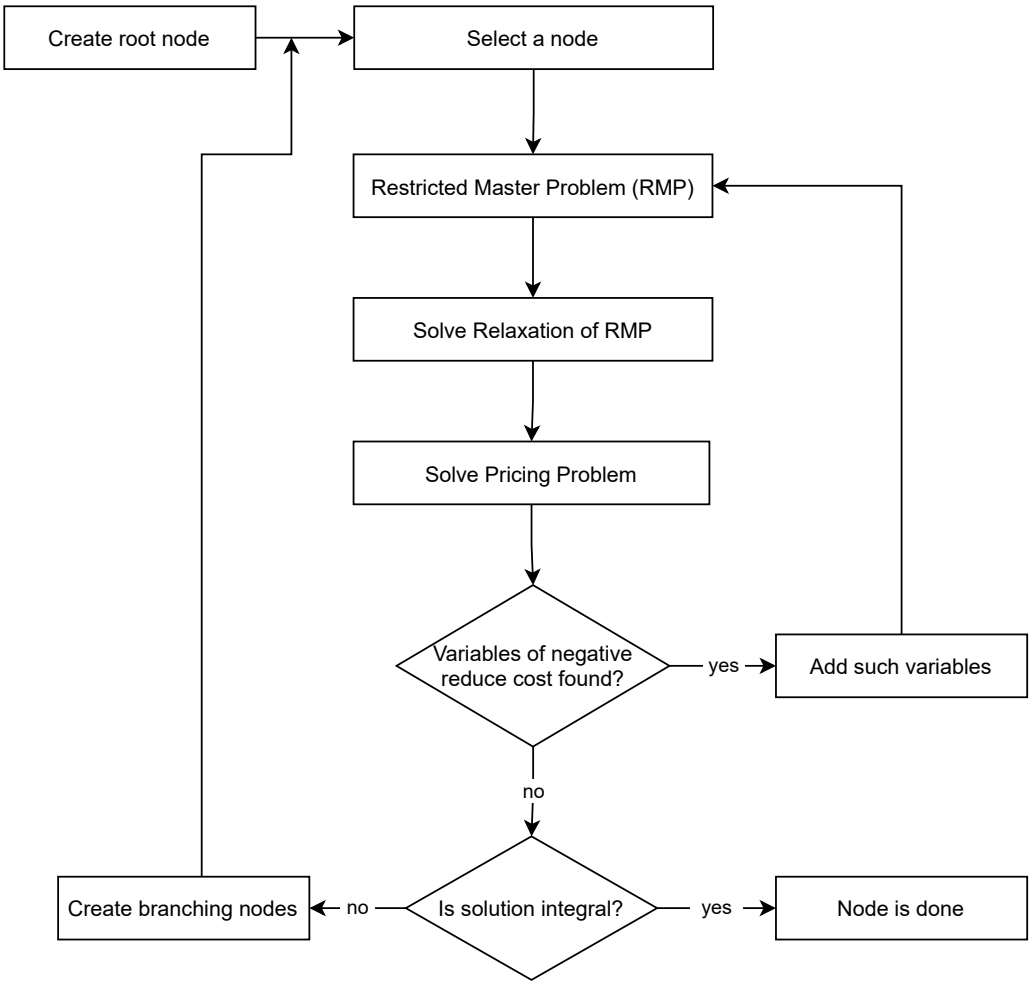


Figure 3.5: Illustration of a Branch-and-Price algorithm. Adapted from [24].

$$\overline{c}_p = 1 - \sum_{f \in \mathcal{A}(P): f \subseteq p} \overline{\alpha}_f. \tag{3.3}$$

In other words, to compute the reduced cost of a variable, we need to sum up the values of the dual variables associated with face constraints covered by the corresponding polygon.

In order to solve the pricing problem, we define a recurrence based on the idea of constructing polygons by joining triangles that share an edge. However, dealing with triangles of zero area involves an intricate implementation of a bookkeeping scheme regarding i-wedges. We opted to leave this as an issue worth investigating (see Section 3.6) and, for the description that follows, we assume that the points in P are in general position.

Let $\Delta(k, l, m)$ denote the reduced cost of a triangle $(k, l, m) \in \mathcal{S}(P)$, with respect to the dual variables $\overline{\alpha}$, given by $\Delta(k, l, m) = \sum_{f \subseteq (k,l,m)} \overline{\alpha}_f$.

We now consider all polygons whose leftmost vertex k is preceded in CCW order by vertices l and m. W.l.o.g., we say that k is the first vertex of such polygons, while l and m are their second-to-last and last vertices, respectively. Since each of these polygons has an associated variable, let $B(k, l, m)$ denote the minimum reduced cost among these variables. We say that (k, l, m) is the *last triangle* of those polygons. See Figure 3.6.

Let us denote $\text{CCW}^-(k, (k_x, y_{max} + 1))$ simply by P_k, notice that those are the points

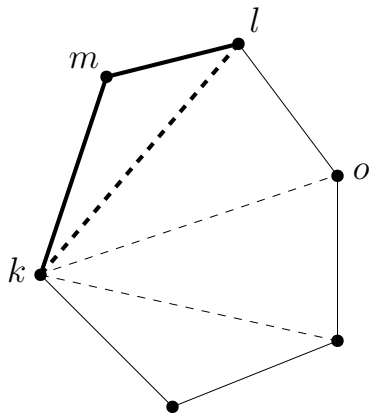


Figure 3.6: Example of a polygon whose last triangle is (k, l, m). Dashed lines show how it can be decomposed into triangles with k as their leftmost vertex, emphasizing the edges of (k, l, m).

to the right of k sorted in CCW order. We use $o \leq_{P_k} l$ to indicate that o precedes l in the sequence P_k. Recall that $\text{CVX}(o, l, m)$ denotes that the sequence (o, l, m) is positively oriented.

See Section 3.7.

Then, $B(k, l, m)$ may be computed by following the recurrence formula:

$$B(k,l,m) = \begin{cases} \infty, \text{ if } m = l \text{ or } k_x > l_x \text{ or } m <_{P_k} l \text{ or } (k,l,m) \notin \mathcal{S} & \text{(3.4a)} \\ \min\limits_{\substack{o \in P_k: \\ o <_{P_k} l \\ \text{CVX}(o,l,m)}} \{0, B(k,o,l)\} + \Delta(k,l,m), \text{ otherwise} & \text{(3.4b)} \end{cases}$$

To explain this formula, we define a polygon and a triangle to be *compatible* if their union is a convex polygon that is empty w.r.t. P. Recurrence (3.4) has two cases. Case (3.4a) deals with invalid triplets of points, while in case (3.4b) we look for a minimum cost polygon that is compatible with triangle (k, l, m).

Now, we discuss how to solve this recurrence in polynomial time using dynamic programming. Recall that $|P| = n$.

There are $O(n^3)$ dynamic programming states, one for each $B(k, l, m)$. The total time complexity for processing each state is the $O(n)$ time spent looking for the best compatible polygon, or deciding that (k, l, m) is not empty, plus the time spent calculating $\Delta(k, l, m)$. Naively, computing $\Delta(k, l, m)$ for a single triangle would take, in the worst case, $O(n^4)$ time, as it would require processing all the faces in the complete arrangement. The final complexity using this approach would be $O(n^7)$.

However, we can take advantage of the structure of the set of P-wedges to reduce the time to compute $\Delta(k, l, m)$ of a triangle to $O(1)$, with $O(n^2)$ preprocessing each time the dual variables change.

This is accomplished by observing that $\Delta(k, l, m)$ is the sum of a range of consecutive i-wedges in each of its vertices i, as shown in Figure 3.7. For each point $i \in P$, we can use a data structure capable of answering range sum queries in $O(1)$ time and which can be built in $O(n)$ time for a given set of dual variable values. Computing $\Delta(k, l, m)$ requires only three such queries. Now, for each of the $O(n^3)$ states, it takes $O(n)$ time to find the compatible polygon with the smallest reduced cost, or decide that (k, l, m) is not empty,

and $O(1)$ time to compute $\Delta(k, l, m)$, leading to a time complexity of $O(n^4)$.

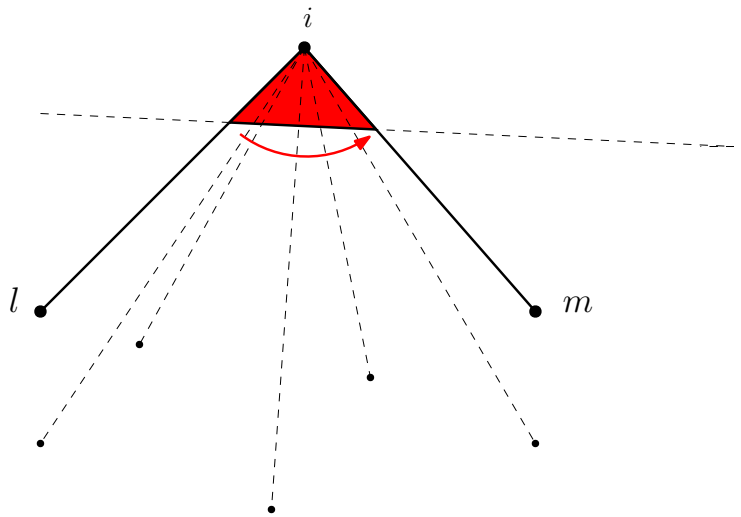


Figure 3.7: An example of the circular order of i-faces around a given vertex i. Each pair of consecutive points in a CCW ordering of $P \setminus \{i\}$ defines an i-wedge, highlighted in red.

To reduce the complexity even further, we adapt the angular sweeping technique presented by Avis and Rappaport [3] to solve the Largest Empty Convex Polygon Problem and we achieve the same time complexity of $O(n^3)$ as they did for their problem. Notice that by setting $\Delta(k, l, m)$ to -1, finding the polygon with minimum negative cost is the same as finding an empty convex polygon with maximum number of vertices. As done in [3], we implement an algorithm to check whether a triangle is empty in $O(1)$ time per query and $O(n^3)$ preprocessing time using visibility graphs. However, since we need to solve this problem multiple times, we store all the empty triangles, increasing the space complexity from $O(n^2)$ to $O(n^3)$.

Finally, we present the pricing algorithm, starting with the description of the angular sweeping technique.

Given k and l, let P_{lk}^+ be the list $\text{CCW}^+(l, k)$ restricted to the points in P_k and, similarly, P_{lk}^- be the list $\text{CCW}^-(l, k)$ restricted to the points in P_k. We compute $B(k, l, m)$ for all possible m's in $O(n)$ total time, using two pointers as follows: m traverses P_{lk}^- and o goes over P_{lk}^+. See Figure (3.8).

The procedure to compute the reduced cost $B(k, l, m)$ is presented in Algorithm 1. To simplify notation, if the triangle (k, l, m) is not empty, we set $\Delta(k, l, m) = \infty$.

Algorithm 1 works as follows. The loops defined by Lines 1 and 2 iterate over the leftmost and second-to-last points of each state, respectively. Pointers m and o are initialized to point to the beginning of their respective lists in Lines 3 and 5, and the best compatible polygon is set in Line 4 to be the empty polygon. The loop between Lines 5 and 9 moves pointer m, corresponding to the last vertex of the polygons being constructed, one step at the time. The other pointer o, corresponding to the third-to-last vertex, is handled by the loop between Lines 6 and 8: o moves forward as far as possible, while the angle $\measuredangle oml$ is convex and updates the best compatible polygon found so far. Finally, Line 9 computes the cost of $B(k, l, m)$.

Algorithm 1: Column Pricing Algorithm

Input: Point set P, lists of points P_k, P_{lk}^+ and P_{lk}^-
Output: Minimum reduced cost $B(k, l, m)$

```
for k ← 1 to n do
    for l ← 1 to |P_k| do
        o ← 1
        bestValueO ← 0
        for m ← 1 to |P_lk^-| do
            while o ≤ |P_lk^+| and CVX(P_lk^+[o], P_k[l], P_lk^-[m]) do
                bestValueO ← min(bestValueO, B(k, P_lk^+[o], P_k[l]))
                o ← o + 1
            end
            B(k, l, m) ← bestValueO + Δ(k, P_k[l], P_lk^-[m])
        end
    end
end
return B
```

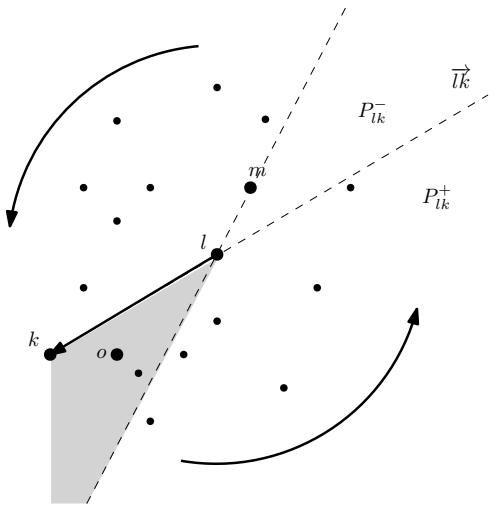


Figure 3.8: To illustrate the execution of the Column Pricing Algorithm 1. consider how P_k is divided into the lists P_{lk}^- and P_{lk}^+ by the oriented line $\overrightarrow{lk}$, as well as the movement of pointers l and o. Points in the gray region are candidates for second-to-last point for polygons whose last triangle is (k, l, m).

To conclude that the total complexity is indeed $O(n^3)$, we need to observe that each one of the variables k, l and m always increases and at most $O(n)$ times. Also, each step is done in $O(1)$ time, including computing $\Delta(k, l, m)$, as previously stated.

Notice that each one of the $O(n^3)$ dynamic programming states determines a polygon with negative cost. Adding multiple variables with negative reduced cost in each pricing iteration can potentially accelerate convergence [22]. This fact was confirmed by our preliminary study. However, reconstructing the polygon given by the dynamic programming table $B(k, l, m)$ takes linear time, since we actually need to find the set of faces that it covers. Doing that for each triplet k, l, m would increase the overall complexity by a

factor of n. To keep the complexity of the procedure at $O(n^3)$, we only consider the best polygon for each pair k, l, limiting the output to $O(n^2)$ polygons.

3.4 Branch-and-Price

In the previous section, we discussed how to solve the pricing problem in polynomial time within a Column Generation framework, allowing for the linear relaxation of Model **M1** to be solved in polynomial time. In practice, the standard way to solve ILP models is to use a Branch-and-Bound algorithm based on linear relaxation. The combination of Branch-and-Bound and Column Generation is known as *Branch-and-Price* (BNP) [5].

In this section, we describe some details and the design choices we made to implement a Branch-and-Price algorithm for Model **M1**.

We also discuss how the addition of variables representing edges to the model leads to a specialized branching rule. Moreover, we elucidate how to find an initial viable solution, called an *incumbent* solution, and how to use this solution to generate an initial set of columns for the RMP. A similar approach is used to find viable solutions at each node of the BNP tree. We conclude the section with additional implementation details that aim to improve the performance of the BNP algorithm in practice.

3.4.1 Introducing Edges

One of the challenges faced when implementing Branch-and-Price algorithms is that branching decisions are handled as additional constraints added to the newly created sub-problems. These constraints have their own dual variables and might change the pricing problem, when the variables being priced are involved. The introduction of variables associated with edges allows for a stronger geometric-based branching rule so as to completely avoid modifying the pricing problem throughout the branching decisions.

The idea to introduce edge variables to the Set Partition Model **M1** comes from the Compact Model introduced in [29]. Let S^C denote the set of pairs of segment of $E(P)$ that cross and $I(P)$ denote the set of points of P in the interior of $CH(P)$. In the Compact Model **M3**, shown below, we associate an edge variable x_e to each edge $e \in E(P)$.

$$\min \sum_{\{i,j\} \in E(P)} x_{ij} \tag{5a}$$

$$\text{s.t.} \quad x_{ij} + x_{k\ell} \leq 1 \qquad \forall \{\{i,j\},\{k,\ell\}\} \in S^c \tag{5b}$$

$$x_{ij} = 1 \qquad \forall \{i,j\} \in CH(P) \tag{5c}$$

$$\sum_{k \in \mathrm{CCW}^+(i,j)} x_{ik} \geq 1 \qquad \forall (i,j) \in A(P), i \in I(P) \tag{5d}$$

$$\sum_{j \in P} x_{ij} \geq 3 \qquad \forall i \in I(P) \tag{5e}$$

$$0 \leq x_{ij} \leq 1 \qquad \forall \{i,j\} \in E(P) \tag{5f}$$

$$x_{ij} \in \mathbb{Z} \qquad \forall \{i,j\} \in E(P) \tag{5g}$$

In the Compact Model **M3**, edge crossings are avoided by Constraints (5b). Constraints (5c) ensure that the edges belonging to the convex hull of P are part of the solution, while Constraints (5d) ensure that the angles incident to a internal vertex are all convex. Constraints (5e) are added to force every internal vertex to have degree at least three. Finally, Constraints (5f) and (5g) guarantee that the variables are binary.

By including edges we can take advantage of the constraints and heuristic presented in [29]. The edges are also natural candidates for branching.

A common drawback of Branch-and-Bound algorithms is the possibility of unbalanced branching trees. When one of the branching choices is much more restrictive than the other, the least restrictive node can have almost no impact in the value of the optimal solution of the relaxation [15]. This happens, in particular, when branching on a single variable, which is the default for commercial solvers.

For partitioning problems, one of the most well known balanced branching approaches is the Ryan-Foster branching rule [15]. According to this rule, we find a pair of constraints that are covered by distinct sets of variables but share at least one (fractional) variable, and branch on two possibilities: forcing those two constraints to be covered by the same variable or to be covered by two different variables.

In this section, we show that for Model **M1**, when the solution is fractional, there is always a pair of adjacent i-wedges that can be used for branching according to the Ryan-Foster rule. This branching can be interpreted geometrically as deciding whether a particular edge of $E(P)$ is part of the solution or not.

Consider edge $ik \in E(P)$ and assume that k is preceded by $k-1$ and succeeded by $k+1$ in the CCW ordering around i. This edge has two corresponding arcs $(i,k),(k,i) \in A(P)$, one for each possible orientation. The polygons in $\mathcal{S}(P)$ can be split into four sets with respect to the arc (i,k): the set of polygons $\mathcal{S}^+_{ik}(P)$ that are supported by $\{i,k\}$ and cover the k-th i-wedge; the set of polygons $\mathcal{S}^-_{ik}(P)$ that are supported by $\{i,k\}$ and cover the $(k-1)$-st i-wedge; the polygons $\mathcal{S}^{over}_{ik}(P)$ that cover both $(k-1)$-st and the k-th i-wedges; and the set of polygons $\mathcal{S}^{disj}_{ik}(P)$ that cover neither the $(k-1)$-st nor the k-th i-wedge. To simplify notation, we omit the point set P from the notation of $\mathcal{S}$ and its subsets when the context makes it clear. Figure 3.9 illustrates this notation. The same reasoning can be used for the reverse arc (i,k), by exchanging the roles played by i and k.

Recall that the i-wedge supported by the edges $i(k-1)$ and ik is the $k-1$-st i-wedge, while the i-wedge between edges ik and $i(k+1)$ is the k-th i-wedge.

By denoting $\sum_{b\in B} u_b$ as $u(B)$ for a given set $B \subseteq \mathcal{S}(P)$, we can rewrite equation (1b) for those i-wedges as:

$$u(\mathcal{S}^-_{ik}) + u(\mathcal{S}^{over}_{ik}) = 1 \tag{3.6}$$

$$u(\mathcal{S}^+_{ik}) + u(\mathcal{S}^{over}_{ik}) = 1, \tag{3.7}$$

respectively. Notice that equality $u(\mathcal{S}^+_{ik}) = u(\mathcal{S}^-_{ik})$ follows from (3.6) and (3.7).

With this in mind, we can extend Model **M1** by adding the set of binary variables x_e for each edge $e \in E(P)$. Denoting that an edge e is in the border of a polygon p by $e \in p$, we also add constraints relating polygons and edges.

An edge ik is in the solution if and only if there are two polygons supported by it, one

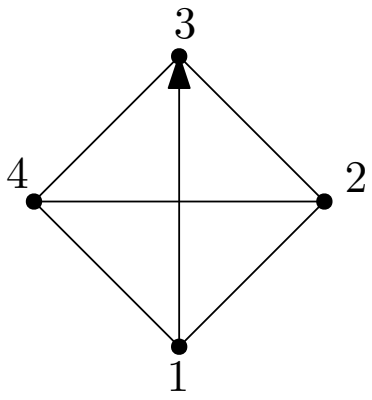


Figure 3.9: Example of an instance with four points inducing four triangles and one quadrilateral. Arc $(1,3) \in A(P)$ splits those polygons into four sets $\mathcal{S}_{13}^{+} = \{134\}$, $\mathcal{S}_{13}^{-} = \{123\}$, $\mathcal{S}_{13}^{over} = \{124, 1234\}$ and $\mathcal{S}_{13}^{disj} = \{234\}$.

on each side. Taking the orientation from i to k, we can express this with the following equations:

$$x_{ik} = u(\mathcal{S}_{ik}^{-}) \tag{3.8}$$

$$x_{ik} = u(\mathcal{S}_{ik}^{+}). \tag{3.9}$$

However, since $u(\mathcal{S}_{ik}^{-}) = u(\mathcal{S}_{ik}^{+})$, we can drop one of the constraints or combine both to simplify notation.

To simplify notation and avoid splitting the constraints, the exterior of $CH(P)$, regarded as a polygon that covers the unbounded face of the arrangement, is included in $\mathcal{S}$. With this, all edges now support exactly two polygons. This allows us to express the extended Model **M4** as follows.

$$\min \quad \sum_{p \in \mathcal{S}} u_p \tag{1a}$$

$$\text{s.a.} \quad \sum_{p \in \mathcal{S}: f \subseteq p} u_p = 1 \qquad \forall f \in \mathcal{A} \tag{1b}$$

$$2x_e = \sum_{p \in \mathcal{S}: e \in p} u_p \qquad \forall e \in E(P) \tag{10a}$$

$$u_p \in \{0,1\} \qquad \forall p \in \mathcal{S} \tag{1c}$$

$$x_e \in \{0,1\} \qquad \forall e \in E(p) \tag{10b}$$

By branching on an edge variable x_e, we are deciding whether the two consecutive i-wedges are going to be covered by the same polygon or by different ones. This is analogous to the Ryan-Foster branch rule, as previously discussed.

Lastly, we now show that we can branch only on edge variables.

Lemma 5. *Let (u^*, x^*) be an optimal solution of the linear relaxation of Model* **M4**. *Then, there is an edge $e \in E(P)$ such that x_e^* is fractional if and only if there is a polygon $p \in \mathcal{S}$ such that u_p^* is fractional.*

Proof. (COUNTERPOSITIVE PROOF OF $\Rightarrow$): observe that if u_p^* is integral for all p, then, if e is any edge in $E(P)$, we either have $u^*(\mathcal{S}_e^{over}) = 0$, which implies $x_e = 1$, or $u^*(\mathcal{S}_e^{over}) = 1$, which ascertains that $x_e = 0$.
(DIRECT PROOF OF $\Leftarrow$): Let p be a polygon such that u_p^* is fractional and $e' = \{i', k'\}$ one of its edges. Assume w.l.o.g. that p covers the $(k'-1)$-st i'-wedge, i.e., $p \in \mathcal{S}_{e'}^-$.

Case 1: If $u^*(\mathcal{S}_{e'}^{over}) > 0$ then, as $u(\mathcal{S}_{e'}^-) > 0$, from Equations (3.6) and (3.7), we have that $u(\mathcal{S}_e^+)$ and $u(\mathcal{S}_e^-)$ must both be fractional. Thus, from $0 < u(\mathcal{S}_e^+) + u(\mathcal{S}_e^-) < 2$ and (10a) we get that x_e is fractional.

Case 2: If $u^*(\mathcal{S}_{e'}^{over}) = 0$, then there is a polygon $h \neq p$ that also covers the $(k'-1)$-st i'-wedge such that $0 < u_h^* < 1$. We can then traverse the edges of p in clockwise order, starting at e', until we find an edge $e = \{i, k\}$ that belongs to p but not to h. Assume w.l.o.g. that h covers both the $(k-1)$-st and the k-th i-wedge (see Figure 3.10), otherwise, just exchange the roles of p and h. By construction, $p \in \mathcal{S}_e^-$ and $h \in \mathcal{S}_e^{over}$ and the corresponding variables u_p and u_h are both positive. Therefore, by Equation (3.6), $u(\mathcal{S}_e^-)$ is fractional and, by Equation (3.8), x_e is also fractional. □

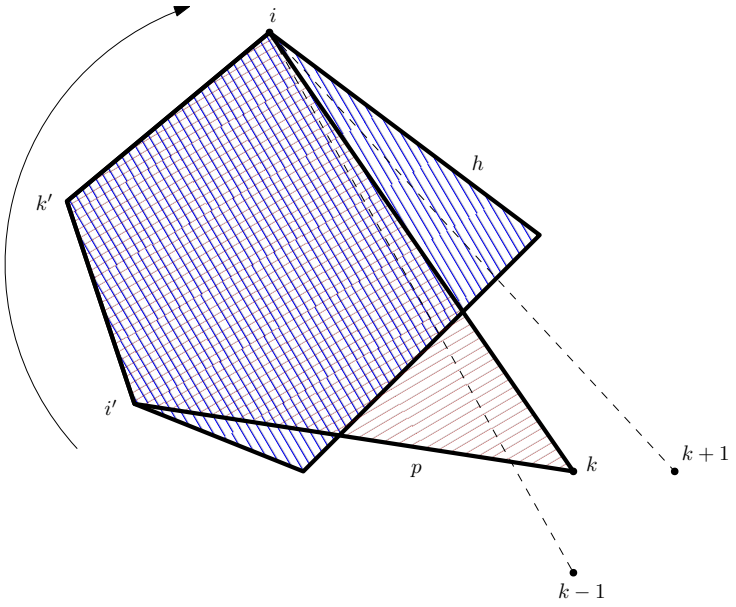


Figure 3.10: Example of two distinct polygons p and h that share edges. Edge ik belongs only to p, but point i belongs to both polygons. Polygon h covers both the $(k-1)$-st and the k-th i-wedge while p only covers the k-th i-wedge.

Hence, the following Corollary is immediate.

Corollary 5.1. *If (u^*, x^*) is a fractional solution of Model* **M4***, then there is a pair of adjacent i-wedges that induce a Ryan-Foster Branching represented by an edge.*

Notice that adding variables is not a necessary step to implement the branching rule. We can simply add the corresponding constraints, replacing x_e by zero or one, to the respective child nodes. However, explicitly adding the variables facilitates the implementation when using a commercial solver as it can take advantage of complex single variable branching rules already in place such as strong branching.

The addition of Constraint (10a) affects the pricing algorithm and it needs to be handled explicitly. On the other hand, the actual branching decisions, which add the constraints $x_e = 0$ or $x_e = 1$ to the child nodes, do not involve polygon variables explicitly. This allows for a slightly modified pricing algorithm that does not change as the branch-and-price progresses.

Let β_e be the set of dual variables associated with Constraints (10a). The new reduced costs for the polygon variables in Model **M4** are:

$$\overline{c_p} = 1 - \sum_{f \in \mathcal{A}: f \subseteq p} \alpha_f - \sum_{e \in p} \beta_e \qquad \forall p \in \mathcal{S}. \tag{3.11}$$

Now, we need to slightly modify the pricing algorithm to accommodate the addition of these constraints. The reduced cost of a triangle (k, l, m) is set to:

$$\Delta(k, l, m) = \beta_{kl} + \beta_{lm} + \beta_{km} + \sum_{f \subset (k,l,m)} \alpha_f. \tag{3.12}$$

Considering that the x variables represent support edges for the polygons, when building the polygon as the union of a triangle and a compatible polygon, the cost of the shared edge needs to be subtracted twice.

Notice that by setting an edge variable x_e to one, we implicitly forbid any edge that crosses it to be part of the solution. Hence, when making this branching decision, we also explicitly set all crossing edges to zero in the corresponding sub-problems. Besides, to prevent polygons with forbidden edges to be generated, we set the dual cost associated with the constraints corresponding to those edges to an arbitrarily large number.

Instead of using the default branching rules, we implemented a simple one based on the geometry of the problem. To that end, we say that a variable x in a given solution is *more fractional* the closer the value of $frac(x) = |0.5 - x|$ is to zero. Among all edge variables x_e such that $frac(x_e)$ is within 0.1 of the most fractional edge variable, we branch on the one whose corresponding edge has the highest number of crossings with all other edges in $E(P)$. This branching rule turned out to be very fast when compared to the default Strong Branching [1] implemented in SCIP.

3.4.2 Primal Heuristic

At the end of each BNP node, after pricing is finished, we use the edge variables to construct a greedy triangulation and solve an instance of the MCPP restricted to a small subset of edges, as described in [29], using Model (**M3**). This model can be modified to handle collinearities, by allowing Constraints (5d) to include points that are collinear with i and j in the set over which the left-hand side summation is applied. In the end of this section, we discuss how to adapt Constraints (5e) to handle collinear points.

To build the greedy triangulation Δ, the edges of $E(P)$ are sorted by descending value of their corresponding variables in the current LP solution of the RMP and inserted, if possible, in Δ, following this ordering. An edge is not inserted when it crosses one of the edges previously added to Δ. The result is an inclusion maximal set of non-crossing edges, which characterizes a triangulation.

A viable integer solution is then obtained by solving the MCPP restricted to the edges of this greedy triangulation and its flip edges. This heuristic runs very fast, taking less than 2 seconds for instances of up to 100 points.

It is possible that the solution found by an edge based heuristic contains polygons that have not yet been included in the RMP. This is addressed by running a secondary pricing algorithm whose purpose is solely to add the variables corresponding to the polygons present in heuristic solutions to the RMP.

3.4.3 Initial Primal Solution

To find an initial primal solution, we use the same heuristic as in the previous section. However, since no LP solution is available in the beginning, we replace the greedy triangulation with the Delaunay Triangulation.

3.4.4 Lower Bound and Early Stopping

At iteration t, the optimal solution of the current RMP z^t is not guaranteed to be a lower bound for Model **M4** unless no negative reduced cost variables were found by the pricing algorithm.

When an upper bound on the sum of the variables being priced $\kappa \geq \sum_{p \in \mathcal{S}} u_p$ is known, a lower bound for the Complete Model **M4** can be computed before pricing is finished. If $\overline{c^t}$ denotes the most negative reduced cost obtained during the t-th iteration of the pricing algorithm, a lower bound for Model **M4** is given by $z^t + \kappa \overline{c^t}$. Since the objective function of Model **M4** is $\sum_{p \in \mathcal{S}} u_p$, the value of any viable integer solution can be expressed as κ. However, this lower bound is poor, possibly even negative, during the first iterations. According to [22], a tighter bound $\underline{z^t}$ for the particular case of unitary cost objective functions, which does not depend on the quality of the incumbent solution, is given by:

$$\underline{z^t} = \frac{z^t}{1 - \overline{c_p^t}}. \tag{3.13}$$

Since all the coefficients in Model **M4** are integers, knowing a lower bound for z^t allows for an early halt of the column generation procedure. If $\lceil z^t \rceil = \lceil \underline{z^t} \rceil$, the integer lower bound at the current node cannot be improved by solving the RMP to optimality, and we can proceed directly to branching.

3.4.5 Stabilization

The lower bound given by (3.13) can oscillate between iterations and its convergence to the optimal value of the relaxation might be slow. Improving this convergence can significantly reduce total solving time of the ILP. This can be accomplished by the use of Stabilization techniques.

We can employ the dual bound given by (3.13) to assess the quality of a dual solution, where a higher lower bound indicates a better solution. One way to stabilize the algorithm is to minimize drastic changes in the dual solution, keeping it close to a known good

solution, also called the *stabilization center*. An in-depth discussion of the topic and different techniques to address the issue can be found in [28].

The dual solution can be stabilized by applying the smoothing technique proposed by Wentges [34]. Instead of using the current dual solution α^t in the pricing subroutine, the following convex combination can be considered

$$\alpha^t_{\text{STAB}} = \alpha^t + \lambda(\alpha_{\text{BEST}} - \alpha^t), \tag{3.14}$$

where $0 \leq \lambda < 1$ is the smoothing factor and α_{BEST} is the stabilization center corresponding to the dual solution with best lower bound found so far. We remark that tuning the parameter for this stabilization was very hard to accomplish, since instances would perform significantly better or worse as the parameter changed, averaging out little change.

Another way to reduce oscillations is to solve the linear relaxations using barrier methods instead of the commonly used Simplex algorithms. By changing the LP algorithm, we can take advantage of the fact that barrier methods find solutions that lie in the center of the optimal face, as opposed to the extreme points encountered by Simplex. In highly degenerate problems, extreme points can oscillate significantly between iterations with the addition of new columns and/or rows, while subsequent central points are uniquely defined and close together. Another advantage of this approach is that no parameter tuning is necessary [25].

On the other hand, when replacing Simplex with barrier methods, we lose the capability of fast re-optimization after branching, one of the most important features of Dual Simplex algorithms explored when implementing branch and cut procedures [35]. Also, due to the nature of the central solutions found by barrier methods, it is very unlikely that they are integral, increasing the need of a good primal heuristic.

In pure branch-and-cut algorithms, when branching is done or violated cutting planes are found, the addition of new constraints makes the current primal solution infeasible while maintaining its dual feasibility. In this case, the use of Dual Simplex as the LP algorithm to optimize the linear relaxations in the child nodes is recommended because, starting from the current dual feasible basis, it usually requires far fewer iterations to reach a new optimal solution than it would be necessary for a Simplex algorithm started from scratch.

Pricing works similarly, however, the addition of columns makes the current dual solution infeasible while maintaining primal feasibility. So, when combining both row and column generation as in a Branch-and-Price algorithm, if a single Simplex algorithm is used to compute relaxations, some re-optimization is inevitable, reducing the negative impact of switching to barrier methods.

In Section 3.5 we show that, for this particular model, the more stable pricing procedure obtained, by replacing Simplex with a Barrier Method, out-weights the re-optimization cost.

3.4.6 Degree Constraints

The addition of edge variables allows for the inclusion of the following degree constraints (5e) introduced in the Compact Model **M3** for point sets in general position:

$$\sum_{j \in P \setminus \{i\}} x_{ij} \geq 3, \forall i \in I(P) \tag{3.15}$$

Notice that these constraints do not involve polygon variables and, therefore, do not require modifications to the pricing problem.

In practice, we verified that the addition of degree constraints improved the quality of the lower bound provided by the Set Partition Model, and the total solving time. However, when using column generation, the simple addition of those constraints resulted in a larger number of calls to the column generation procedure, considerably increasing solving times.

To minimize this negative side effect, we separate the degree constraints after column generation rather than adding them all at once, despite the fact that there are only $O(n)$ of them. The embedding of a cutting plane procedure in the Branch-and-Price algorithm leads to a Branch-Cut-and-Price algorithm. Also, in our implementation, in an attempt to limit the number of pricing rounds, we require that an inequality be violated by at least 0.1 units to be inserted in the current linear relaxation. As shown in [29], the degree constraints can increase the lower bound by 0.5 even in very simple instances that, after rounding, may be just enough to assert a known primal solution as optimal, halting the optimization sooner.

However, Constraint (3.15) is not valid when multiple collinear points are present, as a vertex can have degree two and still be a vertex of a convex polygon.

The set of *neighbors* of i, denoted by $\mathcal{N}_i$, is comprised of the points $j \in P$ such that $\{i, j\} \in E(P)$. Among the neighbors of i, the ones in $\mathcal{N}_i^c = \{j : \text{CVX}(j, i, k) = 0 \text{ for some } k \in \mathcal{N}_i \setminus \{j\}\}$ are candidates for being the endpoints of edges incident to i in a solution where i has degree two. Hence, we replace (3.15) by the following more general (albeit equivalent when P is in general position) set of constraints:

$$\sum_{j \in \mathcal{N}_i \setminus \mathcal{N}_i^c} x_{ij} + \sum_{j \in \mathcal{N}_i^c} 1.5 x_{ij} \geq 3, \forall i \in I(P) \tag{3.16}$$

3.5 Experimental Results

In this section, we show the positive impact of some design choices and compare Model **M1** with Model **M3**, presented in [29].

All experiments were run on an Intel Xeon Silver 4114 at 2.2Ghz, and 32GB of RAM running Ubuntu 16.04. Models and algorithms were implemented in C++ v.11 and compiled with gcc 5.5. Geometric algorithms and data structures were implemented using CGAL 5.1 [32], using Gmpq for exact number representation. The compact model introduced in [29] was implemented using CPLEX 12.10, while the set-partition Model **M4** used SCIP 7.0 [17] with CPLEX as LP solver. A time limit of 3 hours was set for the ILP solver for each instance.

When running times are presented, we consider both the time to generate and to solve the model. Most of the data are presented in a standard boxplot, grouped by size. All data used to generate the figures is available at [4].

To compare the algorithms, we employ the instances from [29], available at [4]. Those instances had been generated by independently sampling x and y coordinates from a uniform distribution, ensuring general position.

In this study, we only use the instances of 65 to 105 points, with 30 instances per size, since smaller instances were too easy, while larger ones were too hard, given our limit of 3 hours of (exclusive) solver time. As we were pushing our models to their limit, we reached, for size 105, a large enough instance size for which multiple failures began to appear. For instances of 105 points, when more than one instance could not be solved by a given configuration, the solving times for this size are omitted. Despite the fact that a few of the instances could not be solved to optimality, we include them as clear outliers in some of the forthcoming figures. Notice that the most basic configuration of Model **M1** solved all instances of 65 points in at most 242 seconds, see Figure 3.12a, while the compact Model **M3** failed to solve any of them in 2400 seconds to provable optimality.

The main reason for the difference in performance is the quality of the lower bound provided by the relaxations of both models. See Figure 3.11.

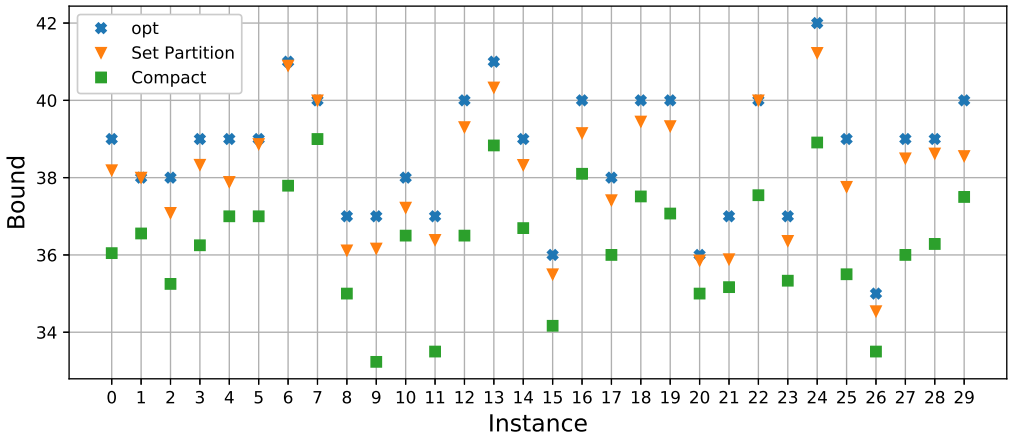
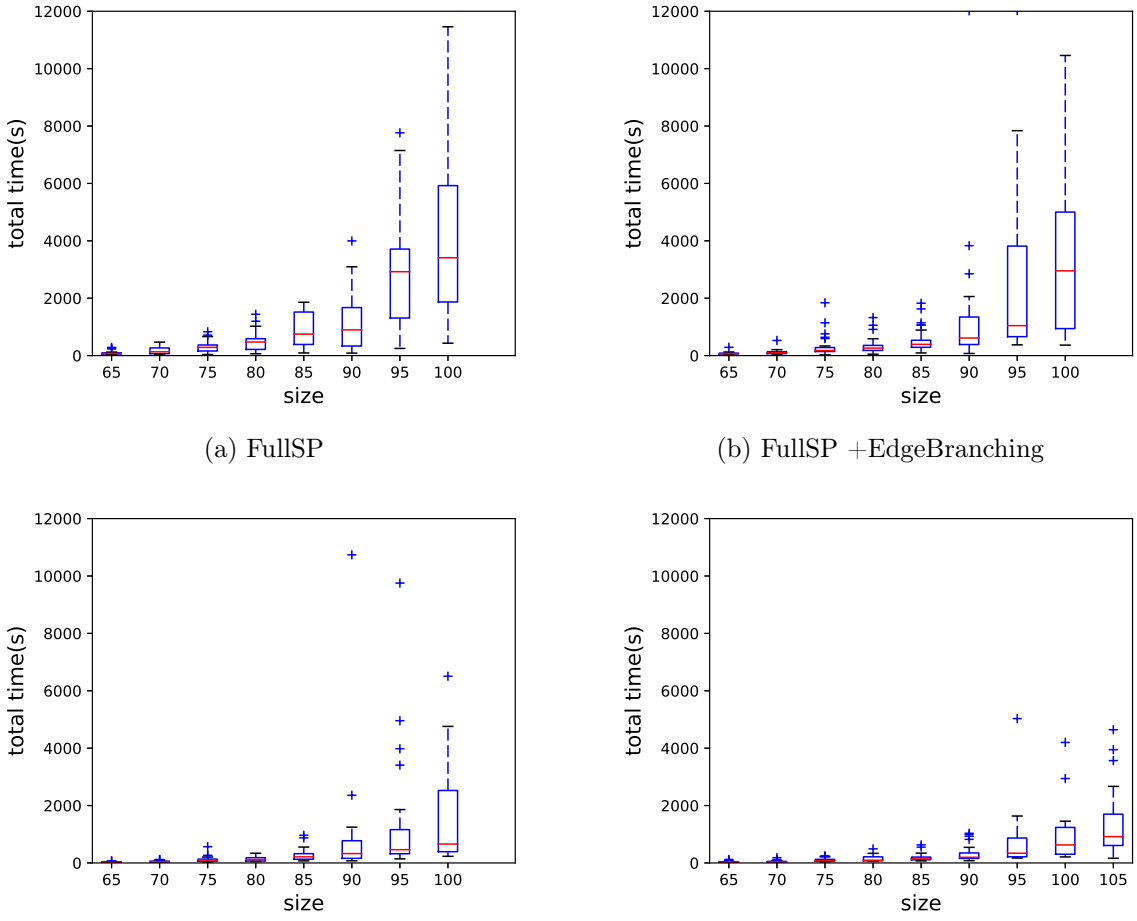


Figure 3.11: Lower bound of the relaxations given by the Compact Model and the Full Set Partition Model for instances of size 55 compared with the optimal solution.

In the first three experiments, we show the impact of improvements to the Model **M1** with all polygon variables, which we call *FullSP*, discussed in Section 3.4: branching on edges, heuristic, and degree constraints. Our next experiment focuses on Model **M4** using column generation, which we call *CGSP*, comparing the different types of stabilization methods and their impact. We conclude the experiments by comparing the memory consumption of FullSP and CGSP. Each experiment includes the features introduced in the previous ones.

Branch on Edge Variables We now discuss the impact of introducing edge variables and the branching rule described in Section 3.4.1 to the basic FullSP model, leading to Model **M4**. Figures 3.12a and 3.12b show the solving times, respectively, with and without the inclusion of edge variables, while Figures 3.13a and 3.13b depict the number

of nodes explored. Although there is an increase in the number of nodes explored, our implementation is faster than the one based on Strong Branch [1], the default branching rule for SCIP, leading to better total running times.



(a) FullSP

(b) FullSP +EdgeBranching

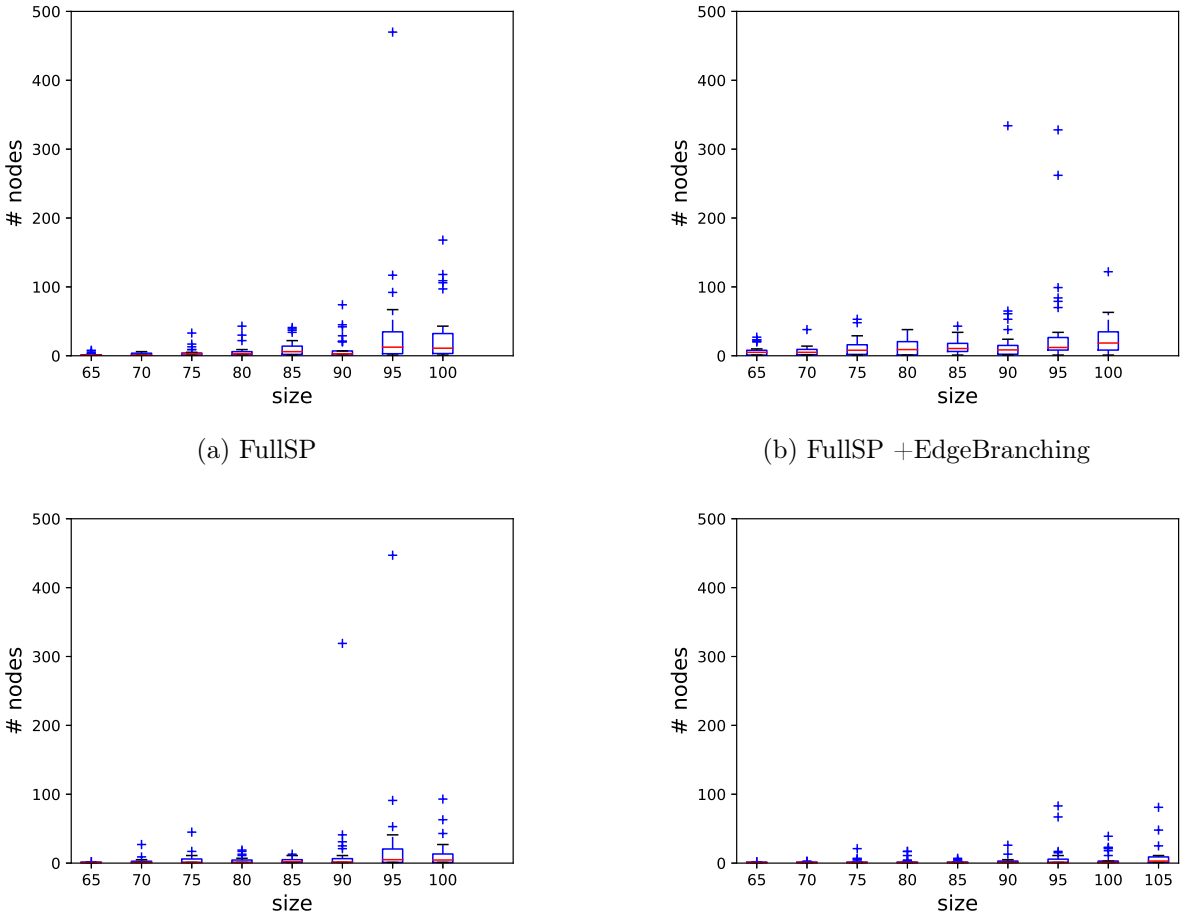
(c) FullSP +EdgeBranching+Heur

(d) FullSP +EdgeBranching+Heur+Deg

Figure 3.12: Total solving time for the FullSP with different configurations.

Primal Heuristic Next, we investigate the effect of introducing a custom primal heuristic. The positive impact of the inclusion of the primal heuristic described in Section 3.4.2 is evidenced by comparing Figures 3.12b and 3.12c. The lower bound provided by the model is very strong. Thus, having good heuristic solutions soon considerably reduces the number of nodes explored. See Figures 3.13b and 3.13c.

Degree Constraints Finally, we discuss the impact of the introduction of constraints (3.15). The performance gain from this inclusion is shown in Figures 3.12c and 3.12d. Degree constraints considerably increase the strength of the model, even improving the lower bound of simple instances, as discussed in [29].This is shown in Figure 3.14, where lower bounds of the FullSP, with and without Degree Constraints, are compared with the optimal values. This stronger formulation leads to fewer nodes being explored during search, as shown in Figure 3.13d. Instances with 105 points were solved, on average $\pm$ std-dev, in 1356 ± 1102 seconds when using degree constraints.



(a) FullSP

(b) FullSP +EdgeBranching

(c) FullSP +EdgeBranching+Heur

(d) FullSP +EdgeBranching+Heur+Deg

Figure 3.13: Number of nodes of the search tree explored during for different configurations of FullSP.

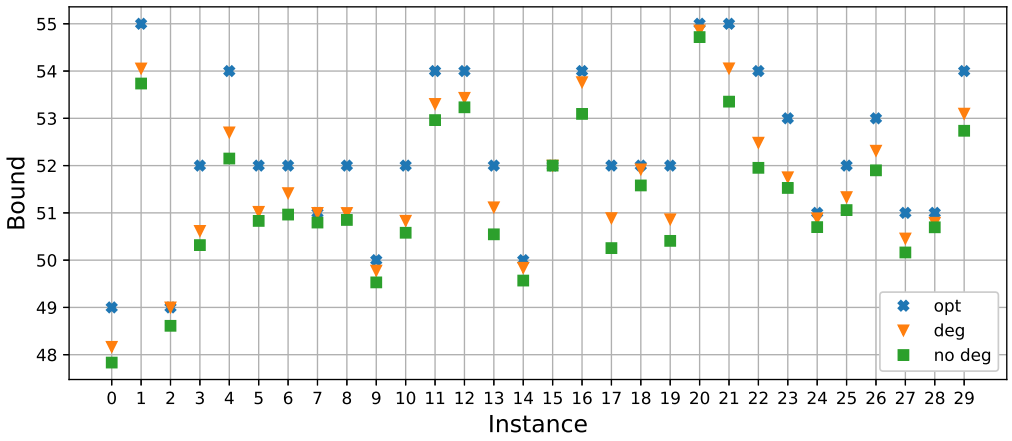
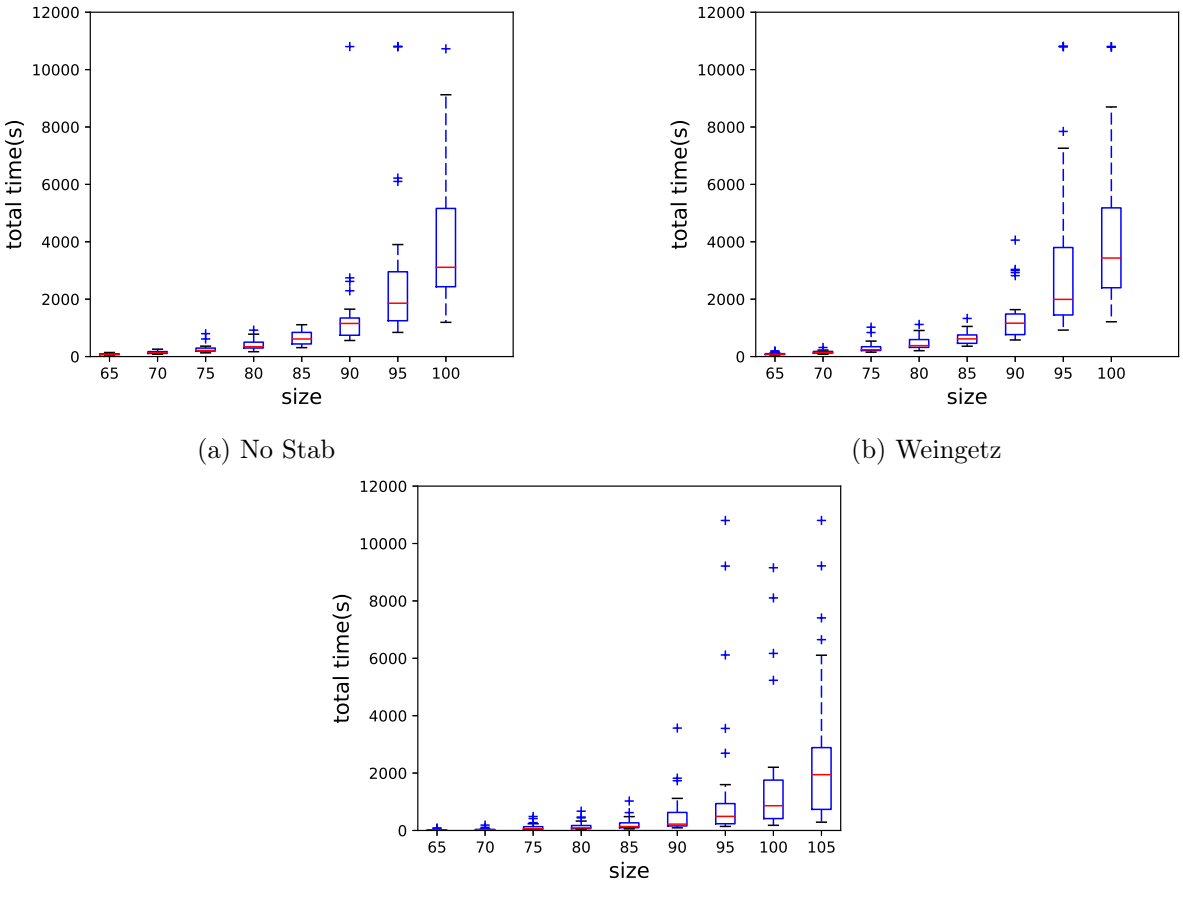


Figure 3.14: Lower bound of the relaxations given by the FullSP with and without Degree Constraints for instances of size 90 compared with the optimal value.

Column Generation All the previously discussed features lead to our best performing FullSP. However, due to the exponential number of columns, the model eventually becomes too large to fit in memory. In our initial testing, instances with 190 points used more than 32GB of RAM during model creation. For instances with 180 points, the solver started running with less than 2GB left, it is very likely that there would not be enough memory available after branching.

With the memory issue in mind, we try to find the best configuration for the CGSP. The RMP is initialized with the set of all triangles and the polygons that belong to the initial heuristic solution.

As mentioned in Section 3.4.6, the simple inclusion of the degree constraints significantly worsen the performance of the model. We overcame this issue by implementing a cutting plane procedure to separate degree constraints. Since the number of constraints is $O(n)$, the separation is made by inspection, only including cuts with a significant violation of at least 0.1.



(a) No Stab

(b) Weingetz

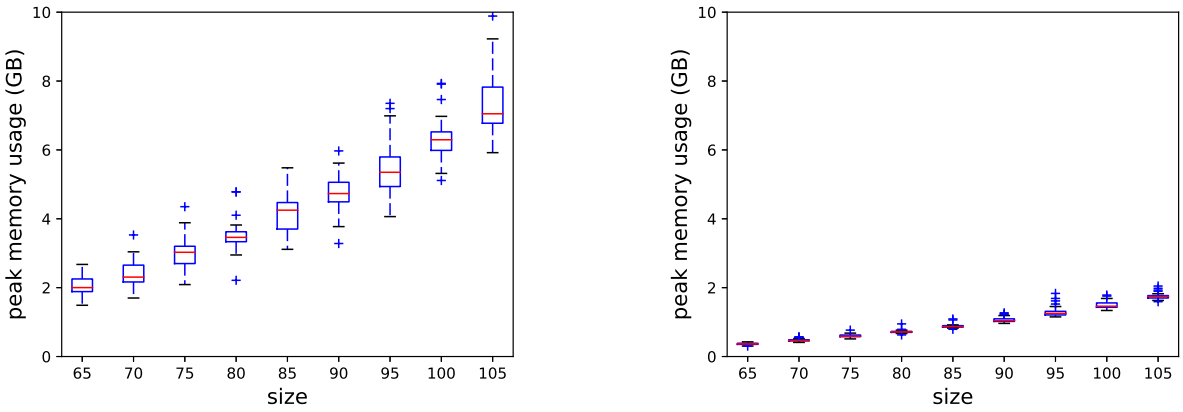
(c) Barrier

Figure 3.15: Total solving time for the CGSP for different set stabilization methods.

Finally, we compare different approaches to stabilize the column generation. As discussed in Section 3.4.5, we studied three options: no stabilization, Weingetz Stabilization, and Barrier Methods. For the Weingetz Stabilization, using the `irace` package [23], we found that the best stabilization parameter is $\alpha = 0.55$. The results are shown in Fig-

ure 3.15. As we can see, although no noticeable difference is observed when Weingetz method is used compared to not applying any stabilization at all, by replacing Simplex with the Barrier Method, we achieve a significant performance improvement. Instances with 105 points were solved, on average ± std-dev, in 2666 ± 2711 seconds when using the Barrier Method.

As expected, when the entire model fits into memory, a better performance is obtained compared to running a column generation algorithm. This is due not only to the time spent solving the pricing problem many times per node, but also to the fact that modern ILP solvers are equipped with extremely powerful pre-processing routines that can reach their maximum potential when the models are completely loaded into memory. However, the trade off between solution time and memory consumption, illustrated in Figure 3.16, reveal that column generation leads to significant savings in memory usage without drastic losses in performance. Thus, the technique is a good alternative when memory becomes the limiting factor to solving instances.



(a) FullSP

(b) CGSP Barrier

Figure 3.16: Peak memory consumption for the two variations of the Set Partition Model: full and column generation using the barrier method.

3.6 Conclusions

In this paper, we discussed a new model for the MCPP featuring variables assigned also to the convex polygons having vertices on the input point set, in contrast to the previous known formulation that only contained variables associated to edges with endpoints in that set. To cope with the exponential number of variables, we proposed a column generation based algorithm and discussed implementation aspects that made it more efficient. The aspects investigated include the deployment of stabilization methods, which led to the use of the Barrier Method for solving linear relaxations, the development of a primal heuristic and of a simple yet effective branching rule. Also, a family of cuts inspired by a previously known ILP model was incorporated to the algorithm which, because of the dynamic and simultaneous inclusion of variables and constraints to the model, is characterized as a *branch-and-cut-and-price* algorithm. All those different aspects were assessed through a series of experiments to show their individual contribution to the algorithm's performance.

As a result, constrained to identical computational resources, the new algorithm was able to solve instances with more than twice the size of what was possible in previous works.

A careful adaptation of the pricing algorithm to deal with multiple collinear points is certainly worth investigating. Even though this might be seen as an implementation issue, we could benefit from more general symbolic perturbation results such as [11] and [30] when handling triangles of zero area. However, in practice, such techniques can incur a high computational cost. A relevant topic for future work is to assess how the model performs when solving more structured instances that are not sampled from a uniform distribution. Other directions for further research include investigating a different stabilization model, and finding facet defining cuts.

Acknowledgments This work was supported in part by grants from: *Brazilian National Council for Scientific and Technological Development* (CNPq), #313329/2020-6, #309627/2017-6, #304727/2014-8; *São Paulo Research Foundation* (FAPESP), #2020 /09691-0, #2018/26434-0, #2018/14883-5. #2014/12236-1; and *Fund for Support to Teaching, Research and Outreach Activities* (FAEPEX).

Bibliography

[1] T. Achterberg. *Constraint Integer Programming*. Doctoral thesis, Technische Universität Berlin, Fakultät II - Mathematik und Naturwissenschaften, Berlin, 2007.

[2] B. Aronov, H. Edelsbrunner, L. J. Guibas, and M. Sharir. The number of edges of many faces in a line segment arrangement. *Combinatorica*, 12(3):261–274, 1992.

[3] D. Avis and D. Rappaport. Computing the largest empty convex subset of a set of points. In *Proceedings of the First Annual Symposium on Computational Geometry*, SCG'85, pages 161–167, New York, 1985. ACM.

[4] A. S. Barboza, C. C. de Souza, and P. J. de Rezende. Minimum Convex Partition of Point Sets – Benchmark Instances and Solutions, 2018. `www.ic.unicamp.br/~cid/Problem-instances/Convex-Partition`.

[5] C. Barnhart, E. L. Johnson, G. L. Nemhauser, M. W. P. Savelsbergh, and P. H. Vance. Branch-and-price: Column generation for solving huge integer programs. *Operations Research*, 46(3):316–329, 1998.

[6] M. W. Bern, H. Edelsbrunner, D. Eppstein, S. A. Mitchell, and T. S. Tan. Edge insertion for optimal triangulations. *Discrete and Computational Geometry*, 10:47–65, 1993.

[7] R. G. Cano, C. C. de Souza, P. J. de Rezende, and T. Yunes. Arc-based integer programming formulations for three variants of proportional symbol maps. *Discrete Optimization*, 18(C):87–110, November 2015.

[8] J. A. De Loera, J. Rambau, and F. Santos. *Triangulations: Structures for Algorithms and Applications*. Springer, 1st edition, 2010.

[9] P. J. de Rezende, C. C. de Souza, S. Friedrichs, M. Hemmer, A. Kröller, and D. C. Tozoni. Engineering art galleries. In L. Kliemann and P. Sanders, editors, *Algorithm Engineering - Selected Results and Surveys*, volume 9220 of *Lecture Notes in Computer Science*, pages 379–417. Springer, 2016.

[10] E. D. Demaine, S. P. Fekete, P. Keldenich, D. Krupke, and J. S. B. Mitchell. Computing convex partitions for point sets in the plane: The CG: SHOP challenge 2020. *CoRR*, abs/2004.04207, 2020.

[11] H. Edelsbrunner and E. P. Mücke. Simulation of simplicity: A technique to cope with degenerate cases in geometric algorithms. *ACM Transactions on Graphs*, 9(1):66–104, 1990.

[12] S. P. Fekete, A. Haas, M. Hemmer, M. Hoffmann, I. Kostitsyna, D. Krupke, F. Maurer, J. S. B. Mitchell, A. Schmidt, C. Schmidt, and J. Troegel. Computing nonsimple polygons of minimum perimeter. *Journal on Computational Geometry*, 8(1):340–365, 2017.

[13] S. P. Fekete, W. Hellmann, M. Hemmer, A. Schmidt, and J. Troegel. Computing maxmin edge length triangulations. *Journal on Computational Geometry*, 9(1):1–26, 2018.

[14] T. Fevens, H. Meijer, and D. Rappaport. Minimum convex partition of a constrained point set. *Discrete Applied Mathematics*, 109(1-2):95–107, 2001.

[15] B. A. Foster and D. M. Ryan. An integer programming approach to the vehicle scheduling problem. *Journal of the Operational Research Society*, 27(2):367–384, June 1976.

[16] S. Friedrichs, M. Hemmer, J. King, and C. Schmidt. The continuous 1.5d terrain guarding problem: Discretization, optimal solutions, and PTAS. *Journal on Computational Geometry*, 7(1):256–284, 2016.

[17] G. Gamrath, D. Anderson, K. Bestuzheva, W.-K. Chen, L. Eifler, M. Gasse, P. Gemander, A. Gleixner, L. Gottwald, K. Halbig, G. Hendel, C. Hojny, T. Koch, P. Le Bodic, S. J. Maher, F. Matter, M. Miltenberger, E. Mühmer, B. Müller, M. E. Pfetsch, F. Schlösser, F. Serrano, Y. Shinano, C. Tawfik, S. Vigerske, F. Wegscheider, D. Weninger, and J. Witzig. The SCIP Optimization Suite 7.0. Technical report, Optimization Online, March 2020.

[18] J. García-López and M. Nicolás. Planar point sets with large minimum convex partitions. In *Abstracts 22nd European Workshop on Computational Geometry*, pages 51–54, 2006.

[19] N. Grelier. Minimum Convex Partition of Point Sets is NP-Hard. *CoRR*, abs/1911.07697, 2019.

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

[21] C. Knauer and A. Spillner. Approximation algorithms for the minimum convex partition problem. In *Proceedings of the 10th Scandinavian Workshop on Algorithm Theory (SWAT)*, volume 4059 of *Lecture Notes in Computer Science*, pages 232–241. 2006.

[22] M. E. Lübbecke and J. Desrosiers. Selected topics in column generation. *Operations Research*, 53(6):1007–1023, 2005.

[23] M. López-Ibáñez, J. Dubois-Lacoste, L. Pérez Cáceres, T. Stützle, and M. Birattari. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58, 2016.

[24] M. Akella, S. Gupta and A. Sarkar. Branch and Price: Column generation for solving huge integer programs, 2021. `https://web.archive.org/web/20100821132913/http://www.acsu.buffalo.edu/~nagi/courses/684/price.pdf`, [Online; accessed 18-May-2021].

[25] P. A. Munari and J. Gondzio. Using the primal-dual interior point algorithm within the branch-price-and-cut method. *Computers & Operations Research*, 40(8):2026–2036, 2013.

[26] V. Neumann-Lara, E. Rivera-Campo, and J. Urrutia. A note on convex decompositions of a set of points in the plane. *Graphs & Combinatorics*, 20(2):223–231, 2004.

[27] M. J. O. Zambon, P. J. de Rezende, and C. C. de Souza. Solving the geometric firefighter routing problem via integer programming. *European Journal of Operational Research*, 274(3):1090–1101, 2019.

[28] A. A. Pessoa, R. Sadykov, E. Uchoa, and F. Vanderbeck. In-out separation and column generation stabilization by dual price smoothing. In V. Bonifaci et al., editor, *12th International Symposium on Experimental Algorithms, SEA, Proceedings*, volume 7933 of *Lecture Notes in Computer Science*, pages 354–365. Springer, 2013.

[29] A. Sapucaia-Barboza, C. C. de Souza, and P. J. de Rezende. Minimum convex partition of point sets. In P. Heggernes, editor, *Algorithms and Complexity - 11th International Conference, CIAC 2019, Rome, Italy, May 27-29, 2019, Proceedings*, volume 11485 of *Lecture Notes in Computer Science*, pages 25–37. Springer, 2019.

[30] R. Seidel. The nature and meaning of perturbations in geometric computing. *Discrete and Computational Geometry*, 19(1):1–17, 1998.

[31] A. Spillner. A fixed parameter algorithm for optimal convex partitions. *Journal of Discrete Algorithms*, 6(4):561–569, 2008.

[32] The CGAL Project. *CGAL User and Reference Manual*. CGAL Editorial Board, 5.1 edition, 2020.

[33] D. C. Tozoni, P. J. de Rezende, and C. C. de Souza. Algorithm 966: A practical iterative algorithm for the art gallery problem using integer linear programming. *ACM Transactions on Mathematical Software*, 43:16:1–16:27, 2016.

[34] P. Wentges. Weighted Dantzig-Wolfe decomposition for linear mixed-integer programming. *International Transactions in Operational Research*, 4(2):151–162, 1997.

[35] L. Wolsey. *Integer Programming*. John Wiley & Sons, Inc., 1998.

3.7 Additional Comments

Errata on Equation 3.4: As expressed in Equation 3.4, $B(k, l, m)$ does not compute the minimum reduce cost, which is instead given by $1 - B(k, l, m)$. Accordingly, in the same equation, min should be replaced with max. This is a presentation error and does not reflect how reduced costs were implemented.

Chapter 4

Convex Bichromatic Quadrangulation of Point Sets

In this paper, we proposed a optimization variant of the Convex Bichromatic Quadrangulation of Point Sets. To tackle this new variation, we introduced an ILP model, adapting some of the results for the arrangement-based mode for the Minimum Convex Partition of Point Sets. We also present heuristic based on this model, and draw connections with matching approaches from the literature.

This chapter corresponds to the paper [17] presented at the *33rd Canadian Conference on Computational Geometry (CCCG 2021)*. It was co-authored with Andre A. Cire[1], Pedro J. de Rezende[2] and Cid C. de Souza[2].

Abstract In this paper, we investigate an $\mathbb{NP}$-hard optimization variant of the well-known convex quadrangulation problem. Given a point set P and a bicoloring C, we wish to find an alternative bicoloring C' that is (a) closest to C with respect to their symmetric difference, and (b) such that P admits a bichromatic convex quadrangulation under the new coloring C', i.e., a partition of the convex hull of P into convex quadrangles where the endpoints of each edge have distinct colors in C'. Thus, the resulting problem combines, and provides solution insights to, two decision problems from the literature: whether P has a convex quadrangulation, and whether P admits a bichromatic convex quadrangulation under the original coloring C.

Due to our optimization perspective, we present two mathematical programming approaches to address the problem. The first is an extension of the standard integer linear model for the quadrangulation problem, and the second is a novel set-partition formulation that enumerates colored convex quadrangles. We also derive new heuristics that exploit the models' linear programming relaxations, drawing connections to existing matching approaches used for quadrangulation problems in the literature. We present an empirical study assessing the effectiveness of the models for both exact and heuristic

[1]Department of Management, University of Toronto Scarborough, Canada

[2]Institute of Computing, University of Campinas, Campinas, Brazil. Supported by CNPq under grants #313329/2020-6, #309627/2017-6, #306454/2018-1; and Fapesp under grants #2018/26434-0, #2018/14883-5, #2014/12236-1.

approaches, along with comparisons between the heuristics. The benchmark used to evaluate the performance of the models and heuristics is also made publicly available.

4.1 Introduction

A *quadrangulation* of a set $P \subset \mathbb{R}^2$ with n points is a partition of its convex hull, denoted by $\mathrm{CH}(P)$, into interior disjoint quadrangles whose vertices belong to P and are *empty*, which means that no points of P lie in their interior. We say that P is *quadrangulable* if it admits a quadrangulation. A necessary condition for a point set P to be quadrangulable is that $\mathrm{CH}(P)$ has an even number of vertices [3].

One of the most studied problems in computational geometry, both in theory and in practice, is the Triangulation Problem, i.e., how to partition the convex hull of a point set into interior disjoint triangles. The study of quadrangulations follows naturally because it allows for the decomposition to use half as many partitions.

A quadrangulation is said to be *convex* if its quadrangles are convex. The Convex Quadrangulation of Point Sets Problem (CQPS) decides whether a point set admits a convex quadrangulation, which is a required property in many practical applications [13]. The question of its complexity, however, is still open. As there are no known polynomial algorithms for this problem, applications often relax the quadrangulation requirement and strive for including as many quadrangles as possible, either allowing some triangles [1, 9] or employing Steiner points to achieve a quadrangulation [6, 10]. An exact algorithm for an optimization version of the problem was proposed in [5], with complexity $O(n^{3h+1})$, where h is the number nested convex hulls of P.

A *coloring* of a point set P is a partition of its points into disjoint subsets. Each of these subsets is referred to as a *color*. A *bicoloration* $C = (R, B)$ of a point set P is a coloring of P into exactly two colors R and B. We say that a point $p \in P$ is colored *red*, with respect to (w.r.t.) $C = (R, B)$ if $p \in R$. Similarly, if $p \in B$, p is said to be colored *blue*. A point set P together with a bicoloration $C = (R, B)$ defines a *bichromatic point set*, denoted $P|C$.

Given a bichromatic point set $P|C$, a quadrangle with vertices in P is said to be *bichromatic* if all its edges have endpoints of different colors. A quadrangulation of a point set P is said to be *bichromatic w.r.t. a bicoloration* C if all its quadrangles are bichromatic. In the Convex Quadrangulation of Bichromatic Point Sets Problem (CQBPS), we decide whether a bichromatic point set admits a *bichromatic quadrangulation*. This problem was proposed in [8], where it was proved to be $\mathbb{NP}$-hard.

All colorings are to be understood as bicolorations henceforth. A *flip* of a point $p \in P$ w.r.t. a coloring C corresponds to changing its color. The *flip distance* between two colorings C and C' is the number of flips necessary to make them equal and is denoted by $F(C, C')$. The flip distance between $C = (R, B)$ and $C' = (R', B')$ is given by $F(C, C') = R \Delta R' = B \Delta B' = F(C', C)$, where Δ denotes the symmetric difference. Two colorings of P are *opposite* if their flip distance is $n = |P|$. It is easy to see that a quadrangulation of a point set P induces two opposite colorings of P.

Problem Statement. In the Convex Bichromatic Quadrangulation of Point Sets with Minimum Color Flips Problem (CQBPSMF), given a point set P and a coloring C, we want to find a coloring C' of P, if it exists, with minimum flip distance from C such that P

admits a bichromatic convex quadrangulation with respect to C'.

Unlike the previously introduced quadrangulation problems, the CQBPSMF is an optimization problem and can have three possible answers. Either P does not admit any convex quadrangulation and the sought after coloring C' of P does not exist; or P admits a bichromatic convex quadrangulation w.r.t. C, i.e., $C' = C$; or C' exists and $F(C, C') > 0$, meaning that P admits convex quadrangulations but no bichromatic convex quadrangulation w.r.t. C. It follows that the CQBPSMF is also $\mathbb{NP}$-hard from the hardness proof of the CQBPS [8].

Our Contributions. In this paper, we introduce a new optimization $\mathbb{NP}$-hard variant of the quadrangulation problem and present two Integer Linear Program models to solve it. We also propose several heuristics that build upon such exact models. These heuristics are compared with the matching approach for other quadrangulation problems from the literature. A comprehensive empirical study assessing the models for both exact and heuristic approaches is reported, as well as comparisons between the heuristics. To allow for future comparisons to our results, the complete benchmark used to appraise the performance of the algorithms is made publicly available.

The work is organized as follows. Section 4.2 presents an Integer Linear Programming (ILP) model for the CQBPSMF, while Section 4.3 introduces a set-partition based formulation. Section 4.4 studies heuristics based on the previously introduced models. An empirical study of the exact and heuristic methods is shown in Section 4.5, and Section 4.6 presents our conclusions.

4.2 Integer Linear Programming Model

In this section, we introduce a formulation (model) for the the CQBPSMF. This is done by starting from a standard set-partition formulation for the quadrangulation problem and adding constraints and variables to assign colors and count flips. We begin by defining important ILP concepts and notations.

A *linear relaxation* of an ILP model is obtained by dropping the integrality requirement of its variables. In the case of binary variables, instead of assuming values in the set {0,1}, relaxed variables assume values in the interval [0,1]. In this text, we will refer to the linear relaxation simply as relaxation.

Relaxations are a major component of the most popular algorithm for solving ILP models, known as *Branch-and-Bound*. In a Branch-and-Bound algorithm, the set of solutions of a model is searched in a rooted tree-like form. At each node, a linear relaxation is solved obtaining an optimal solution x^*. If the relaxation has a worse objective function value than the best known integral solution or is infeasible, nothing is done. If there are any variables assuming a fractional value, one of them, say x_i^*, is selected for branching leading to two sub-problems, one with the additional constraints $x_i \leq \lfloor x_i^* \rfloor$ and the other with additional constraints $x_i \geq \lceil x_i^* \rceil$. Starting from a root node consisting of the relaxation of the original model with no additional constraints, the algorithm proceeds until there are no nodes left for processing. Additional heuristics are run at each node with the goal of finding improved integral solutions, which results in fewer nodes having to be explored.

A *set-partition constraint* is of the form $\sum_{x\in\mathcal{X}} x = 1$, where $\mathcal{X}$ is a set of binary variables. A *set-partition formulation* is a formulation comprised exclusively of set-partition constraints.

Modern ILP solvers are equipped with different heuristics, preprocessing techniques, and methods to derive additional constraints that reduce solving times of many formulations, such as the ones investigated here.

Given a set P of n points, let $L(P)$ denote the set of $\Theta(n^2)$ line segments $\overline{jk}$, where j and k are distinct points in P. The *complete (geometric) graph* induced by P is $G(P) = (P, E(P))$, where $E(P) = \{\{j,k\} : \overline{jk} \in L(P)\}$. We refer to a segment $\overline{jk} \in L(P)$ and the corresponding edge $\{j,k\} \in E(P)$ interchangeably. An edge $\{j,k\} \in E(P)$ is said to be *bichromatic with respect to a coloring* $C = (R, B)$ of P if its endpoints have different colors, i.e., $1 = |\{j,k\} \cap R| = |\{j,k\} \cap B|$.

Let $\mathcal{A}(P)$ denote the set of bounded faces of the planar arrangement induced by $L(P)$, and $\mathcal{Q}(P)$ denote the set of $O(n^4)$ empty convex quadrangles with endpoints in P. We say that a quadrangle $q \in \mathcal{Q}$ *covers* a face $f \in \mathcal{A}(P)$, denoted by $f \subset q$, if the interior of f is contained in the interior of q.

We associate a binary variable r_i to each point $i \in P$, such that point i is assigned the color red if, and only if, $r_i = 1$. To each edge $e \in E(P)$, we associate a binary variable x_e so that e is an edge of the quadrangulation if, and only if, $x_e = 1$. Finally, to each quadrangle $q \in \mathcal{Q}(P)$, we associate a binary variable u_q that takes value 1 if, and only if, q is part of the quadrangulation. This allows us to express our Model (4.1) as follows.

$$\min \quad \sum_{i\in B} r_i + \sum_{i\in R}(1 - r_i) \tag{4.1a}$$

$$\text{s.t.} \quad x_{ij} + r_i + r_j \le 2 \qquad \forall \{i,j\} \in E(P) \tag{4.1b}$$

$$x_{ij} \le r_i + r_j \qquad \forall \{i,j\} \in E(P) \tag{4.1c}$$

$$\sum_{q\in\mathcal{Q}(P):f\subseteq q} u_q = 1 \qquad \forall f \in \mathcal{A}(P) \tag{4.1d}$$

$$\sum_{q\in\mathcal{Q}(P):\{i,j\}\in q} u_q = 2x_{ij} \qquad \forall \{i,j\} \in E(P)\setminus \mathrm{CH}(P) \tag{4.1e}$$

$$\sum_{q\in\mathcal{Q}(P):\{i,j\}\in q} u_q = x_{ij} \qquad \forall \{i,j\} \in \mathrm{CH}(P) \tag{4.1f}$$

$$r_i \in \{0,1\} \qquad \forall i \in P \tag{4.1g}$$

$$u_q \in \{0,1\} \qquad \forall q \in \mathcal{Q}(P) \tag{4.1h}$$

$$x_{ij} \in \{0,1\} \qquad \forall \{i,j\} \in E(P) \tag{4.1i}$$

Constraints (4.1b) and (4.1c) ensure that an edge can only be selected if its endpoints have different colors. Constraints (4.1d) enforce the partitioning of $\mathrm{CH}(P)$ into empty convex quadrangles. Finally, Constraints (4.1e) and (4.1f) links edges and quadrangles.

In [11], we showed that for problems where the convex hull of a point set P has to be partitioned into polygons with endpoints in P, only $O(n^2)$ faces of $\mathcal{A}(P)$ need to be considered (as in Constraints (4.1d)). This will be discussed in detail in the next section.

Constraints (4.1d), together with quadrangle variables, define a standard set-partition formulation, with a known strong linear relaxation [7]. A similar formulation was used

to solve the Minimum Convex partition Problem [11]. Formulations for the CQPS and CQBPS can be trivially obtained from them.

On the other hand, Constraints (4.1b) and (4.1c), together with the edge and color variables, correspond to a formulation for the classical *MaxCut* problem with positive weights [4]. In the MaxCut problem, the goal is to find a bi-partition of the graph (in our case each part corresponds to a color) such that the total weight of the edges between both parts is maximized. This formulation only works for positive weights, as we are allowed to not select an edge even if its endpoints are in different parts. Notably, this MaxCut formulation is known for its weak linear relaxation. We will show in Section 4.5 that the linear relaxation of Model (4.1) behaves poorly as the size of instances grows, resulting in the total solving times increasing very quickly.

4.3 Pure Set-Partition Formulation

In the previous section, we presented a model for the CQBPSMF and discussed that it is not a strong one due to a subset of its constraints that define a weak formulation for the MaxCut problem.

In this section, we introduce a new model that consists entirely of set-partition constraints, which are known to lead to a strong linear relaxation. To this end, we introduce colored quadrangles and explore properties of the set partition constraints (4.1d).

First, notice that each quadrangle in $\mathcal{Q}(P)$ can be colored in only two ways such that its edges are bichromatic. Let us denote the set of all *colored empty convex quadrangles* of P by $\mathcal{Q}_c(P)$.

As proved in [11], to reach a partition of $\mathrm{CH}(P)$, we are only required to use Constraints (4.1d) for faces of $\mathcal{A}(P)$ with at least one vertex in P. The faces with $i \in P$ as one of their vertices are called i-wedges and we say that i is an *anchor* of these i-wedges. Note that faces may have multiple anchors. We denote the set of anchors of an i-wedge f by $\mathrm{ANC}(f)$. Finally, let us denote by $\mathcal{W}$ the set of i-wedges for all $i \in P$.

Given an i-wedge $f \in \mathcal{W}$ and an anchor $i \in \mathrm{ANC}(f)$, we can split the colored quadrangles that cover f into two sets based on the color assigned to i in their coloration. We denote by R_i^f and B_i^f the sets of colored quadrangles that cover f where point i is colored red and blue, respectively. Figure 4.1 illustrate an arrangement and its i-wedges.

We write Constraints (4.1d) for the i-wedges in $\mathcal{W}$ considering the set of colored convex quadrangles as

$$\sum_{q \in \mathcal{Q}_C : f \subset q} u_q = 1, \forall f \in \mathcal{W}.$$

The summation on the left-hand side can be split based on the color assignments to the anchors of f, leading to

$$\sum_{q \in R_i^f} u_q + \sum_{q \in B_i^f} u_q = 1, \forall f \in \mathcal{W}, \forall i \in \mathrm{ANC}(f)$$

However, in any integral solution, there is only one colored polygon covering each face f and, for all anchors $i \in \mathrm{ANC}(f)$, it must be from R_i^f when $r_i = 1$, and from B_i^f,

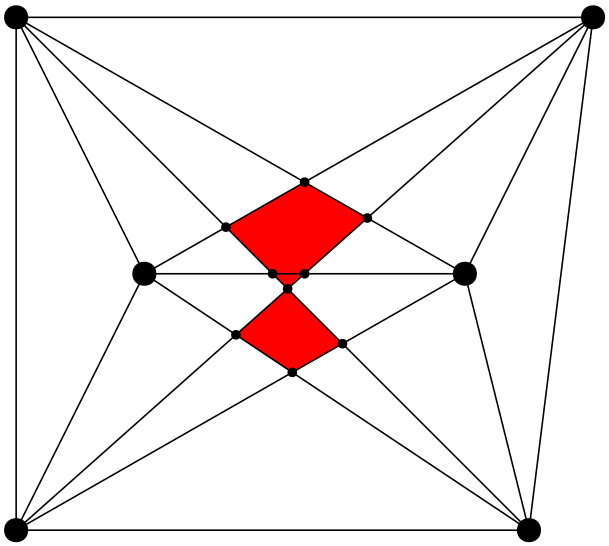


Figure 4.1: Illustration of the arrangement induced by $L(P)$ for the set P of points indicated by the larger disks. The smaller disks show vertices of the arrangement that are crossings of line segments. Faces of the arrangement that are not i-wedges are colored red.

otherwise. This allows us to split these constraints, leading to the following model.

$$\min \quad \sum_{i \in B} r_i + \sum_{i \in R} (1 - r_i) \tag{4.2a}$$

$$\text{s.t.} \quad \sum_{q \in R_i^f} u_q = r_i, \qquad \forall f \in \mathcal{W}, \forall i \in \text{ANC}(f) \tag{4.2b}$$

$$\sum_{q \in B_i^f} u_q = 1 - r_i, \qquad \forall f \in \mathcal{W}, \forall i \in \text{ANC}(f) \tag{4.2c}$$

$$r_i \in \{0, 1\} \qquad \forall i \in P \tag{4.2d}$$

$$u_q \in \{0, 1\} \qquad \forall q \in \mathcal{Q}(P) \tag{4.2e}$$

For better visualization of the model as a pure set-partition formulation and to better interpret its constraints, an equivalent model can be obtained by adding binary variables b_i for all $i \in P$ such that $b_i = 1 - r_i$, allowing us to rewrite the model as follows.

$$\min \quad \sum_{i \in B} r_i + \sum_{i \in R} b_i \tag{4.3a}$$

$$\text{s.t. } r_i + b_i = 1 \qquad \forall i \in P \tag{4.3b}$$

$$\sum_{q \in R_i^f} u_q + b_i = 1, \qquad \forall f \in W, \forall i \in \mathrm{ANC}(f) \tag{4.3c}$$

$$\sum_{q \in B_i^f} u_q + r_i = 1, \qquad \forall f \in W, \forall i \in \mathrm{ANC}(f) \tag{4.3d}$$

$$r_i \in \{0,1\} \qquad \forall i \in P \tag{4.3e}$$

$$b_i \in \{0,1\} \qquad \forall i \in P \tag{4.3f}$$

$$u_q \in \{0,1\} \qquad \forall q \in \mathcal{Q}(P) \tag{4.3g}$$

Constraints (4.3c) ensure that either point i is assigned the color blue or there is exactly one colored quadrangle covering the i-wedge f where i is colored red. A similar interpretation can be given to (4.3d) by exchanging the colors. Constraints (4.3b) where added to ensure that a single color is assigned to each point.

This model performs significantly better in practice than the one introduced in the previous section, as we will show in Section 4.5.

4.4 Linear Relaxation Based Heuristics

In this section, we propose a heuristic framework for the CQBPSMF, based on the strength of the Model (4.2).

Our heuristics work in three steps. First, we solve the relaxation of Model (4.2), obtaining a solution u^*. Based on u^*, a set of colored quadrangles $S_c \subset \mathcal{Q}_c$ is constructed. Finally, we solve Model (4.2) restricted to the colored quadrangles in S_c. Notice that if the relaxation is infeasible, so is the restricted problem.

This is motivated by the global view that an linear programming relaxation provides, as opposed to merely local properties provided by, for instance, projected Delaunay Tetrahedralizations [13]. Thus, even when fractional, the values assigned to variables are still within the constraints of the model, and might give a good guidance for heuristic decisions.

An advantage of relaxation-based heuristics is the possibility of integrating them with a Branch-and-Bound algorithm. This is possible since a solution to the linear relaxation is always available. In particular, modern solvers are capable of finding good set-partition solutions using their own internal heuristics. However, if new constraints and variables are added with the goal of strengthening the model, the solver might face difficulties in finding good solutions. With this in mind, even if the relaxation takes a considerable amount time to be solved, having a fast relaxation-based algorithm can significantly speedup the solution to an ILP model.

The choice of colored quadrangles in S_c plays an important role for the success of the heuristic. In particular, a poor choice of quadrangles might lead to an infeasible model. Conversely, if too many quadrangles are included, the problem may become as hard as

the unrestricted version. We propose three different ways to construct S_c: one is based on quadrangles, one relies on edges and another is grounded on triangulations.

4.4.1 Quadrangulation-based

A trivial way to construct S_c, given a solution u^* to its relaxation, is to include all quadrangles whose variables have positive value in the relaxation, i.e., $S_c = \{q : u^*_q > 0\}$. However, this choice fails to lead to feasible solutions for many instances.

This approach works best when there are ways to ensure feasibility by adding a specific set of variables to the ILP model, or when good procedures to repair feasibility are known.

4.4.2 Edge-based

As an attempt to increase the number of feasible solutions found, one might expand the quadrangles selected. In our case, we can explore a neighborhood of the polygons whose values are positive in the relaxation. One way to achieve this is to project the values of polygon variables onto edges, similarly to how it is done in Model (4.1).

For Model (4.2), the projected value of an edge e can be obtained, based on Constraint (4.1f) and (4.1e), as $\sum_{u_t : e \in t} \frac{u_q}{2}$. This encourages us to include all polygons whose edges have positive projected values, i.e., $S_c = \{q : \sum_{u_t : e \in t} u_q > 0, \forall e \in q\}$. Indeed, this approach results in a higher number of feasible solutions being found. However, the growth in the number of quadrangles substantially increases solving times, leading to a heuristic that does not scale well with the size of the set of points.

4.4.3 Triangulation-based

Lastly, we propose a third approach based on triangulations that tries to combine the benefits of the previous heuristics. This insight comes from the fact that most of the heuristics for the CQPS are based on triangulations.

We begin with a greedy triangulation based on the value of the relaxation projected onto the edges and find quadrangles supported by those edges. Actually, when starting from a triangulation, the ILP model is not even necessary for solving the resulting restricted model, as it becomes equivalent to finding a matching of adjacent triangles, a standard approach used in the literature for quadrangulation problems [3, 9].

Let $x_e = \sum_{u_t : e \in t} u^*_q$ denote the projected value of edge $e \in E(P)$. A greedy triangulation can be constructed by sorting the edges of $E(P)$ by their projected values in descending order and attempting to add them, in that order, to an initially empty partial triangulation. An edge is successfully added if it does not cross any previously inserted edge, otherwise, it is rejected. The result is a greedy triangulation.

An edge of a triangulation admits an *edge-flip* if the two triangles adjacent to it can be merged to form a convex quadrangle – one of whose diagonals is the given edge. Its edge-flip is the other diagonal of this convex quadrangle.

If T denotes the set of all edges of a given (say, greedy) triangulation $\mathcal{T}$ and F denotes the set of edge-flips of T, we can identify the set of the aforementioned quadrangles as $S_c = \{q : e \in T \cup F, \forall e \in q\}$.

4.4.4 Matchings and Edge-Flips

In the literature on the CQPS, to the best of our knowledge, there are no prior attempts at edge-flips. Most of the work based on triangle matching is either focused on specific triangulations, such as Delaunay and Serpentine [2], or includes randomized components. The larger angles on a Delaunay triangulation lead to more pairs of triangles that can be matched. The Serpentine quadrangulation, in turn, admits a perfect match if convexity is not required, provided that the convex hull of the point set has an even number of points.

We recall that a matching takes place on a subgraph $H = (\Delta, D)$ of the dual graph of a triangulation $\mathcal{T}$, where each vertex in Δ corresponds to a triangle, and the edges[3] in D are those that connect pairs of vertices that represent two triangles that share a side thereby forming a convex quadrangle. A partial quadrangulation can be constructed by finding a matching M on the graph H. However, this approach usually leaves unmatched vertices of Δ, i.e., triangles of $\mathcal{T}$.

Geometrically speaking, to exchange an edge e of $\mathcal{T}$ for its edge-flip $e^{\perp}$ can be seen as an operation that locally modifies $\mathcal{T}$ creating a new triangulation $\mathcal{T}'$. Let H' be the subgraph of the dual graph of $\mathcal{T}'$ constructed in a fashion analogous to the description of the previous paragraph. On H', most of the matching M is preserved. However, by searching for augmenting paths for M on H', we may still be able to further reduce the number of unmatched triangles. This can be thought of as a local search procedure.

In this context, solving an ILP restricted to the quadrangles obtained from both the edges of a triangulation and their edge-flips is an attempt at globally navigating a neighborhood (by edge-flips) of a given triangulation.

As we show in the next section, the resulting ILP can easily be solved by modern solvers and this approach can straightforwardly be adapted for the CQPS.

4.5 Experiments

In this section, we compare the solving time and strength of the relaxation of our two exact models. We also show how our heuristics behave in practice.

To do this, we generated a benchmark comprised of two sets of instances, ranging in size from 40 to 120 points, in steps of 10. We were able to determine quite fast which point sets were not quadrangulable and collected them in the first set of instances. More on this, below. The feasible ones were gathered in the second set, since these are, obviously, the challenging instances. In both cases, 20 point sets of each size are generated by sampling points uniformly on the $[0, 1] \times [0, 1]$ square. Next, each point set is assigned three random colorings, also chosen uniformly. This results in 60 instances per size in each set of instances. All these instances and their respective solutions are available at [12].

All experiments were run on a computer featuring an Intel Xeon Silver 4114 at 2.2Ghz CPU with 10 cores, and 32GB of RAM, running Ubuntu 16.04. Models and algorithms were implemented in C++ v.11 and compiled with gcc 5.5. Geometric algorithms and data structures were implemented using CGAL 5.2 [14], and Gmpq for exact number

[3]The term *edge* is used in this text to refer to line segments between points, in a geometric context, and as arcs between vertices, in the context of a graph.

representation. The ILP models and relaxations were coded using CPLEX 12.10 with the default parameters. A time limit of 30 minutes was set for each CPLEX run.

Throughout this section, data will be presented mostly as standard boxplots[4]. Averages will be expressed in the form avg $\pm$ std, where avg is the average and std is the standard deviation. All data used to generate the plots are accessible at [12].

In the first experiment, we compared Model (4.1), which we call NATURAL Model, and Model (4.2), referred to as SET-PARTITION Model.

Firstly, deciding that a point set with at most 120 points does not admit a quadrangulation was accomplished within 20 seconds using either model since the solver was able to prove infeasibility during preprocessing. This turned out to be an efficient way to separate our two sets of instances. As this is quite swift compared to optimizing (color) flips in a feasible solution, the remaining of this section will be restricted to the set of feasible instances.

To shed light on the ILP models being solved and on the intricacies our benchmark, Table 4.1 shows the average number of empty convex quadrangles, number of i-wedges and faces in the arrangement. For the sake of simplifying the implementation, i-wedges were considered with multiplicities (for each anchor), leading to exactly $n(n-1)$ i-wedges, which CPLEX can easily remove as duplicated rows. We confirmed that restricting our model to the i-wedges reduced the number of constraints drastically.

Table 4.1: Analysis of the instances in the quadrangulable benchmark, averaged for each instance size, where '# quads' indicate the number of empty convex quadrangles, '# i-wedges' the number of i-wedges and '# faces' the number of faces of the arrangement.

	40	50	60	70	80	90	100	110	120
# quads	3455 ± 323	5659 ± 474	8671 ± 731	12329 ± 1023	16790 ± 980	21546 ± 640	27493 ± 1303	33675 ± 1127	41359 ± 1284
# i-wedges	1560	2450	3540	4830	6320	8010	9900	11990	14280
# faces	64381 ± 1809	161467 ± 5184	338360 ± 11217	636033 ± 13104	1097777 ± 25193	1760375 ± 41057	2717477 ± 48764	4006901 ± 58169	5726520 ± 86319

We say that the optimal value of a relaxation is *tight* if its ceiling (in the case of a minimization problem) is equal to the optimal integral value. Table 4.2 shows how many instances each model was capable to solve per size within our specified time limit. No number is shown when at least 30% of the instances failed to be solved. The last two rows show for how many instances the relaxation produced a tight bound. Figures 4.2 and 4.3 show, for the instances that *were* solved, the total solving time, grouped by instance size for the NATURAL and SET-PARTITION models, respectively. We can see that the SET-PARTITION model was much faster. While the NATURAL model began to founder for instances of just 70 points, the SET-PARTITION model only failed to solve instances of 110 or more points. As discussed in Section 4.3, this difference in performance is mostly attributed to the stronger relaxation of the SET-PARTITION Model.

If we denote the value of an optimal solution for an instance by OPT and the value of an optimal solution of the linear relaxation of a given model for the same instance by RLX, the *relative duality gap* can be defined as $\frac{\text{OPT}-\text{RLX}}{\text{RLX}}$. The relative gap for the NATURAL and SET-PARTITION models in shown in Figures 4.4 and 4.5, respectively. As can be seen, while the relative duality gap of the SET-PARTITION model shows very little variation as the size of instances increases, staying within 0.2, the relative duality gap

[4]See: `https://en.wikipedia.org/wiki/Box_plot`.

Table 4.2: Number of solved instances and of tight relaxations for the NATURAL and SET-PARTITION Model per size.

	40	50	60	70	80	90	100	110	120
# solved NATURAL	60	60	60	57	39	-	-	-	-
# solved SET-PARTITION	60	60	60	60	60	60	60	51	32
# tight NATURAL	32	14	9	9	3	-	-	-	-
# tight SET-PARTITION	54	58	47	52	44	41	35	24	26

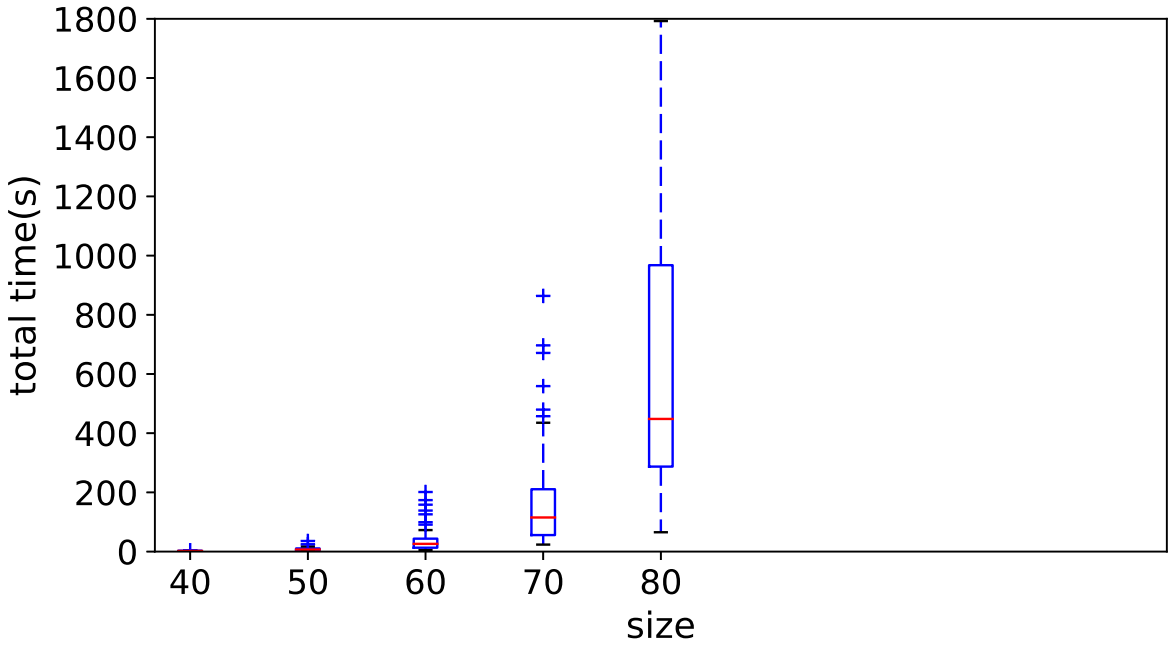


Figure 4.2: Solving time per instance size for the NATURAL Model.

of the NATURAL model surpasses 0.4 for much smaller instances. The number of tight relaxations shown in Table 4.2 corroborates with the observation that the SET-PARTITION model provides a much stronger relaxation.

The quality of the relaxation is a key element when solving ILP models. Poor relaxations result in a larger number of nodes being explored in the search tree and more difficulty in finding good integral solutions.

The *support* of a solution of a linear program is the set of variables assigned positive values. Next, we discuss the support of the optimal solutions for both the NATURAL and the SET-PARTITION model.

Tables 4.3 and 4.4 show the average distribution of (projected) edge values in the optimal solutions found for the NATURAL and the SET-PARTITION models, as well as the size of the corresponding support. For the purpose of deriving good heuristics, solutions of the relaxation must have a large number of variables with high values (in the $[0, 1]$ range), as they are more likely to appear in an integral solution. In this case, while most of the edge variables of the NATURAL model have value less than 0.1, the projected edge value of the SET-PARTITION model presented a much better distribution and are much more informative. As a consequence, we confine the analysis of our heuristic to the relaxation

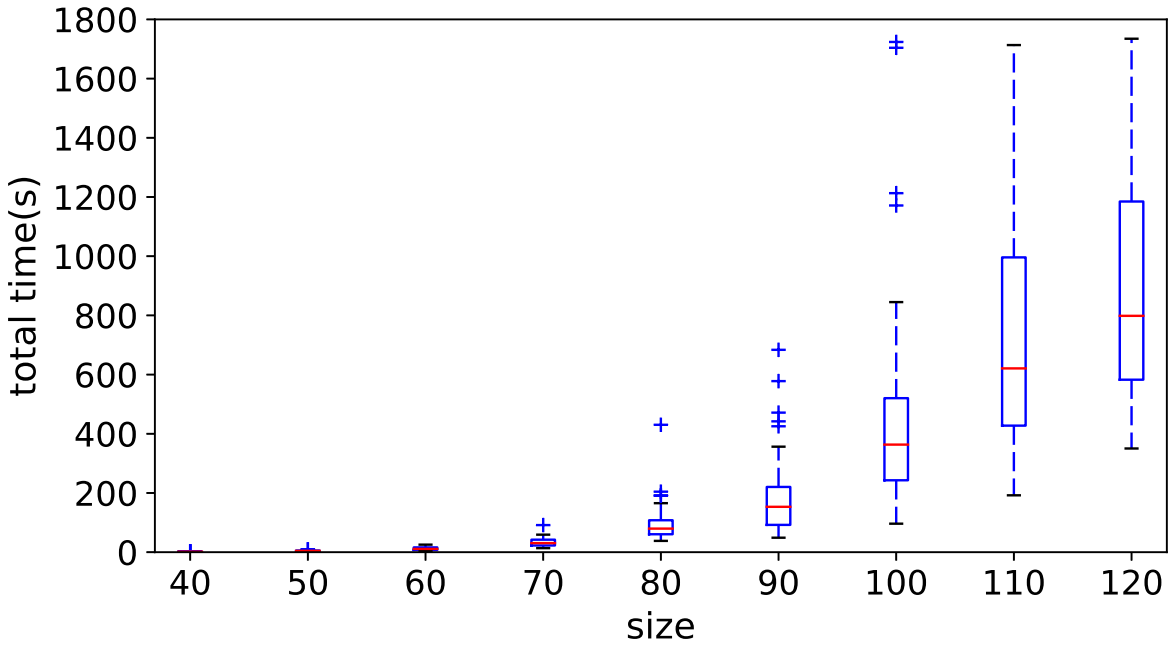


Figure 4.3: Solving time per instance size for the SET-PARTITION Model.

of the SET-PARTITION model.

Even though the quality of the relaxation of the SET-PARTITION model is significantly higher, it suffers from one of the major drawback of set-partition models: slow convergence. This is exacerbated as the model grows with instance size.

Total solving time for the relaxation of the SET-PARTITION Model is shown in Figure 4.6. Comparing the solving time of the relaxation with the median solving time of the exact model, we can see that solving the linear relaxation is a significant bottleneck.

Lastly, we show how the proposed heuristics perform on the benchmark. As small instances were too easy to solve, the experiments with the heuristics are limited to instances with at least 70 points.

We denote our quadrangle-based heuristic *quad*, our edge-based heuristic *edge* and our heuristic based on the greedy triangulation *greedytri*. We also include the Delaunay triangulation as an alternative to the greedy triangulation, denoted by *Delaunay*, for a baseline comparison. For each heuristic that relies on triangulations, we append the suffix *-flips* to indicate an alternative when edge-flips were added.

Table 4.5 shows the average number of quadrangles selected by each heuristic. As can be seen, all heuristics use only a small fraction of the total number of quadrangles. Notably, for small instances, where the relaxation performed well, the edge-based heuristic starts with a small number of quadrangles, but it rapidly increases. As expected, the Delaunay triangulation admits a larger number of empty convex quadrangles when compared to the greedy triangulation.

Table 4.6 shows how many feasible solutions each heuristic found. The quad heuristic only found solutions where the relaxation itself had already produced an integral one. Both triangulation-based heuristics had a hard time finding quadrangulations without edge-flips. The greedytri heuristic was the only one competitive with the edge-based one.

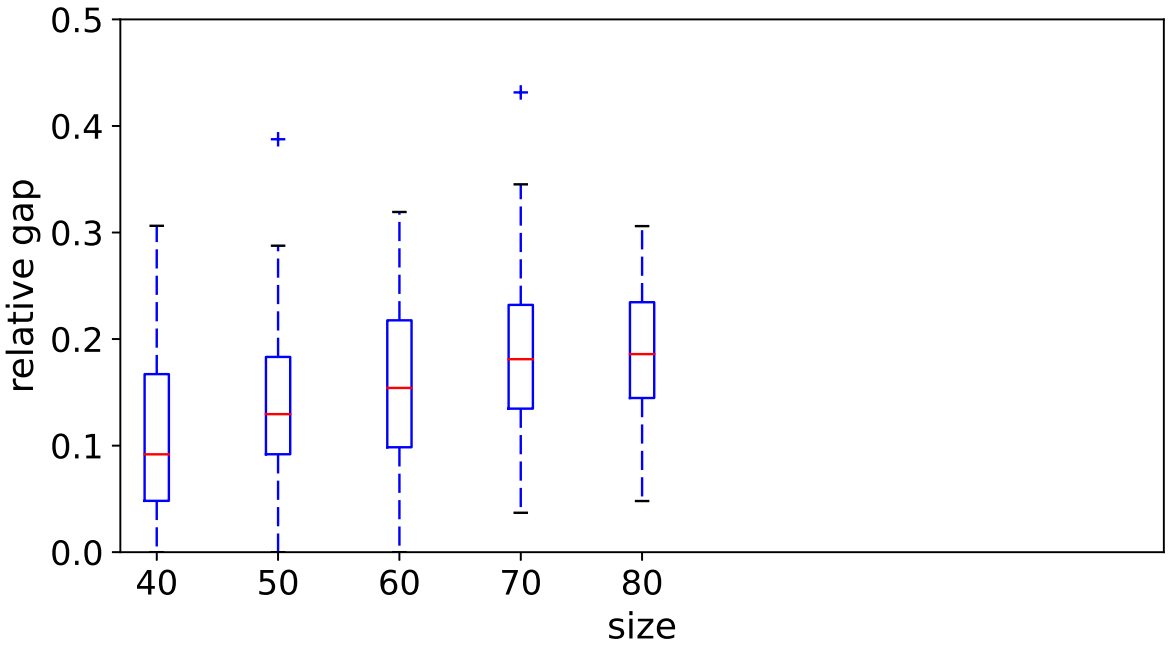


Figure 4.4: Relative duality gap per instance size for the NATURAL Model.

The solving time for each heuristic is shown in Table 4.7. We can see that the time just to solve the relaxation is more than 200 times larger than the specific solving time of each of the heuristics.

One way to take advantage of this discrepancy in solving times is to add randomization to the heuristics. For instance, one might consider adding perturbations to the weights before running the greedy triangulation. Another approach is to run the heuristic before the relaxation is solved optimally. With the exception of the edge-based one, all other heuristics performed fast enough to be executed multiple times during the solution of the relaxation of a node in a Branch-and-Bound algorithm. Alternative methods to solve the linear relaxation approximately are also worth of consideration.

Let HEU be the number of (color) flips in a solution computed by a heuristic for a given instance and recall that OPT denotes the optimal number of flips for that instance. The *relative primal gap* is given by $\frac{\text{HEU}-\text{OPT}}{\text{OPT}}$, similarly to the relative duality gap.

Table 4.8 shows the relative primal gap for all the heuristics. Even though the heuristic based on the greedy triangulation with edge-flips proved competitive, yielding non-trivial solutions for *all* instances, it is clear that the edge-based one was superior in finding solutions of the highest quality.

4.6 Concluding Remarks

In this paper, we introduced and studied a new optimization variant of the quadrangulation problem. This study focused on providing exact ILP models and heuristics derived from their linear relaxations. Both approaches provide insight into two quadrangulation decision problems from the literature.

We proposed two ILP models to solve this novel problem: one that arises naturally from a model for the more general convex partitioning problem and another that is

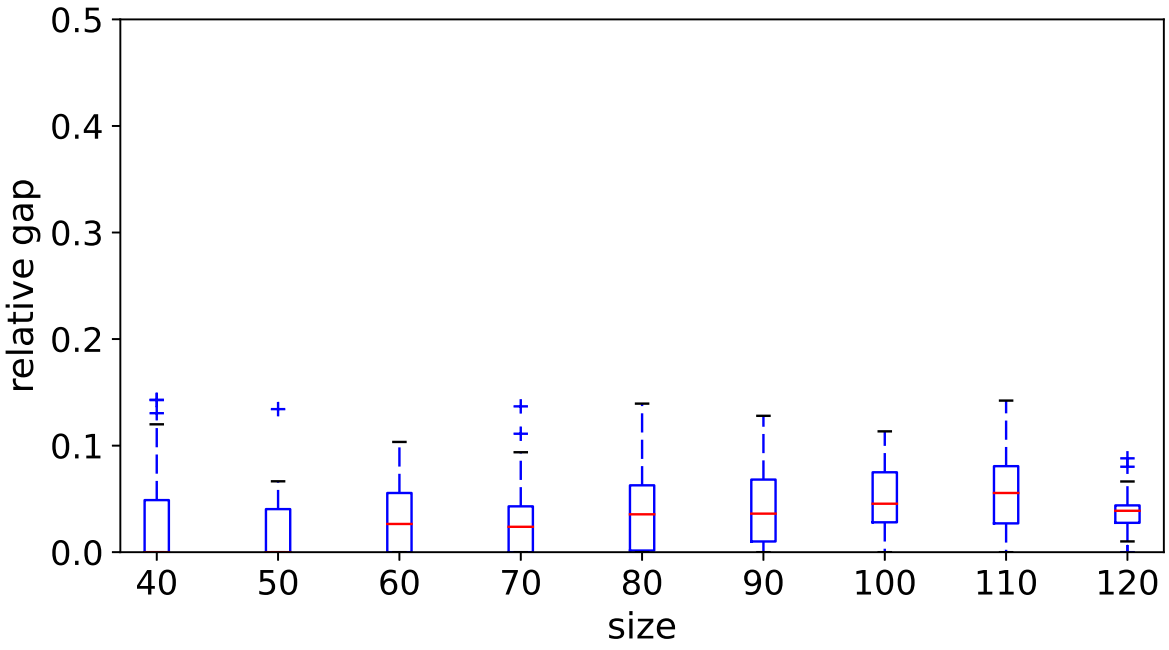


Figure 4.5: Relative duality gap per instance size for the SET-PARTITION Model.

obtained from formulating the problem as a set-partition problem.

While the natural model failed to solve instances with 70 points within 30 minutes of computing time, the set-partition model was successful in solving all instances of up to 100 points. This is due to a much stronger linear relaxation obtained from the set-partition formulation.

We also proposed several heuristics based on the linear relaxation of the set-partition model. These heuristics use a solution to the linear relaxation of the set-partition model as a guide to select only a subset of all quadrangles and, subsequently, solve much smaller instances of the set-partition model restricted to them.

Among these heuristics, our edge-based one shows superior performance in finding many good quality feasible solutions. On average, it solved most of the benchmark instances and arrived at solutions at most 3% worse than the optimal. Therefore, it is an excellent candidate to be included in a Branch-and-Bound algorithm, to be run at the end of the computation in each node.

The heuristic that is based on the greedy triangulation with edge-flips found a large number of feasible solutions for all sizes, and was notably faster than the edge-based one. As a result, it might be useful for successive executions, say, within a randomization scheme.

Overall, the quality of the relaxation-based heuristic was also quite good. Finding faster ways of solving or even approximating the linear relaxation is clearly a good venue for future research related to heuristics.

Another topic of investigation that can benefit both the heuristics and the exact algorithm is to look for preprocessing routines capable of fixing variables a priori, leading to a reduced model size.

Table 4.3: Description of the support of optimal linear relaxations found by the NATURAL Model as a function of instance size. The first ten lines indicate the percentage of edge values within the given interval. The last two lines show the number of edge variables in the support and the total number of edges.

	40	50	60	70	80
% in (0.0,0.1]	30 ± 12	40 ± 13	46 ± 12	57 ± 6	60 ± 6
% in (0.1,0.2]	16 ± 8	17 ± 7	17 ± 6	16 ± 5	16 ± 5
% in (0.2,0.3]	10 ± 5	10 ± 5	9 ± 3	7 ± 3	7 ± 2
% in (0.3,0.4]	7 ± 4	6 ± 3	6 ± 2	5 ± 2	4 ± 1
% in (0.4,0.5]	5 ± 4	4 ± 2	4 ± 3	3 ± 1	3 ± 1
% in (0.5,0.6]	3 ± 2	3 ± 2	3 ± 1	2 ± 1	2 ± 1
% in (0.6,0.7]	3 ± 2	3 ± 1	2 ± 1	2 ± 1	2 ± 1
% in (0.7,0.8]	2 ± 2	2 ± 2	2 ± 1	2 ± 1	1 ± 1
% in (0.8,0.9]	2 ± 2	2 ± 2	2 ± 1	2 ± 1	1 ± 1
% in (0.9,1.0]	21 ± 27	13 ± 19	9 ± 13	5 ± 3	4 ± 2
Support size	215 ± 55	342 ± 74	482 ± 96	699 ± 78	913 ± 107
# edges	780	1225	1770	2415	3160

Table 4.4: Description of the support of optimal linear relaxations found by the SET-PARTITION Model as a function of instance size. The first ten lines indicate the percentage of projected edge values within the given interval. The last two lines show the number of projected edges variables in the support and the total number of edges.

	40	50	60	70	80	90	100	110	120
% in (0.0,0.1]	1 ± 3	3 ± 7	4 ± 9	8 ± 12	14 ± 15	14 ± 15	23 ± 17	27 ± 16	32 ± 15
% in (0.1,0.2]	3 ± 8	7 ± 12	7 ± 10	9 ± 12	10 ± 9	13 ± 12	16 ± 9	15 ± 9	17 ± 7
% in (0.2,0.3]	4 ± 10	7 ± 12	6 ± 10	5 ± 7	9 ± 9	9 ± 9	9 ± 6	11 ± 7	11 ± 5
% in (0.3,0.4]	4 ± 10	7 ± 12	9 ± 15	7 ± 10	9 ± 12	8 ± 10	7 ± 6	10 ± 9	7 ± 3
% in (0.4,0.5]	17 ± 24	8 ± 14	13 ± 19	11 ± 15	11 ± 14	9 ± 12	10 ± 11	9 ± 11	7 ± 9
% in (0.5,0.6]	1 ± 4	2 ± 4	2 ± 4	3 ± 4	3 ± 4	3 ± 3	4 ± 3	4 ± 2	3 ± 2
% in (0.6,0.7]	2 ± 6	4 ± 8	4 ± 7	3 ± 6	4 ± 6	4 ± 5	4 ± 3	4 ± 4	4 ± 2
% in (0.7,0.8]	2 ± 4	3 ± 6	2 ± 3	2 ± 3	2 ± 3	3 ± 4	3 ± 3	3 ± 2	4 ± 3
% in (0.8,0.9]	0 ± 1	1 ± 1	1 ± 2	2 ± 3	1 ± 2	2 ± 3	3 ± 3	3 ± 2	3 ± 2
% in (0.9,1.0]	67 ± 35	57 ± 37	52 ± 37	49 ± 36	36 ± 34	35 ± 35	22 ± 22	16 ± 18	13 ± 11
Support size	96 ± 31	142 ± 54	185 ± 74	234 ± 101	330 ± 138	380 ± 161	513 ± 182	628 ± 197	757 ± 218
# edges	780	1225	1770	2415	3160	4005	4950	5995	7140

Table 4.5: Number of quadrangles in each heuristic, and the number of quadrangles in an unrestricted model per instance size.

	70	80	90	100	110	120
Delaunay	137 ± 4	158 ± 5	182 ± 5	202 ± 6	227 ± 7	247 ± 6
Delaunay-flips	538 ± 19	628 ± 16	719 ± 15	815 ± 23	919 ± 21	1000 ± 23
quad	163 ± 106	252 ± 148	293 ± 173	423 ± 203	499 ± 208	637 ± 243
greedytri	113 ± 10	135 ± 12	154 ± 14	177 ± 13	197 ± 12	217 ± 12
greedytri-flips	376 ± 40	448 ± 48	519 ± 52	601 ± 52	676 ± 50	743 ± 53
edges	293 ± 288	502 ± 419	597 ± 514	951 ± 647	1122 ± 687	1565 ± 889
unrestricted	12329 ± 1023	16790 ± 980	21546 ± 640	27493 ± 1303	33675 ± 1127	41359 ± 1284

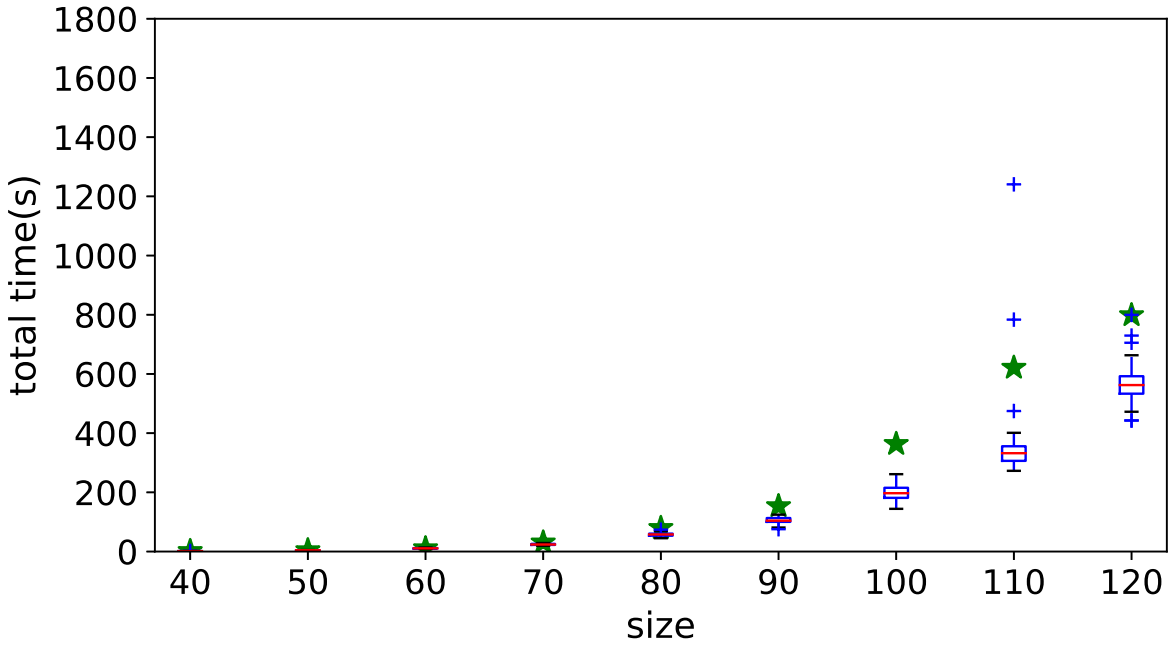


Figure 4.6: Solving time per instance size for the relaxation of the SET-PARTITION Model. Green stars indicate the median solving time for the exact model.

Table 4.6: Number of feasible solutions found by each heuristic (out of 60).

	70	80	90	100	110	120
Delaunay	0	0	0	0	0	0
Delaunay-flips	3	3	0	0	2	0
quad	17	10	10	1	2	1
greedytri	19	10	11	1	2	0
greedytri-flips	51	44	51	48	42	37
edges	**52**	**55**	**52**	**53**	**46**	**48**

Table 4.7: Solving time (in seconds) for each heuristic by instance size. For the ones based on the relaxation, (+) indicates that the average time spend solving the relaxation is shown separately.

	70	80	90	100	110	120
Delaunay	0.07 ± 0.00	0.09 ± 0.00	0.10 ± 0.00	0.12 ± 0.00	0.14 ± 0.01	0.16 ± 0.01
Delaunay-flips	0.11 ± 0.04	0.13 ± 0.04	0.15 ± 0.01	0.18 ± 0.01	0.22 ± 0.04	0.25 ± 0.01
quad (+)	0.08 ± 0.04	0.13 ± 0.05	0.15 ± 0.06	0.22 ± 0.08	0.27 ± 0.09	0.37 ± 0.14
greedytri (+)	0.08 ± 0.01	0.09 ± 0.01	0.10 ± 0.01	0.12 ± 0.01	0.14 ± 0.01	0.17 ± 0.01
greedytri-flips (+)	0.17 ± 0.04	0.20 ± 0.05	0.25 ± 0.05	0.31 ± 0.08	0.35 ± 0.10	0.39 ± 0.11
edges (+)	0.25 ± 0.35	0.51 ± 0.61	0.63 ± 0.78	1.31 ± 1.62	1.46 ± 1.52	3.12 ± 3.80
(+) relax time	24.02 ± 2.36	57.35 ± 5.61	105.90 ± 9.53	197.42 ± 24.73	363.48 ± 140.25	610.19 ± 128.95

Table 4.8: Relative primal gap for each heuristic per instance size. Heuristics that did not find solutions when the relaxation was not integral are indicated by *.

	70	80	90	100	110	120
Delaunay	-	-	-	-	-	-
Delaunay-flips	0.74 ± 0.17	0.57 ± 0.10	-	-	-	-
quad	*	*	*	*	*	*
greedytri	*	*	0.03 ± 0.11	*	*	-
greedytri-flips	0.12 ± 0.15	0.21 ± 0.23	0.22 ± 0.24	0.26 ± 0.17	0.33 ± 0.21	0.30 ± 0.20
edges	0.02 ± 0.06	0.02 ± 0.04	0.01 ± 0.03	0.02 ± 0.07	0.03 ± 0.07	0.01 ± 0.02

Bibliography

[1] A. Biniaz, A. Maheshwari, and M. H. M. Smid. Compatible 4-holes in point sets. In S. Durocher and S. Kamali, editors, *Proceedings of the 30th Canadian Conference on Computational Geometry, CCCG, August 8-10, 2018, University of Manitoba, Winnipeg, Canada*, pages 346–352, 2018.

[2] P. Bose, S. Ramaswami, G. T. Toussaint, and A. Turki. Experimental results on quadrangulations of sets of fixed points. *Computer Aided Geometric Design*, 19(7):533–552, 2002.

[3] P. Bose and G. T. Toussaint. No quadrangulation is extremely odd. In J. Staples, P. Eades, N. Katoh, and A. Moffat, editors, *Proceedings of the 6th International Symposium on Algorithms and Computation, ISAAC, Cairns, Australia, December 4-6, 1995*, volume 1004 of *Lecture Notes in Computer Science*, pages 372–381. Springer, 1995.

[4] W. F. de la Vega and C. Kenyon-Mathieu. Linear programming relaxations of Maxcut. In N. Bansal, K. Pruhs, and C. Stein, editors, *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA, New Orleans, USA, January 7-9, 2007*, pages 53–61. SIAM, 2007.

[5] T. Fevens, H. Meijer, and D. Rappaport. Minimum convex partition of a constrained point set. *Discrete Applied Mathematics*, 109(1-2):95–107, 2001.

[6] V. M. Heredia and J. Urrutia. On convex quadrangulations of point sets on the plane. In J. A. et al., editor, *Discrete Geometry, Combinatorics and Graph Theory, 7th China-Japan Conference, Tianjin, China, Nov 18-20, 2005*, volume 4381 of *Lecture Notes in Computer Science*, pages 38–46. Springer, 2005.

[7] K. L. Hoffman and M. Padberg. Solving airline crew scheduling problems by branch-and-cut. *Management Science*, 39(6):657–682, June 1993.

[8] A. Pilz and C. Seara. Convex quadrangulations of bichromatic point sets. *International Journal on Computational Geometry and Applications*, 29(4):289–299, 2019.

[9] S. Ramaswami, P. A. Ramos, and G. T. Toussaint. Converting triangulations to quadrangulations. *Computational Geometry*, 9(4):257–276, 1998.

[10] S. Ramaswami, M. Siqueira, T. A. Sundaram, J. H. Gallier, and J. C. Gee. Constrained quadrilateral meshes of bounded size. *International Journal on Computational Geometry and Applications*, 15(1):55–98, 2005.

[11] A. Sapucaia, P. J. de Rezende, and C. C. de Souza. Solving the minimum convex partition of point sets with integer programming. *Computational Geometry*, 99:101794, 2021.

[12] A. Sapucaia, C. C. de Souza, and P. J. de Rezende. Convex bichromatic quadrangulation of point sets: Benchmark instances. www.ic.unicamp.br/∼cid/Problem-instances/BichromaticPartition, 2021.

[13] T. Schiffer, F. Aurenhammer, and M. Demuth. Computing convex quadrangulations. *Discrete and Applied Mathematics*, 160(4-5):648–656, 2012.

[14] The CGAL Project. *CGAL User and Reference Manual*. CGAL Editorial Board, 5.2 edition, 2020.

Chapter 5

Solving the Coarseness Problem by ILP Using Column Generation

In this work, we proposed an ILP model for the Coarseness Problem. We then derived an iterative preprocessing algorithm that combined the relaxation of this model, implemented using column generation, and an heuristic from the literature. The resulting algorithm was able to reduce the size of the model significantly, and even solve to proven optimality a large portion of our benchmark.

This chapter corresponds to the paper [18] presented at the *21st International Conference on Computational Science and its Applications (ICCSA 2021)*. It was co-authored with Andre A. Cire[1], Pedro J. de Rezende[2] and Cid C. de Souza[2].

Abstract A core problem in machine learning is the classification of points in high dimensions. In an attempt to minimize the resources required and facilitate data visualization, a recent effort has been made to project the points non-linearly onto the plane and apply classification methods. The quality of any such projection can be measured by evaluating how the resulting points of different classes are set apart. In this paper, we study integer linear programming (ILP) models to determine the coarseness of sets of bicolored points in the plane, which measures how well-separated these two classes are. The complexity of computing the coarseness of a point set is unknown, but conjectured to be NP-hard. We present a base ILP model for the problem with an exponential number of variables and constraints, followed by a series of improvements in the quality of its relaxation and close with an efficient column generation implementation. These modifications allow us to solve instances with three times as many points as the base model, within the same time and memory limits. By combining of our preprocessing techniques and a heuristic from the literature, we are even able to solve to proved optimality many instances of our benchmark in polynomial time. A comprehensive experimental study is presented to evaluate the impact of each improvement.

[1]Department of Management, University of Toronto Scarborough, Canada

[2]Institute of Computing, University of Campinas, Campinas, Brazil. Supported by CNPq under grants #313329/2020-6, #309627/2017-6, #304727/2014-8; and Fapesp under grants#2020/09691-0, #2018/26434-0, #2018/14883-5, #2014/12236-1.

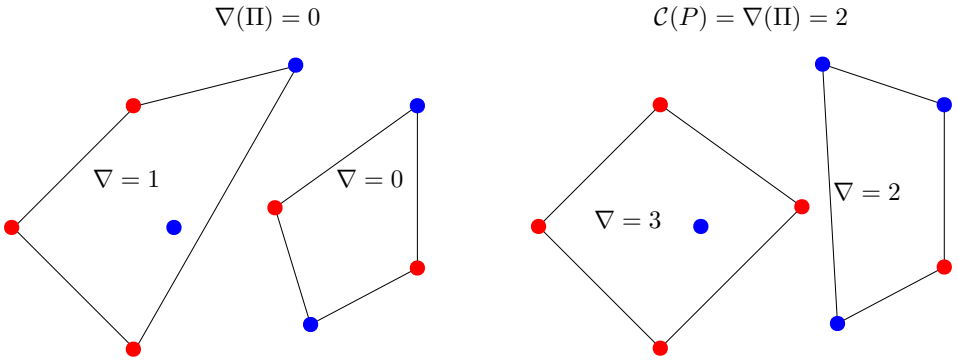
5.1 Introduction

In this paper, we study integer linear programming (ILP) models to determine the coarseness of sets of bicolored points in the plane. Given a bipartite planar point set $P = R \cup B$, with $R \cap B = \emptyset$, we call R (red points) and B (blue points) *classes of points* and say that P is a *bicolored set*. Intuitively, the *coarseness* of P is a measure of how blended the two classes of points of P are.

A core problem in machine learning is the classification of points in high dimensions, where two of the main issues are the necessary resources, time or memory, and the difficulty in visually representing the data. Non-linear projection techniques, such as t-SNE [18], are capable of projecting points from a high dimension onto the plane while attempting to preserve neighborhood relationships. These projections allow for the development of algorithms that first map the points to the plane and then work on classifying and presenting them. Successful applications of this approach can be found in [4], [12] and [3].

In this context, non-linear projections can be evaluated by how well separated the classes of points are in the plane after projection. This evaluation is particularly useful for parameter tuning for the projection algorithms. The most popular metric for evaluating projections for classification problems is Normalized Cuts [16], which is based on graph theory. A geometric-based alternative metric is given by the *coarseness* of a point set that is discussed below.

As before, let $P = R \cup B$, with $R \cap B = \emptyset$, be a bicolored planar point set. From here on, unless stated otherwise, we assume that the points in P are in general position, i.e., no three points in P are colinear. A subset $Q \subseteq P$ is an *island* if there is a convex region C such that $Q = C \cap P$. The set of all islands of P is denoted by $\mathcal{I}(P)$. We assume that the sequence of vertices of the convex hull of island Q, denoted $\mathrm{CH}(Q)$, is always given in counter-clockwise (CCW) order with respect to (w.r.t.) a point in its interior. Also, $\mathrm{INT}(Q)$ denotes the subset of points of Q that do not lie on the boundary of $\mathrm{CH}(Q)$. As island Q can be uniquely characterized by its convex hull, we may sometimes refer to Q simply by the CCW sequence of vertices of $\mathrm{CH}(Q)$. Two islands are called *disjoint* if their convex hulls are disjoint.



(a) Suboptimal Partition (b) Optimal Partition

Figure 5.1: Point set P and two partitions of its points into disjoint islands.

We associate a *discrepancy* with each island $Q \in \mathcal{I}(P)$, given by $\nabla_Q = ||R \cap Q| - |B \cap Q||$. Moreover, the discrepancy of a partition $\Pi = \{Q_1, Q_2, \ldots Q_k\} \subseteq \mathcal{I}(P)$ of P into k

pairwise disjoint islands, denoted by ∇_Π, is given by $\nabla_\Pi = \min_{Q \in \Pi} \nabla_Q$. The *coarseness* of P, denoted by $\mathcal{C}(P)$, is the maximum discrepancy among all possible partitions of P into pairwise disjoint islands. Figure 5.1 illustrates the concepts of discrepancy and coarseness.

Problem Statement. Given a bicolored planar point set, the *Coarseness Problem* consists in determining its coarseness.

This problem was first introduced in [5]. In that paper, the authors conjectured that the problem is NP-hard and studied theoretical bounds for the coarseness of point sets. They also proposed exact polynomial-time algorithms for two special cases: when the points are in convex position and when the partition is restricted to exactly two islands. The algorithm for the later case is reviewed in Section 5.4. Also, a work leading to polynomial algorithms for two other special cases is presented in [6], where the islands are restricted to lie within strips or within boxes. We are not aware of any studies in the literature that report experimental results regarding this problem.

Our Contributions. In this paper, we present an ILP model to compute the coarseness of bicolored planar point sets. We also investigate a number of improvements to the initial model, including a column generation approach, followed by a comprehensive experimental study that shows the impact of each refinement in terms of computational efficiency. The final model is able to solve instances with three times as many points when restricted to the same computational resources. Many of those instances are solved to proved optimality in polynomial time during preprocessing due to tight upper and lowerbound estimates. The benchmark used to assess the performance of the model is made publicly available to allow for future comparisons to our results.

This paper is organized as follows. In Section 5.2, we present a base ILP model for the Coarseness Problem. In Section 5.3, we improve the base model. Section 5.4 introduces a column generation algorithm. Lastly, Section 5.5 shows the empirical results comparing the improvements previously introduced.

5.2 ILP Model for the Coarseness Problem

In this section, we present the base ILP model for the Coarseness Problem.

Given a set P of n points, let $L(P)$ denote the set of $\Theta(n^2)$ line segments $\overline{jk}$ where j and k are distinct points in P. The *complete (geometric) graph* induced by P is $G(P) = (P, E(P))$, where $E(P) = \{\{j,k\} : \overline{jk} \in L(P)\}$. In this text, we refer to a segment $\overline{jk} \in L(P)$ and the corresponding edge in $\{j,k\} \in E(P)$ interchangeably. We denote by $S^C(P)$ the set of edge pairs whose corresponding line segments share an interior point, i.e., they cross each other.

Let LB and UB represent lower and upper bounds, respectively, for the coarseness of a given bicolored point set P. Associating a positive variable u_Q to each island $Q \in \mathcal{I}(P)$, a binary variable x_e for edge edge $e \in E(P)$ and a binary variable z_d for each possible discrepancy value d, we can express the Coarseness Problem as Model (5.1).

Constraint (5.1b) ensures that only one discrepancy value is selected, while Constraints (5.1c) ascertain that if an island is selected, its discrepancy is an upper bound on the discrepancy of the solution. Constraints (5.1d) guarantee that the points in P are partitioned into islands. Constraints (5.1e) enforce that if an island is being used,

its edges are part of the solution. Constraints (5.1f), together with Constraints (5.1d), require the islands to be disjoint.

Notice that there is potentially an exponential number of islands.

From Constraints (5.1d) and (5.1e), it follows that having integer values for edge variables, (5.1h), implies that the island variables necessarily assume integer values. If edge variables are integral, their corresponding line segments define a planar subdivision. Now, assume that Q_1 and Q_2 are two different islands that share an edge and are fractional, i.e., $0 < u_{Q_1} \leq 1$ and $0 < u_{Q_2} < 1$, and $u_{Q_1} + u_{Q_1} < 1$. Since one of those islands must have and edge that is not shared with the other, at least one of the edge variables must assume a fractional value, which is a contradiction.

$$\max \sum_{LB \leq d \leq UB} dz_d \tag{5.1a}$$

$$\text{s.t.} \sum_{LB \leq d \leq UB} z_d = 1 \tag{5.1b}$$

$$u_Q \leq \sum_{LB \leq d \leq \nabla_Q} z_d \qquad \forall Q \in \mathcal{I}(P) \tag{5.1c}$$

$$\sum_{Q \in \mathcal{I}: p \in Q} u_Q = 1 \qquad \forall p \in P \tag{5.1d}$$

$$\sum_{Q \in \mathcal{I}: e \in \mathrm{CH}(Q)} u_Q = x_e \qquad \forall e \in E(P) \tag{5.1e}$$

$$x_e + x_f \leq 1 \qquad \forall \{e, f\} \in S^C(P) \tag{5.1f}$$

$$u_Q \in \mathbb{R}^+ \qquad \forall Q \in \mathcal{I}(P) \tag{5.1g}$$

$$x_e \in \{0, 1\} \qquad \forall e \in E(P) \tag{5.1h}$$

$$z_d \in \{0, 1\} \qquad \forall d \in \{LB, LB+1, \ldots, UB\} \tag{5.1i}$$

Prior to investigating the geometrical structure of P, we can set $LB = 1$ and $UB = \max_{Q \in \mathcal{I}(P)} \nabla_Q$ and solve the model by standard branch-and-bound (BnB) techniques, which rely on solving its linear relaxation obtained by dropping the integrality constraints on the variables. However, the relaxation is very weak, leading to a dual bound very close to UB. Moreover, another feature of this model that makes it hard to solve in practice is the exponential number of variables and constraints.

To overcome some of the deficiencies noted above, in the next section, we improve the quality of the relaxation by finding, in polynomial time, tighter values for LB and UB, and by adding a new class of constraints. Besides, in a subsequent section, a column generation approach to address the exponential size of the model is presented.

5.3 Improving the Base Model

In this section, we propose methods to find better values for the bounds, LB and UB, for the coarseness, as well as new constraints to strengthen Model (5.1).

As discussed in the previous section, the relaxation of our Model leads to a very weak dual bound. For many instances, this poor bound is attained by assigning to z_{UB} a value close to 1 and setting $z_{LB} + z_{UB} = 1$. Notice that the value of z_{LB} is an upper bound on

all island variables due to Constraints (5.1c).

To illustrate this point, let us make reference to a benchmark of 30 randomly generated planar bicolored point sets of sizes $\{10, 15, \ldots, 60\}$ that we built for testing purposes. For each point, both x and y coordinates are randomly chosen in the interval $[0, 1]$ using uniform distribution and the colors are also assigned at random, with equal probability. This benchmark is available at [14]. Firstly, we observed that instances of size 20 are the largest that could be solved within one hour on a fairly standard computer (see Section 5.5) using the base model. Figure 5.2 shows the optimal value, the relaxation dual bound and the upper bound $UB = \max_{Q \in \mathcal{I}(P)} \nabla_Q$ for all thirty instances of size 20. For those instances, the difference between UB and the relaxation value, given as avg±stddev, is 0.017 ± 0.004.

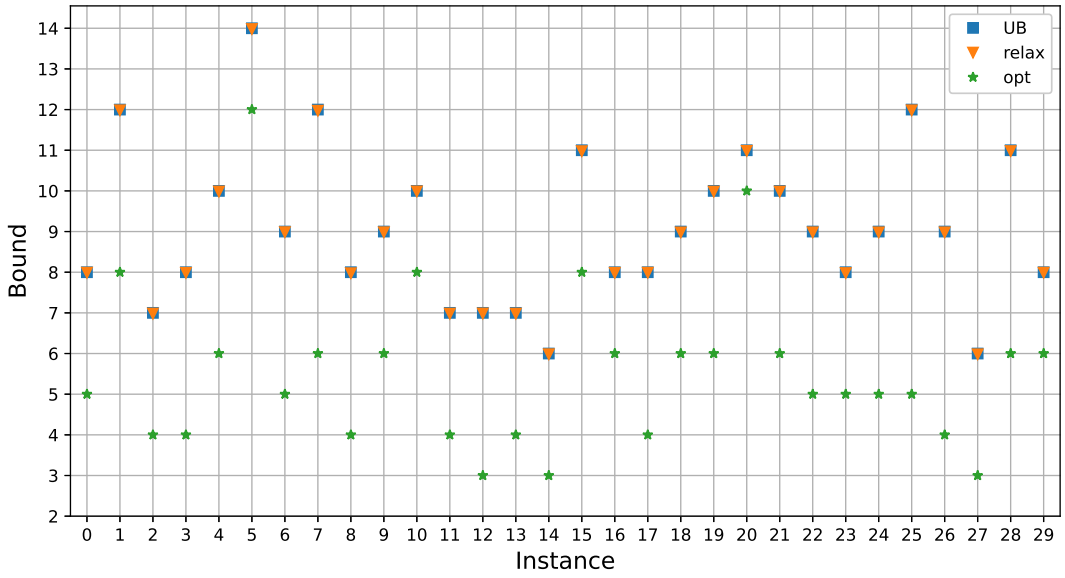


Figure 5.2: Coarseness, UB and value of the relaxation given by the base model for instances with 20 points.

Based on these observations, we can infer that the quality of the relaxation could be improved by tightening the LB and UB bounds.

5.3.1 Lower Bound (LB)

By finding a tighter value for LB, we not only get a stronger bound, but can also eliminate from the model any island whose discrepancy is smaller than LB, effectively increasing solving speed and reducing memory requirements.

Clearly, the discrepancy of any partition of P into islands, i.e., a primal solution for Model (5.1), can be used to provide a lower bound. For example, a trivial solution would simply be to regard P as comprised of a single island.

Now, when the two classes of points that comprise P are linearly separable, there exists a partition that splits the points into two monochromatic islands, which obviously is an optimal solution to the coarseness problem for P. As an educated guess, corroborated

by the aforementioned extreme case, we suspect that partitions into at most two islands might lead to good quality solutions.

As discussed in the literature review (see the latter part of Section 5.1), there are results regarding bounds for the coarseness of a point set that may serve our purpose. In particular, we decided to use a polynomial time algorithm for the special case where the partitions are restricted to two islands, as it is efficient and straightforward to be implemented. That algorithm consists in finding a line that divides the plane into two islands with the largest minimum discrepancy. In spite of the $O(n^2)$-time algorithm from [5], for solving this special case, we opted for a trivial $O(n^3)$-time procedure since, for the instances at hand, $n = |P|$ is quite small. This algorithm inspects the partitions obtained by using the lines supported by the $\Theta(n^2)$ segments in $L(P)$, corresponding to edges in $E(P)$ and returns the best value found.

5.3.2 Upper Bound (UB)

Denote by z^*_{UB} the optimal solution to the relaxation of Model (5.1) for an instance P using LB as described previously and an upper bound UB. Note that since z^*_{UB} is an upper bound on the coarseness of P, we can take $UB = \lfloor z^*_{UB} \rfloor$ as a new upper bound, since coarseness is an integral number. This leads to an iterative procedure that successively finds decreasing upper bounds UB by solving the linear relaxation until this process converges, i.e., $UB = z^*_{UB}$. As the coarseness of P is bounded by $|P| = n$, there are a linear number of iterations which consist on solving the relaxation of Model (5.1).

In the next section, we show how to modify Model (5.1) so its relaxation can be solved in polynomial time through column generation. This will allow the iterative procedure we just discussed to be performed in polynomial time as well.

5.3.3 Least Discrepancy Constraints

We now improve Model (5.1) by observing that any optimal partition of discrepancy d must include at least one island of discrepancy d. This requirement can be expressed by the following linear constraints

$$\sum_{Q:\nabla_{Q=d}} u_Q \geq z_d \qquad \forall d \in \{LB, LB+1, \ldots, UB\}, \tag{5.2}$$

which complements Constraints (5.1c).

Adding Constraints (5.2) prevents the solver from setting the value of z_{UB} close to 1 without actually including an island of discrepancy UB. Indeed, we observed that Constraints (5.2) improve the quality of the relaxation, as discussed in Section 5.5.

5.4 A Column Generation Algorithm

The number of variables and constraints in Model (5.1) makes it hard to solve instances in practice due not only to the high memory requirements, but also to the time spent enumerating them and building the model. In this section, we show how to use column

generation to solve the linear relaxation of this model in polynomial time. For this, we need to improve a transient weakened version of the original model.

To create the set of all islands, we use an iterative procedure that generates islands by the size of their convex hull. Initially, we generate all single points, segments and islands whose convex hull is a triangle. Then, new islands are generated by adding a new point to the islands of the preceding convex hull size in such a way that the resulting island has all vertices from the previous island plus the new point as vertices. The algorithm stops when no new island can be expanded in this fashion.

Instead of enumerating all variables, we can solve the relaxation of Model (5.1) by starting with a small subset of variables and generating new ones when necessary. This approach, known as *Column Generation* [2], consists of iterating between two problems: the Restricted Master Problem (RMP) and the Pricing problem. The RMP solves the original model restricted to a subset of variables. On the other hand, given an optimal dual solution to the RMP, the Pricing problem seeks variables with positive (in the case of maximization problems) reduced cost that, when added to the RMP, will potentially increase the value of the objective function or, alternatively, certifies that no such variables exist, in which case the procedure stops. The solution of the last RMP is optimal for the relaxation of the complete model and, if the pricing algorithm runs in polynomial time, then the relaxation also is solved in polynomial time [10].

By associating dual variables $\alpha_Q \geq 0$ to Constraints (5.1c), β_p to Constraints (5.1d), γ_e to Constraints (5.1e) and $\delta_d \geq 0$ to Constraints (5.2), given a dual solution $(\overline{\alpha}, \overline{\beta}, \overline{\gamma}, \overline{\delta})$ we can write the pricing problem, obtained from the dual problem, as:

$$\overline{c} = \max_{Q \in \mathcal{I}} \left\{ -\overline{\alpha}_Q - \sum_{p \in Q} \overline{\beta}_p - \sum_{e \in \mathrm{CH}(Q)} \overline{\gamma}_e + \overline{\delta}_{\nabla_Q} \right\}. \tag{5.3}$$

If $\overline{c} > 0$, then Q has a positive reduced cost and must be added to the RMP.

Notice that before island $Q \in \mathcal{I}$ is included in the RMP, $\alpha_Q = 0$, and after it is included, its reduced cost is non-positive in any optimal solution of the RMP.

This being the case, to solve the pricing problem in polynomial time, we can use dynamic programming. A similar approach proved successful for solving the pricing problem in [13].

To express the pricing problem as a recurrence suitable for an efficient dynamic programming implementation, it suffices to weaken the model by combining discrepancy constraints (5.1c) in such a way that each dual variable is associated with a discrepancy value, similar to the dual variable associated with Constraints (5.2), instead of to individual islands. With this change, we can solve a recurrence that finds an island with the maximum reduced cost that has k as its leftmost and first point, ℓ and m as the second-to-last and last points and discrepancy d. Notice that, as is, we lack a necessary property to devise divide-and-conquer algorithms for the pricing problem (5.3), since knowing whether we have $\alpha_Q > 0$ is only possible after island Q is constructed by combining solutions to sub-problems, changing the reduced cost of Q.

This is done by adding together all Constraints (5.1c) with the same discrepancy d and noticing that, since we are partitioning n points of P into islands, we can have at most $\frac{n}{d}$

islands of discrepancy d in a solution. By doing this, we obtain the following constraints:

$$\sum_{Q:\nabla_Q=d} u_Q \leq \frac{n}{d} \sum_{LB\leq s\leq d} z_s \qquad \forall d \in \{LB, LB+1, \ldots, UB\}. \tag{5.4}$$

We can lift the new constraints by including all islands with discrepancy lower than d, leading to these constraints:

$$\sum_{Q:\nabla_Q\leq d} \nabla_Q u_Q \leq n \sum_{LB\leq s\leq d} z_s \qquad \forall d \in \{LB, LB+1, \ldots, UB\}. \tag{5.5}$$

By replacing Constraints (5.1c) with Constraints (5.5) and associating to them a dual variable η, we obtain the following final pricing problem:

$$\overline{c} = \max_{Q\in\mathcal{I}} \left\{ -\sum_{p\in Q} \overline{\beta}_p - \sum_{e\in \mathrm{CH}(Q)} \overline{\gamma}_e + \nabla_Q \sum_{s\geq \nabla_Q} \overline{\eta}_s - \overline{\delta}_{\nabla_Q} \right\}. \tag{5.6}$$

We now solve this pricing problem using dynamic programming. We first provide an $O(n^5)$ algorithm and show how to improve the time complexity to $O(n^4)$.

To this end, we extend the concept of discrepancy so that we know the exact value of the resulting discrepancy when two islands are combined. For that, we introduce the notation $\nabla^{\pm}$ as the signed discrepancy. By assigning $\nabla^{\pm}(r) = +1$ for $r \in R$ and $\nabla^{\pm}(b) = -1$ for $b \in B$, the *signed discrepancy* of island $Q \in \mathcal{I}(P)$ is given by $\nabla^{\pm}(Q) = |R \cap Q| - |B \cap Q| = \sum_{p\in Q} \nabla^{\pm}(p)$.

Let $B(k, \ell, m, d)$ denote the maximum reduced cost from among all the islands with k as the leftmost point (and first in CCW order), ℓ as the second-to-last, m as the last point, and d as the signed discrepancy (w.r.t. the dual solution $(\overline{\beta}, \overline{\gamma}, \overline{\delta}, \overline{\eta})$). We say that (k, ℓ, m) is the *last triangle* of such islands. Figure 5.3 illustrates these concepts.

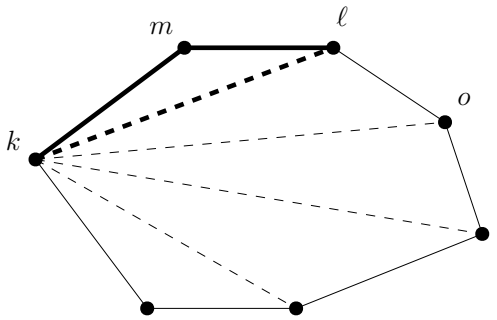


Figure 5.3: An island whose last triangle is (k, ℓ, m). Dashed lines show how it is decomposed into triangles with k as the leftmost vertex. The edges of (k, ℓ, m) are emphasized.

To simplify notation, let $D(d) = |d|(\sum_{|d|\leq s\leq UB} \overline{\eta}_s) - \overline{\delta}_{|d|}$ denote the sum of the dual variables associated with (signed) discrepancy d in (5.6) and $\Delta(k, \ell, m) = -\overline{\gamma}_{k,\ell} - \overline{\gamma}_{\ell,m} - \overline{\gamma}_{m,k} - \sum_{p\in(k,\ell,m)} \overline{\beta}_p + D(\nabla(k, \ell, m))$ denote the reduced cost of triangle (k, ℓ, m).

Given a triangle (k, ℓ, m) and a discrepancy d, we wish to determine whether there exists an island with maximum reduced cost that, once merged with triangle (k, ℓ, m),

leads to discrepancy d. We can compute the discrepancy necessary for such an island to be mergeable with triangle (k, ℓ, m), and denote it by $d'(k, \ell, m, d) = d - \nabla^{\pm}((k, \ell, m)) + \nabla^{\pm}(k) + \nabla^{\pm}(\ell)$. Note that since the points are in general position and k and ℓ are the only points shared by both the island and the triangle, the last two additive terms are necessary. To compute the resulting reduced cost of combining the triangle and an island, we need to consider that both share edge $\{k, \ell\}$, and also account for the reduced cost associated to their own discrepancy. Lastly, since a triangle can be regarded as an island, when its discrepancy is equal to the target discrepancy, island merging becomes optional.

Let P_k denote the list of points to the right of k, in increasing order of abscissae. Given a sequence of three points (k, ℓ, m) in the plane, we say that they are *colinear*, *positively oriented*, or *negatively oriented* depending on the value of the cross product $\overrightarrow{k\ell} \times \overrightarrow{\ell m}$ being 0, positive or negative. For simplicity, we say that $\mathrm{CVX}(k, \ell, m) = \text{true}$ (or simply $\mathrm{CVX}(k, \ell, m)$) if (k, ℓ, m) are positively oriented. We can combine an island whose last triangle is (k, o, ℓ) and triangle (k, ℓ, m) to form an island, as long as $\mathrm{CVX}(k, o, \ell)$, so that (k, o, ℓ) is a triangle given in CCW order, and $\mathrm{CVX}(o, \ell, m)$, to ensure convexity. Let $O_{k\ell m}$ denote the set $\{o \in P_k : \mathrm{CVX}(k, o, \ell) \wedge \mathrm{CVX}(o, \ell, m)\}$.

To simplify notation, we introduce an auxiliary recurrence $B_u(k, \ell, m)$, which computes a maximum reduced cost island that has (k, l, m) as its last triangle and discrepancy d by merging this triangle with another island. Using the aforementioned notation, these observations lead the following recurrence:

$$B_u(k, \ell, m, d) = \left\{ \begin{array}{ll} -\infty, & \text{if } O_{k\ell m} = \emptyset \\ \left.\begin{array}{c} \max\limits_{o \in O_{k\ell m}} \{B(k, o, \ell)\} + \Delta(k, \ell, m) + D(d) \\ + 2\overline{\gamma}_{k,\ell} + \overline{\beta}(k) + \overline{\beta}(\ell) \\ - D(\nabla^{\pm}(k, \ell, m)) - D(\nabla^{\pm}(d'(k, \ell, m, d) \end{array}\right] & \text{otherwise} \end{array} \right\} \tag{5.7}$$

$$B(k, \ell, m, d) = \begin{cases} \max\{B_u(k, \ell, m, d), \Delta(k, \ell, m)\} & \text{if } d = \nabla^{\pm}((k, \ell, m) \quad (5.8a) \\ B_u(k, \ell, m, d) & \text{otherwise} \quad (5.8b) \end{cases}$$

A naive implementation of this algorithm would have a time complexity of $O(n^5)$. This is the result of having $O(n^4)$ states that require finding a maximum over $O(n)$ candidates plus the sum of costs associated with a triangle for each of them.

To calculate the costs associated with a triangle in constant time, we need to calculate $\sum_{p \in (k,\ell,m)} \overline{\beta}_p$ and $\nabla^{\pm}(k, \ell, m) = \sum_{p \in (k,\ell,m)} \nabla^{\pm}(p)$ in $O(1)$. Notice that the triangles are not necessarily empty. This can be realized using a technique presented in [8] for: given a set P of n weighted points, after an $O(n^2)$-time preprocessing, compute, in $O(1)$ time, the sum of the weights of all points within a query triangle with vertices in P. The basic idea is to precompute the sum above each segment in $L(P)$ and use an inclusion-exclusion principle to calculate the sum inside any triangle, based on its edges. To determine the signed discrepancy of a triangle, we assign weights $+1$ and -1 to points in R and B, respectively, and to compute the sum of the β variables of points within a triangle we associate to each point its β value.

To reach our proposed time complexity target, we adapt an angular sweeping tech-

nique first introduced to find the Maximum Cardinality Empty Convex Polygon [1], and recently employed in [13] to solve the pricing problem of an ILP model for the Minimum Convex Partition Problem. This technique allows us to compute the maximum within Equation (5.7), which corresponds to finding a third-to-last point for the island being constructed, in amortized constant time.

While referring to Figure 5.4, notice that, for fixed k and ℓ, if we visit the possible last points m in counter-clockwise order around ℓ, the list of candidates third-to-last points o can also be visited in counter-clockwise order in such a way that each candidate is only considered once. For further details, we refer the reader to [1] and [13].

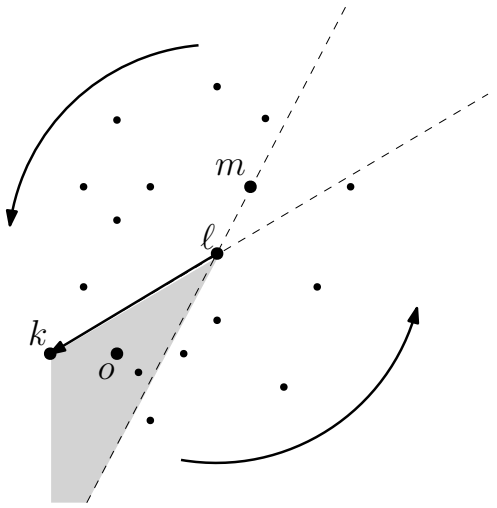


Figure 5.4: Illustration of the angular sweep technique.

Finally, we use a technique of early stop to accelerate the convergence of the RMP. Instead of halting the column generation algorithm when the maximum reduced cost $\overline{c}$ is zero, we stop when it is small enough so that the floor of the current optimal of the RMP, z^*_{RMP}, cannot change. We can calculate a dual bound for the complete model based on the maximum reduced cost as $z^* \geq z^*_{\text{RMP}} + \overline{c}\kappa$, where κ is an upper bound on the number of islands in an optimal solution. Since we know that $z^* \geq LB$, we must have $\kappa \leq \lfloor \frac{n}{LB} \rfloor$. Finding a better LB reduces significantly the tailing-off effect, which is a frequent issue for column generation, when the value of z^*_{RMP} is very close to z^* and yet many iterations are needed to achieve a reduced cost of zero [11].

Now, we can solve to integrality the model that results from replacing Constraints (5.4) with (5.5) and adding Constraints (5.2), by using Branch-and-Price (BnP) [2]. We start the model with the islands corresponding to each individual point and use the pricing as described. When solving multiple relaxation, as in the case of the iterative procedure to find UB, we start the new relaxation with all islands generated to solve the previous one. Branching is done on edge and discrepancy variables.

5.5 Experimental Results

In this section, we show the positive impact of the modifications to the basic model described in the previous sections. All experiments were run on an Intel Xeon E5-2420 at 1.9GHz and 32GB of RAM, running Ubuntu 18.04. Models and algorithms

were implemented in C++ v.11 and compiled with `gcc` 7.5. Geometric algorithms and data structures were implemented using CGAL 5.2 [17], using Gmpq for exact number representation. ILP models were implemented using SCIP 7.0.2 [9] with CPLEX 12.10 as the LP solver. In each experiment, we use the instances from the benchmark described in Section 5.3. A time limit of one hour was set for the ILP solver for each instance.

When running times are presented, we consider both the time to generate and to solve the model. Most of the data are presented in a standard boxplot, grouped by size. All data used to generate the figures is available at [14].

We begin by showing the impact of tightening the LB and UB bounds in the relaxation. In Figures 5.5 and 5.6, the modifier +heuLB indicates that LB was found by a heuristic solution, +itUB means that the iterative procedure was used to find UB and the addition of the least discrepancy constraints is denoted by the +cut. The calculation LB and UB are considered the *preprocessing phase*.

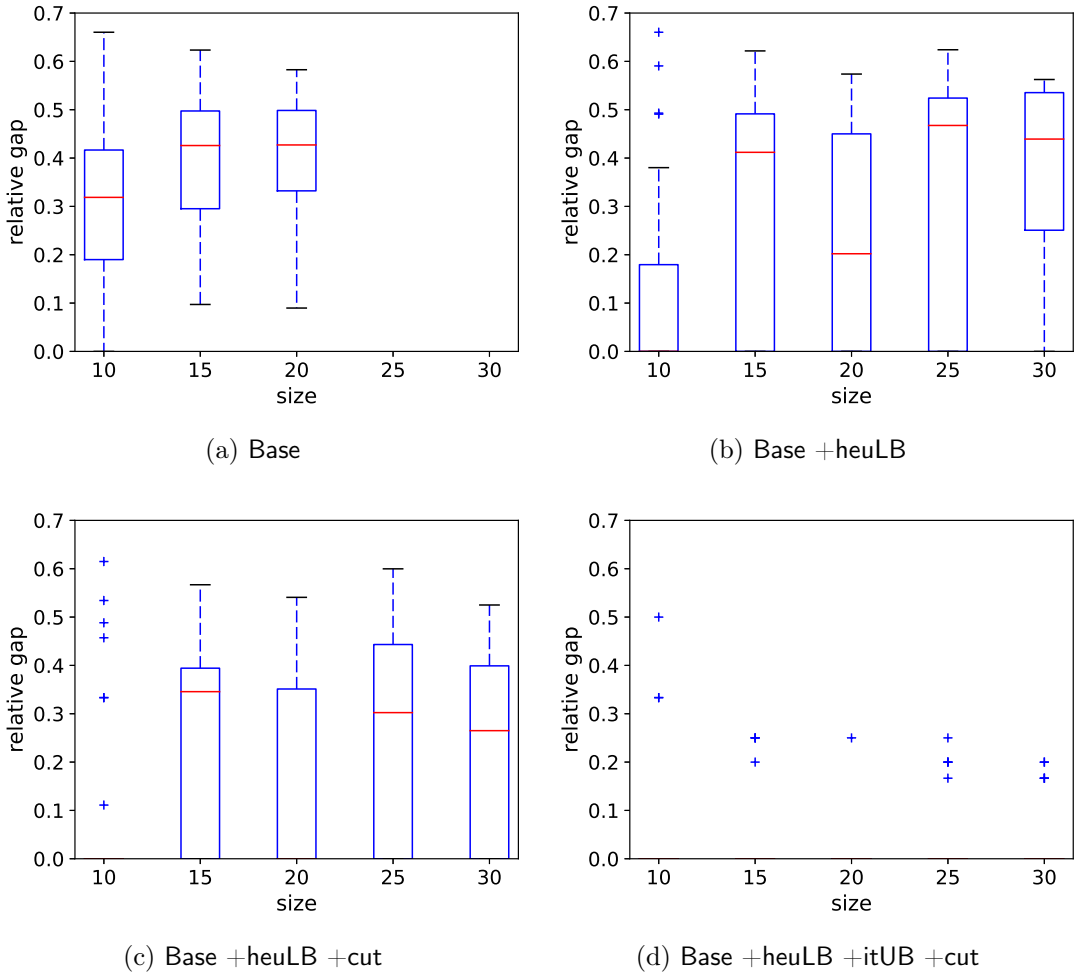
Figure 5.5 shows the relative gap, computed as $\frac{\text{RELAX}-\text{OPT}}{\text{RELAX}}$, where OPT and RELAX are the values of an optimal integer solution and optimal relaxation solution, respectively, for instances of different sizes. The Base model could not solve instances with 25 or more points within one hour, while instances with at least 35 points did not fit in 32GB of memory without column generation.

As expected, improving LB by finding the best partition into 1 or 2 islands has a positive impact in reducing the relative gap. The benefit of the least discrepancy constraints can also be noted. The iterative algorithm to find a better UB yields a tight bound for most instances.

Similarly, the total solving time for the same configurations is shown in Figure 5.6. In order not to distort the scale and hinder details in the graph, two outlier instances with 30 points are not shown for the Base +heuLB +itUB +cut graph as they took 2359 and 3980 seconds, respectively, to be solved. This experiment shows the impact in performance of the improved LB values, as that allows for the removal of a large number of variables, drastically reducing both solving time and memory consumption, which, in turn, raises the size of instances being solved from 20 to 30. Notably, the high quality bounds obtained by using the iterative algorithms to find UB come with a cost in overall time. This extra time is greatly reduced, along with the expected memory requirement, when solving the relaxations using column generation, as we will see.

Next, we show, in Figure 5.7, the solving time to find exact solutions with and without the tighter UB, while still using the least discrepancy constraints and the LB found by the heuristic. As before, so as not to distort the scale, two outlier instances are omitted in Figure 5.7b as they took 2594 and 4369 seconds, respectively, to be solved. Instances of more than 30 points could not fit in 32GB of memory. Here, for most instances, we observe the advantage of using the iterative algorithm to find UB when it comes to solving the ILP. In such cases, the stronger bound compensates the extra time during preprocessing leading to an overall performance gain.

Lastly, as our paramount result, we focus now on the solving times using the same bounds as before, but employing column generation for both the iterative procedure to find UB and for the actual ILP, as depicted in Figure 5.8a. Having thus lessened the memory limitation, larger instances could now be solved and we set aside a time limit of



(a) Base

(b) Base +heuLB

(c) Base +heuLB +cut

(d) Base +heuLB +itUB +cut

Figure 5.5: Relative gap for different configurations of the Base model.

one extra hour just for the iterative procedure to find a tight UB value.

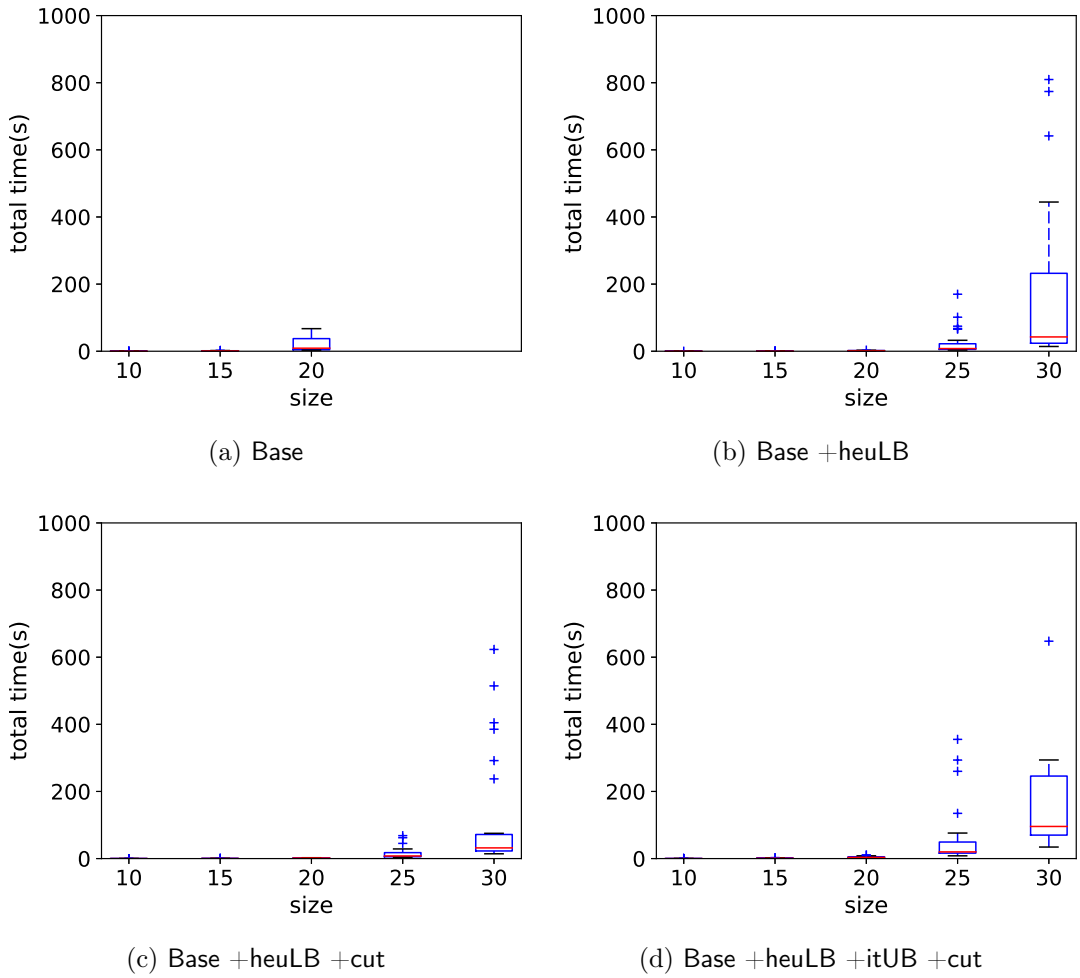
For the actual memory consumption, we refer the reader to Figure 5.8b. Notice that the most remarkable development is the great reduction in memory usage, which allowed us to solve instances with three times the number of points as before, using one tenth of the total memory limit, on average. Moreover, the use of column generation sped up the procedure to find UB. For more details, see Table 5.1.

Notably, the combination of itUB and heuLB was able to solve to proved optimality a high number of randomly generated instances in polynomial time. This table also shows that even when limited to a total of one hour for preprocessing and solving, the usage of itUB increased the number of instances solved.

Final Remarks

In conclusion, we showed that by using column generation, in conjunction with the improvements to the bounds on coarseness, we were able to solve to optimality instances of triple the size (up to 60 points) from our benchmark that the initial model could solve, with remarkably small memory requirement.

As to future work, we intend to investigate how to speed up the column generation



(a) Base

(b) Base +heuLB

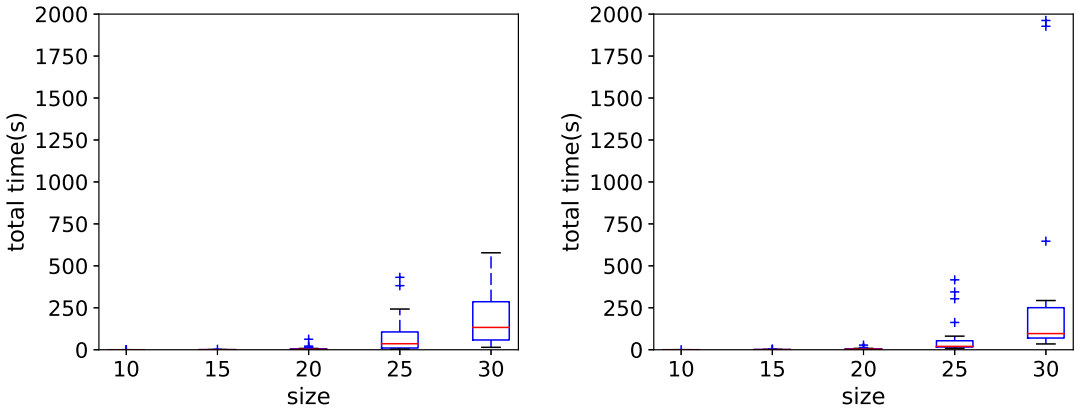
(c) Base +heuLB +cut

(d) Base +heuLB +itUB +cut

Figure 5.6: Total solving time for the relaxation of different configurations of the model.

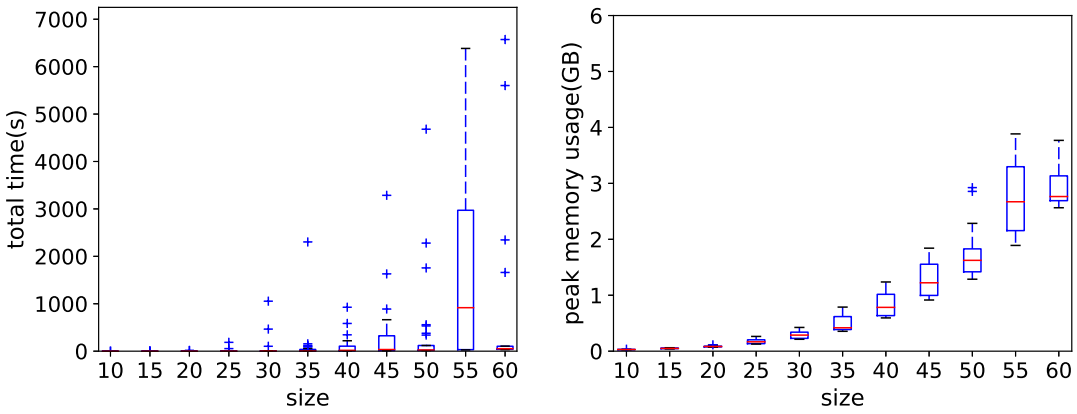
procedure since its convergence is central for solving large instances quickly. Improving the heuristic is another venue of research, which might not only reduce the solving time for the relaxation but also lead to a larger number of instances that can be solved during preprocessing due to establishing a tighter lower bound.

Despite the fact that randomly generated point sets have a high probability of being in general position, it is important to consider how our model might be adapted for instances where this property may not be guaranteed. Besides the obviously intricate approach of considering all different cases of collinearity that may appear during island construction, it would be worth considering the alternative of applying symbolic perturbation schemes such as [7] or [15], notwithstanding their well known impact in time complexity, in practice.



(a) Without itUB (b) With itUB

Figure 5.7: Total time for Base +heuLB +cut with and without itUB.



(a) Time (s) (b) Peak Memory Usage (GB)

Figure 5.8: Total solving time and peak memory usage for Base +heuLB +itUB +cut using Column Generation. Only instances that were solved to optimality are included.

Table 5.1: Number of instances solved by different methods using column generation for different instance sizes. Solved in preprocessing indicates instances where $UB = LB$.

Method ↓ | **Size** →	**10**	**15**	**20**	**25**	**30**	**35**	**40**	**45**	**50**	**55**	**60**
# solved in preprocessing	27	25	28	26	26	25	23	21	27	19	18
# solved with itUB	30	30	30	30	30	29	28	23	28	21	20
# solved in 1h with itUB	30	30	30	30	30	29	28	23	27	19	18
# solved without itUB	30	30	30	30	30	28	27	20	26	13	17

Bibliography

[1] D. Avis and D. Rappaport. Computing the largest empty convex subset of a set of points. In *Proceedings of the First Annual Symposium on Computational Geometry*, SCG'85, pages 161–167, New York, 1985. ACM.

[2] C. Barnhart, E. L. Johnson, G. L. Nemhauser, M. W. P. Savelsbergh, and P. H. Vance. Branch-and-price: Column generation for solving huge integer programs. *Operations Research*, 46(3):316–329, 1998.

[3] B. C. Benato, J. F. Gomes, A. C. Telea, and A. X. Falcão. Semi-supervised deep learning based on label propagation in a 2d embedded space. *CoRR*, abs/2008.00558, 2020.

[4] B. C. Benato, A. C. Telea, and A. X. Falcão. Semi-supervised learning with interactive label propagation guided by feature space projections. In *Proceedings of the 31st Conference on Graphics, Patterns and Images (SIBGRAPI)*, pages 392–399, October 2018.

[5] S. Bereg, J. M. Díaz-Báñez, D. Lara, P. Pérez-Lantero, C. Seara, and J. Urrutia. On the coarseness of bicolored point sets. *Computational Geometry*, 46(1):65–77, 2013.

[6] J. M. Díaz-Báñez, M. A. López, C. Ochoa, and P. Pérez-Lantero. Computing the coarseness with strips or boxes. *Discrete Applied Mathematics*, 224:80–90, 2017.

[7] H. Edelsbrunner and E. P. Mücke. Simulation of simplicity: A technique to cope with degenerate cases in geometric algorithms. *ACM Transactions on Graphs*, 9(1):66–104, 1990.

[8] D. Eppstein, M. H. Overmars, G. Rote, and G. J. Woeginger. Finding minimum area k-gons. *Discrete Computational Geometry*, 7:45–58, 1992.

[9] G. Gamrath, D. Anderson, K. Bestuzheva, W.-K. Chen, L. Eifler, M. Gasse, P. Gemander, A. Gleixner, L. Gottwald, K. Halbig, G. Hendel, C. Hojny, T. Koch, P. Le Bodic, S. J. Maher, F. Matter, M. Miltenberger, E. Mühmer, B. Müller, M. E. Pfetsch, F. Schlösser, F. Serrano, Y. Shinano, C. Tawfik, S. Vigerske, F. Wegscheider, D. Weninger, and J. Witzig. The SCIP Optimization Suite 7.0. Technical report, Optimization Online, March 2020.

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

[11] M. E. Lübbecke and J. Desrosiers. Selected topics in column generation. *Operations Research*, 53(6):1007–1023, 2005.

[12] A. Z. Peixinho, B. C. Benato, L. G. Nonato, and A. X. Falcão. Delaunay triangulation data augmentation guided by visual analytics for deep learning. In *Proceedings of the 31st SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI)*, pages 384–391, October 2018.

[13] A. Sapucaia, P. J. de Rezende, and C. C. de Souza. Solving the minimum convex partition of point sets with integer programming. *Computational Geometry*, 99:101794, 2021.

[14] A. Sapucaia, C. C. de Souza, and P. J. de Rezende. Coarseness of point sets – Benchmark instances and solutions, 2021. `http://www.ic.unicamp.br/~cid/Problem-instances/Coarseness`.

[15] R. Seidel. The nature and meaning of perturbations in geometric computing. *Discrete and Computational Geometry*, 19(1):1–17, 1998.

[16] J. Shi and J. Malik. Normalized cuts and image segmentation. In *1997 Conference on Computer Vision and Pattern Recognition (CVPR'97), June 17-19, 1997, Puerto Rico*, pages 731–737. IEEE Computer Society, 1997.

[17] The CGAL Project. *CGAL User and Reference Manual*. CGAL Editorial Board, 5.2 edition, 2020.

[18] L. van der Maaten and G. Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9(86):2579–2605, 2008.

Chapter 6

Covering Points with Unit Disks under Color Constraints

In this work, we proposed an ILP model for the Minimum 3-Colorable Discrete Unit Disk Cover Problem and solved it using Logic-Based Benders Decomposition. We provided methods to derive strong cuts leveraging the geometry of the problem, leading to a very efficient iterative implementation. We also provided theoretical insights into the performance of this method.

This chapter corresponds to the paper submitted to the journal *Discrete Applied Mathematics*. It was co-authored with Andre A. Cire[1], Pedro J. de Rezende[2], Susanna F. de Rezende[3] and Cid C. de Souza[2].

Abstract In this paper, we study a wireless network-inspired set cover problem with color constraints, called Minimum 3-Colorable Discrete Unit Disk Cover. In this problem, given a point set P and a unit disk set D, one is asked to find a minimum cover for the points of P using a 3-colorable subset of D. We describe an ILP model and show how it can be extended and implemented using Logic-based Benders Decomposition. This decomposition leads to a set-cover master problem and a 3-coloring subproblem. We show that an iterative implementation of this decomposition outperform a more traditional one based on cuts. From an theoretical point of view, we prove that the solution of the first iteration, which is a pure set cover problem, uses at most 18 times the minimum number of colors from among all covers of P that use disks of D. We also present graph-theoretical and geometric algorithms to derive stronger cuts that significantly reduce the running time. A comprehensive experimental study is presented to show the impact of our design choices on instances sampled from both uniform and population density distributions.

[1]Department of Management, University of Toronto Scarborough, Canada
[2]Institute of Computing, University of Campinas, Campinas, Brazil
[3]Department of Computer Science, Lund University, Lund, Sweden

6.1 Introduction

The design of wireless networks typically involves the decision on where to install antennas so that all customer regions are covered. There are also many different secondary goals to consider such as minimizing the number of antennas, or ensuring that the network remains operational even when some of the antennas are down. For the problem discussed in this paper, the goal is to cover a set of clients using the minimum number of antennas while avoiding destructive interference.

The most popular frequency band (2.4GHz) for WiFi is the IEEE 802.11 standard where there are 11 channels available, each one with a slightly different frequency, for the antennas to transmit and receive signals. Destructive interference between two antennas occurs when they use the same channel and are too close to each other. To maximize the spread of frequencies in order to reduce interference, it is recommended to use only three channels (1, 6 and 11), as discussed in [7].

In this work, customer regions are modeled as points in the plane while a potential position for installing an antenna is modeled by a disk representing its coverage area. Colors are then used to represent the channel frequencies to be assigned to the antennas associated with the disks. In this context, the overlap of two disks assigned to the same color characterizes destructive interference between their corresponding antennas.

The above modeling corresponds to the optimization problem defined below that we investigate in this article.

Minimum 3-Colorable Discrete Unit Disk Cover (M3CDUDC) Given a set P of n points in the plane and a set D of m disks of the same radius and fixed centers, find a minimum cardinality subset D' of D so that each point in P is covered by at least one disk in D', subject to the requirement that each disk in D' is assigned one of three colors and no two disks of D' with the same color overlap.

For an illustration of an instance of the M3CDUDC problem and its solution, see Figure 6.1. A more detailed description of the M3CDUDC is given in Section 6.2 to allow a better understanding of the mathematical formulations and algorithms we propose to solve the problem.

Literature Review The decision version of this problem, called 3-Colorable Discrete Unit Disk Cover (3CDUDC) Problem, where the goal is to decide whether there is a 3-colorable cover, is an NP-hard problem introduced in [5]. In the same work, a 2-approximation algorithm, with respect to the number of colors, that runs in $O(nm^{25})$ time, was proposed. In other words, if there is a 3-colorable cover, the algorithm is guaranteed to produce one with at most six colors, in polynomial time. A faster 2-approximation algorithm for the 3CDUDC, with time complexity $O(nm^{18})$, was proposed in [29] in the context of solving a more general problem of finding k-colorable covers, where k is given *a priori*.

Due to the impracticality of these high degree polynomial time complexities, we chose not to implement these algorithms as they do not scale to the instance sizes that we solve in this paper.

A closely related problem is the Minimum Ply Covering (MPC) Problem, where we are given an instance comprised of a disk set and a point set, and wish to cover the point

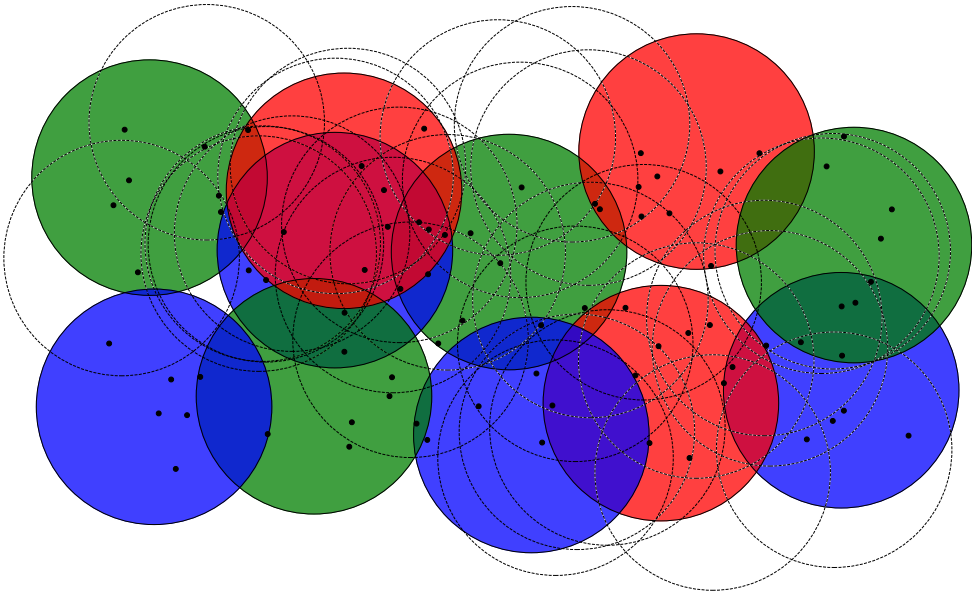


Figure 6.1: Example of an instance of the M3CDUDC with 80 points and 40 disks and an optimal solution with 11 disks. Dashed circles correspond to disks in the input that are not part of the solution.

set using some of the disks. However, the objective there is to minimize the maximum number of disks covering any single point in the plane, called the *ply* of that instance. This problem was also investigated in [5], where a 2-approximation was proposed under the assumption that the objective function is bounded by a constant.

Comparatively, while in the MPC we wish to minimize the worst interference, in the M3CDUDC we want to find a minimum cover of the points such that, when limited to the three recommended channels, no destructive interference occurs. Notably, it cannot be guaranteed that an instance for which the ply is three can be covered by a set of 3-colorable disks [5].

Similar problems, where the modelling of the antennas is done using triangles instead of disks, are studied in [2].

Next, we review the literature on wireless network inspired problems that capture some of the features of the M3CDUDC.

To the best of our knowledge, the literature on disk covering is limited to unit disks. A *Unit Disk Graph* is an intersection graph whose vertices correspond to equal-sized disks and edges encode the fact that a pair of disks overlap. A comprehensive review of properties of unit disk graphs can be found in [9]. These properties allow for strong theoretical results, leading to efficient algorithms for problems that are hard to solve for general graphs.

The problem of finding minimum discrete unit disk covers of point sets without coloring constraints, which is a geometric variation of the classic Set Cover Problem, has been well investigated. This problem is NP-hard and several approximation algorithms have been proposed, among which we highlight those with the best ratios: an 18-approximation [11] that runs in $O(mn)$ time and a $(9+\varepsilon)$ PTAS [6].

There are also geometric set cover problems without coloring constraints where either

the point set or the disk set is not finite. The variation where the input consists only of a point set and we want to find a minimum cover comprised of any unit disks in the plane was studied in [6] and [17]. The first shows a simple 4-approximation that runs in $O(n \log n)$ time while the latter presents a 7-approximation that runs in $O(n)$ time. The problem where the input consists of a disk set and a set of regions to be covered and we want to find a minimum cover comprised of disks in the input was studied in [10].

Elsewhere, the coloring problem on unit disk graphs, which is also an NP-hard problem, has also been studied and a 3-approximation algorithm was presented in [18]. An alternative and less restrictive definition of coloring, called *conflict-free coloring*, has been investigated in [14], where it is shown that six colors are always sufficient.

For discussions on applications of unit disks for wireless network modelling, the reader may refer to [3]. For a comprehensive text on frequency assignment problems, we recommend [19].

In this paper, we study a new optimization version of a discrete unit disk cover problem with coloring constraints. While our main goal is to investigate coverage by unit disks, our models and algorithms can easily be adapted to handle disks with different radii. Whenever possible, we show how our analysis can be adapted for the general case, as well.

We were not able to find empirical studies that combine both covering and coloring, which are the focus of our research. In the literature, instances for frequency assignment are usually given as graphs without geometric information[23, 24], while the geometric instances we found for related problems were restricted to either disk sets or point sets [10, 13, 15].

Our Contributions In this paper, we introduce an optimization variant of the 3-Colorable Discrete Unit Disk Cover. We investigate distinct Integer Linear Programming models enhanced with improvements that leverage the geometry of the problem. We also provide the first publicly available benchmark for instances composed of both disk sets and point sets. Using this benchmark, we show that our best model, an iterative Logic-Based Benders Decomposition approach, solved instances with thousands of points and disks to optimality in under 20 minutes on a standard desktop machine. This model compares favorably with the approximation algorithms in the literature for the decision version of the problem, which are often not practical. We also present theoretical results showing that the first iteration of our algorithm, which corresponds to a pure set cover problem, finds solutions with at most a constant factor (18 times) the minimum number of colors of any set cover.

This text is organized as follows: Section 6.2 introduces an initial model based on geometric arrangements. In Section 6.3, we discuss how to modify the initial model and solve it using projection methods. Heuristics are studied in Section 6.4. Section 6.5 is dedicated to discussing implementation details and to analysing the behavior of our models under some extreme circumstances. Section 6.6 contains a comprehensive empirical evaluation of the algorithms proposed. Concluding remarks are presented in Section 6.6.

6.2 Solving the M3CDUDC using ILP

In this section, we propose a natural formulation using arrangement of circles and discuss how to avoid enumerating the arrangement by using lazy constraints.

Unless we explicitly mention a different radius, we assume that all disks have the same radius and, w.l.o.g., that this radius is 1, since this can be ensured by properly scaling the instances.

6.2.1 Notation and Concepts

Let P be a set of n points and D be a set of m (closed) disks in the plane. Each $a \in D$ is described by its center $c(a)$. The (euclidean) distance between two points p and q is denoted by $\text{dist}(p, q)$. We abuse this notation by writing $\text{dist}(a, q)$ (or $\text{dist}(q, a)$) to denote the distance between the center of disk a and point q and similarly $\text{dist}(a, b)$ to denote the distance between the centers of the disks a and b. Using this notation, a disk a is said to *overlap* a disk b if, and only if, $\text{dist}(a, b) \leq 2$. Similarly, a disk a is said to *cover* a point p, denoted by $p \in a$, if, and only if $\text{dist}(a, p) \leq 1$. Finally, a disk set D is said to cover a point set P if for each point $p \in P$ there is a disk $d \in D$ that covers p.

The *conflict graph of a disk set* D is a graph where each vertex corresponds to a disk in D and there is an edge between two vertices if the corresponding disks overlap. The *chromatic number* of a graph is the minimum number of colors sufficient to color all vertices such that no two adjacent vertices share the same color. The chromatic number of a disk subset $D' \subseteq D$, denoted by $\chi(D')$, is the chromatic number of its conflict graph.

We generalize these notions by defining the *chromatic cover number* of an instance of the M3CDUDC comprised of point set P and disk set D as the smallest chromatic number among all subsets of D that cover P.

Given a set D of m disks, the *arrangement of* D is the planar subdivision induced by the circles corresponding to the borders of the disks in D. Figure 6.2 illustrates this definition. An arrangement of m disks is comprised of $O(m^2)$ faces, arcs and points. Each face f of the arrangement is an *atomic open region* relative to disk coverage, in the sense that all points in f are covered by the same subset of disks in D. Thus, a point $p \in f$ is covered by a disk $a \in D$ if and only if f is fully contained in the interior of a. In this case, we say that *a covers f*. Due to this atomicity property, we can represent each face of the arrangement by a single one of its points. Let us then denote by $F(D)$ a set obtained by picking one (arbitrary) point in each face of the arrangement of D.

To simplify exposition and avoid intersections of disks that are single points (i.e., zero dimensional), we increase the radius of each disk to $1 + \varepsilon$, in such a way that all pairs of non-overlapping unit disks remain non-overlapping and, moreover, any point of P that is not covered by a given disk in D remains not covered by the larger disks. Given a disk set D and a point set P, we can choose ε as follows.

$$\varepsilon = \min \left\{ \min_{a,b \in D : \text{dist}(a,b) > 2} \big\{ \text{dist}(a, b) - 2 \big\}, \min_{d \in D, p \in P : \text{dist}(a,p) > 1} \big\{ \text{dist}(a, p) - 1 \big\} \right\} / 3.$$

Clearly, this change of radius does not alter any coverage or overlap. This allows us to

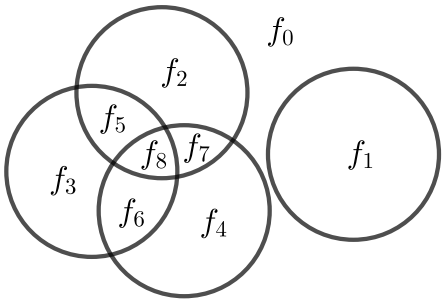


Figure 6.2: Example of an arrangement of four disks. Face f_0 is the unbounded face. Faces f_1 and f_8 are convex. The remaining faces are non-convex.

determine whether two disks overlap by finding a (two dimensional) face in A that is covered by both disks.

Thus, for the remainder of this paper, we will consider intersections of disks to be two dimensional.

6.2.2 Arrangement-based Model

Using the concept of arrangements, we can express an initial model for this problem as follows. Denote by $\mathcal{C} = \{R, G, B\}$ the set of three recommended channels ("colors") mentioned in the introduction. We associate a binary variable x_d^c to each disk $d \in D$ and each color $c \in \mathcal{C}$, indicating whether disk d belongs to the cover and is assigned color c. Then, we can write a model for the M3CDUDC that minimizes the number of disks in the cover as follows:

$$\min \quad \sum_{d \in D, c \in \mathcal{C}} x_d^c \tag{6.1a}$$

$$\text{s.t.} \quad \sum_{c \in \mathcal{C}, d \in D: p \in d} x_d^c \geq 1 \qquad \forall p \in P \tag{6.1b}$$

$$\sum_{d \in D: p \in d} x_d^c \leq 1 \qquad \forall p \in F, \forall c \in \mathcal{C} \tag{6.1c}$$

$$x_d^c \in \{0, 1\} \qquad \forall d \in D, \forall c \in \mathcal{C}. \tag{6.1d}$$

The objective function (6.1a) minimizes the total number of disks in the cover. Constraints (6.1b) ensure that each point in P is covered while Constraints (6.1c) guarantee that overlapping disks are not assigned the same color.

6.2.3 Symmetry Breaking Constraints

One of the main drawbacks of this model is its symmetry with respect to the colors. One simple way to reduce symmetry is to add constraints to prioritize colors, as follows:

$$\sum_{d \in D} x_d^B \geq \sum_{d \in D} x_d^G \tag{6.2}$$

$$\sum_{d \in D} x_d^G \geq \sum_{d \in D} x_d^R. \tag{6.3}$$

6.2.4 Convex Faces of the Arrangement

Another performance bottleneck for the model is the number of Constraints (6.1c), which may be quadratic on the number of disks. Arrangements are very rich geometric structures that allow the expression of many different problem as ILPs. However, many successful applications of arrangements to the solution of hard problems avoid associating constraints to each face of the arrangement [32, 12]. If we were to fully build the arrangement of thousands of disks – as in many of our instances – using exact arithmetic (to prevent inconsistencies), we would incur in a great computational cost since the number of faces would be quadratic on the number of disks and might require overcoming redundancies.

To reduce the number of Constraints (6.1c) in the model, we can limit ourselves to the *convex faces* of the arrangement. A face f is not convex if and only if there is an arc of circle in its border that belonging to a disk d that does not cover f. In this case, the constraint corresponding to the face f' adjacent to f and covered by d dominates the one corresponding to f. This allows us to discard constraints that do not correspond to convex faces. See Figure 6.2.

However, to the best of our knowledge, we cannot avoid computing the entire arrangement. In fact, the question of enumerating convex faces appears to be closely related to the question of counting disjoint *lenses*, which are faces corresponding to the intersection of two circles, in an arrangement of circles. The number of disjoint lenses is conjectured to be linear in the number of circles and this conjecture has been an open problem for a long time [1]. However, even if the conjecture is proved true, the result is only useful to our purposes if an algorithm exists that enumerates all the convex faces in sub-quadratic time. As no such algorithm is known so far, we actually have to compute the whole arrangement. This makes the process of building the model too resource intensive, both in time and memory, even for a modest number of disks.

6.2.5 Lazy Coloring Constraints

We can avoid constructing the entire arrangement by separating Constraints (6.1c) lazily, similarly to the iterative approach presented in [10] for another disk cover problem. By starting the formulation with very few coloring constraints, we can drastically reduce the time spent constructing the model. The remaining constraints, dubbed *lazy* ones, are added to the model on the fly. In fact, although lazy constraints are required to ensure the model's correctness, they only become part of it when a primal solution is found during the execution of the branch-and-bound algorithm that violates them.

The identification of a violated constraint, also called a *cut*, is performed by a *separation routine* that works as follows. Suppose we have an integral solution to an M3CDUDC model where not all coloring constraints (6.1c) are considered. To decide whether there is a violated coloring constraint among those not present in the incomplete formulation, we can iterate over all pairs of disks that are assigned the same color. Clearly,

if a coloring constraint exists that is violated by the current solution, there is a pair of disks (a, b) of the same color, say c, that overlap. In this case, the current solution violates the coloring constraint $x_a^c + x_b^c \leq 1$. Of course, when the intersection of the two disks is non-empty, there is a point p that belongs to both of them. Stronger and more general coloring constraints can be derived from the one given above if we consider the set of all disks covering p and all possible colors. This leads to the set of constraints

$$\sum_{d \in D : p \in d} x_d^c \leq 1, \forall c \in \mathcal{C}. \tag{6.4}$$

A practical issue to be considered when implementing the separation routine refers to the computation of the coordinates of the point p from the discussion above. Even if all input values are given as rational numbers, when finding p, one should avoid the use of square roots, as they may lead to non-rational algebraic numbers. As a consequence, since we are doing exact computations, square roots can become very expensive to operate with. However, under the assumption of rational input data, non-rational algebraic representations of p are unnecessary provided that we proceed as follows.

As before, let a and b be two overlapping disks and recall that their centers are denoted by $c(a)$ and $c(b)$, respectively. To encompass the general case of non unit disks, let us denote by $r(a)$ and $r(b)$ the radius of a and b, respectively. A point p that belongs to the interior of both of these disks can be obtained as follows:

$$p = c(a) + (c(b) - c(a)) \frac{r(a)}{r(a) + r(b)}. \tag{6.5}$$

Also, when restricted to unit disks, p can be chosen as the midpoint of the segment that connects the centers of the disks, i.e., this formula simplifies to:

$$p = \frac{c(a) + c(b)}{2}. \tag{6.6}$$

Moreover, since the elements of the arrangement of m disks are atomic, be them faces, arcs or points, we generate at most $O(m^2)$ lazy constraints.

6.3 Extended Model

In the previous section, we discussed how to address the symmetry of Model (6.1) as well as the difficulties related to enumerating the arrangement. In this section, we modify the previous model allowing it to be decomposed in a way that tackles both issues simultaneously.

To do this, we associate a binary variable y_d to each disk $d \in D$ that is assigned the value one if, and only if, d is part of the solution. Recalling that F is the set of points representing the faces of the arrangement of D, we can write the model as follows:

$$\min \quad \sum_{d \in D} y_d \tag{6.7a}$$

$$\text{s.t.} \quad y_d - \sum_{c \in \mathcal{C}} x_d^c = 0 \qquad \forall d \in D \tag{6.7b}$$

$$\sum_{d \in D : p \in d} y_d \geq 1 \qquad \forall p \in P \tag{6.7c}$$

$$\sum_{d \in D : p \in d} x_d^c \leq 1 \qquad \forall p \in F, \forall c \in \mathcal{C} \tag{6.7d}$$

$$y_d \in \{0, 1\} \qquad \forall d \in D \tag{6.7e}$$

$$x_d^c \in \{0, 1\} \qquad \forall d \in D, \forall c \in \mathcal{C}. \tag{6.7f}$$

Notice that Constraints (6.7b) are the link constraints while Constraints (6.7c) and (6.7d) do not share variables of different types. We can apply Benders Decomposition to this model [35], by splitting it into a master problem that has y variables and a subproblem that contains only the x variables. This can be interpreted as the master problem being responsible for covering the points while the subproblem handles colorings. More specifically, the purpose of the subproblem is to decide whether the conflict graph derived from the disks in a solution to the master problem admits a 3-coloring. When it does not, the subproblem must provide a cut that separates this solution.

However, since deciding whether a graph is 3-colorable is NP-hard, we cannot rely on LP duality to derive Benders Cuts. Alternatively, we can combinatorially construct our cuts, which makes this an application of Logic-Based Benders Decomposition [21]. Notably, since the variables in the subproblem do not appear in the objective function, we need only to provide feasibility cuts.

In this context, a *violation* v is a subset of disks D_v and an integer S_v such that if there are more than S_v disks of D_v in the solution, the resulting conflict graph is not 3-colorable. Let $\mathcal{V}$ denote the set of all violations. We can express the master problem as the following ILP.

$$\min \quad \sum_{d \in D} y_d \tag{6.8a}$$

$$\text{s.t.} \quad \sum_{d \in D : p \in d} y_d \geq 1 \qquad \forall p \in P \tag{6.8b}$$

$$\sum_{d \in D_v} y_d \leq S_v \qquad \forall v \in \mathcal{V} \tag{6.8c}$$

$$y_d \in \{0, 1\} \qquad \forall d \in D. \tag{6.8d}$$

Let D' be a solution to (6.8) and recall that $F(D')$ denotes a set of points corresponding to faces in the arrangement of D'.

By associating a ternary variable $c_d \in \{R, G, B\}$ to disk $d \in D'$, we can write the

following Constraint Programming (CP) model to solve the coloring subproblem

$$\textit{all\_different}\left(\{c_d : p \in d\}\right) \qquad \forall p \in F(D') \tag{6.9a}$$

$$c_d \in \{R, G, B\} \qquad \forall d \in D', \tag{6.9b}$$

where the *all_different* constraints (c.f. [30]) ensure that disks covering the same face are assigned different colors.

If the coloring subproblem of the conflict graph obtained from D' is infeasible, we can add the violation v where $D_v = D'$ and $S_v = |D'| - 1$ to the set $\mathcal{V}$ of violations. This cut, called a *no-good cut*, is notably weak and will be further improved later in this section.

We have determined that this new decomposition leads to enormous speed ups. With fewer variables and no color assignment, the master problem is close enough to a pure set-covering problem that it is often easily computed by modern ILP solvers. The resulting subproblem is a system of *all_different* constraints, a core constraint in Constraint Programming, that is also easy to be solved by modern constraint programming solvers [4].

With this decomposition, we avoid having to explicitly enumerate the conflict graph, or the arrangement, for the full set of disks while also removing coloring symmetries from the master problem. Even if we still need to enumerate an arrangement and deal with coloring symmetries for the resulting subproblems, by restricting ourselves to integral solutions, we not only drastically reduce the number of disks involved, but we can also use Constraint Programming to solve it. This performance gain can be further explained by the fact that modern Constraint Programming solvers have built-in symmetry breaking mechanisms, such as [27]. Thus, the resulting speedup is a consequence of combining the strengths of both the ILP and the CP paradigms.

6.3.1 Solving the Subproblems

The success of this decomposition relies on the ability to identify small subsets of a solution to the master problem that are not 3-colorable. By finding such small subsets, we can cut many similar solutions at the same time, reducing the number of subproblems that need to be solved.

In this section, we discuss how to generate strong cuts and present instances where strong cuts avoid exploring an exponential number of solutions. To accomplish this, we propose the methods described below.

Moreover, the impact of these methods on a family of very hard instances is discussed in detail in Section 6.5.

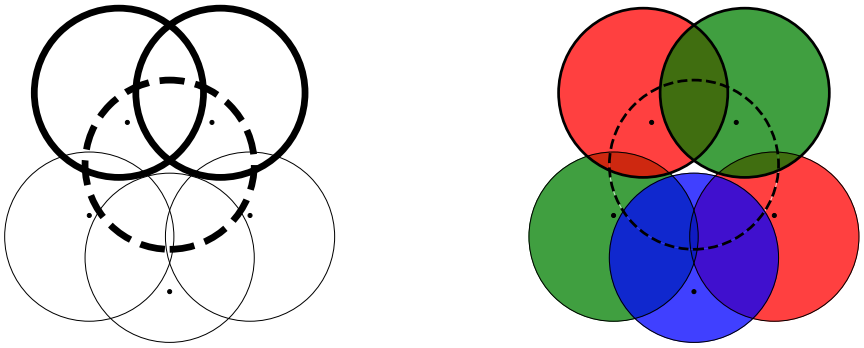
Block-Cut Tree Decomposition

Let G be a graph, a block-cut tree decomposition is obtained by collapsing its biconnected components. The resulting graph is a tree, called a *block-cut tree*. Each biconnected component can be colored separately and a coloring for G can be obtained by traversing the block-cut tree and permuting the colors of each biconnected component so as to avoid conflicts [20].

This decomposition not only accelerates the coloring, which can be done in parallel on smaller graphs, but also leads to multiple stronger cuts. Actually, each non 3-colorable biconnected component provides a violation, exempting us from having to consider the entire solution as the violation found.

Now, consider the gadget shown in Figure 6.3a. We argue that it admits two minimal covers. The disks not in bold must be part of any feasible solution, since each one is the sole disk covering a particular point of the instance. Among the three bold disks in the upper part of the instance, there are two alternative minimal covers: the dashed disk or the two solid ones. A minimum solution including the dashed disk is smaller, but its conflict graph is a clique of four vertices, which is not 3-colorable. If, instead, we choose the two solid disks, the resulting solution is larger, but is 3-colorable, as shown in Figure 6.3b. Moreover, it is easy to see that the conflict graph of the disks in this gadget is biconnected.

By carefully chaining together several of these gadgets side-by-side, in such a way that each pair of consecutive gadgets have a pair of overlapping disks, we can construct arbitrarily large instances that admit an exponential number of non 3-colorable covers that are smaller than the unique 3-colorable one, illustrated in Figure 6.4. All these non 3-colorable solutions must be identified by the decomposition algorithm before proving optimality.



(a) Instance (b) Optimal Solution

Figure 6.3: Instance with five points and six disks and its optimal solution.

By removing any one of the overlapping disks shared by two gadgets, the conflict graph of the ensuing instance becomes disconnected. This implies that each gadget is a biconnected component and the block-cut tree decomposition algorithm will generate one cut for each gadget, leading to a linear number of cuts being added before the instance is solved to optimality. For example, for the instance comprised of 10 gadgets shown in Figure 6.4, at least 1024 non 3-colorable solutions must be explored before optimality can be proved.

Odd-wheel Separation

An *odd-wheel* is a graph composed of an odd cycle with at least 3 vertices and a central vertex connected to all vertices of the cycle. Odd-wheels form a class of non 3-colorable

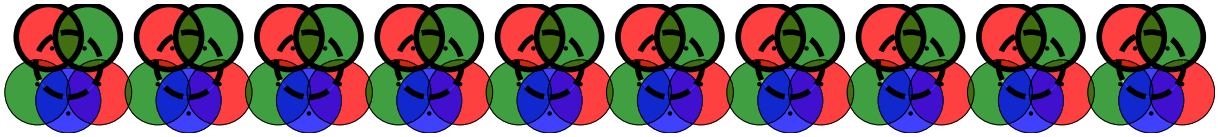


Figure 6.4: Solution to a hard instance constructed from 10 gadgets. The displayed 3-colorable cover contains 50 disks, while the minimum cover containing 40 disks has chromatic number four.

graphs that can be identified as a subgraph in polynomial time. Notably, any clique with more than three vertices contains an odd-wheel. We call a wheel whose largest odd-cycle has size n an *n-wheel*. In the remaining of this section, we will refer to a disk and its corresponding vertices in the conflict graph interchangeably, as the context makes it clear to which element we are referring. We denote the clique with four vertices by K_4.

To separate the odd-wheels of a given graph, we consider each one of its vertices as a central vertex, and look for odd cycles in its immediate neighborhood. Odd cycle detection can be done in linear time on the size of the graph using a Breadth-First search. The complexity of this odd-wheel detection algorithm for graphs with t vertices is $O(t\mathcal{N}^2)$, where $\mathcal{N}$ denotes the size of the largest vertex neighborhood in the graph.

Let $G = (V, E)$ be a graph and denote by $W = (C, k)$ an odd-wheel with odd cycle $C \subseteq V$ and central vertex $k \in V$. The following constraint is a valid inequality to separate 3-colorable graphs:

$$\sum_{u \in C} y_u + y_k \leq |C|. \tag{6.10}$$

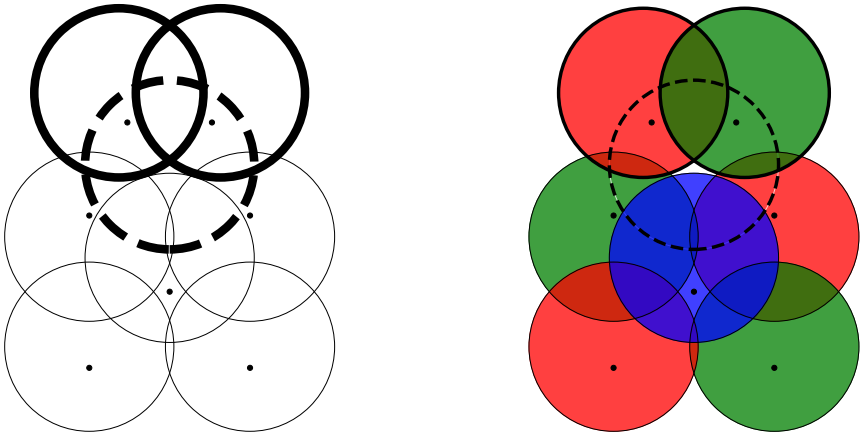
The separation is done independently on each biconnected component. We first use the odd-wheel separation, and resort to solving Model (6.9) exactly only if it fails to find any violations.

Similarly to what we did for the Block Graph Decomposition, we now show a family of instances that required an exponential number of solutions to be explored by the decomposition without the Odd-wheel Separation.

Now, observe the gadget depicted in Figure 6.5a, obtained by adding to the previously discussed gadget two points and two disks that uniquely cover those points. Clearly, these disks must be part of any cover of the instance shown. As before, it admits two minimal covers: one requiring four colors and the other is 3-colorable, as shown in Figure 6.5b. Now, when constructing instances by chaining together several of these gadgets, there is no one disk that can be removed to make a resulting solution disconnected. Thus, any solution must be comprised of a single biconnected component and, hence, the Block Graph Decomposition can have no impact on it.

On the other hand, the non 3-colorable solution for each gadget is a 5-wheel. As a consequence, an execution of the odd-cycle detection algorithm will find one odd-wheel per gadget, taking a linear number of iterations to solve these instances.

As an example, an instance comprised of 10 gadgets is shown in Figure 6.6.



(a) Instance (b) Optimal Solution

Figure 6.5: Instance with seven points and eight disks and its optimal solution.

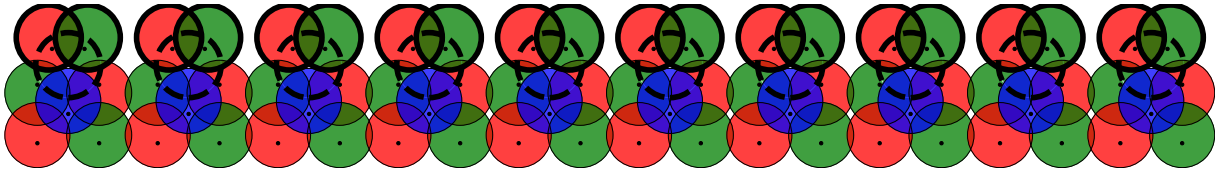


Figure 6.6: Solution to a hard instance constructed from 10 gadgets. The displayed 3-colorable cover contains 70 disks, while the minimum cover containing 60 disks has chromatic number four.

Lifting Odd-wheels

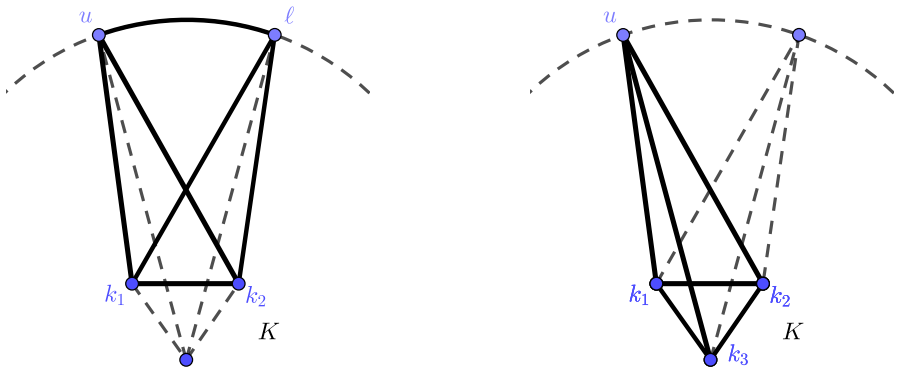
The resulting odd-wheel cuts can be lifted by combining odd-wheels that share the same odd cycle and have pairwise adjacent central vertices. Lifting of odd-wheel inequalities has been studied in the context of strengthening stable set inequalities [8, 28]. Since it is an NP-hard problem for general graphs, this lifting is usually done heuristically. Next, we show how to perform the lifting for unit disk graphs exactly in polynomial time.

Let $K \subseteq V$ be a clique and $C \subseteq V \setminus K$ be an odd cycle such that (C, k) is an odd-wheel in G for all $k \in K$. Then, the following inequality is valid to separate 3-colorable graphs.

$$\sum_{u \in C} y_u + \sum_{k \in K} y_k \leq |C|. \tag{6.11}$$

The validity of this lifting can be proved by showing that there is no 3-colorable subgraph G' of G induced by $|C| + 1$ vertices of $C \cup K$. To accomplish this, we split the analysis into cases based on the number of vertices of K in G'. If it is exactly one, then we have an odd-wheel. If there are two vertices k_1 and k_2 of K in G', and u and ℓ are any pair of adjacent vertices in C, then $\{k_1, k_2, u, \ell\}$ induces a K_4, see Figure 6.7a. If there are three vertices of K in G', say k_1, k_2 and k_3, then $\{k_1, k_2, k_3, u\}$ also induces a K_4 for any $u \in C$, see Figure 6.7b. Finally, if there are four or more vertices of K in G', they form a clique with at least four vertices.

Given an odd-wheel (C, k), we can find the strongest lifting exactly in polynomial time



(a) Two vertices from K (b) Three vertices from K

Figure 6.7: Remaining cases for lifting proof based on the number of vertices of a subgraph G' that belong to clique K.

using the following geometric construction.

Let W be an odd-wheel composed of a cycle C and a central vertex k. An arrangement of disks is obtained by including, for every $w \in W$, a disk of radius 2 with the same center as the disk represented by w. This construction is illustrated in Figure 6.8. Denote by $L \subseteq D$ the set of disks whose centers belong to the face that contains k. Notice that this face is covered by all disks in the construction. It is easy to see that each one of these vertices is central to an odd-wheel with cycle C and their corresponding disks overlap the disk represented by k. Thus, we only need to find a maximum clique in L. Although a maximum clique can be found in polynomial time for unit disk graphs [9], we chose to find it here by solving the standard ILP formulation for maximum clique simply because its solving time was negligible for all instances we encountered.

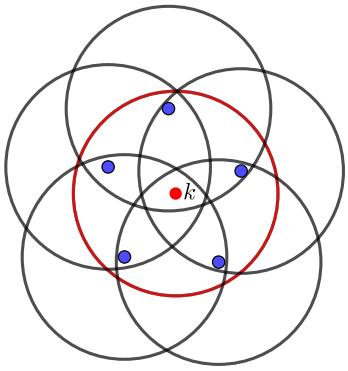


Figure 6.8: Illustration of the lifting algorithm. All disks here have radius 2. A maximum clique of the vertices corresponding to disk centers that lie in the face containing k leads to the strongest lifting.

6.4 Heuristics

When solving very large instances, we determined that it is difficult for the solver to find good primal solutions. Since finding feasible solutions became an issue, we propose

three heuristics. One is based on repairing primal infeasible solutions discarded by the subproblem, the second is based on the solution to the linear relaxation, and the third is designed as a polynomial greedy algorithm. The heuristics based on solutions, be them to the relaxation or to the master problem, work by fixing a large number of variables and solving a much smaller instance of the original problem.

6.4.1 Violation Repair

Since the master problem is a combinatorial relaxation of the original problem, we expect its solution that are infeasible – for the original problem – to still provide useful information. This expectation naturally leads us to try to fix the infeasibilities present in discarded (infeasible) solutions.

To do this, we force all disks not involved in violations to be part of the solution, ignoring color assignments except when they are the result of branching decisions. Then, we solve a smaller instance restricted to the uncovered points.

As long as the number of violations is small, the reduced problem is quite fast to solve. However, in the case Model (6.1), very few solutions to the master problem are found, which reduces the impact of this heuristic.

6.4.2 Relaxation-based reduction

A similar heuristic can be employed by using the current relaxation to create a reduced instance, instead of an infeasible solution.

This is done by removing all disks whose corresponding variables have value zero in the relaxation. Again, color assignments are only kept if there were branch decisions.

6.4.3 Greedy Heuristic

Next, we discuss a polynomial time heuristic called GreedyMinCol. In this heuristic, we construct a solution one color at a time, as follows. The algorithm starts with all available disks and a color. At each iteration, a disk is chosen greedily so as to maximize the number of uncovered points that would be covered. Finally, we remove all disks that overlap the chosen disk. When there are no more candidates or the remaining disks do not cover any new points, we move to the next color, starting with disks not belonging to the partial cover being constructed.

When starting the algorithm, or when moving to a new color, the algorithm computes the number of uncovered points for each disks, which takes $O(nm)$ time. At each iteration, we must scan the $O(m)$ disks to find the best candidate and then remove all the $O(m)$ disks that intersect with it, as well as all the $O(n)$ points that it covers. Finally, we must update the number of uncovered points that covered by each available disk. Since each one of the $O(n)$ disks might cover $O(m)$ points, this update takes $O(nm)$ time.

Since at each iteration a new point is covered, it takes at most n iterations, each taking $O(nm)$ time. Thus, the final time complexity is $O(n^2m)$.

In the case of unit disks, we can speed up this algorithm in practice by keeping two 2D-trees, one for the points and one for the centers of the disks. By doing so, the points

and disks to be removed are the nearest neighbors of the disk center with distance at most 2, in the case of removing overlapping disks, or distance at most 1.

If there are any uncovered points after the third color, we employ local search. This is done by looking for a disk that can be swapped for a disk already in the solution so that the number of uncovered points is reduced. We also remove disks whose covered points are already covered by another disk, i.e., redundant disks. We repeat this process until no disk can be swapped or removed.

To further improve the performance of the heuristic, we introduce randomization, and allow it to be run several times. This randomization is introduced by choosing not the best disk, but one among a fixed number of best candidates. The algorithm is executed until a certain number of iterations have been processed without improving the best solution.

Since the corresponding decision problem is known to be NP-hard and the current literature contains only rather impractical algorithms, the above greedy procedure proved to be the most effective one for finding initial feasible solutions for the models.

6.5 Analysing the decomposition

In this section, we investigate theoretical aspects of the decomposition. We first discuss how the decomposition is implemented. Next, we prove that for any instance the number of colors required to color a minimum cover, which corresponds to the solution found in the first iteration of the algorithm, is within a constant factor of the chromatic cover number of the instance.

6.5.1 Implementing Benders Decomposition

There are two standard ways of implementing Benders Decomposition. The first one consists in implementing lazy constraints for the master problem that attempt to detect violations for each feasible solution found, while traversing the search tree. The second consists in solving the master problem to optimality, adding violations found only in the computed optimal solution. This procedure is repeated until no violations are found. More sophisticated implementations have also been proposed. See [26] for an in-depth discussion on how to implement the Benders Decomposition algorithm.

The lazy constraints approach has the advantage of always keeping a valid dual bound and using a single search tree.

In contrast, the iterative approach has to restart the search each time a violation is found. However, as the new ILP is entirely known in the beginning of each iteration, we can use all the preprocessing techniques available in modern solvers. Also, a large number of constraints may be added by the lazy approach that do not appear in near-optimal solutions. Despite proving optimality not being necessary, these lazy constraints make linear programs solved during enumeration harder to compute, leading to increased solving time. For more details on how lazy constraints are implemented in modern solvers, see [22].

The decision of which approach to use depends on the trade-off between the number of restarts and the impact of the preprocessing techniques.

The results in Section 6.6 reveal that for the M3CDUDC, not only very few iterations are necessary on average, but also that modern solvers are capable of taking advantage of the fact that the master problem consists almost entirely of set cover constraints. These observations shed light on why the iterative approach outperformed the lazy one in our experiments.

Now that we understand how the decomposition is implemented, we can better understand its performance. In particular, in which situations the algorithm is expected to perform poorly.

Intuitively, it seems reasonable to assume that a hard instance for the decomposition approach is one for which several iterations are required before the subproblem fails to provide a no-good cut. Moreover, such a bad case is likely to occur when the sizes of a minimum disk cover of the points in P and a minimum 3-colorable disk cover of the same points differ by a large amount. This difference is exacerbated when there are many solutions of the same size, since the computation of one subproblem may be needed to eliminate each one of those solutions. In the next sections, we take a deep dive into these questions.

6.5.2 Coloring Minimum Covers

Due to the inherent geometry of unit disk graphs, we will show that the number of colors in a minimum cover is bounded by a constant factor w.r.t. the minimum number of colors among all covers. This constant factor approximation can be interpreted as the first master problem and the original model having a somewhat similar goal.

Given a point set P, a disk set D and a minimum-sized cover $D' \subseteq D$ of P, consider a greedy coloring algorithm that sorts the disks in D' by the x-coordinates of their centers, and performs a horizontal left-to-right sweep to assign to each disk encountered the first compatible color available. Without loss of generality, let us assume that there is no pair of disks with the same x-coordinate, which can be accomplished by carefully rotating the instance.

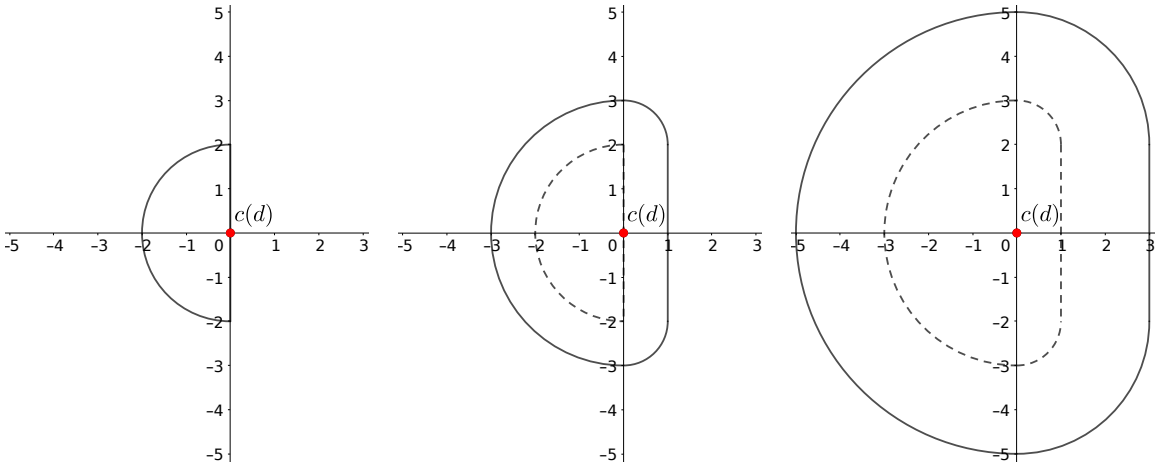
Given a disk d in D', with center $c(d)$, we denote by L_d the set of disks that overlap d and lie to its left. These disks get assigned colors by the greedy algorithm before d does. We now argue that the number of disks in L_d is at most some linear function of the chromatic number of any cover of P comprised of disks in D.

Since the (unit) disks in L_d overlap d and are to its left, their centers must lie in a half-circle of radius 2 with center $c(d)$. This region, denoted by $R(L_d)$, is illustrated in Figure 6.9a.

Denote by $P_d \subseteq P$ the set of points that are not covered by $D' \setminus (L_d \cup \{d\})$. Since, D' is a minimum cover, P_d cannot be empty. Such points must lie in the region given by the union of all unit disks centered at points in $R(L_d)$. We denote this region by $R(P_d)$ (see Figure 6.9b).

Now, denote by χ the chromatic number of the instance comprised of point set P and disk set D, i.e., χ is the minimum chromatic cover number among all covers of P using disks in D. Also, denote by D^χ a cover of P with chromatic number χ, and by D_d^χ a set of disks in D^χ that covers the points in P_d. Since these disks must at least touch $R(P_d)$,

they have to lie entirely in the region given by the union of all unit disks that overlap $R(P_d)$. This region, which we denote by $R(D_d^\chi)$, is illustrated in Figure 6.9c.



(a) R(L_d) (b) $R(P_d)$ (c) $R(D_d^\chi)$

Figure 6.9: Geometric construction used to establish an upper bound on the chromatic number of a minimum cover.

We proceed by computing the maximum number of non-overlapping disks that can fit in $R(D_d^\chi)$. From elementary geometry, the area of $R(D_d^\chi)$, denoted by $Area(D_d^\chi)$, is

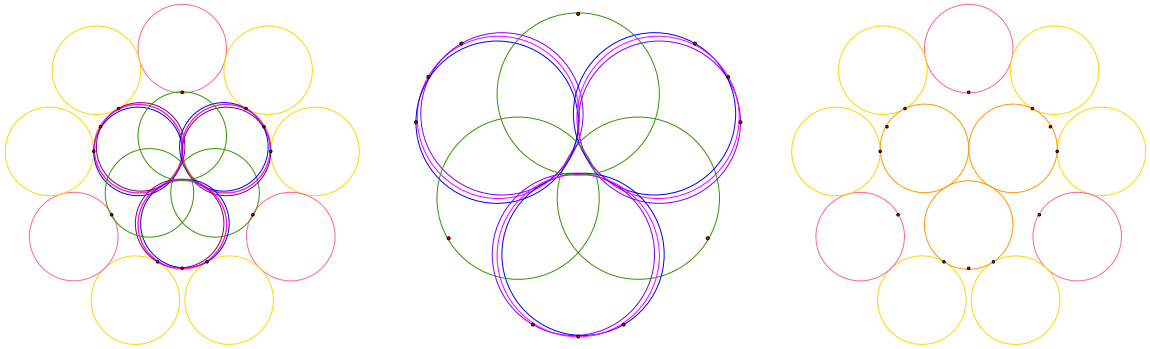
$$\frac{\pi 5^2}{2} + 2\frac{\pi 3^2}{4} + 3 \times 4 = 17\pi + 12.$$

It follows from [34], as discussed in [16], that the convexity of $R(D_d^\chi)$ implies that its area may contain at most $\left\lfloor \frac{Area(D_d^\chi)}{\sqrt{12}} \right\rfloor = 18$ non-overlapping disks. Therefore, the largest number of χ-colorable disks that can fit in $R(D_d^\chi)$ is 18χ. Hence, $|D_d^\chi| \leq 18\chi$.

Finally, since D' is a minimum cover, it follows that $|L_d \cup \{d\}| \leq |D_d^\chi|$. This leads us to conclude that $L_d \cup \{d\} \leq |D_d^\chi| \leq 18\chi$, implying that a minimum cover can be greedily colored using at most 18χ colors. Therefore, the number of colors a minimum cover requires is at most 18 times the number of colors required by any cover.

This result can be generalized for disks of different sizes, assuming that the difference in sizes is bounded by a constant. Indeed, assume that the largest disk u in D has radius 1 while the smallest has radius ρ, for a constant $0 < \rho \leq 1$. In the worst case scenario, if there is a cover comprised only of disks of radius ρ, we can bound the size of $L_d \cup \{u\}$ to at most $\left\lfloor \frac{Area(D_d^\chi)}{\rho^2\sqrt{12}} \right\rfloor = \left\lfloor \frac{17\pi+12}{\rho^2\sqrt{12}} \right\rfloor$.

Unfortunately, we have not been able to determine how tight this bound actually is, even in the case of unit disks. We were able, though, to construct an instance for which the ratio between the minimum number of colors required by a minimum-sized disk cover and the minimum number of colors required by any disk cover is 12. See Figure 6.10. Nonetheless, this ratio is still below the theoretical value computed above.



(a) Instance

(b) Cover with chromatic number 12 (zoomed in)

(c) Cover with chromatic number 1

Figure 6.10: An instance (a) showing a minimum cover (b) requiring 12 colors when a cover that minimizes colors (c) requires only one.

6.6 Empirical Evaluation

In this section, we show empirical data to evaluate our models and algorithms in practice. We begin by introducing our experimental setup and instances, followed by a comparison of our algorithms using a benchmark generated from a uniform distribution, and conclude with results on a benchmark generated from population density data.

Environment set up All experiments were run on a computer featuring an Intel Xeon Silver 4114 processor with 12 cores at 2.2Ghz and 32GB of RAM, running Ubuntu 16.04. Models and algorithms were coded in C++ v.14 and compiled with gcc 5.5. Geometric algorithms and data structures were implemented using CGAL 5.4.1 [33], and the Gmpq library for exact number representation. To solve the ILP models, we employed CPLEX version 20.1.

Generating benchmarks and algorithm performance measures We present two sets of instances, or benchmarks. The first was generated by uniformly sampling points and disk centers in the unit square, and we call it the *uniform benchmark*. The second was sampled from data obtained from the population density of France made available at [36], and is named herein the *population benchmark*. All instances, as well as the data used to generate the tables and figures in this section, are available at [31].

For both benchmarks, in order to evaluate the impact on the solving times of the ratio between the area of the region where the points are sampled and the area covered by each disk, we consider different values for the radii. This is done without loss of generality, since we can scale each instance such that the resulting disk set is a unit disk set.

Thus, an instance is characterized by three parameters: number of points, n, number of disks, m, and disk radius, r. For each configuration (n, m, r), ten instances were generated.

For each instance, we fixed our running time limit to 20 minutes. If an instance was not solved to optimality or shown to be infeasible within this time limit, it was regarded as *not solved*. It is worth noting that it is challenging to generate feasible instances for some combinations of small number of disks and small disk sizes, even when each point

was covered by at least one disk. For each algorithm being evaluated, and for each pair of radius r and number of points n, we increased the number of disks up until the algorithm failed to solve at least three instances of the given configuration. This was regarded as the maximum number of disks in an instance that the algorithm could solve, for the given pair of values of r and n. When comparing multiple algorithms, we considered all instances from the benchmark that were solvable by at least one of them.

Testing the arrangement-based model We start by shedding light on the limitations of the initial arrangement-based model described in Section 6.2 on the uniform benchmark. For this benchmark, the set of values for numbers of points and disks is {100, 300, 700, 1000, 3000, 7000, 10000, 30000, 700000, 100000}, while the values for the radii are 0.0625, 0.1250, 0.2500 and 0.5000. However, instances with radius 0.5000 only appear early in the experiments, as they proved to be trivial for most of our algorithms, while instances with radius 0.0625 had to be introduced towards the end of the tests to help further differentiate the performance of our best algorithms.

Table 6.1 shows the performance of the arrangement-based model on instances with radius at least 0.1250. Column "Cover Size" gives statistics on the number of disks in a solution for each value of radius, while column "Build Time" displays the time required to build the formulation. Note that aggregated data are presented as *mean* ± *standard deviation*, with the standard deviation being omitted when null. This format is also used in the subsequent tables.

Table 6.1: Results for the arrangement-based model: all instances from the configurations shown were solved to optimality.

r	n	m	**Cover Size**	**Build Time (s)**	**Total Time (s)**
0.125	100	300	19.10 ± 0.88	2.27 ± 0.13	4.71 ± 0.30
0.125	300	300	24.90 ± 0.88	2.34 ± 0.11	6.29 ± 0.73
0.125	700	300	28.80 ± 0.79	2.21 ± 0.12	10.59 ± 3.63
0.125	1000	300	29.90 ± 0.74	2.22 ± 0.12	21.87 ± 24.95
0.125	3000	300	32.70 ± 0.67	2.18 ± 0.09	105.73 ± 92.40
0.250	100	300	6.90 ± 0.32	9.56 ± 0.39	165.92 ± 17.62
0.250	300	300	7.90 ± 0.32	9.55 ± 0.56	209.49 ± 40.30
0.250	700	300	8.40 ± 0.52	9.45 ± 0.56	246.94 ± 114.75
0.250	1000	300	9.00	9.39 ± 0.45	223.95 ± 49.23
0.250	3000	300	9.00	9.43 ± 0.41	303.77 ± 109.20
0.500	100	300	3.00	27.86 ± 0.28	365.53 ± 19.19
0.500	300	300	3.00	27.85 ± 0.25	378.07 ± 47.37
0.500	700	300	3.00	27.80 ± 0.26	388.32 ± 30.62
0.500	1000	300	3.00	27.95 ± 0.20	374.08 ± 31.68
0.500	3000	300	3.90 ± 0.32	28.04 ± 0.21	381.22 ± 18.62

The arrangement-based model was unable to solve instances with 700 or more disks due to high memory usage, while instances with 100 disks were easily solved in less than 20 seconds. For this reason, the results reported in Table 6.1 are limited to instances

with 300 disks. Throughout the experiments, we observed that even for a small set of points, the algorithm had difficulties in dealing with the large number of constraints in the model due to the superabundance of disk overlaps. We also noticed that the total execution time grows rapidly with the radius of the disks, as can be seen, for instance, by comparing rows 1, 6 and 11 of Table 6.1. Clearly, the complexity of the arrangement is the main bottleneck for the scalability of the model. Larger disks, just like an increase in the number of disks, cause more overlaps, which amplifies the complexity of the arrangement. In fact, here, the build time alone is quite high when compared to the total solving time for much larger instances when solved by algorithms presented later in this section.

The benefits of lazy separation and primal heuristics Next, we show how lazy separation and the inclusion of the primal heuristics proposed in Section 6.4 impact the computation time of the algorithm.

During preliminary testing with the lazy separation, we observed that the main issue preventing instances from being solved was the lack of primal feasible solutions. This was expected, as the introduction of lazy constraints disables most preprocessing procedures and makes it hard even for a modern solver to handle the model. This was the main motivation for the development of primal heuristics for which some implementation details are presented below.

The randomized greedy heuristic (Greedy) is executed before invoking the solver. At each iteration, it randomly chooses a candidate among the best four, according to the greedy criteria, and runs until 40 iterations are executed with no solution improvement. The choice of these parameter values was made after tuning them with the `irace` package [25].

Both the repair heuristic (Repair) and the relaxation heuristic (Relax) were implemented using the solver's callback interface and are called several times during execution of the enumeration algorithm. Repair is called when a primal solution is rejected by the lazy separation, while Relax is called according to the solver's inherent heuristics manager.

For the subsequent experiments, we cast aside instances with a small number of disks and restrict the number of points to the maximum quantity for which configurations with 3000 disks of radius 0.25 are still solvable.

To present the comparison between the algorithms on the uniform benchmark in a compact manner, we group configurations by radius, r and number of points, n. Having fixed these two parameters, we exhibit in Table 6.2 the maximum number of disks, hereafter denoted by **Max** m, that results in a solvable configuration. Thus, for each pair (r, n), the best performing algorithm is the one with the largest **Max** m. Ties are broken by the highest number of instances solved, denoted by *Sol*, followed by the lowest average total time. The aggregated data displayed is restricted to the instances that were solved.

In Table 6.2, for each pair (r, n), we can perceive that the largest number of disks the algorithm can handle has grown, clearly indicating that the memory issues are greatly diminished, as expected, when lazy constraints are used. Contrarily to when the entire arrangement is enumerated, complexity decreases as the radius shrinks when coloring

constraints are implemented in a lazy fashion. In fact, instances with radius 0.5 were so trivial to be solved that we saw no reason to pollute the table with their results. Overall, the maximum number of disks of the solved instances increases drastically when both the number of points and the radius are fixed. Also, we can see that the number of lazy cuts given in the "# Lazy Cuts" column is much smaller when the heuristics are included, as they prevent the solver from separating covers with a large number of disks. In particular, the number of cuts when using GREEDY is the smallest, which indicates that the solutions produced by this heuristic are of very high quality.

Table 6.2: Instances solved by the LAZY model alone and in combination with primal heuristics. For each pair (r, n), best performing algorithm has its **Max** m value in bold.

		LAZY				REPAIR			
r	n	**Max** m	**Sol**	**Total Time(s)**	**# Lazy Cuts**	**Max** m	**Sol**	**Total Time (s)**	**# Lazy Cuts**
0.125	100	70000	9	564.73 ± 334.41	19.56 ± 23.44	70000	9	574.05 ± 340.87	19.22 ± 23.04
0.125	300	**10000**	10	248.99 ± 267.50	69.40 ± 32.34	10000	9	241.48 ± 181.40	63.11 ± 20.39
0.125	700	700	9	324.86 ± 276.56	178.56 ± 77.90	700	9	260.38 ± 283.49	156.67 ± 80.85
0.125	1000	**700**	7	569.74 ± 353.59	189.43 ± 76.80	300	9	140.02 ± 270.94	119.89 ± 26.01
0.125	3000	**100**	10	0.97 ± 0.11	0.00	-	-	-	-
0.250	100	100000	9	635.85 ± 292.57	2.11 ± 4.14	100000	9	634.30 ± 292.31	2.11 ± 4.14
0.250	300	30000	10	369.63 ± 289.29	14.50 ± 13.65	**30000**	10	316.20 ± 197.12	13.70 ± 12.28
0.250	700	10000	10	425.74 ± 348.36	15.30 ± 16.88	10000	10	339.23 ± 231.23	12.60 ± 12.02
0.250	1000	7000	8	461.45 ± 274.02	12.50 ± 11.81	3000	9	271.27 ± 332.22	8.33 ± 10.44
0.250	3000	1000	8	228.76 ± 256.33	5.00 ± 4.54	-	-	-	-

		RELAX				GREEDY			
r	n	**Max** m	**Sol**	**Total Time(s)**	**# Lazy Cuts**	**Max** m	**Sol**	**Total Time (s)**	**# Lazy Cuts**
0.125	100	**100000**	8	437.54 ± 179.88	6.50 ± 8.14	30000	10	189.40 ± 84.31	13.00 ± 8.72
0.125	300	3000	8	13.12 ± 4.39	37.12 ± 17.54	10000	8	224.49 ± 244.34	40.12 ± 27.59
0.125	700	300	8	249.80 ± 259.51	31.88 ± 10.83	**1000**	7	459.81 ± 304.86	167.00 ± 82.84
0.125	1000	100	10	2.29 ± 4.73	5.50 ± 7.32	300	10	213.37 ± 333.62	118.40 ± 26.66
0.125	3000	-	-	-	-	100	10	4.90 ± 0.71	0.00
0.250	100	**100000**	10	325.80 ± 211.40	1.70 ± 2.87	100000	9	711.95 ± 235.81	0.78 ± 0.83
0.250	300	30000	9	271.74 ± 233.27	13.00 ± 13.46	30000	10	324.69 ± 136.41	6.60 ± 4.77
0.250	700	**10000**	10	220.20 ± 49.22	8.90 ± 5.97	10000	10	287.93 ± 169.61	3.60 ± 4.53
0.250	1000	**10000**	10	405.06 ± 164.59	6.20 ± 6.36	7000	8	528.42 ± 346.53	2.88 ± 2.75
0.250	3000	3000	8	704.27 ± 309.81	4.12 ± 4.26	**3000**	10	546.15 ± 124.86	0.40 ± 0.97

To further compare the heuristics, we show in Figure 6.11 how many instances (from those in Table 6.2) were solved as a function of time. Instances with radius 0.5 were excluded on account of their triviality. Although not part of the present analysis, the graph also exhibits the curve relative to the ITERATIVE_BENDERS algorithm, which we will later discuss.

Now, back to the algorithms currently being compared, we can see in Figure 6.11 that the REPAIR heuristic causes a large overhead, leading to a significant decrease in the number of solved instances for any given amount of time. Both the GREEDY and the RELAX heuristics are comparable with the implementation with no heuristics, represented by the LAZY curve, but GREEDY slightly outperforms the other two methods. However, we can observe that the relative performance of the GREEDY heuristic deteriorates on easier instances because, according to the parameters chosen for this heuristic, it iterates at least 40 times before stopping.

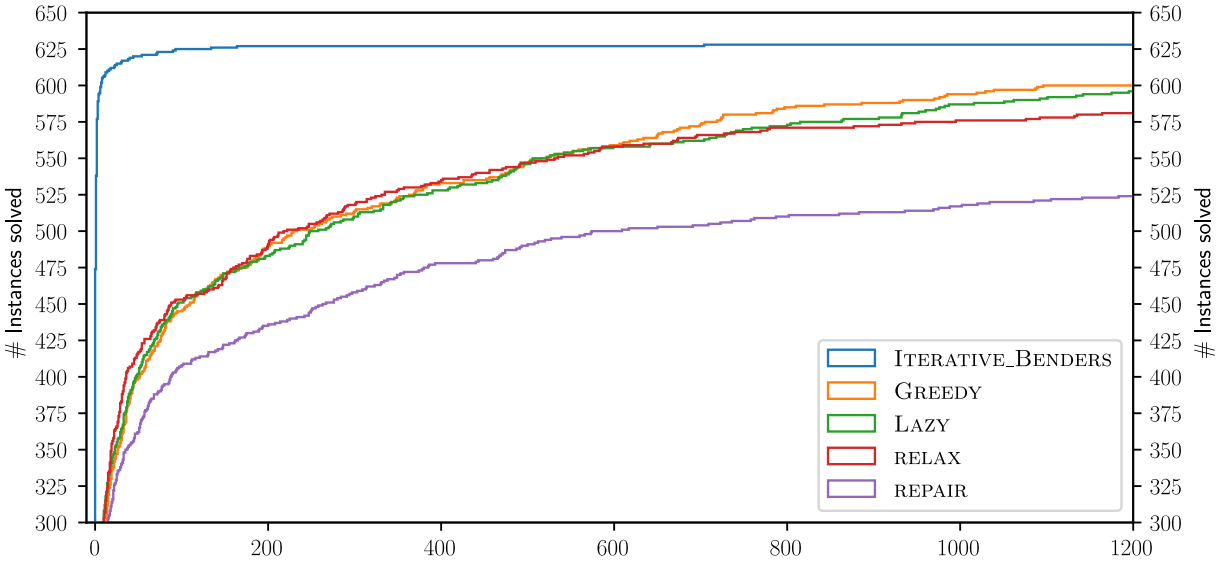


Figure 6.11: Number of instances solved as a function of time by the LAZY model, in combination with the heuristics REPAIR, RELAX and GREEDY, and the ITERATIVE_BENDERS algorithm. For a clearer presentation, we start with # Instances solved at 300.

Gains obtained by the Benders Decomposition Next, we examine the two implementations of the Benders decomposition discussed in Section 6.5.1: the lazy and the iterative one, called LAZY_BENDERS and ITERATIVE_BENDERS, respectively. Unless stated otherwise, we consider the inclusion of both block-cut tree decomposition and separation of odd-wheels procedures. For the LAZY_BENDERS implementation, we incorporated the GREEDY heuristic. However, since preliminary experiments showed that the ITERATIVE_BENDERS implementation did not benefit from employing primal heuristics, they were not applied in this case.

As previously stated, to further differentiate these methods, we decided to include harder instances, of radius 0.0625, into the benchmark and to remove all those with radius 0.5, since the former are much more challenging than the latter.

Table 6.3 shows how many instances each method solved. Clearly, the ITERATIVE_BENDERS implementation largely outperforms the LAZY_BENDERS. The advantage of the iterative approach becomes even more evident when one examines Figure 6.12 that shows the number of solved instances over time by both algorithms. This gain in performance is likely due to the small number of iterations required before reaching a feasible solution, when compared to the necessary number of lazy cuts, and to the preservation of the solver's preprocessing capabilities when lazy constraints are disabled.

However, when a larger number of disks is available, the number of non 3-colorable solutions found by both implementations is very low, as shown in columns "# Cuts" and "# Iterations". For the iterative implementation, there are many instances for which the algorithm iterates only once, unveiling that often a minimum cover is 3-colorable when there is a large number of well distributed disks. In these cases, the computational effort is almost entirely dedicated to finding a minimum cover.

Finally, we turn our attention to the comparison between the arrangement-based and

the Benders decomposition models. Comparing Tables 6.2 and 6.3, we observe that both implementations of the Benders decomposition are able to solve instances with many more disks than the best implementation of the arrangement-based model, everything else being equal. The difference in performance becomes even more perceptible when we examine the curves in Figure 6.11.

Table 6.3: Instances solved by the LAZY_BENDERS model and ITERATIVE_BENDERS. For each pair (r, n), the best performing algorithm in indicated by its **Max** m value being in bold. The number of lazy cuts found by the LAZY_BENDERS implementation is denoted by "# Cuts" while the number of iterations performed by the ITERATIVE_BENDERS implementation is denoted by "# Iterations".

		LAZY_BENDERS					ITERATIVE_BENDERS				
r	n	**Max** m	**Sol**	**Cover Size**	**Total Time (s)**	**# Cuts**	**Max** m	**Sol**	**Cover Size**	**Total Time (s)**	**# Iterations**
0.0625	100	100000	10	37.40 ± 1.26	154.86 ± 30.41	2.30 ± 0.48	**100000**	10	37.40 ± 1.26	0.60 ± 0.02	1.00
0.0625	300	30000	10	55.80 ± 1.69	44.60 ± 10.55	3.10 ± 0.88	**30000**	10	55.80 ± 1.69	0.37 ± 0.06	1.00
0.0625	700	7000	9	72.44 ± 0.73	286.63 ± 243.27	7.78 ± 2.28	**7000**	10	72.40 ± 0.70	41.62 ± 57.70	1.00
0.0625	1000	1000	10	91.80 ± 1.23	18.42 ± 8.35	31.70 ± 23.89	**3000**	7	81.57 ± 1.37	502.73 ± 417.07	1.00
0.0625	3000	300	10	infeasible	33.23 ± 11.37	3.10 ± 2.02	**300**	10	infeasible	0.22 ± 0.17	3.80 ± 2.44
0.1250	100	100000	10	15.80 ± 0.42	153.02 ± 21.76	2.50 ± 0.71	**100000**	10	15.80 ± 0.42	0.90 ± 0.03	1.00
0.1250	300	30000	10	20.50 ± 0.71	78.38 ± 45.52	4.00 ± 0.67	**30000**	10	20.50 ± 0.71	0.75 ± 0.12	1.00
0.1250	700	3000	10	24.00	13.95 ± 5.27	6.00 ± 1.49	**3000**	10	24.00	2.80 ± 1.20	1.00
0.1250	1000	7000	9	24.00	270.97 ± 358.70	6.33 ± 1.58	**10000**	10	24.20 ± 0.42	82.55 ± 131.85	1.00
0.1250	3000	700	7	29.86 ± 0.38	610.70 ± 407.62	601.00 ± 450.56	**1000**	9	28.89 ± 0.33	303.78 ± 205.10	19.67 ± 25.47
0.1250	7000	**300**	8	34.38 ± 0.52	223.27 ± 271.90	2661.62 ± 3187.26	300	7	34.43 ± 0.53	351.10 ± 357.09	508.00 ± 462.50
0.2500	100	100000	10	6.10 ± 0.32	161.25 ± 10.79	1.90 ± 0.32	**100000**	10	6.10 ± 0.32	2.06 ± 0.04	1.00
0.2500	300	30000	10	7.20 ± 0.42	125.79 ± 126.00	2.90 ± 0.74	**30000**	10	7.20 ± 0.42	15.23 ± 41.95	1.10 ± 0.32
0.2500	700	10000	10	8.00	48.22 ± 25.85	4.40 ± 1.65	**10000**	10	8.00	2.25 ± 0.59	1.10 ± 0.32
0.2500	1000	10000	10	8.00	56.24 ± 17.07	4.80 ± 1.48	**10000**	10	8.00	4.59 ± 2.05	1.10 ± 0.32
0.2500	3000	10000	9	9.00	390.59 ± 250.05	7.67 ± 2.00	**10000**	9	9.00	258.85 ± 216.34	2.22 ± 1.56
0.2500	7000	**7000**	8	9.00	474.80 ± 144.93	5.25 ± 1.67	7000	7	9.00	704.98 ± 309.55	1.43 ± 0.53

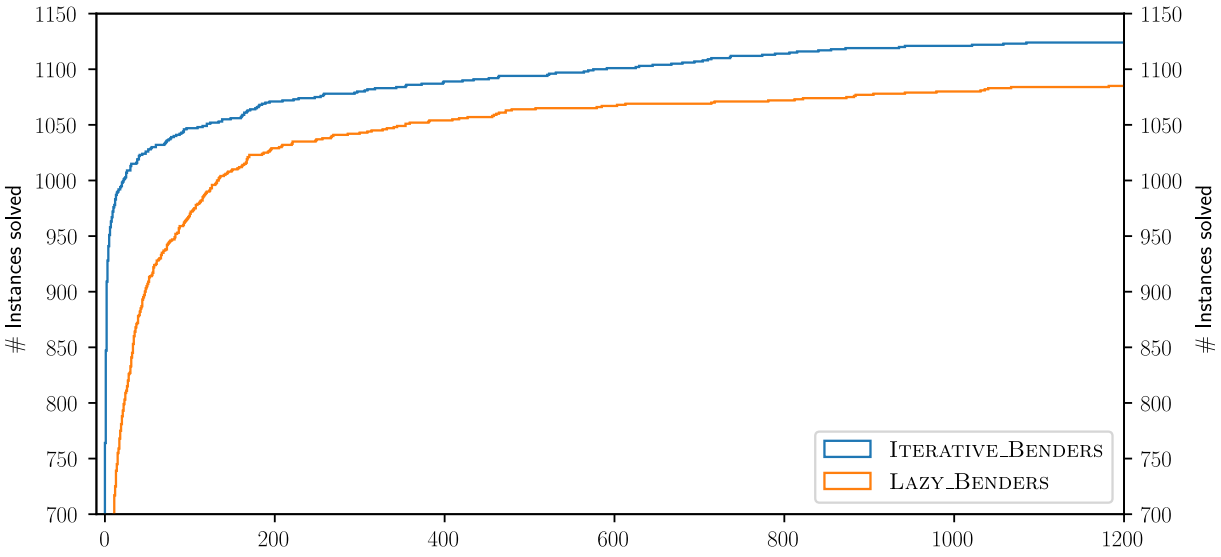


Figure 6.12: Number of instances solved as a function of time for the LAZY_BENDERS and ITERATIVE_BENDERS. For a clearer presentation, we start Instances solved at 700.

Testing on the population benchmark We now show how the ITERATIVE_BENDERS algorithm performs on instances from the population benchmark. These instances are intended to more closely reproduce those arising from the wireless network design application described in the introduction.

As said above, the population benchmark was sampled from the population density data of France whose territory is represented by a 1700 by 1000 rectangular grid, where each cell corresponds to a 1 km by 1 km region and has an associated population density. When a cell is sampled, be it for a disk center or for a point, we assign to it the coordinates of the cell's top right corner. To generate the instances, we set the number of points to 3000, the number of disks to 1000, 5000 or 10000, and the radius to 30, 50 or 100 km. As before, we ensure that all points are covered by at least one disk. A sample instance is shown in Figure 6.13.

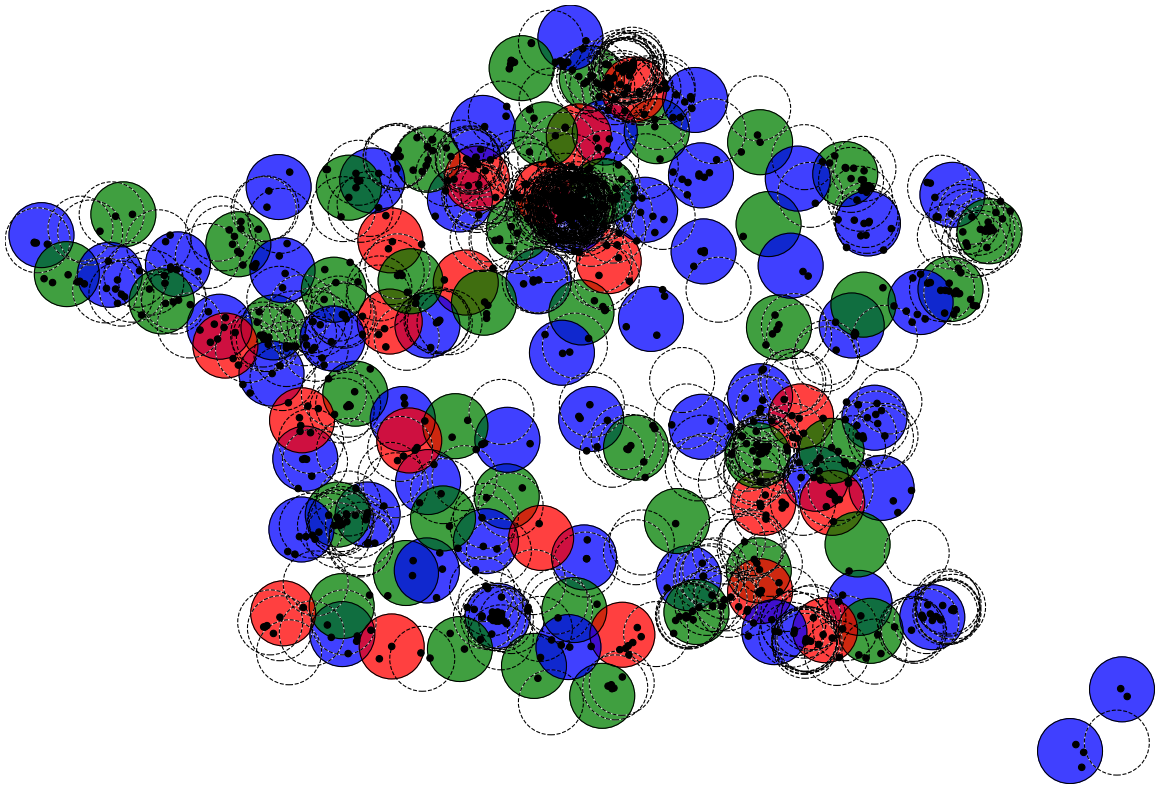


Figure 6.13: Example of an instance sampled from a population density distribution with 1000 points and 250 disks. An optimal solution with 115 disks is also shown, and dashed circles represent disks in the input that are not part of the solution.

Table 6.4 displays detailed information generated when using the ITERATIVE_BENDERS algorithm on our population benchmark. We can see that when either the number of disks available or their radius increase, the number of iterations required by the algorithm drops significantly, approaching one. When there are many large disks to choose from, a solution can consist of disks that are more spread out and fewer in number, while the overlaps are minimized. On the other hand, fewer alternatives for covering each point can lead to more overlaps, requiring many smaller subsets of non 3-colorable disks to be separated, or, ultimately, making the instance infeasible.

Moreover, we see from the results that instances with radius 50 were harder to solve than those with radius 30 and 100. This can be understood by observing that disk size affects the hardness of instances in two ways. Larger disks induce smaller covers, which are easier to solve due to the efficacy of the lifting procedure for odd-wheel constraints in separating small infeasible solutions. On the other hand, smaller disks lead to solutions

with fewer overlaps, which are more likely to be 3-colorable.

Table 6.4: Instances from the population benchmark solved by the ITERATIVE_BENDERS. Instances not solved have been proven to be infeasible.

r	n	m	Sol	Cover Size	Total Time (s)	# Iterations	# Wheels found	Wheel Lift	# Bicon. Comp.
30	3000	1000	10	infeasible	0.65 ± 0.35	3.60 ± 1.78	2.60 ± 1.78	8.78 ± 4.87	0.00
30	3000	5000	10	292.30 ± 3.62	4.03 ± 2.80	3.90 ± 2.85	2.80 ± 2.70	27.38 ± 31.81	151.70 ± 19.74
30	3000	10000	10	273.40 ± 3.37	4.23 ± 1.77	1.70 ± 0.95	0.70 ± 0.95	41.40 ± 54.16	173.40 ± 11.55
50	3000	1000	10	120.70 ± 63.69	1.98 ± 1.18	10.10 ± 5.90	9.10 ± 5.90	26.70 ± 23.04	12.80 ± 8.18
50	3000	5000	10	127.10 ± 1.79	44.53 ± 41.35	1.70 ± 1.25	0.70 ± 1.25	8.82 ± 13.10	29.10 ± 7.16
50	3000	10000	10	120.90 ± 1.37	168.42 ± 91.25	1.10 ± 0.32	0.10 ± 0.32	25.80 ± 81.59	39.10 ± 9.16
100	3000	1000	10	44.20 ± 1.14	2.90 ± 2.91	5.80 ± 6.09	4.30 ± 4.92	20.39 ± 20.76	3.88 ± 1.43
100	3000	5000	10	38.90 ± 0.57	27.35 ± 16.85	1.10 ± 0.32	0.10 ± 0.32	0.80 ± 2.53	4.80 ± 2.04
100	3000	10000	10	37.90 ± 0.32	80.01 ± 88.11	1.10 ± 0.32	0.00	-	4.80 ± 1.03

Analysing the merits of block-cut tree decomposition and odd-wheel separation Recall that the number of iterations is one more than the number of violations found. By comparing columns "# Iterations" and "# Wheels found" in Table 6.4, we see that almost all violations were separated using odd-wheels. We also perceive that, for sparser instances, with smaller radius and larger cover size, the number of biconnected components is higher. Thus, block-cut tree decomposition keeps the size of coloring subproblems, which must be solved exactly using constraint programming, small enough to prevent this step from becoming a bottleneck in solving such instances.

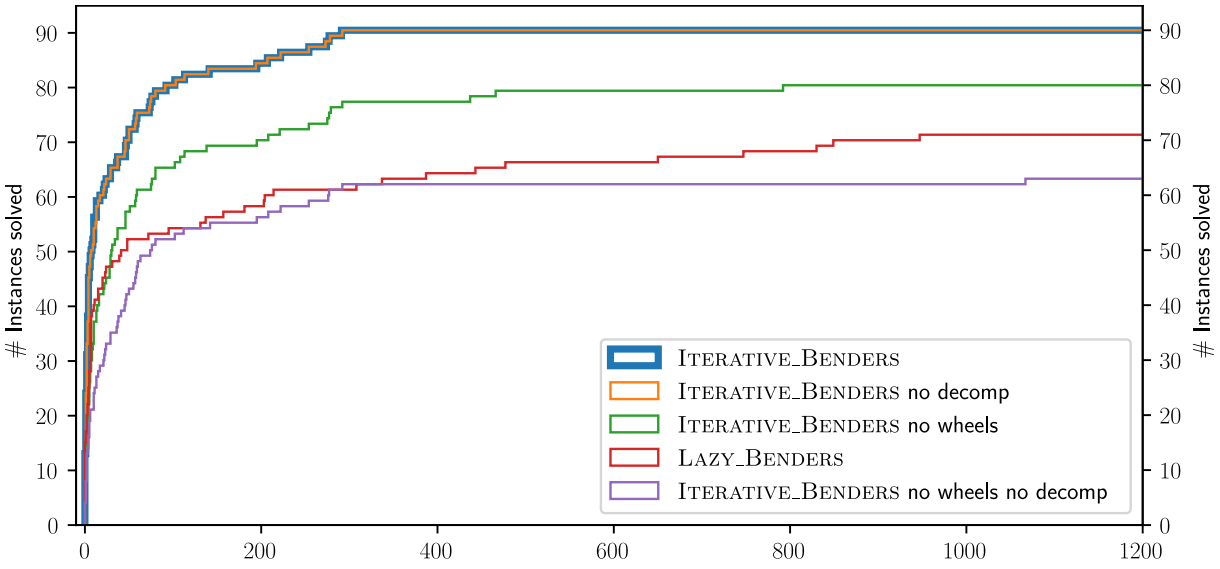


Figure 6.14: Number of instances solved, from the population benchmark, as a function of time for ITERATIVE_BENDERS and LAZY_BENDERS. The exclusion of odd-wheels separation is denoted by *no wheels* and the exclusion of block-cut tree decomposition by *no decomp*.

To measure the impact of both block-cut tree decomposition and odd-wheel separation, we compare the number of instances solved with and without these improvements. The results obtained by the different versions of ITERATIVE_BENDERS for the population benchmark are compiled in Figure 6.14. For comparison purposes, the results obtained by LAZY_BENDERS are also exhibited in the figure. Notice that, without the improvements,

Iterative_Benders is significantly worse than its lazy counterpart, probably due to the much larger number of iterations it takes to converge. The inclusion of block-cut tree decomposition leads to more instances being solved. Also, note that the lazy implementation of the Benders decomposition with all the improvements performs better than a naive iterative implementation. However, adding either the block-cut tree decomposition or the odd-wheel separation to the iterative implementation suffices for it to significantly outperform the lazy implementation.

Another relevant observation is that odd-wheel separation is so effective in boosting the performance of Iterative_Benders, that there is no noticeable difference when it is combined with block-cut tree decomposition. This can be surmised from Figure 6.14 by noticing that the two corresponding curves, denoted by "Iterative_Benders" and "Iterative_Benders no decomp", are almost identical, differing only for small runtimes.

To further demonstrate the benefits of the proposed improvements, we discuss their impact when solving the configuration with 3000 points, 1000 disks and radius 30. In this case, we were not able to generate feasible instances. While Iterative_Benders with all improvements proved infeasibility in less than one second on average, the naive implementation, without the improvements, failed to do so for any of these instances even after 20 minutes. For the ten instances with this particular configuration, the inclusion of block-cut decomposition alone proved enough for solving seven of them, while odd-wheel separation on its own led to *all* instances being solved.

This last analysis highlights the importance of the wheel separation and, to a lesser extent, of the block-cut decomposition for an efficient implementation of the Benders decomposition.

Concluding Remarks

In this paper, we introduced mathematical models for the Minimum 3-Colorable Discrete Unit Disk Cover Problem (M3CDUDC) and devised various algorithmic strategies to compute these models efficiently in practice. Moreover, we provide theoretical insights on their performance. From our findings, we conclude that an extended formulation can be solved by a Benders Decomposition algorithm for large instances with thousands of points and disks, as long as the implementation includes the procedures described herein. This is particularly prominent when compared with the approximation algorithms from the literature that present prohibitively high time complexity [5, 29].

We also present the first empirical study of the problem. Through rigorous experiments, we also gained insight on some factors that influence the hardness of instances. Surprisingly, we observed that while smaller disks lead to larger covers, which increase the hardness of the covering aspect of the problem, they may decrease the overlaps, leading to cover solutions that are easier to color. Likewise, while a larger number of disks in the input makes the covering aspect of the problem harder to solve, the more scattered they are, the more dispersed the cover found will be, which also leads to fewer required colors.

With these results and observations, we lay the groundwork for further investigation of the M3CDUDC problem, while also providing an efficient algorithm that can be used as a

baseline for future comparisons and to find exact solutions that can help the investigation of new heuristics.

Acknowledgments This work was supported in part by grants from: *Brazilian National Council for Scientific and Technological Development* (CNPq) #313329/2020-6, #306454/2018-1; *São Paulo Research Foundation* (FAPESP) #2018/26434-0, #2018/14883-5, #2014/12236-1; *Knut and Alice Wallenberg Foundation* KAW 2021.0307; *ELLIIT Excellence Center*; and *Natural Sciences and Engineering Research Council of Canada* (NSERC) Discovery Grant RGPIN-2020-06054.

Bibliography

[1] P. K. Agarwal, E. Nevo, J. Pach, R. Pinchasi, M. Sharir, and S. Smorodinsky. Lenses in arrangements of pseudo-circles and their applications. *Journal of the ACM*, 51(2):139–186, March 2004.

[2] M. Basappa and S. Mishra. Interference reduction in directional wireless networks. In D. Goswami and T. A. Hoang, editors, *Distributed Computing and Internet Technology - 17th International Conference, ICDCIT 2021, Bhubaneswar, India, January 7-10, 2021, Proceedings*, volume 12582 of *Lecture Notes in Computer Science*, pages 114–126. Springer, 2021.

[3] M. Baysan, K. Sarac, R. Chandrasekaran, and S. Bereg. A polynomial time solution to minimum forwarding set problem in wireless networks under unit disk coverage model. *IEEE Transactions on Parallel and Distributed Systems*, 20(7):913–924, 2009.

[4] D. Bergman and J. N. Hooker. Graph coloring inequalities from all-different systems. *Constraints*, 19(4):404–433, 2014.

[5] T. Biedl, A. Biniaz, and A. Lubiw. Minimum ply covering of points with disks and squares. *Computational Geometry*, 94:101712, 2021.

[6] A. Biniaz, P. Liu, A. Maheshwari, and M. Smid. Approximation algorithms for the unit disk cover problem in 2d and 3d. *Computational Geometry*, 60:8–18, 2017.

[7] P. Brass, F. Hurtado, B. J. Lafreniere, and A. Lubiw. A lower bound on the area of a 3-coloured disk packing. *International Journal of Computational Geometry & Applications*, 20(3):341–360, 2010.

[8] S. S. Brito and H. G. Santos. Preprocessing and cutting planes with conflict graphs. *Computers & Operations Research*, 128:105176, 2021.

[9] B. N. Clark, C. J. Colbourn, and D. S. Johnson. Unit disk graphs. *Discrete Mathemathics*, 86(1-3):165–177, 1990.

[10] C. Contardo and A. Hertz. An exact algorithm for a class of geometric set-cover problems. *Discrete Applied Mathematics*, 300:25–35, 2021.

[11] G. K. Das, R. Fraser, A. López-Ortiz, and B. G. Nickerson. On the discrete unit disk cover problem. *International Journal of Computational Geometry & Applications*, 22(5):407–420, 2012.

[12] P. J. de Rezende, C. C. de Souza, S. Friedrichs, M. Hemmer, A. Kröller, and D. C. Tozoni. Engineering art galleries. In L. Kliemann and P. Sanders, editors, *Algorithm Engineering - Selected Results and Surveys*, volume 9220 of *Lecture Notes in Computer Science*, pages 379–417. Springer, 2016.

[13] E. D. Demaine, S. P. Fekete, P. Keldenich, D. Krupke, and J. S. B. Mitchell. Computing convex partitions for point sets in the plane: The CG: SHOP challenge 2020. *CoRR*, abs/2004.04207, 2020.

[14] S. P. Fekete and P. Keldenich. Conflict-free coloring of intersection graphs. *International Journal of Computational Geometry & Applications*, 28(3):289–307, 2018.

[15] R. Friederich, M. Graham, A. Ghosh, B. Hicks, and R. Shevchenko. Experiments with unit disk cover algorithms for covering massive pointsets. *CoRR*, abs/2205.01716, 2022.

[16] Z. Gáspár and T. Tarnai. Upper bound of density for packing of equal circles in special domains in the plane. *Periodica Polytechnica Civil Engineering*, 44(1):13–32, 2000.

[17] A. Ghosh, B. Hicks, and R. Shevchenko. Unit disk cover for massive point sets. In I. S. Kotsireas, P. M. Pardalos, K. E. Parsopoulos, D. Souravlias, and A. Tsokas, editors, *Analysis of Experimental Algorithms - Special Event, SEA 2019, Kalamata, Greece, June 24-29, 2019, Revised Selected Papers*, volume 11544 of *Lecture Notes in Computer Science*, pages 142–157. Springer, 2019.

[18] A. Gräf, M. Stumpf, and G. Weißenfels. On coloring unit disk graphs. *Algorithmica*, 20(3):277–293, 1998.

[19] W. Hale. Frequency assignment: Theory and applications. *Proceedings of the IEEE*, 68(12):1497–1514, 1980.

[20] D. S. Hochbaum. Why should biconnected components be identified first. *Discrete Applied Mathematics*, 42(2):203–210, 1993.

[21] J. N. Hooker and G. Ottosson. Logic-based Benders decomposition. *Mathematical Programming*, 96:33–60, 2003.

[22] IBM Corporation. Differences between user cuts and lazy constraints. `https://www.ibm.com/docs/en/icos/20.1.0?topic=pools-differences-between-user-cuts-lazy-constraints`. Accessed: 06-21-2022.

[23] A. E. Kiouche, M. Bessedik, F. B. Tayeb, and M. R. Keddar. An efficient hybrid multi-objective memetic algorithm for the frequency assignment problem. *Engineering Applications of Artificial Intelligence*, 87, 2020.

[24] F. Luna, C. Estébanez, C. León, J. M. Chaves-González, E. Alba, R. Aler, C. Segura, M. A. Vega-Rodríguez, A. J. Nebro, J. M. Valls, G. Miranda, and J. A. G. Pulido. Metaheuristics for solving a real-world frequency assignment problem in GSM networks. In C. Ryan and M. Keijzer, editors, *Genetic and Evolutionary Computation Conference, GECCO 2008, Proceedings, Atlanta, GA, USA, July 12-16, 2008*, pages 1579–1586. ACM, 2008.

[25] M. López-Ibáñez, J. Dubois-Lacoste, L. Pérez Cáceres, T. Stützle, and M. Birattari. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58, 2016.

[26] S. J. Maher. Implementing the branch-and-cut approach for a general purpose Benders' decomposition framework. *European Journal of Operational Research*, 290(2):479–498, 2021.

[27] C. Mears, M. Garcia De La Banda, B. Demoen, and M. Wallace. Lightweight dynamic symmetry breaking. *Constraints*, 19(3):195–242, 2014.

[28] S. Rebennack. Stable set problem: branch & cut algorithms. In C. A. Floudas and P. M. Pardalos, editors, *Encyclopedia of Optimization*, pages 3676–3688. Springer, Boston, MA, 2009.

[29] M. S. Reyunuru, K. Jethlia, and M. Basappa. The k-colorable unit disk cover problem. *Computing Research Repository (CoRR)*, abs/2104.00207, 2021.

[30] F. Rossi, P. van Beek, and T. Walsh, editors. *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*. Elsevier, 2006.

[31] A. Sapucaia, A. A. Cire, P. J. de Rezende, S. F. de Rezende, and C. C. de Souza. Minimum 3-Colorable Discrete Unit Disk Cover – Benchmark Instances and Solutions, 2022. `www.ic.unicamp.br/~cid/Problem-instances/Colorable-Unit-Disk-Cover`.

[32] A. Sapucaia, P. J. de Rezende, and C. C. de Souza. Solving the minimum convex partition of point sets with integer programming. *Computational Geometry*, 99:101794, 2021.

[33] The CGAL Project. *CGAL User and Reference Manual*. CGAL Editorial Board, 5.4.1 edition, 2022.

[34] L. F. Tóth. *Lagerungen in der Ebene auf der Kugel und im raum*. Springer, Berlin, 1953.

[35] L. Wolsey. *Integer Programming*. John Wiley & Sons, Ltd, 2nd edition, 2020.

[36] World Pop. Population Density. `https://hub.worldpop.org/geodata/summary?id=41221`. Accessed: 06-30-2022.

Chapter 7

Conclusions

In this dissertation, we presented the first ILP formulations for several geometric decomposition problems. The results obtained for these problems are likely to generalize to other similar problems, laying the groundwork for studying other problems of a similar nature.

For the MCPP, in Chapter 3, we presented a smaller set of constraints for convex hull partitioning problems together with a data structure for computing the sum of dual variables very efficiently. This data structure allowed us to solve the pricing problem in polynomial time and avoided the need to completely enumerate arrangements. We also showed how a novel geometric interpretation of the Ryan Foster branching rule significantly reduces solving time, while fitting the pricing scheme without introducing new complicating dual variables. The combination of these results sets up a framework for solving convex hull partition problems that can easily be adapted, as we did for two of the other three problems studied. These results were only possible due to the inclusion of many features of the first model for the problem, presented in Chapter 2, including its heuristics.

Regarding the CQBPS, in Chapter 4, we showed how the techniques developed for the MCPP generalize well for convex hull partitioning problems, even when the comparatively small number of variables does not require column generation. We showed that the coloring aspect of the problem could be reinterpreted as set-partitioning, leading to a pure set-partitioning ILP and significantly decreasing solving time. Based on geometric and graph-theoretical heuristics from the literature, we also showed how novel heuristics could be proposed by solving the ILP restricted to a very small number of selected polygons.

The Coarseness problem, presented in Chapter 5, was the first problem that we studied that is not a variant of convex hull partitioning. Even in this case, we showed that some of the core ideas from the other problems still applied, such as the combination of arrangement-based and graph-based components. We also managed to adapt the pricing scheme used for the MCPP, even though it required a novel approach for the fast computation of sums of dual variables.

Lastly, in the article that comprises Chapter 6, we presented our results for the M3CDUDC. This problem is the only one we studied that has disks as the primary geometric objects, as well as the only one where the objects being chosen are explicitly given as part of the input. Once again, avoiding the enumeration of an arrangement was

crucial for the performance of the model. This was done using row generation, not only by simply employing Lazy Cuts, but also by applying Logic-Based Benders Decomposition. The decomposition worked very well indeed, enabling us to separate the covering and the coloring aspects of the problem. However, as the coloring subproblem was an ILP, it required problem-specific cuts to be introduced. The final algorithm significantly outperformed the naive Lazy Cuts implementation, after several improvements were introduced. We also showed how the solution to the first iteration of this algorithm used at most 18 times the minimum number of colors used by all possible covers using disks in D.

Additionally, for each of the problems studied, we were first to make publicly available all instances and solutions, together with solving information used to compare performance. This clearly allows for the comparison against future exact algorithms, but also helps to better understand the practical quality of solutions found by approximation algorithm and heuristics.

Notably, throughout this work, we observed how well both row and column generation work for solving geometric variants of decomposition problems. This is mostly a consequence of how objects are implicitly given through geometric constructions.

Also, the master problem preserves most of the structure of the general decomposition problems and can benefit from their vast literature. Besides, the subproblem can leverage the geometry as a self-contained problem that can either be solved in polynomial time or very efficiently in practice.

This approach worked very well for the classes of problem studied here and are likely to be a good contender as a first approach when solving other hard geometric problems.

However, we also showed that these ideas can only achieve their full potential after careful implementation, while being very sensible to problem specific improvements.

In particular, our formulations demonstrated how well arrangements work when modeling geometric decomposition problems. As a tool for discretization of the space, it allows the expression of constraints very naturally. However, the study of which faces are indeed necessary for a given problem is a good research direction to improve performance.

For future work, all problems studied could benefit from adapting recent advances from the literature, specially the ones that improve either column or row generation. This includes modifying the pricing problems [14], or the lazy cut generation for Benders Decomposition [12], apart from using fast numerical methods for solving the first relaxations [3]. Alternatively, studying better choices of initial constraints/variables and the inclusion of new classes of constraints are proven directions to speed up both row and column generation implementations.

Bibliography

[1] E. D. Andersen. How to use farkas' lemma to say something important about infeasible linear problems. Technical report, Mosek ApS, 2011.

[2] D. L. Applegate, R. E. Bixby, V. Chvatál, and W. J. Cook. *The Traveling Salesman Problem: A Computational Study*. Princeton University Press, 2006.

[3] D. L. Applegate, M. Díaz, O. Hinder, H. Lu, M. Lubin, B. O'Donoghue, and W. Schudy. Practical large-scale linear programming using primal-dual hybrid gradient. In M. Ranzato, A. Beygelzimer, Y. N. Dauphin, P. Liang, and J. W. Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 20243–20257, 2021.

[4] C. Barnhart, E. L. Johnson, G. L. Nemhauser, M. W. P. Savelsbergh, and P. H. Vance. Branch-and-price: Column generation for solving huge integer programs. *Operations Research*, 46(3):316–329, 1998.

[5] B. M. Chazelle. *Computational Geometry and Convexity*. PhD thesis, Yale University, USA, 1980. AAI8110021.

[6] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer, 3rd edition, 2008.

[7] E. D. Demaine, S. P. Fekete, P. Keldenich, D. Krupke, and J. S. B. Mitchell. Computing convex partitions for point sets in the plane: The CG: SHOP challenge 2020. *CoRR*, abs/2004.04207, 2020.

[8] Z. Drezner and Z. Drezner. *Facility Location: A Survey of Applications and Methods*. Springer Series in Operations Research and Financial Engineering. Springer, 1995.

[9] T. A. Granberg, T. Polishchuk, V. Polishchuk, and C. Schmidt. Integer programming–based airspace sectorization for terminal maneuvering areas with convex sectors. *Journal of Air Transportation*, 27(4):169–180, 2019.

[10] D. Halperin and M. Sharir. Arrangements. In J. Goodman, J. O'Rourke, and C. Tóth, editors, *Handbook of Discrete and Computational Geometry*, chapter 28. CRC Press LLC, 3rd edition, 2017.

[11] J. N. Hooker and G. Ottosson. Logic-based Benders decomposition. *Mathematical Programming*, 96:33–60, 2003.

[12] E. Karlsson and E. Rönnberg. Strengthening of feasibility cuts in logic-based benders decomposition. In P. J. Stuckey, editor, *Integration of Constraint Programming, Artificial Intelligence, and Operations Research - 18th International Conference, CPAIOR 2021, Vienna, Austria, July 5-8, 2021, Proceedings*, volume 12735 of *Lecture Notes in Computer Science*, pages 45–61. Springer, 2021.

[13] M. Akella, S. Gupta and A. Sarkar. Branch and Price: Column generation for solving huge integer programs, 2021. `https://web.archive.org/web/20100821132913/http://www.acsu.buffalo.edu/~nagi/courses/684/price.pdf`, [Online; accessed 18-May-2021].

[14] S. J. Maher and E. Rönnberg. Integer programming column generation: Accelerating branch-and-price using a novel pricing scheme for finding high-quality solutions in set covering, packing, and partitioning problems, 2021. Preprint.

[15] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. Wiley Interscience Series in Discrete Mathematics and Optimization. Wiley, 1988.

[16] L. Palios. *Decomposition Problems in Computational Geometry*. Doctoral thesis, Princeton University, USA, 1992. UMI Order No. GAX92-30247.

[17] A. Sapucaia, A. A. Ciré, P. J. de Rezende, and C. C. de Souza. Convex bichromatic quadrangulation of point sets with minimum color flips. In M. He and D. Sheehy, editors, *Proceedings of the 33rd Canadian Conference on Computational Geometry, CCCG 2021, August 10-12, 2021, Dalhousie University, Halifax, Nova Scotia, Canada*, pages 185–194, 2021.

[18] A. Sapucaia, P. J. de Rezende, and C. C. de Souza. Solving the coarseness problem by ILP using column generation. In O. Gervasi, B. Murgante, S. Misra, C. Garau, I. Blecic, D. Taniar, B. O. Apduhan, A. M. A. C. Rocha, E. Tarantino, and C. M. Torre, editors, *Computational Science and Its Applications - ICCSA 2021 - 21st International Conference, Cagliari, Italy, September 13-16, 2021, Proceedings, Part V*, volume 12953 of *Lecture Notes in Computer Science*, pages 30–45. Springer, 2021.

[19] A. Sapucaia, P. J. de Rezende, and C. C. de Souza. Solving the minimum convex partition of point sets with integer programming. *Computational Geometry*, 99:101794, 2021.

[20] A. Sapucaia-Barboza, C. C. de Souza, and P. J. de Rezende. Minimum convex partition of point sets. In P. Heggernes, editor, *Algorithms and Complexity - 11th International Conference, CIAC 2019, Rome, Italy, May 27-29, 2019, Proceedings*, volume 11485 of *Lecture Notes in Computer Science*, pages 25–37. Springer, 2019.

[21] The CGAL Project. *CGAL User and Reference Manual*. CGAL Editorial Board, 5.4.1 edition, 2022.

[22] R. R. Vemuganti. Applications of set covering, set packing and set partitioning models: A survey. In D.-Z. Du and P. M. Pardalos, editors, *Handbook of Combinatorial Optimization: Volume1–3*, pages 573–746, Boston, MA, 1998. Springer.

[23] L. Wolsey. *Integer Programming.* John Wiley & Sons, Inc., 1998.

[24] L. Wolsey. *Integer Programming.* John Wiley & Sons, Ltd, 2nd edition, 2020.