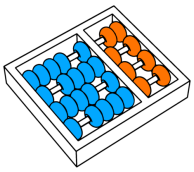
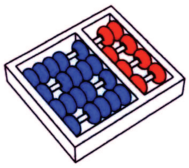


Davi Colli Tozoni

"Solving the Art Gallery Problem: A Practical and Robust Method for Optimal Point Guard Positioning"

"*Resolução do Problema da Galeria de Arte: Um Método Prático e Robusto para o Posicionamento Ótimo de Guardas-Ponto*"

CAMPINAS
2014



University of Campinas
Institute of Computing

Universidade Estadual de Campinas
Instituto de Computação

Davi Colli Tozoni

"Solving the Art Gallery Problem: A Practical and Robust Method for Optimal Point Guard Positioning"

Supervisor(s)/Orientador(es)
Prof. Dr. Cid Carvalho de Souza
Prof. Dr. Pedro Jussieu de Rezende

"Resolução do Problema da Galeria de Arte: Um Método Prático e Robusto para o Posicionamento Ótimo de Guardas-Ponto"

MSc Dissertation presented to the Post Graduate Program of the Institute of Computing of the University of Campinas to obtain a Master's degree in Computer Science.

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Computação da Universidade Estadual de Campinas para obtenção do título de Mestre em Ciência da Computação.

This volume corresponds to the final version of the Dissertation defended by Davi Colli Tozoni, under the supervision of Prof. Dr. Cid Carvalho de Souza.

Este exemplar corresponde à versão final da Dissertação defendida por Davi Colli Tozoni, sob orientação de Prof. Dr. Cid Carvalho de Souza.

Supervisor's signature

Assinatura do Orientador(a)

CAMPINAS
2014

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

T669s

Tozoni, Davi Colli, 1988-
Solving the art gallery problem : a practical and robust method for optimal point guard positioning / Davi Colli Tozoni. – Campinas, SP : [s.n.], 2014.

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

1. Geometria computacional. 2. Otimização combinatória. 3. Programação inteira. 4. Algoritmos. I. Souza, Cid Carvalho de,1963-. II. Rezende, Pedro Jussieu de,1955-. III. Universidade Estadual de Campinas. Instituto de Computação. IV. Título.

Informações para Biblioteca Digital

Título em outro idioma: Resolução do problema da galeria de arte : um método prático e robusto para o posicionamento ótimo de guardas-ponto
Palavras-chave em inglês:
Computational geometry
Combinatorial optimization
Integer programming
Algorithms
Área de concentração: Ciência da Computação
Titulação: Mestre em Ciência da Computação
Banca examinadora:
Cid Carvalho de Souza [Orientador]
Luiz Henrique de Figueiredo
Fábio Luiz Usberti
Data de defesa: 25-07-2014
Programa de Pós-Graduação: Ciência da Computação

TERMO DE APROVAÇÃO

Defesa de Dissertação de Mestrado em Ciência da Computação, apresentada pelo(a) Mestrando(a) **Davi Colli Tozoni**, aprovado(a) em **25 de julho de 2014**, pela Banca examinadora composta pelos Professores Doutores:

Prof(a). Dr(a). Luiz Henrique de Figueiredo
Titular

Prof(a). Dr(a). Fábio Luiz Usberti
Titular

Prof(a). Dr(a). Cid Carvalho de Souza
Presidente

Institute of Computing /*Instituto de Computação*
University of Campinas /*Universidade Estadual de Campinas*

Solving the Art Gallery Problem: A Practical and Robust Method for Optimal Point Guard Positioning

Davi Colli Tozoni[1]

July 25, 2014

Examiner Board/*Banca Examinadora*:

- Prof. Dr. Cid Carvalho de Souza (Supervisor/*Orientador*)
- Prof. Dr. Pedro Jussieu de Rezende (Co-Supervisor/*Co-Orientador*)
- Prof. Dr. Luiz Henrique de Figueiredo
  Instituto Nacional de Matemática Pura e Aplicada - IMPA
- Prof. Dr. Fábio Luiz Usberti
  Instituto de Computação - UNICAMP
- Prof. Dr. Carlos Eduardo Ferreira
  Instituto de Matemática e Estatística - USP (Substitute/*Suplente*)
- Prof. Dr. Orlando Lee
  Instituto de Computação - UNICAMP (Substitute/*Suplente*)

[1]Financial support: CNPq scholarship (process 146025/2012-1) 2012 and FAPESP scholarship (processes 2012/18384-7 and 2013/13534-3) 2013–2014

Abstract

In this dissertation, we present our research on the Art Gallery Problem (AGP), one of the most investigated problems in Computational Geometry. The AGP, which is a known $\mathbb{NP}$-hard problem, consists in finding the minimum number of guards sufficient to ensure the visibility coverage of an art gallery represented as a polygon. In the version of the problem treated in this work, usually called Art Gallery Problem with Point Guards, the guards can be placed anywhere in the polygon and the objective is to cover the whole region, which may or not have holes.

We studied how to apply Computational Geometry concepts and algorithms as well as Integer Programming techniques in order to solve the AGP to optimality. This work culminated in the creation of a new algorithm for the AGP, whose idea is to iteratively generate upper and lower bounds for the problem through the resolution of discretized versions of the AGP, which are reduced to instances of the Set Cover Problem.

The algorithm was implemented and tested on more than 2800 instances of different sizes and classes of polygons. The technique was able to solve in minutes more than 90% of all instances considered, including polygons with thousands of vertices, greatly increasing the set of instances for which exact solutions are known. To the best of our knowledge, in spite of the extensive study of the AGP in the last four decades, no other algorithm has shown the ability to solve the AGP as effectively as the one described here.

Resumo

Nesta dissertação, apresentamos nossa pesquisa sobre o Problema da Galeria de Arte (AGP), um dos problemas mais estudados em Geometria Computacional. O AGP, que é um problema $\mathbb{NP}$-difícil, consiste em encontrar o número mínimo de guardas suficiente para garantir a cobertura visual de uma galeria de arte representada por um polígono. Na versão do problema tratada neste trabalho, usualmente chamada de Problema da Galeria de Arte com Guardas-Ponto, os guardas podem ser posicionados em qualquer lugar do polígono e o objetivo é cobrir toda a região, que pode ou não conter buracos.

Nós estudamos como aplicar conceitos e algoritmos de Geometria Computacional, bem como Técnicas de Programação Inteira, com a finalidade de resolver o AGP de forma exata. Este trabalho culminou na criação de um novo algoritmo para o AGP, cuja ideia é gerar, de forma iterativa, limitantes superiores e inferiores para o problema através da resolução de versões discretizadas do AGP, que são reduzidas a instâncias do Problema de Cobertura de Conjuntos.

O algoritmo foi implementado e testado em mais de 2800 instâncias, de diferentes tamanhos e classes. A técnica foi capaz de resolver, em minutos, mais de 90% de todas as instâncias consideradas, incluindo polígonos com milhares de vértices, e ampliou em muito o conjunto de casos para os quais são conhecidas soluções exatas. Até onde sabemos, apesar do extensivo estudo do AGP nas últimas quatro décadas, nenhum outro algoritmo demonstrou a capacidade de resolver o AGP de forma tão eficaz como a técnica aqui descrita.

Acknowledgements

Firstly, I want to thank my advisors, Professors Cid and Rezende, for all the help during my Master's Degree. Their hard work and commitment to research and teaching will certainly be an example for all my life. Thank you for the weekly meetings, the conversations and for this great learning opportunity.

A special thanks also goes to my parents, who always supported me in my studies, taught me the importance of committing myself to the maximum in each and every activity and always cared about giving the best conditions so that I could achieve my goals.

I also want to thank all other professors of the Institute of Computing at UNICAMP, who taught me a lot during undergraduate and graduate school. In particular, I thank Professor Arnaldo, my Scientific Initiation advisor, for introducing me to research. Moreover, I thank Professor Christiane that, in addition to supervising my teaching internship (PED), made me like the teaching profession even more and became a great friend and motivator.

Furthermore, I would like to thank the research group at TUBS for earnestly welcoming me during my research internship in Germany and for the partnership developed during this project. I thank Professor Alexander Kröller, my internship supervisor, Professor Sándor Fekete, head of the Algorithms Group, my friends Michael Hemmer and Stephan Friedrichs and the entire staff of the Algorithms Group at TUBS.

I couldn't forget to say thank you to all my colleagues from LOCo (Laboratory of Optimization and Combinatorics) that were by my side during much of this work. The coffee breaks, conversations and moments of laugh were surely very important. In addition, I thank my family, all my friends and especially my girlfriend Thamiris for their support.

Finally, I thank the University of Campinas for being my second home since I was 17 years old and the research funding agencies FAPESP and CNPq for the financial support of my research.

Agradecimentos

Primeiramente, gostaria de agradecer aos meus orientadores, Professores Cid e Rezende, por toda ajuda durante o período de mestrado. O trabalho sério e o comprometimento deles com a pesquisa e com o ensino com certeza me servirão de exemplo por toda a vida. Obrigado pelas reuniões semanais, pelas conversas e por esta grande oportunidade de aprendizado.

Um agradecimento especial vai também para os meus pais, que sempre me apoiaram muito nos estudos, me ensinaram a importância de se dedicar ao máximo em toda e qualquer atividade e sempre se preocuparam em dar as melhores condições para que eu conseguisse atingir os meus objetivos.

Agradeço também a todos os demais professores do Instituto de Computação da Unicamp, que muito me ensinaram durante a graduação e a pós graduação. Em especial, agradeço ao Professor Arnaldo, orientador de minha Iniciação Científica, por ter me introduzido ao universo da pesquisa e agradeço também à Professora Christiane que, além de supervisionar o meu estágio docente (PED), me fez gostar ainda mais do trabalho de um professor e se tornou uma grande amiga e incentivadora.

Gostaria de agradecer ao grupo de pesquisa da TUBS pela ótima recepção que tive durante o meu estágio de pesquisa na Alemanha e pela parceria desenvolvida neste projeto. Agradeço ao Professor Alexander Kröller, supervisor de meu estágio, ao Professor Sándor Fekete, chefe do grupo de algoritmos, aos amigos Michael Hemmer e Stephan Friedrichs e também aos demais funcionários do grupo de algoritmos da TUBS.

Não poderia esquecer de dizer obrigado a todos os colegas do LOCo (Laboratório de Otimização Combinatória) que estiveram ao meu lado durante grande parte deste trabalho. Os cafés, as conversas e as risadas com certeza foram muito importantes. Além disso, agradeço aos meus familiares, aos colegas do curso de Computação na Unicamp, demais amigos e principalmente à minha namorada Thamiris por todo o apoio.

Finalmente, agradeço à Unicamp por ter se tornado a minha segunda casa desde os 17 anos de idade e às entidades de fomento FAPESP e CNPq pelo suporte financeiro de meus estudos.

Contents

Abstract **ix**

Resumo **xi**

Acknowledgements **xiii**

Agradecimentos **xv**

1 Introduction **1**
- 1.1 The Problem . . . 2
- 1.2 Related Work . . . 3
- 1.3 Our Contribution . . . 7
- 1.4 Text Organization . . . 8

2 Preliminaries **9**
- 2.1 Computational Geometry . . . 9
- 2.2 Integer Programming . . . 14
- 2.3 The Art Gallery Problem . . . 16
- 2.4 Basic Theorems . . . 18

3 An Algorithm for the AGP **21**
- 3.1 Sketch of the Algorithm . . . 21
- 3.2 Solving the AGPW . . . 22
- 3.3 Solving the AGPFC . . . 23
- 3.4 Solving the AGPWFC (SCP) . . . 25
- 3.5 Witness Management . . . 28
- 3.6 Guard Candidate Management . . . 32
- 3.7 Resulting Algorithm . . . 34

4 Implementation and Computational Results **39**
4.1 Implementation . . . 39
4.1.1 I1: Implementation for the first paper [34] . . . 40
4.1.2 I2: Implementation for the second paper [33] . . . 41
4.1.3 I3: Implementation for the third paper [17] . . . 43
4.2 Computational Environment . . . 44
4.3 Instances . . . 44
4.3.1 Simple . . . 45
4.3.2 Orthogonal . . . 45
4.3.3 Simple-simple . . . 45
4.3.4 Ortho-ortho . . . 46
4.3.5 von Koch . . . 46
4.3.6 Spike . . . 47
4.4 Results and Analysis . . . 47
4.4.1 General Results . . . 50
4.4.2 Witness Management . . . 56
4.4.3 Guard Candidate Management . . . 62
4.4.4 SCP Resolution . . . 62
4.5 Comparison With Other Techniques . . . 67
4.5.1 Comparison with Bottino et al.'s technique [10] . . . 67
4.5.2 Comparison with Kröller et al.'s technique [26] . . . 69

5 Conclusions **75**

Bibliography **77**

List of Tables

4.1 Summary of instances 49
4.2 Optimality Rates of I1, I2 and I3 in environment M2. 51
4.3 Average Time of I1, I2 and I3 in environment M2. 52
4.4 Optimality rate, average cardinality of optimal guard sets, average number of iterations and average time spent results with E(I3,M2). 53
4.5 Percentage of executions with I3 in M2 within different optimality gaps. . . 57
4.6 Average number of iterations (main loop) until an optimal solution is found for each initial discretization strategy with E(I1,M1). 58
4.7 Average running time until an optimal solution is found for each initial discretization strategy with E(I1,M1). 59
4.8 Optimality rate and average time results for CV and CP discretization strategies with $E(I3, M2)$. 61
4.9 Optimality rate and average time results for CG and BG guard candidate selection strategies with $E(I3, M2)$. 63
4.10 Optimality rate and average time results when using or not the Lagrangian Heuristic to help CPLEX with $E(I3, M2)$. 68
4.11 Comparison between the method of Bottino et al. [10] and I2. 69
4.12 Optimality Rate of our implementations and AGP Solvers from [26] in environment M2. 71

List of Figures

1.1 Satellite photo of UNICAMP. 3
1.2 Representation of UNICAMP Campus (top); an optimal guard positioning (bottom). 4
1.3 Timeline including the major recent algorithmic achievements for the AGP. 5

2.1 Simple polygons (left); non-simple polygons (right). 10
2.2 Polygons with holes. 10
2.3 Convex (purple) and reflex (blue) vertices. 11
2.4 Two points visible (green) and two other that are not visible (red) to each other (left); the visibility polygon of a point (right). 11
2.5 The arrangement induced by a finite set S of points (left); the set S and its covered (light green) and uncovered (white) regions (right). 12
2.6 The arrangement induced by S with the *light* (blue) (left); and *shadow* (red) AVPs (right). 13
2.7 A point (left) and an edge (right) (both represented in magenta) which are, at the same time, light and shadow AVPs. 13
2.8 An illustration of the four different variants of the Art Gallery Problem: (a) AGP; (b) AGPFC; (c) AGPW; (d) AGPWFC. 17

3.1 **Algorithm 2:** (a) AGPW(D); (b) The light AVPs of Arr(D); (c) The guard candidate set; (d) An optimal solution $G \subseteq V_{\mathcal{L}}(D)$ for AGPW(D). . 24
3.2 **Algorithm 3:** A sequence of AGPWFC(D,C) instances is generated until a viable solution for the AGP is obtained. 26
3.3 Examples of AGPWFC instances where the column reduction procedure was applied, showing the removed guard candidates (gray) and the remaining ones (blue). 28
3.4 Examples of AGPWFC instances where the row reduction procedure was applied, showing the removed witnesses (gray) and the remaining ones (red). 28

3.5 Examples of the initial set D when using each one of the following discretization strategies: (a) All-Vertices (AV); (b) Convex-Vertices (CV); (c) Chwa-Points (CP); (d) Chwa-Extended (CE). 30
3.6 Solution of an AGPWFC instance (left); New witnesses chosen (violet) for the next iteration of the AGPFC algorithm (right). 31
3.7 Solution of an AGPW instance (left); New witnesses chosen (violet) for the next iteration of the AGP algorithm (right). 32
3.8 Examples of the guard candidate set C when using each of the following discretizations: Boundary-Guards (BG) (left); Center-Guards (CG) (right); 33
3.9 AGPW instance where it is not possible to find an optimal solution using only interior points of faces that are light AVPs as guard candidates. . . . 33
3.10 **Solving the AGPW (lower bound):** (a) An orthogonal polygon; (b) the initial witness set D (Convex-Vertices); (c) the arrangement Arr(D) and the light AVPs; (d) the guard candidate set C; (e) witnesses and guard candidates forming an instance of AGPWFC(D,C); (f) the solution to AGPW(D). 36
3.11 **Solving the AGPFC (upper bound):** Iterations of AGPFC(C) resolution. (a,c,e) Updated witness set D_f; (b,d,f) the optimal solution to AGPWFC($D_f, V_{\mathcal{L}}(D) \cup V$); (f) a viable solution to the AGP. 37
3.12 **Solving the AGPW (lower bound):** (a) The new witness set D; (b) the arrangement Arr(D) and the light AVPs; (c) the guard candidate set C; (d) witnesses and guard candidates forming an instance of AGPWFC(D,C); (e) the solution to AGPW(D); (f) region not covered by current AGPW solution. 38

4.1 Example of the fraction simplification procedure where an initial witness (red) placed in the interior of an uncovered region is turned into another with a much smaller representation (violet). 41
4.2 Levels of edges based on the modified von Koch curve. 47
4.3 Examples of instances from different classes. (a) Simple; (b) Orthogonal; (c) Simple-simple; (d) Ortho-ortho; (e) von Koch; (f) Spike. 48
4.4 Average time spent in each of the major tasks of the technique when solving simple polygons of 1000 vertices. 54
4.5 Run time information of E(I3,M2) for polygons of 1000 vertices. 55
4.6 Percentage of time used in ILP resolution with each of our three implementations in M2, when solving simple polygons with 1000 vertices. 64

4.7 Comparison of average run time with $E(I2, M1)$ when using Lagrangian Heuristic or only XPRESS on simple (top) and orthogonal (bottom) polygons. 65
4.8 Comparison of average run time with $E(I2, M1)$ when using Lagrangian Heuristic or only GLPK on simple (top) and orthogonal (bottom) polygons. 66
4.9 Performance comparison between I3 and BS3 in M2 when solving the following classes: (a) Simple; (b) Orthogonal; (c) Simple-simple; (d) Ortho-ortho; (e) von Koch; (f) Spike. Here, only fully solved instances are considered. 72

Chapter 1

Introduction

Computational Geometry (CG) is the field of Computer Science and Discrete Mathematics dedicated to study the computational aspects of geometric problems, which involves, among other things, the design and analysis of algorithms. Many of the problems treated in CG are also optimization problems and, for some of these, there are known polynomial algorithms.

On the other hand, there are many CG problems that are in the $\mathbb{NP}$-hard class. Traditionally, researchers have dealt with geometric $\mathbb{NP}$-hard problems mainly using approximation algorithms. This type of approach remains widely employed to this day, as seen in recently published works by Mitchell [29] on Watchman Routes and Bartal and Gottlieb [4] on the Euclidean Traveling Salesman Problem (ETSP). Moreover, when the objective is to solve problems in a practical manner, heuristics are also often applied. As an example, a recent work by Laahover proposes a randomized Delaunay triangulation heuristic for the Euclidean Steiner Tree Problem [27], one of the most studied problems in the field.

Finally, it is also possible, although much less frequently, to find in the literature exact methods for $\mathbb{NP}$-hard optimization problems arising in CG. The lack of studies on exact approaches leave unanswered the question of how difficult it is to actually solve these problems (to optimality) in practice. The employment of exact techniques in other areas, such as in Operations Research (OR), has already achieved proven success, which can be seen in the resolution of well known OR problems, such as the Traveling Salesman Problem (TSP) and the Vehicle Routing Problem (VRP). In these cases, experiments showed the possibility of solving instances of very large size in a reasonable amount of time (see [3] and [20]). To achieve these outcomes, the use of Integer Linear Programming (ILP) techniques was of great importance. The performance breakthrough obtained by ILP solvers in recent years has enabled effective and efficient solutions to $\mathbb{NP}$-hard problems.

1.1 The Problem

In 1973, Victor Klee proposed a new geometric puzzle, which basically consisted in answering how many guards would be sufficient to ensure that an art gallery (represented by a polygon) be fully guarded, assuming that a guard's field of view covers 360 degrees as well as an unbounded distance limited only by the walls of the gallery. This question became known as the Art Gallery Problem (AGP) and, in the course of time, turned into one of the most investigated problems in computational geometry. The breadth of this research can be perceived from the many important works that appeared, including O'Rourke's classical book [30], a recent text by Ghosh on visibility algorithms [22] and surveys by Shermer in 1992 [31] and Urrutia in 2000 [35].

It is easy to see that by varying the concept of visibility, changing what constitutes a coverage or even by pre-defining a discrete set of guard candidates, we are led to a number of variations of the AGP. In the original problem, often called the *Art Gallery Problem with Point Guards*, the objective is to completely cover the interior of the input gallery using guards that may be positioned anywhere in the polygon, which is why they are also referred to as *point guards*. Meanwhile, in the *Art Gallery Problem with Fixed Guard Candidates* (AGPFC), a viable solution consists of a set guards that guarantee surveillance of the entire polygon while having their placement restricted to a finite number of predefined potential positions in the polygon. If only guards on vertices (*vertex guards*) are allowed, we have the AGP variant known as the *Art Gallery Problem with Vertex Guards* (AGPV). In contrast, consider the *Art Gallery Problem with Witnesses* (AGPW) that asks for a set of guards able to observe merely a given finite set of points inside the polygon, while guard placement may be unrestricted.

In order to illustrate the original AGP, consider the campus of the University of Campinas (UNICAMP), shown in Figure 1.1. Imagining the streets of the university as the interior of a polygon, we are led to the representation of the campus presented in the top picture of Figure 1.2.

Now, suppose that the president of the university needs to prepare an edict for a public bidding in which cameras will be purchased to guard the streets of the campus and, for this purpose, wants to know the smallest sufficient number of cameras to be acquired. In this situation, assuming that cameras can be positioned anywhere on the streets, what would the smallest number of cameras be and where should they be placed to completely cover this area (polygon)? In Figure 1.2, an optimal solution using 57 cameras is shown for this case.

Besides the natural application just described, the study and resolution of the AGP and its variations may also be of value in other areas. Consider, for instance, an application where one seeks to place the smallest number of nodes forming a sensor network that

Figure 1.1: Satellite photo of UNICAMP.

covers an area under the restriction that the viewing range is limited (in angle and in depth). Similarly, a robot equipped with a 3D scanner (of bounded range) may be used to map out a building floor, a plaza or an entire town. The problem of minimizing the number of scan positions is also a straightforward variation of the AGP (see [7]).

In this dissertation, we will show how to solve some of these versions of the AGP and how the resolution of such variants can help to solve the original problem. Before that, we present a brief review of the literature in the context in which our work is situated.

1.2 Related Work

After Klee's proposition of the AGP, the efforts first focused on the theoretical analysis of the problem. As early as 1975, Chvátal proved that $\lfloor n/3 \rfloor$ guards are always sufficient and sometimes necessary for covering a simple polygon of n vertices. Roughly a decade later, Lee and Lin proved the $\mathbb{NP}$-hardness of the AGP for vertex guards [28]. The case of point guards is also known to be $\mathbb{NP}$-hard, as proved by Aggarwal et al. in 1988 [1].

On the algorithmic side, several techniques have been proposed for different variants of the problem, including approximation algorithms, heuristics and even some exact meth-

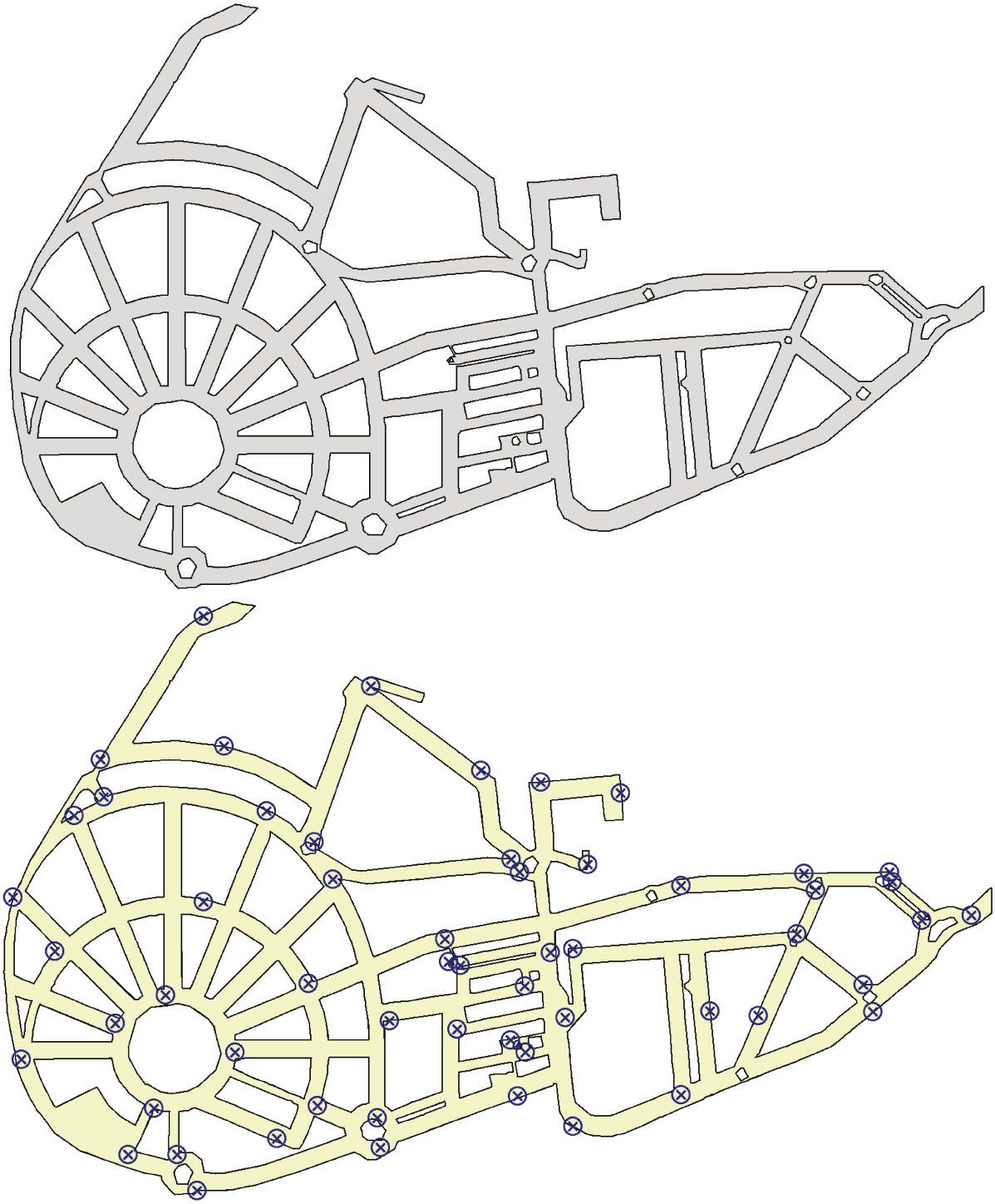


Figure 1.2: Representation of UNICAMP Campus (top); an optimal guard positioning (bottom).

ods. Figure 1.3 shows a timeline of the major recent algorithmic achievements for the problem.

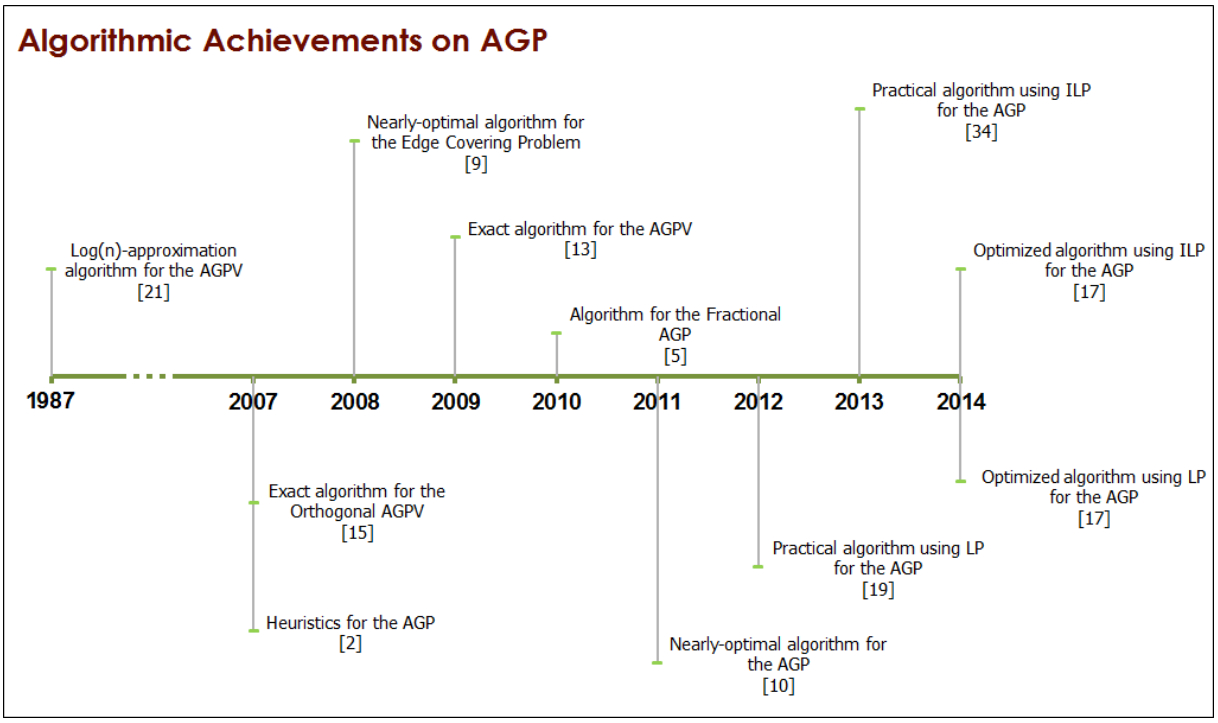


Figure 1.3: Timeline including the major recent algorithmic achievements for the AGP.

As shown in Figure 1.3, Ghosh, in 1987 [21] (and in a revised version in 2010 [23]), proposed a $\log n$-approximation algorithm for the AGP with vertex or edge guards. The idea, also exploited in more recent works for exact algorithms, was to reduce the AGP to the *Set Cover Problem* (SCP) and to apply an already known approximation for the SCP. Further approximation results are found in Eidenbenz's work [18], which describes algorithms designed for several variations of terrain guarding problems.

On the other hand, in 2007, Amit et al. [2] presented a series of heuristic techniques for the AGP based on greedy strategies and methods that employ polygon partition. According to the authors, some of these algorithms, specially the greedy ones, achieved good results for a large set of instances and, in many cases, optimal solutions. Basically, the idea of the greedy heuristics was to iteratively select new guards based on some cost ranking, which changes over the iterations.

In Figure 1.3, it is possible to see two major contributions by Bottino and Laurentini of the last few years [9, 10]. The first one, from 2008 [9], consisted in an algorithm for the Edge Covering Problem (EC), a variant of the AGP where the objective is to cover only the walls of an art gallery, being unnecessary to watch the whole interior. The

method proposed obtains viable solutions for EC by solving instances of the Integer Edge Covering Problem (IEC), a similar version with one additional restriction that any edge of the polygon must be entirely seen by one of the guards selected. At the end of each iteration of the EC resolution, the cardinality of the current guard set is compared to a lower bound and if the gap equals zero, the solution is returned as optimal.

The most recent work by these authors on this subject was published in 2011 [10] and contains a heuristic for the original AGP. The idea was to adapt the solution produced by the EC solver [9] so it can also be viable for the AGP. The algorithm achieved good results and was compared with Amit et al.'s approach in [10]. From the comparison, it was shown that the method by Bottino and Laurentini usually behaves better.

Recently, however, the search for algorithms that are able to find provably optimal solutions for specific formulations has intensified. In this line, we highlight the work by Couto et al. on the AGPV, where only vertex guards are allowed. Their first work, from 2007 [15], presented an exact algorithm for solving orthogonal polygons, which was then extended in 2009 [13, 14] to treat all classes of hole-free polygons. The iterative technique developed was guaranteed to converge to an optimal solution and, while the maximum number of iterations was proven to be polynomial in the number of vertices in the worst case, actual convergence happened after just a few iterations. The first interesting insight was to show that the infinite points of the polygon to be watched could be discretized into a finite set of points, called *witnesses*. The AGPV, having a finite set of potential locations for guards and a finite number of witnesses to be covered, can be reduced to a sequence of SCP instances that are solved to optimality using an Integer Programming solver. These instances start off with a small subset of witnesses and additional witnesses are included whenever the previous solution is not viable for the original instance. An alternative approach consisted of including all pre-determined witnesses into a single SCP instance and solving the problem to exactness in a single iteration. In fact, in [14], a clever reduction of the witness set is also presented, which makes this alternative approach much more timewise competitive. The algorithm for the AGPV was tested on more than ten thousand polygons of various classes and, for up to 2500 vertices, optimal solutions were attained in just a few minutes of computing time.

In 2013, we applied the technique of Couto et al. [14] as a tool in a new algorithm for the AGP with Point Guards, where guards can be positioned anywhere in the polygon. Our new algorithm, published in [34] and in [33], initially discretizes the polygon into a witness set and then iteratively searches for new lower bounds, through the resolution of AGPW instances, and new upper bounds, by solving the AGPFC. Both AGPW and AGPFC instances can be reduced to the SCP, as will be described in Chapter 3, which makes it possible to formulate the instances as ILPs and solve them using an ILP solver, like XPRESS, CPLEX or GLPK. At the end of each iteration of the algorithm, the current

lower and upper bounds are compared and the gap between them is verified. If it is zero, an optimal solution was found and the program halts. Otherwise, the initial discretization is updated and the whole process starts over. The technique was tested on more than two thousand instances from different classes and optimal solutions were obtained for a great majority of them.

Another technique that used similar strategies in the search for exact solutions for the AGP appears in the work by Kröller et al., first published in [5] (see also [26]). In this technique, the authors also sought to reach optimal solutions for the AGP by discretizing the infinitely many possibilities for witnesses and guard candidates into finite sets, solving discretized versions of the problem, which can be reduced to SCP instances as mentioned above. In their first work [5], the main idea was to solve, at each iteration, the linear relaxation of the primal and dual formulations of an SCP instance, producing upper and lower bounds for the original problem. This method adds new witnesses and guards at each iteration, until convergence is achieved or a maximum time frame is exceeded. Although converging on some instances, the method was unable to find integer optimal solutions in the majority of cases.

In 2012, the same authors modified the initial algorithm, aiming at a better optimality rate. In this work, a group of new restrictions (facets) of the discretized AGP (or SCP, for that matter) was included in the dual formulation, in order to improve the lower bound computed. Moreover, the new technique employed IP heuristics to find viable solutions for the AGP and, consequently, better upper bound values. The resulting algorithm, which was presented in [19], was tested and obtained better results than the previous version.

Finally, last year, the authors of [34, 33] along with those of [5, 19] worked together to produce a Survey paper focused on algorithms for the AGP [17]. In this joint work, we presented optimized versions of the previous implementations and were able to test and compare them also with older techniques. We discuss the major results obtained in this survey in Section 4.5.2.

This brief summary of the state of the art on algorithms for the AGP sets the stage for describing the contributions that this dissertation brings forth.

1.3 Our Contribution

In this dissertation, we detail a new robust method for solving the Art Gallery Problem with Point Guards. Our new algorithm, which had already been presented in the papers [34, 33], iteratively solves discretized versions of the AGP (AGPW and AGPFC) making use of ILP solvers to quickly obtain new lower and upper bounds for the problem, until an optimal solution is found or a timeout is reached.

The technique was tested in different opportunities and, as seen in Section 4.4, it

lead to good results. In total, 2880 publicly available instances with sizes reaching 5000 vertices were tested and the method achieved an optimality rate of more than 90%, which means a significant improvement over all previously published techniques.

Furthermore, due to the success of our implementation, we have recently released a free source version of our code in the project website [16]. This action is a milestone in the quest for practical solvers for the AGP, since, to our knowledge, it is the first time that an implementation with verified efficiency is made publicly available. Hopefully, this will be a catalyst in the search for new techniques to the problem and an incentive for other experimentation projects.

1.4 Text Organization

This dissertation is organized as follows. The concepts, definitions and theorems necessary to understand the algorithm to be described are presented in the next chapter. In Chapter 3, the technique we developed is explained in detail. Chapter 4 focus on how the algorithm was implemented, on the environment and instances used for testing and also on the most significant results obtained, including a comparison with other state of art techniques. Finally, some conclusions are presented in Chapter 5, as well as some ideas for future research on the Art Gallery Problem.

Chapter 2

Preliminaries

Before digging into the algorithm and its particulars, it is necessary to fully understand some important concepts, being they in the computational geometry field or part of combinatorial optimization. In the next sections, the background related to the Art Gallery Problem and our solver is explained.

2.1 Computational Geometry

The objective of the AGP is to watch over an art gallery, which may be represented as a simple polygon or as a polygon with holes. A *simple polygon* consists of straight, non-intersecting line segments that are joined pair-wise to form a closed path. The set of vertices V contains all points where consecutive segments are joined. Figure 2.1 displays examples of simple and non-simple polygons. On the other hand, a *polygon with holes* P_h can be described using a simple polygon P_b as the outer boundary and a set of m disjoint simple polygons $H_1, H_2, ..., H_m$ totally contained inside P_b as the holes. In this case, P_h consists in the set $P_b - \bigcup_{i=1}^{m} H_i$. Two examples of polygons with holes are displayed in Figure 2.2. In addition to these general types, some polygons may receive special names due to other characteristics. It is the case of the so called *orthogonal polygons*, which are simple polygons where all edges are parallel to the x-axis or y-axis (see the bottom left polygon in Figure 2.1 for an example).

A vertex in a polygon can also receive a special name depending on the angle between its two incident edges with the interior of the polygon. If this angle is less than 180°, than the vertex is called *convex*. Otherwise, it is a *reflex* vertex. Figure 2.3 presents some examples. Note that, in the case of a hole H_i, the convex vertices of H_i are actually reflex in relation to the entire polygon P_h. Obviously, the reverse is also true for reflex vertices.

In the AGP, we say that a position is *surveilled* (*watched*, *guarded* or *covered*) by a guard g if this position is visible to g. The concept of visibility can be described as follows:

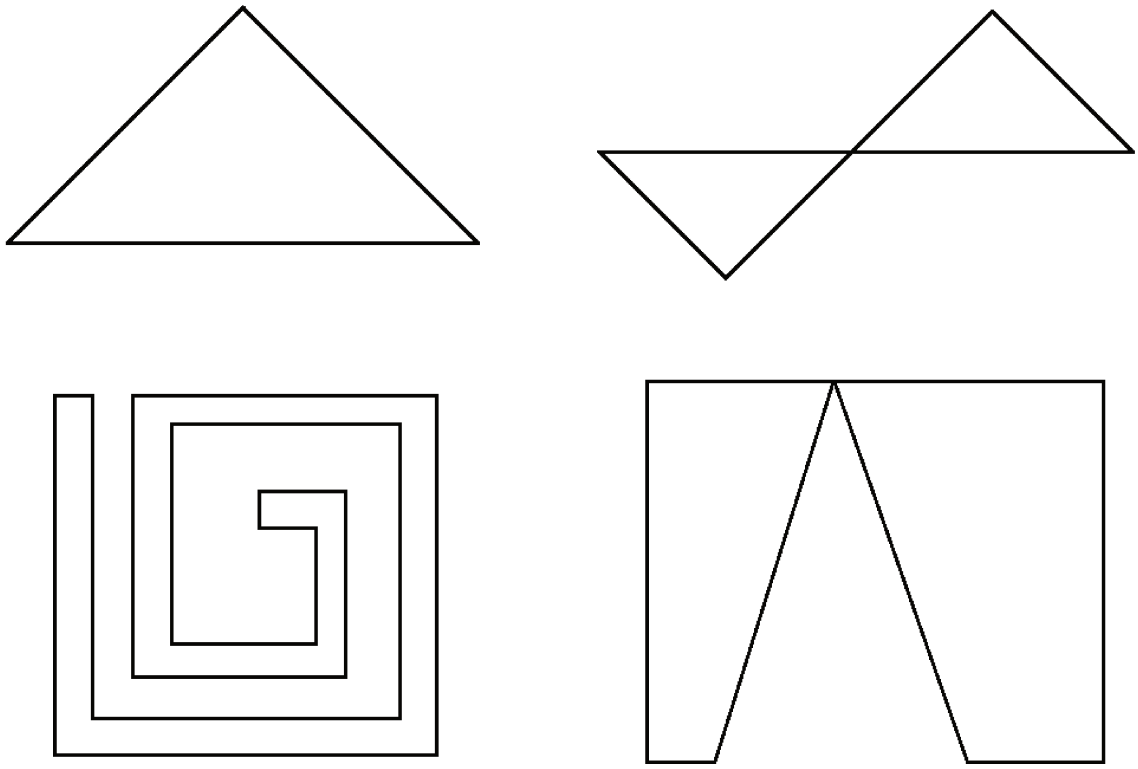


Figure 2.1: Simple polygons (left); non-simple polygons (right).

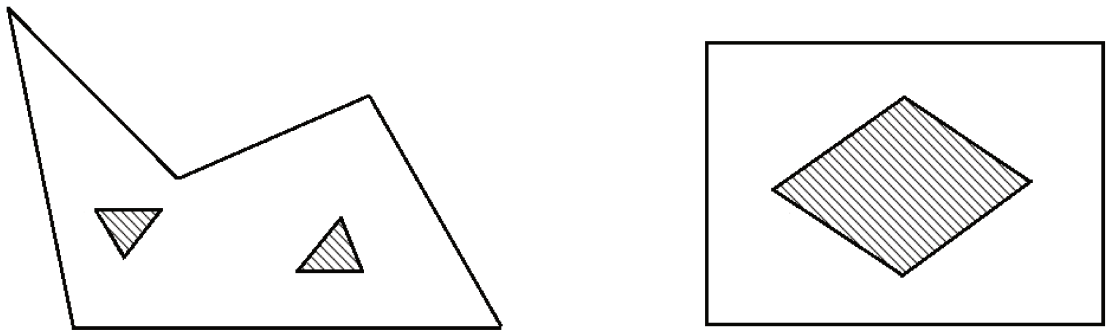


Figure 2.2: Polygons with holes.

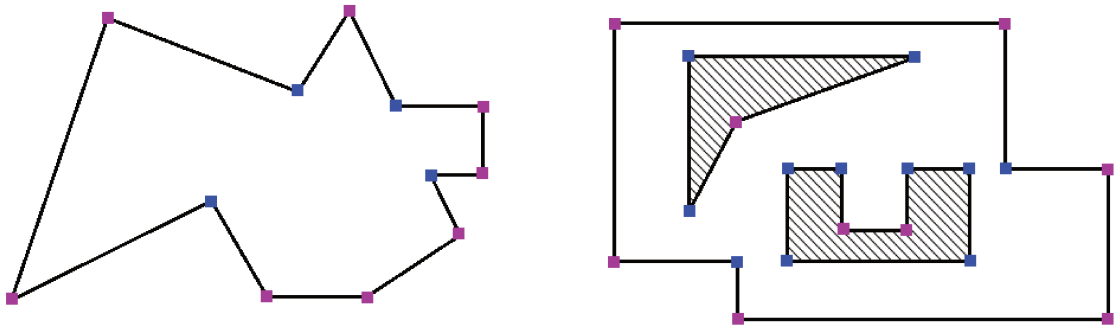


Figure 2.3: Convex (purple) and reflex (blue) vertices.

two points in a polygon P are *visible* to each other if the line segment that joins them does not intersect the exterior of P. Thereafter, the *visibility polygon* of a point $p \in P$, denoted by $\text{Vis}(p)$, is the set of all points in P that are visible from p. The edges of $\text{Vis}(p)$ are called *visibility edges* and are said to be *proper* for p if they are not contained in any edges of P. Figure 2.4 illustrates the concept of visibility.

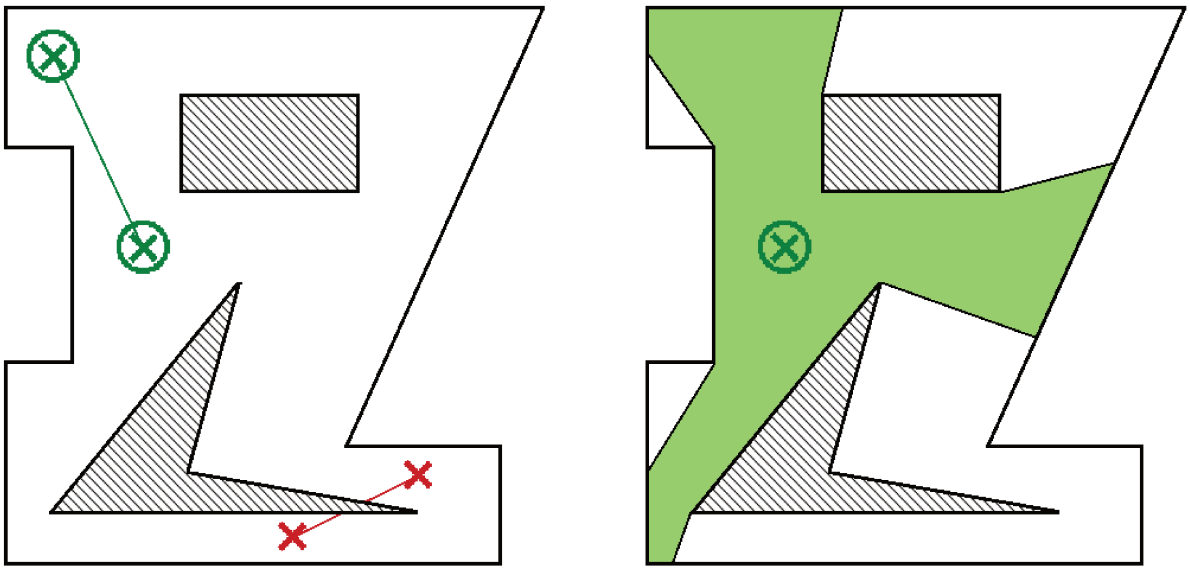


Figure 2.4: Two points visible (green) and two other that are not visible (red) to each other (left); the visibility polygon of a point (right).

After defining visibility, the next step is to define the coverage of a given region. Given a finite set S of points in P, a *covered* (respectively, *uncovered*) region induced by S in P is a maximal connected region in $\bigcup_{p \in S} \text{Vis}(p)$ (respectively, $P - \bigcup_{p \in S} \text{Vis}(p)$). Knowing this, we say that S *fully covers* the polygon P if the covered region induced by S in P equals

P. In addition, we can also define $C_{\mathcal{U}}(S)$ as a set containing exactly one interior point of each uncovered region induced by S.

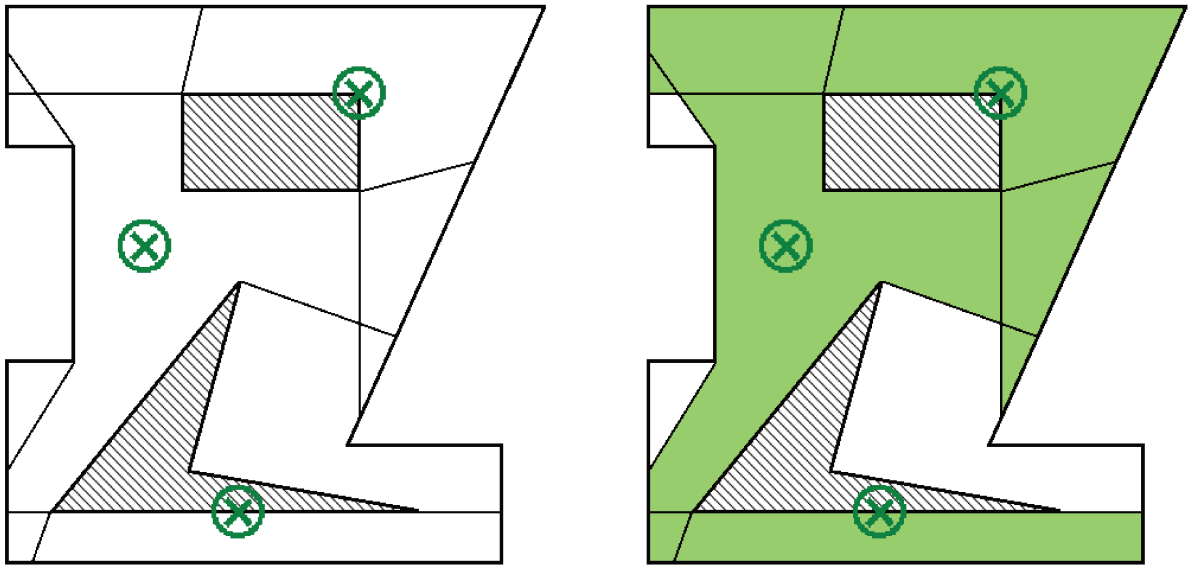


Figure 2.5: The arrangement induced by a finite set S of points (left); the set S and its covered (light green) and uncovered (white) regions (right).

Moreover, the geometric arrangement defined by the visibility edges of the points in S, denoted $\mathrm{Arr}(S)$, partitions P into a collection of closed polygonal faces called *Atomic Visibility Polygons* or simply *AVP*s. Clearly, the edges of an AVP are either portions of edges of P or portions of proper visibility edges of points of S. Denote by $C_f \subset S$ the set of points in S that cover an AVP f. Define a partial order $\prec$ on the faces of $\mathrm{Arr}(S)$ as follows. Given two AVPs f and f', we say that $f \prec f'$ if and only if $C_f \subset C_{f'}$. We call f a *shadow AVP* (*light AVP*) if f is minimal (maximal) with respect to $\prec$. Applying these concepts, it is possible to define $C_{\mathcal{L}}(S)$ as a set containing exactly one point within each light face of $\mathrm{Arr}(S)$ and $V_{\mathcal{L}}(S)$ as the set of all vertices of light AVPs. Figures 2.5 and 2.6 illustrate these concepts.

It is worth noticing that there may be degenerate cases of shadow and light AVPs. Figure 2.7 illustrates two of these situations.

In our work, the complexity of the resulting arrangement is of great importance for the solver's performance. In [8], Bose et al. showed that, for the case where the set of vertices V induces the arrangement in a simple polygon, the arrangement complexity is $\Theta(|P|^3)$. This result can be easily adapted to prove that, for a generic set S in a hole-free polygon P, the complexity of constructing $\mathrm{Arr}(S)$, as well as the number of AVPs in the arrangement, is $\Theta(|S|^2 \cdot |P|)$. In the general situation, where P can have holes, we did not

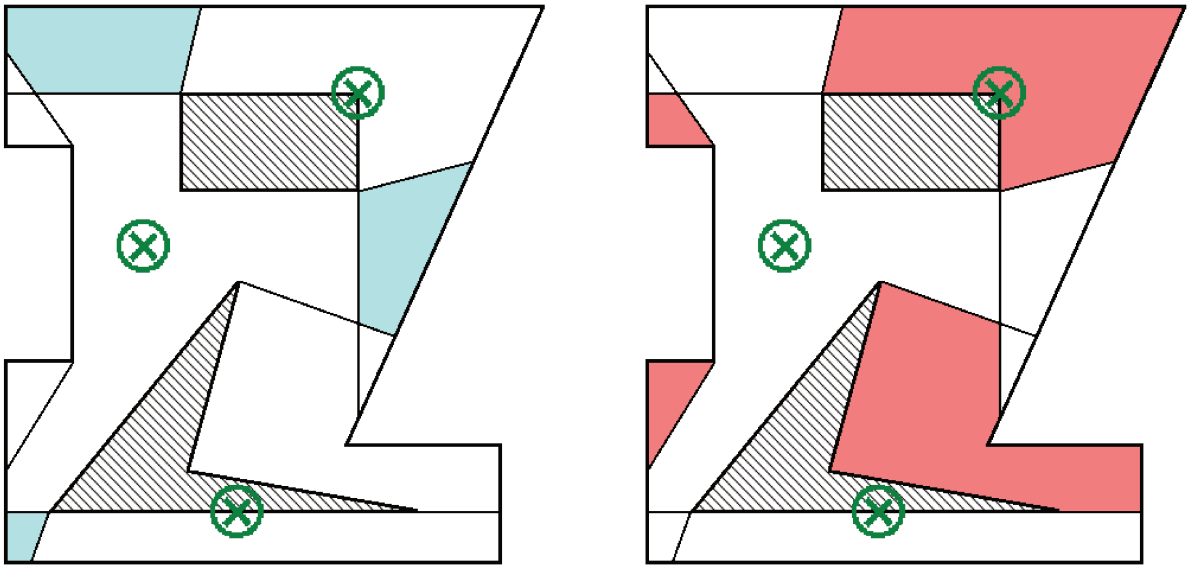


Figure 2.6: The arrangement induced by S with the *light* (blue) (left); and *shadow* (red) AVPs (right).

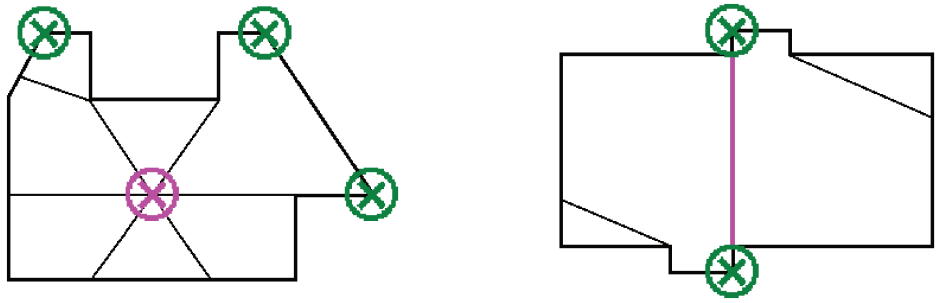


Figure 2.7: A point (left) and an edge (right) (both represented in magenta) which are, at the same time, light and shadow AVPs.

find any complexity results in literature. However, as we have $O(|S| \cdot |P|)$ visibility edges in the polygon, in the worst case, considering that all of them intersect, we will have a final complexity of $O(|S|^2 \cdot |P|^2)$.

2.2 Integer Programming

Recall that in Section 1.2 we reported that Ghosh's approximation technique [21] employs a reduction of a discrete version of the AGP to the Set Cover Problem (SCP). A similar reduction is also used in the method developed in this Master's project. Therefore, it is important to know the best ways to solve an SCP instance, since the performance of the technique chosen to accomplish this task can have a great influence in the overall performance of our AGP solver.

The SCP is one of the most famous combinatorial problems. Given a set of elements U, called Universe, and a set A containing subsets of U, the SCP asks for the minimum number of sets from A whose union equals U. As proved by Karp in 1972, the Set Cover Problem is $\mathbb{NP}$-complete [25], which means that obtaining an algorithm with polynomial worst-case complexity is not possible, unless $\mathbb{P} = \mathbb{NP}$. Nevertheless, in practice, a good option for solving an SCP instance is to model the problem as an Integer Linear Program (ILP), even though the cost of solving a general ILP is theoretically exponential. Today, several ILP solvers are capable of finding optimal solutions for large SCP instances, with thousands of variables and constraints, in just a few seconds or minutes. Below the classic ILP model for the SCP is given.

$$\begin{aligned} \min \quad & \sum_{s \in A} x_s \\ \text{s.t.} \quad & \sum_{\substack{s \in A \\ e \in s}} x_s \geq 1, \ \forall e \in U \\ & x_s \in \{0, 1\}, \ \forall s \in A \end{aligned}$$

In the model, the variable x_s has value 1 if the set s is chosen to be part of the resulting collection of subsets and 0 otherwise. The objective is to minimize the sum of variables x_s, which actually means to minimize the number of sets selected from A. Finally, the set of constraints presented in the model ensures that, for every element $e \in U$, at least one of subsets containing e is chosen, which gives rise to a viable solution.

Although ILP solvers are normally a good choice for solving SCP instances, there are cases where their use may not be so efficient. In these situations, a technique that can take advantage of particular characteristics of the problem can behave better and be used to improve the ILP solver's performance. In our AGP solver, we implemented some techniques with this purpose.

One well tested method to find good viable solutions for SCP instances and that was implemented in this project is a Lagrangian Heuristic. The heuristic implemented is based on the work by Beasley [6], which, among other results, presents a *Lagrangian Relaxation* for the SCP. The idea of the relaxation is to move all coverage constraints into the objective function and use penalties u_e (for each constraint), the Lagrangian Multipliers (LM), resulting in the following *Lagrangian Primal Problem* (LPP):

$$z(u) = \min \sum_{s \in A} x_s + \sum_{e \in U} u_e \left(1 - \sum_{\substack{s \in A \\ e \in s}} x_s\right)$$
$$x_s \in \{0, 1\}, \forall s \in A$$

After this step, the LPP (with no actual constraints) can be solved by inspection. It is well-known that the optimum of this problem gives a lower bound for the original SCP instance. The new quest is to find the values for the LM (the u_e variables) that maximize this lower bound. The optimization problem of finding the best Lagrangian Multipliers is called the *Lagrangian Dual Problem* (LDP):

$$Z = \max z(u)$$
$$u_e \geq 0, \quad \forall e \in U.$$

A classical approach to tackle the LDP is to apply the *subgradient method* (c.f., [6]), an algorithm in which the LM values are updated iteratively using a subgradient of the objective function. This way, at each iteration, a new LPP instance is solved and, based on this result, we apply a primal heuristic, which attempts to find a good viable solution for the original SCP instance. The primal heuristic consists in a greedy procedure that uses the Lagrangian costs obtained during the last LPP resolution to define which candidates will be selected to join the new viable solution (see [6] for more details). In summary, at each iteration, a new lower bound for the SCP is obtained from the LPP resolution and a new upper bound is obtained by the primal heuristic. In practice, the subgradient method is stopped when either a proven optimal solution is found or a maximum number of iterations has been reached.

Apart from the techniques employed to find viable solutions for the SCP, others can be used to simplify the problem, before an actual solver is used. For example, after a problem is reduced to an ILP instance, it is possible to search for redundant variables or constraints and remove them from the original matrix. This type of operation can normally be done quickly and may greatly minimize the size of the initial instance, probably improving the performance of the ILP solver. In Section 3.4, we show how this can be done in the case of the AGP.

2.3 The Art Gallery Problem

After discussing important geometric and combinatorial concepts, it is now possible to discuss the AGP in a more formal way.

In a geometric setting, the AGP can be restated as the problem of determining a smallest set of points $G \subset P$ such that $\bigcup_{g \in G} \text{Vis}(g)$ equals P. This leads to a reduction from the AGP to the SCP, in which the points in P are the elements to be covered (set U) and the visibility polygons of the points in P are the sets used for covering (which compose the collection A). Accordingly, we can use this reduction to construct an ILP formulation for the AGP:

$$\begin{aligned} \min \quad & \sum_{c \in P} x_c \\ \text{s.t.} \quad & \sum_{\substack{c \in P \\ w \in \text{Vis}(c)}} x_c \geq 1, \ \forall w \in P \\ & x_c \in \{0, 1\}, \ \forall c \in P \end{aligned}$$

However, for non-trivial cases, this formulation has an infinite number of constraints and an infinite number of variables, rendering it of no practical benefit. A natural idea that arises is to make at least one of these quantities finite. By fixing only the guard candidates to be a finite set, we obtain the so-called *Art Gallery Problem With Fixed Guard Candidates* (AGPFC). On the other hand, by restricting solely the witness set to a finite set, we end up with the special AGP variant known as the *Art Gallery Problem With Witnesses* (AGPW). In principle, in the first case, we are still left with an infinite number of constraints while, in the second case, we still have an infinite number of variables. However, in order to have a tractable SCP instance, one should have both the witness and the guard candidate sets of finite size. The AGP variant that fulfills this property is named the *Art Gallery Problem with Witnesses and Fixed Guard Candidates* (AGPWFC). Examples of these three versions of the AGP are shown in Figure 2.8. Therein, the witnesses and the guard candidates are identified by the symbols "$\times$" and "$\otimes$", respectively. Darker guard candidates refer to guards present in an optimal solution of the corresponding problem and, when appropriate, have their visibility polygons also depicted.

To assist in the description of the algorithm in the next chapter, we introduce here some other useful notations. Let D and C be finite witness and guard candidate sets, respectively. We denote the AGPW, AGPFC and AGPWFC problems defined for the sets C and D by AGPW(D), AGPFC(C) and AGPWFC(D, C), respectively. The SCP

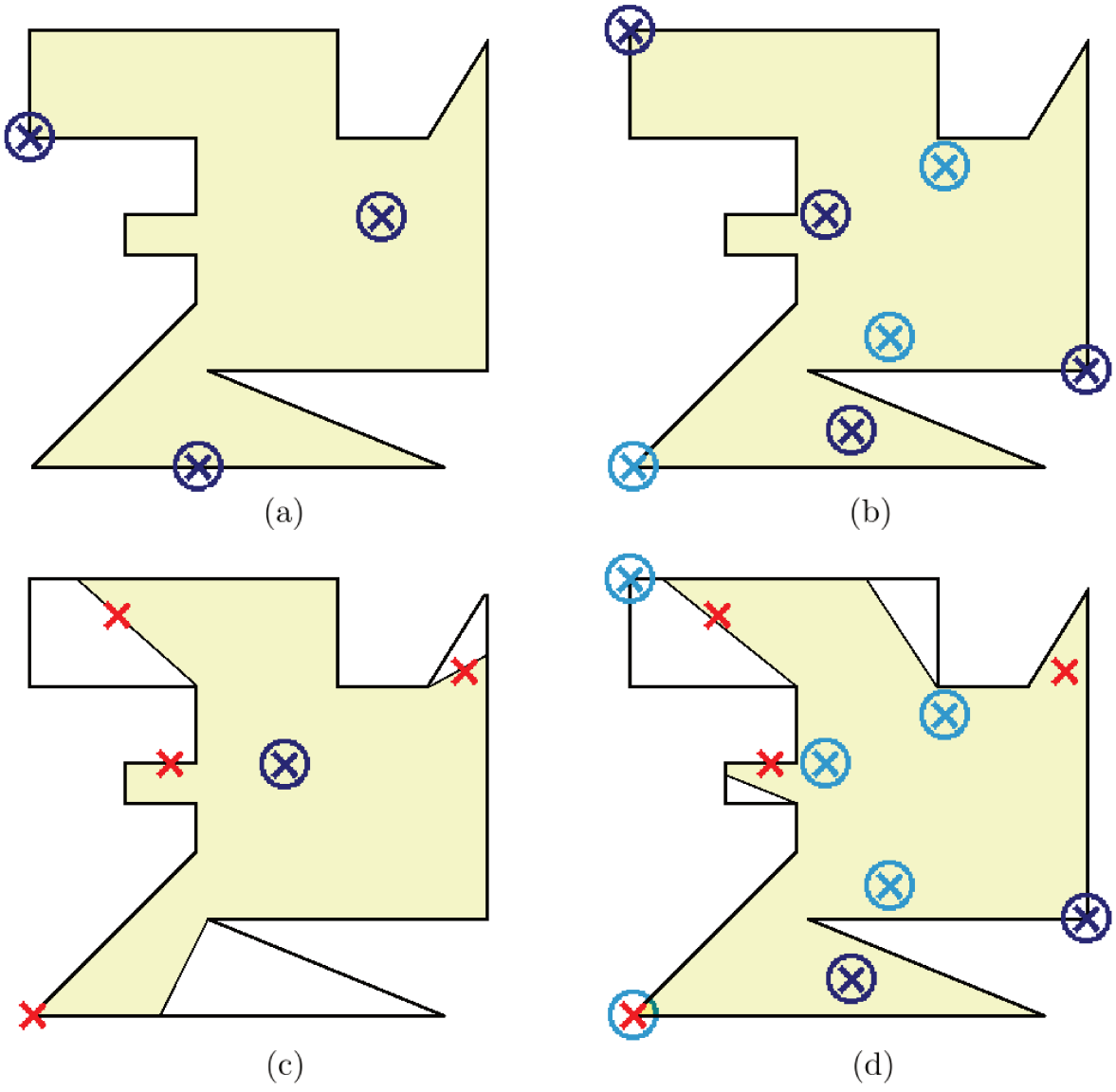


Figure 2.8: An illustration of the four different variants of the Art Gallery Problem: (a) AGP; (b) AGPFC; (c) AGPW; (d) AGPWFC.

model associated to AGPWFC(D, C) is then

$$\begin{aligned} \min \quad & \sum_{c \in C} x_c, \\ \text{s.t.} \quad & \sum_{\substack{c \in C \\ w \in \text{Vis}(c)}} x_c \geq 1, \ \forall w \in D, \\ & x_c \in \{0, 1\}, \ \forall c \in C. \end{aligned}$$

2.4 Basic Theorems

To close this chapter, we briefly introduce the theorems that form the basis of our algorithmic solution, which will be fully explained in Chapter 3. The following theoretical results allow us to apply reductions of the AGPFC and the AGPW to the AGPWFC (SCP) and also guarantee its usage to find correct bounds for the original AGP. It is noteworthy that Theorems 2.2, 2.3 and 2.4 are actually adaptations of results presented in the work of Couto et al. [14], where the specific problem called AGPV (AGPFC(V)) was treated.

Theorem 2.1. *Let D be a finite subset of points in P. Then, there exists an optimal solution for AGPW(D) in which every guard belongs to a light AVP of* Arr(D).

Proof. Let G be an optimal (cardinality-wise) set of guards that covers all points in D. Suppose there is a guard g in G whose containing face f in Arr(D) is not a light AVP. This means that f is not maximal with respect to the order relation $\prec$ (see Section 2.1). In other words, there exists a face f' of Arr(D) that shares an edge with f such that $f \prec f'$, i.e., a point in f' sees more points of D than one in f does. An inductive argument suffices to show that this process eventually reaches a light AVP (maximal w.r.t. $\prec$) wherein lies a point that sees at least as much of D as g does, i.e., g may be replaced by a guard that lies on a light AVP. The Theorem then follows, by induction, on the number of guards of G. □

Theorem 2.2. *Let C be a finite subset of points in P. Consider the set D composed of a point of each AVP of* Arr(C). *Then, $G \subseteq C$ guards D if and only if G is a guard set for P.*

Proof. The necessity part is trivial since $D \subset P$, therefore, we focus on the proof of sufficiency. By the construction process of Arr(C), all interior points of a given AVP A_i are visible to the same set $S_i \in C$. Otherwise, there would be an edge of Arr(C) separating two different points in A_i, which is obviously not possible. Consequently, if a set G guards one interior point of A_i, G directly covers the entire AVP. Thus, since the

union of all faces of $\mathrm{Arr}(C)$ equals P, G can watch over the whole polygon by simply covering one interior point within each AVP. □

Theorem 2.3. *Let C be a finite subset of points in P. Consider the set D composed of a point of each shadow AVP of* $\mathrm{Arr}(C)$*. Then, $G \subseteq C$ guards D if and only if G is a guard set for P.*

Proof. The necessity part is trivial since $D \subset P$, therefore, we focus on the proof of sufficiency. Suppose $G \subset C$ guards D, but not P. Thus, there exist regions of P that are not covered by any of the points of G. Let R be a maximal connected region not covered by G. Note that R is the union of (disjoint) AVPs. To prove that at least one of those AVPs is of type shadow, notice that the entire region R is not seen by any point in G whose proper visibility edges spawn R. If R is an AVP, it is by definition a shadow AVP. Otherwise, there is a candidate $c_i \in C$ which has a proper visibility edge e_{ci} that intersects and partitions R in two other regions. One of these regions matches the side of e_{ci} visible from c_i while the opposite one does not. Hence, through an inductive argument, by successively partitioning R, at least one shadow AVP is bound to be contained in R and therefore uncovered. This contradicts the hypothesis since G guards D, which is comprised of interior points of all shadow AVPs. □

Theorem 2.4. *Let D and C be two finite subsets of P, so that C fully covers P. Assume that $G(D, C)$ is an optimal solution for AGPWFC(D, C). If $G(D, C)$ also covers P, then $G(D, C)$ is an optimal solution for AGPFC(C).*

Proof. Assume that $G(D, C)$ covers P, but it is not an optimal solution for AGPFC(C). Then, there exists $G' \subseteq C$ with $|G'| < |G(D, C)|$ such that G' is a feasible solution for AGPFC(C), i.e., G' covers P. This implies that G' is also a feasible solution for AGPWFC(D, C), contradicting the fact that $G(D, C)$ is optimal for this problem. □

Chapter 3

An Algorithm for the AGP

In this chapter, we employ the concepts and theorems presented in Chapter 2 to explain, in details, our technique for solving the original AGP. Initially, we present a simplified and shortened version of the algorithm and discuss the main idea proposed. After this, we dig into the most important steps of the technique, thoroughly showing how the computation occurs.

3.1 Sketch of the Algorithm

The core idea of our algorithm consists in computing increasing lower bounds and decreasing upper bounds for the AGP until a proven optimal solution is found or a pre-established maximum time limit is exceeded. The procedure for obtaining these bounds involves the resolution of discretized versions of the AGP. To find a new lower bound, an AGPW instance is solved, while in the upper bound case, an AGPFC instance in which the guard candidate set covers the polygon is worked out. One important fact to remember is that an optimal solution for such an instance of AGPFC is also viable for AGP, since the AGPFC asks for a solution that guards the entire polygon. Consequently, reducing the gap between bounds to zero means reaching an optimal solution for the original AGP.

In Algorithm 1, we summarize how our technique works. After initializing the witness and guard candidate sets, the algorithm enters its main loop (lines 4 to 10). At each iteration, new lower and upper bounds are computed and, if optimality has not been proved, the witness and guard candidate sets are updated in line 8. Later we will see how this can be done in a way that the optimality gap decreases monotonically through the iterations. It is also worth noticing that, as we do not have a proof of convergence for the algorithm, the parameter `MAXTIME` is used to limit its running time.

In the following two sections, the procedures for solving AGPW (line 5) and AGPFC (6) instances are described in details. Subsequently, Section 3.4 describes the resolution

Algorithm 1 AGP (Sketch)

```
1: Set UB ← |V| and LB ← 0
2: Select the initial witness set D
3: Select the initial guard candidate set C ⊇ V
4: while (UB ≠ LB) and (MAXTIME not reached) do
5:    Solve AGPW(D), set G_w ← optimal solution of AGPW(D) and
      LB ← max{LB, |G_w|}
6:    Solve AGPFC(C), set G_f ← optimal solution of AGPFC(C) and
      UB ← min{UB, |G_f|}
7:    if (UB ≠ LB) then
8:       Update D and C
9:    end if
10: end while
11: return G_f
```

method for AGPWFC instances through ILP techniques. As said in the previous chapter, both the AGPW and the AGPFC can be cast as an AGPWFC, justifying why we focus on the latter. In Section 3.5, we present how the management of the witness set is done and, in the following section, we discuss the selection of guard candidates. Section 3.7 gathers all the algorithmic information presented previously and describes the complete algorithm for the AGP. Finally, we illustrate the step by step of the algorithm with an example, where a polygon from our benchmark instances is resolved to optimality in a few iterations.

3.2 Solving the AGPW

The resolution of an AGPW on D allows for the discovery of a new lower bound for the AGP, since fully covering P requires at least as many guards as the minimum sufficient to cover the points in $D \subset P$. However, despite being a simplification of the original AGP problem, we are still dealing with an infinite number of potential guard positions, which does not allow for a direct reduction to the set cover problem. Thus, our approach is based on discretizing the guard candidate set, creating an AGPWFC instance from our original AGPW. To do this, we apply Theorem 2.1, presented in Section 2.4.

From Theorem 2.1, one concludes that there exists an optimal solution for AGPW(D) in which all the guards are in Light AVPs of the arrangement induced by D. Besides, as every vertex of an AVP can see at least the same set of witnesses observed by the points inside it, we can state that there is an optimal solution G where each point in G belongs to the set $V_{\mathcal{L}}(D)$ of all vertices from the light AVPs of Arr(D) (see in Figure 3.9

an example where a vertex of a Light AVP can see more witnesses than the points inside it). Therefore, an optimal solution for AGPW(D) can be obtained simply by solving AGPWFC($D, V_{\mathcal{L}}(D)$), as illustrated in the example of Figure 3.1. As seen before, the latter problem can be modeled as an ILP, where the numbers of constraints and of variables are polynomial in $|D|$. This follows, as mentioned in Section 2.1, from the fact that the number of AVPs (and vertices) in Arr(D) is bounded by a polynomial in $|D|$ and $|P|$. Algorithm 2 shows a pseudo-code of the AGPW resolution method.

Algorithm 2 AGPW(D)

1: Arr(D) $\leftarrow$ construct the arrangement from the visibility polygons of the points in D
2: $V_{\mathcal{L}}(D) \leftarrow$ identify the vertices of the light AVPs of Arr(D)
3: $C \leftarrow V_{\mathcal{L}}(D)$
4: Solve AGPWFC(D, C): set $G_w \leftarrow$ optimal solution of AGPWFC(D, C)
5: **return** G_w

As will be shown in Section 3.6, the guard candidate set used in the implemented algorithm for the AGP is not actually equal to $V_{\mathcal{L}}(D)$. The final set C includes additional points and, depending on the discretization strategy used, may employ points from $C_{\mathcal{L}}(D)$, which are located in the interior of light faces, instead of the ones in $V_{\mathcal{L}}(D)$.

3.3 Solving the AGPFC

As we now know how to generate dual bounds for the AGP, the next task is to compute good upper bounds for the problem. A possible way to achieve this is through the resolution of an AGPFC instance in which the guard candidate set is known to cover the polygon. Under this condition, an AGPFC solution is always viable for the original problem.

In contrast to what was discussed regarding the AGPW, we now have a finite number of guard candidates and an infinite number of points to be covered. Therefore, a direct resolution method using a reduction to an SCP is not possible. To circumvent this, our algorithm discretizes the original AGPFC instance, relying on what Theorem 2.4 establishes. Theorem 2.4 shows that an optimal solution for the AGPFC may be obtained through the resolution of an AGPWFC instance, provided it fully covers P. Whenever an optimal solution for the simplified version (AGPWFC) leaves uncovered regions in P, additional work is required. To guarantee that we attain an optimal solution for the AGPFC, we will employ here a technique designed by Couto et al. [14] to solve the AGPV, a special case of the AGPFC where $C = V$, but which may be used to handle the general case without significant changes.

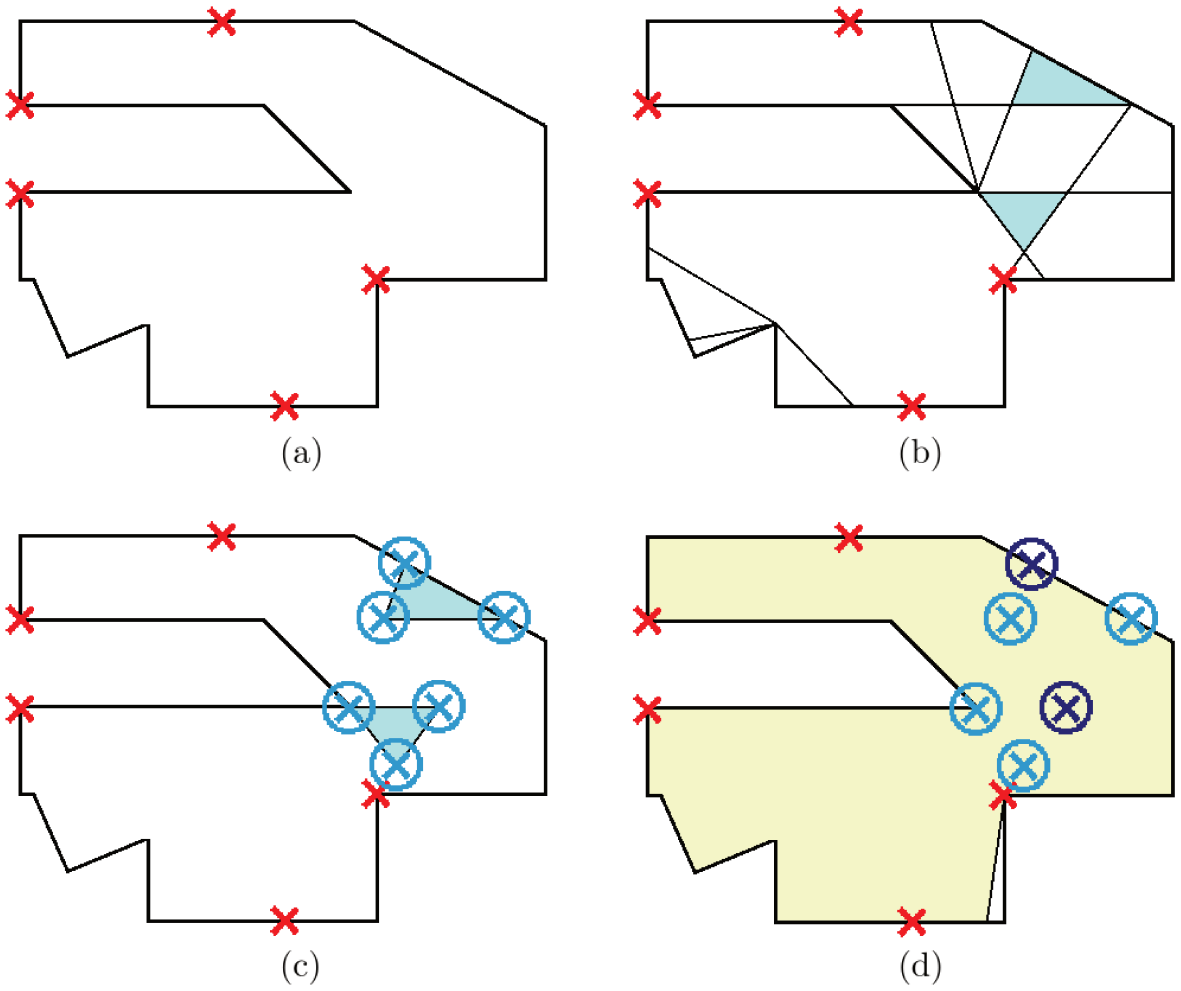


Figure 3.1: **Algorithm 2:** (a) AGPW(D); (b) The light AVPs of Arr(D); (c) The guard candidate set; (d) An optimal solution $G \subseteq V_{\mathcal{L}}(D)$ for AGPW(D).

Initially, consider that we have a finite witness set D. Using the guard candidate set C, we can create and solve the AGPWFC(D,C) instance. If the solution fully covers the polygon, we have satisfied the hypothesis of Theorem 2.4 and, consequently, we have an optimal solution for the AGPFC. Otherwise, there are regions of the polygon that remain uncovered. We now update the witness set, adding new points within the uncovered regions, denoted $C_{\mathcal{U}}(G)$, and repeat the process.

As demonstrated in [14] by Couto et al., the iterative method for solving the AGPFC converges in polynomial time. To clarify this point, consider Theorem 2.2 and its proof. This theorem states that constructing D by choosing only one point within each AVP of Arr(C) is enough to ensure the whole coverage of P. As the number of AVPs is polynomial (see Section 2.1) and we iteratively construct D by choosing witnesses in the interior of uncovered AVPs of Arr(C), it is straightforward that the number of iterations is bounded by the polynomial complexity of Arr(C). Although the convergence time for AGPFC is theoretically guided by this complexity, Couto et al. showed throw extensive experiments that, in practice, the number of necessary iterations is much lower. Moreover, it can even be argued that it suffices to select one point from each shadow AVP of the arrangement induced by C (see Theorem 2.3). Figure 3.2 shows how the algorithm for the AGPFC iteratively adds new witnesses in different AVPs until the polygon is fully covered.

A pseudo-code for the algorithm employed to solve the AGPFC is shown in Algorithm 3.

Algorithm 3 AGPFC(C)

```
1: D_f ← initial witness set
2: repeat
3:    Solve AGPWFC(D_f, C): set G_f ← optimal solution
4:    if G_f does not fully cover P then
5:       D_f ← D_f ∪ C_U(G_f)
6:    end if
7: until G_f fully covers P
8: return G_f
```

3.4 Solving the AGPWFC (SCP)

Having reduced the AGPW and the AGPFC into AGPWFC instances in order to obtain the desired bounds, the objective becomes solving the latter efficiently. Since AGPWFC(D, C) can be easily reduced to an SCP, where the witnesses in D are the elements to be covered and the visibility polygons of the guard candidates in C are the subsets considered, we will make use of the ILP formulation for SCP presented in Section 2.2.

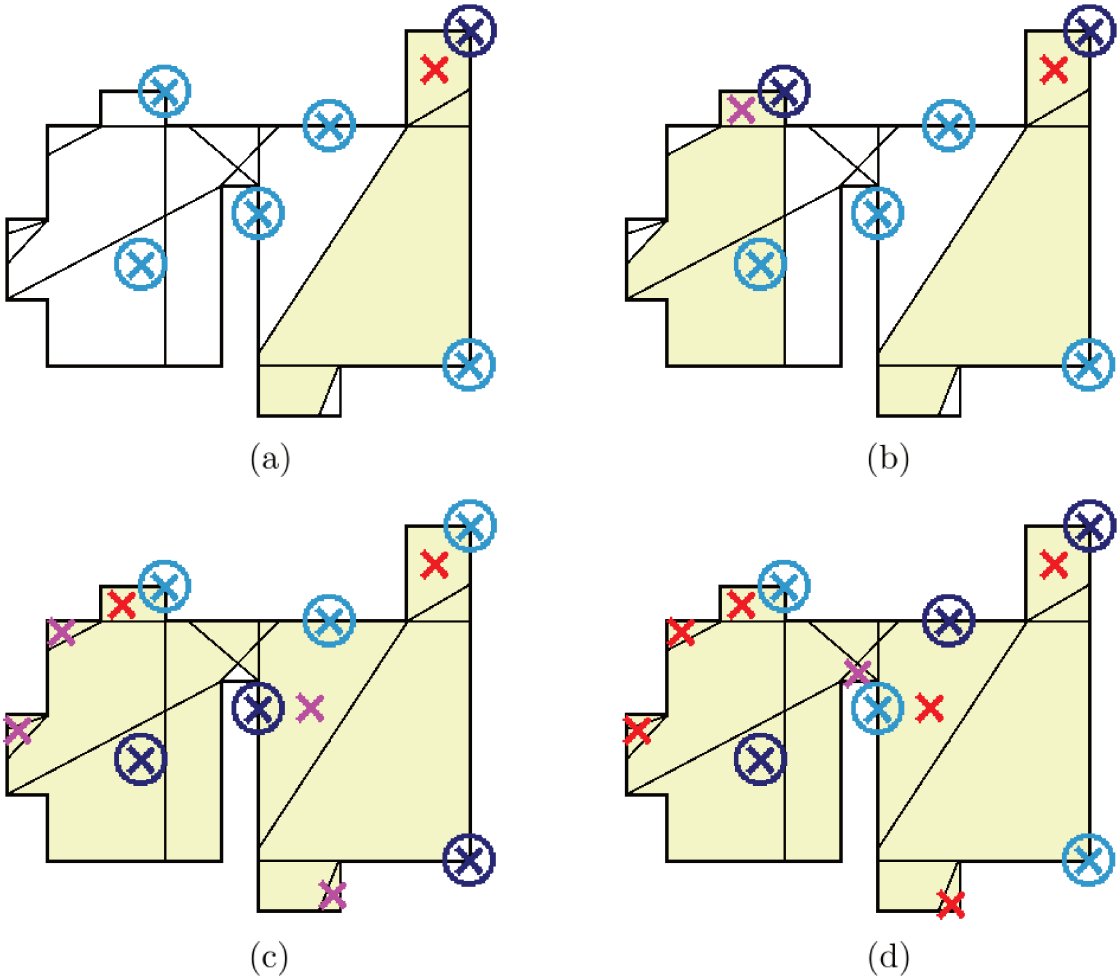


Figure 3.2: **Algorithm 3:** A sequence of AGPWFC(D,C) instances is generated until a viable solution for the AGP is obtained.

A simple and straightforward approach would be to directly use an ILP solver, such as XPRESS, CPLEX or GLPK, since even large instances of the ($\mathbb{NP}$-hard) SCP can be solved quite efficiently by many modern integer programming solvers. However, as observed in our experiments, some AGP instances can generate significantly complex and very large SCP instances, rendering the solvers less efficient. For this reason, some known techniques were implemented to improve the solvers' running time. Among these, the most effective consisted in the reduction on the number of constraints and variables in the initial model.

The method used for reducing constraints and variables is based on containment relationships between columns (guard candidates) and between rows (witnesses) of the Boolean constraint matrix corresponding to the ILP model of the SCP instance.

Firstly, we search for and discard *redundant guard candidates (columns)*. A guard candidate g_r is *redundant* if there is another candidate that covers the same witness set as g_r. For this step, we divide all guard candidates into groups, according to the light AVP they belong to. Guard candidates from the same AVP are then tested pairwisely for redundancy, and, when one is found, the corresponding column (variable) is removed from the original matrix (ILP). Here there is a considerable advantage when solving AGPW instances. Recall that when lower bounds are computed, the witness set D remains fixed while the guard candidates are points (possibly vertices) of the light AVPs of $\mathrm{Arr}(D)$. In this case, there is usually a great number of points from a given light AVP that covers the same subset of witnesses. Actually, despite degenerate cases, all but one of the corresponding variables are redundant in the SCP instance, i.e., we are left with a single candidate from each light AVP. This considerably reduces the size of the guard candidate set.

Figure 3.3 shows three examples where the guard set is reduced employing the procedure just described. In the picture on the left, only one candidate from the light AVP remains represented in the ILP matrix. The second picture displays the special case where one candidate (in the intersection of three light AVPs) remains necessary. In the last example, despite the fact that the remaining two candidates watch the same witnesses, our implementation maintains both of them in the final set because the method employed analyzes each AVP separately, not testing candidates from different faces of the arrangement.

After the removal of columns, we also test each pair of rows in search for removable ones. We say that a row represents an *easy witness* w whenever the set of guard candidates that see w properly contains the whole set of candidates that are visible to some other witness. Figure 3.4 illustrates the removal of easy witnesses.

It is important to notice that the test for redundant candidates (columns) and easy witnesses (rows) can be performed simply by using very fast bit string operations.

Besides conducting matrix reduction, our algorithm also uses an initial *Lagrangian*

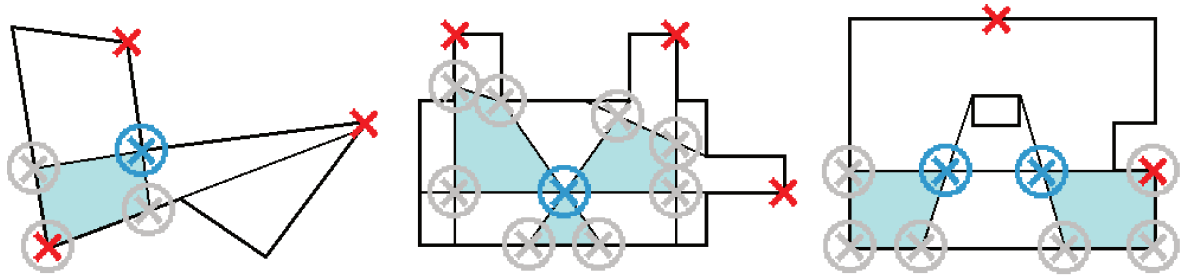


Figure 3.3: Examples of AGPWFC instances where the column reduction procedure was applied, showing the removed guard candidates (gray) and the remaining ones (blue).

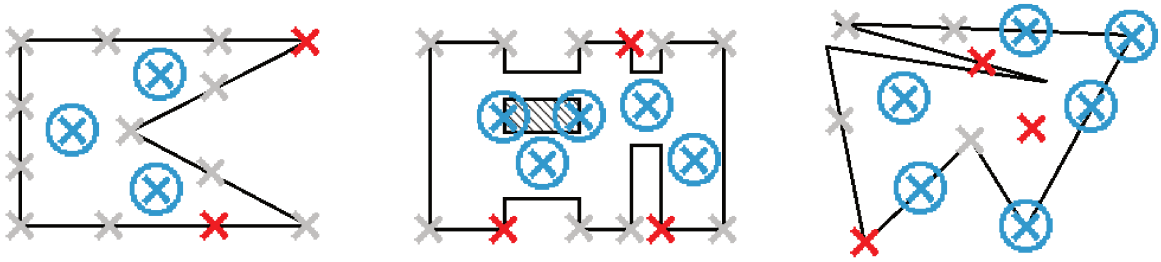


Figure 3.4: Examples of AGPWFC instances where the row reduction procedure was applied, showing the removed witnesses (gray) and the remaining ones (red).

Heuristic (LH) with the goal of finding good viable solutions, possibly optimal, for the AGPWFC. A simplified description of the Lagrangian Heuristic applied is presented in Section 2.2.

After running LH, we have a reduced ILP matrix, a viable solution and an upper bound for the AGPWFC. All this information was used in our implementation to improve the efficiency of the ILP solver. Algorithm 4 summarizes the steps for solving the AGPWFC instance, using matrix reduction and a Lagrangian Heuristic.

3.5 Witness Management

The witnesses selected during the execution of our algorithm play an important role in the search for optimal solutions for the AGP. The witness set D is not only decisive for producing good lower bounds through the resolution of AGPW(D), but it is crucial to find tight upper bounds, since the candidate set C of AGPFC(C) is constructed from Arr(D). In this context, choosing D wisely may lead to a lower gap between the bounds and, consequently, to a lower number of iterations of Algorithm 1.

Recall that, before the first iteration of Algorithm 1, an initial witness set is chosen and, in the following ones, it gets suitably updated (line 5). In this section, we present

Algorithm 4 AGPWFC(D,C)

1: $M \leftarrow$ Boolean matrix of the SCP model constructed using D and C
2: $R \leftarrow$ matrix obtained from M after applying the reduction procedure
3: Set $I \leftarrow$ SCP instance associated to R
4: Solve I using LH
5: Set $\text{LB}_{\text{SCP}} \leftarrow$ best lower bound found for I by the subgradient method
6: Set $G_v \leftarrow$ best viable solution found for I by the primal heuristic
7: $G_s \leftarrow G_v$
8: **if** $|G_s| \neq \text{LB}_{\text{SCP}}$ **then** {LH was not able to prove optimality}
9: Solve I using an ILP solver with the primal bound G_v
10: Set $G_s \leftarrow$ optimal solution found for the ILP model
11: **end if**
12: **return** G_s

all the initialization alternatives that were considered since our first work [34] and, after this, we discuss how the set D is updated.

The first initialization choice, called *All-Vertices* (AV), consists in using all vertices of the polygon as witnesses, i.e., $D = V$. Besides the easy construction of this set, it was confirmed in tests that the coverage of such points is usually a good start for covering the whole polygon.

In an attempt to begin with a smaller number of witnesses, we also considered initializing D with only the convex vertices of P. This strategy is referred to as the *Convex-Vertices* (CV) initialization. The reason for reducing the initial witness set lies on the fact that a smaller set D is likely to lead to a lower number of visibility polygon calculations, to a less complex visibility arrangement and, consequently, to a simpler SCP model. In addition, we decided to choose only convex vertices because the reflex ones are usually more exposed due to its wider visibility angle.

The third alternative is based on a work by Chwa et al. [12]. In this paper, a polygon P is defined to be *witnessable* when there exists a finite witness set $W \subset P$ with the property that any set of guards that covers W must also cover the entire polygon. The authors also present an algorithm that computes a minimum witness set for a polygon whenever it is witnessable. The method for constructing this minimum witness set consists in placing a witness in the interior of every *reflex-reflex* edge of P and on the convex vertices of every *convex-reflex* edge. The terms *convex* and *reflex* here refer to the interior angles at a vertex or at the endpoints of an edge. Based on these selection criteria, we devised our third discretization method, called *Chwa-Points* (CP), which assembles the initial witness set for our algorithm from the midpoints of all reflex-reflex edges and all convex vertices from convex-reflex edges.

It follows from the results in [12] that, when the Chwa-Points discretization is used

for a witnessable input polygon, our AGP algorithm will find an optimal solution in a single iteration. However, as observed in our experiments, non-witnessable polygons are the norm. In fact, among our random benchmark instances, they constitute the vast majority.

Finally, an additional discretization was created based on CP, in an attempt to improve the results previously obtained. In this strategy, called *Chwa-Extended* (CE), besides all points from CP, we also include all reflex vertices of P in the initial discretization.

An example of each one of the four discretization strategies implemented is presented in Figure 3.5. Notice that, when characterizing vertices of a hole, the terms *convex* and *reflex* have their meaning inverted.

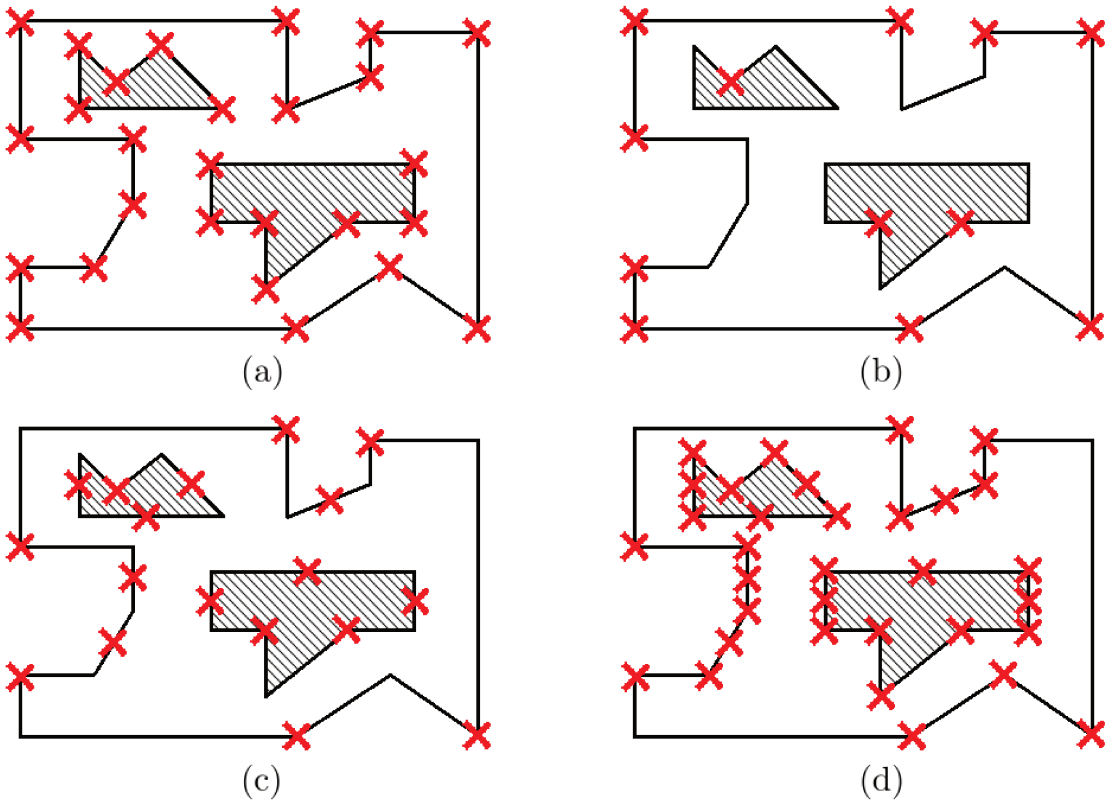


Figure 3.5: Examples of the initial set D when using each one of the following discretization strategies: (a) All-Vertices (AV); (b) Convex-Vertices (CV); (c) Chwa-Points (CP); (d) Chwa-Extended (CE).

Let us now focus on the updating process of the witness set throughout the algorithm. This procedure takes place in two different occasions: while solving an AGPFC instance (in line 5 of Algorithm 3) and when jumping to the next iteration of the AGP algorithm (line 8 of Algorithm 1). In the first case, as presented in Section 3.3, new witnesses are

positioned considering the current solution of AGPWFC(D_f,C). Here, we add to D_f one point placed in the interior of each uncovered region, which (as explained in Section 3.3) is enough to guarantee the convergence of the AGPFC resolution method. See Figure 3.6 for an example.

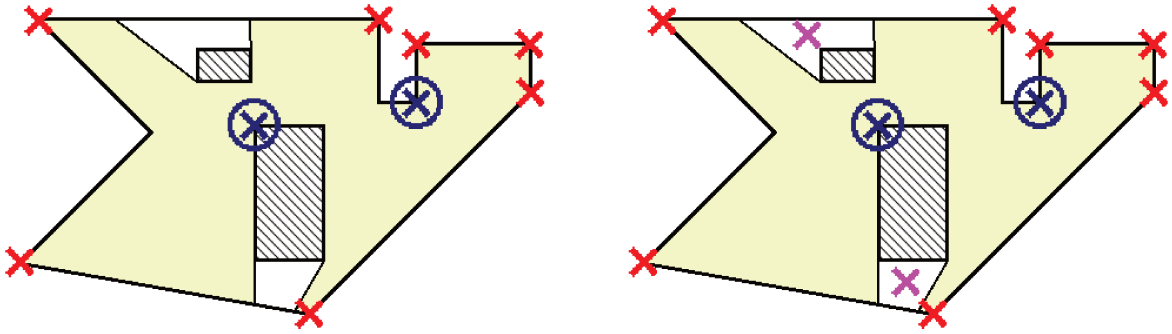


Figure 3.6: Solution of an AGPWFC instance (left); New witnesses chosen (violet) for the next iteration of the AGPFC algorithm (right).

On the other hand, a deeper analysis is necessary when dealing with the update procedure of the main algorithm, because, as discussed at the beginning of this section, the selection of D affects all significant parts of the technique. In a summarized form, the better the choice of a new set of points for inclusion into the witness set, the better the lower and the upper bounds obtained would be and, consequently, the faster the convergence.

In essence, the process consists of adding points from the uncovered regions arising from the solution of the previous AGPW instance. Our first attempt was to imitate the strategy adopted by the AGPFC algorithm and to position one new witness inside each uncovered region. However, our experiments later showed that the inclusion of only these points were not sufficient to lead the algorithm to good performance and convergence. The shortfall was traced to the absence of new points on the boundary of the polygon, which proved to be useful to the evolution of the bounds obtained. Therefore, whenever an edge of an uncovered region is found to be contained on the boundary of the polygon, its vertices and its midpoint are also selected to increment the witness set throughout the iterations. These points along with an interior point of each uncovered region form the whole set M of new witness. Note that the selection of midpoints in this procedure is arbitrary and, in general, can have a better or worse effect than choosing any other non-extreme point of the segment. Figure 3.7 displays an example of how this updated procedure works.

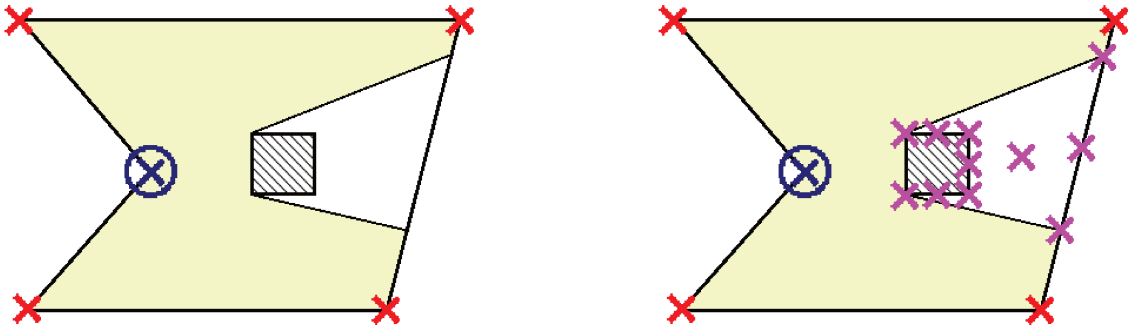


Figure 3.7: Solution of an AGPW instance (left); New witnesses chosen (violet) for the next iteration of the AGP algorithm (right).

3.6 Guard Candidate Management

After constructing the arrangement induced by the set of witnesses D and identifying the corresponding light AVPs, our algorithm is able to build the guard candidate set C. This set must be built in such a way that guarantees that an optimal solution G for AGPW(D) is contained in C. After solving AGPW(D), C is maintained and also used in AGPFC(C), contributing to the discovery of a new upper bound. In this section, we present, in details, how these candidates are selected.

Since our first published work [34], two different discretization strategies for C have been experimented. Both of them follow the idea presented in Theorem 2.1 and construct C using at least one point from each existing light AVP, thereby ensuring that AGPW(D) is optimally solved. In addition, as using only points from Light AVPs may not guarantee the existence of a solution that covers the entire polygon (a necessary requirement for the AGPFC solvability), both strategies also include all vertices of P ($V \subset C$).

Our first strategy, named *Boundary-Guards* (BG), contains, besides the vertices of P, all points from $V_{\mathcal{L}}(D)$ ($C = V \cup V_{\mathcal{L}}(D)$). This discretization was originally the only method used for choosing C in the two papers that describe our algorithm [34, 33].

However, recall that the arrangement does not grow linearly with the number of witnesses in D. This way, in our experiments with large polygons, the tasks which depend on C, such as the computation of visibility between pairs of points, become increasingly time consuming. For example, in a simple polygon with 5000 vertices, we may have to compute more than 100 million visibility tests between candidates and witnesses in a single iteration.

In this context, a new discretization with a lower number of guard candidates was experimented in [17]. This time, the points from $V_{\mathcal{L}}(D)$ were not included in C. The idea, named *Center-Guards* (CG), was to replace the vertices of a given light AVP by an

internal point of it, reducing the number of these candidates by a factor of at least 3. For an example of BG and CG strategies, see Figure 3.8.

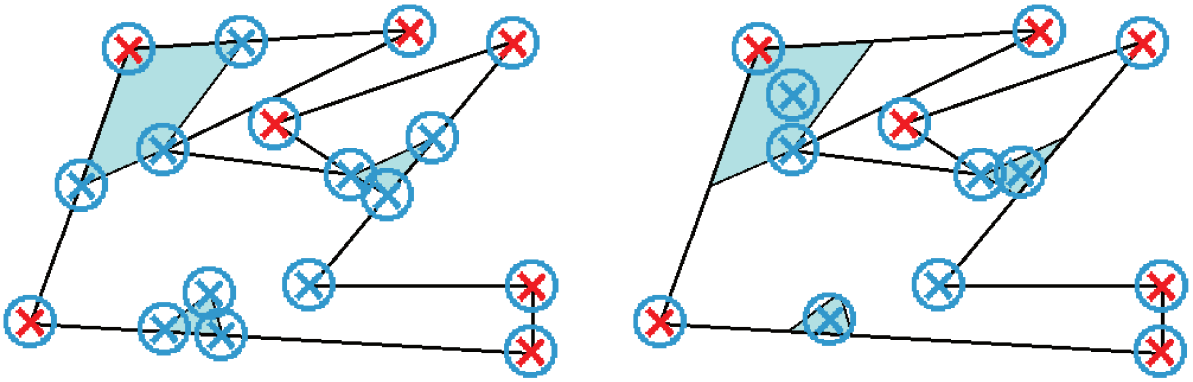


Figure 3.8: Examples of the guard candidate set C when using each of the following discretizations: Boundary-Guards (BG) (left); Center-Guards (CG) (right);

One must though be careful when implementing this second discretization. As seen in the AGPW instance shown in Figure 3.9, two or more Light AVPs can intersect at a point. If we blindly followed the idea of CG discretization, we would choose the centroid of each of the AVPs as candidates, what would lead to a non-optimal solution for the corresponding AGPWFC instance. In this case, the vertex where the intersection occurs should be chosen to be part of C, since it is able to guard all witnesses watched by the light AVPs that are incident to it. If we look more carefully, we will see that Figure 3.9 actually exemplifies a degeneracy case, where the vertex itself is a light AVP and should be identified as an interior point of its own.

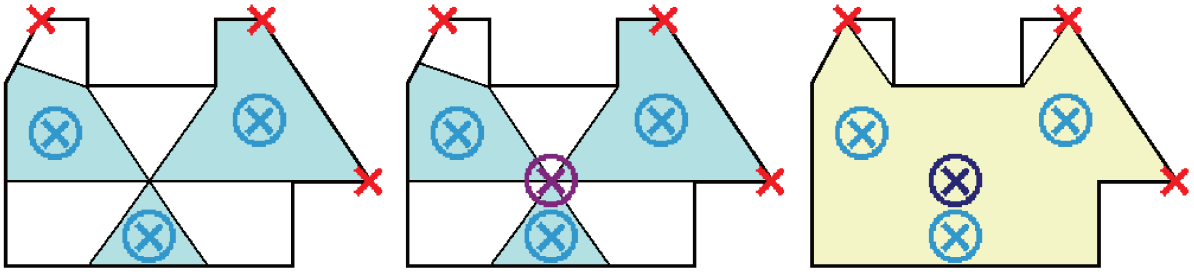


Figure 3.9: AGPW instance where it is not possible to find an optimal solution using only interior points of faces that are light AVPs as guard candidates.

3.7 Resulting Algorithm

Once each of the main steps of the algorithm is understood, we are able to describe how these parts fit together to comprise the complete algorithm. Algorithm 5 sums up how the process as a whole works.

Algorithm 5 AGP(P)

1: $D \leftarrow$ initial witness set {see Section 3.5}
2: Set LB $\leftarrow 0$, UB $\leftarrow n$ and $G^* \leftarrow V$
3: **loop**
4: Solve AGPW(D): set $G_w \leftarrow$ optimal solution and $z_w \leftarrow |G_w|$ {see Section 3.2}
5: **if** G_w is a coverage of P **then**
6: **return** G_w
7: **end if**
8: LB $\leftarrow \max\{\text{LB}, z_w\}$
9: **if** LB = UB **then**
10: **return** G^*
11: **end if**
12: $C \leftarrow V_{\mathcal{L}}(D) \cup V$ (or $C_{\mathcal{L}}(D) \cup V$) {see Section 3.6}
13: $U \leftarrow C_{\mathcal{U}}(G_w)$ {one additional point per uncovered region}
14: $D_f \leftarrow D \cup U$
15: Solve AGPFC(C), using D_f: set $G_f \leftarrow$ optimal solution and $z_f \leftarrow |G_f|$ {see Section 3.3}
16: UB $\leftarrow \min\{\text{UB}, z_f\}$ and, if UB $= z_f$ then set $G^* \leftarrow G_f$
17: **if** LB = UB **then**
18: **return** G_f
19: **end if**
20: $D \leftarrow D \cup U \cup M$ {see Section 3.5}
21: **end loop**

It is important to notice that the set of guard candidates used in the AGPW resolution is actually the set C from the AGPFC(C) instance solved on Line 15. This means that the AGPW resolution is actually the first iteration of the AGPFC algorithm (Algorithm 3). Thus, all information obtained during the solution of AGPW(D) can be reused for AGPFC(C), which improves the performance of the implementation.

Another relevant aspect of this algorithm is that information on bounds may be used throughout the iterations in order to skip unnecessary steps. For instance, if an upper bound UB was found in a previous iteration and a new AGPFC instance is being solved, whose current solution is not lower than UB, then we may stop the AGPFC resolution before obtaining an optimal solution since the upper bound can not be improved.

Figures 3.10, 3.11 and 3.12 illustrate the execution of the AGP algorithm on an orthogonal polygon. Note that, even though the final solution of Figure 3.12 was not feasible for the AGP (due to the uncovered region in picture (f)), it was able to raise the lower bound to 7, proving that the solution found in Figure 3.11 is actually optimal.

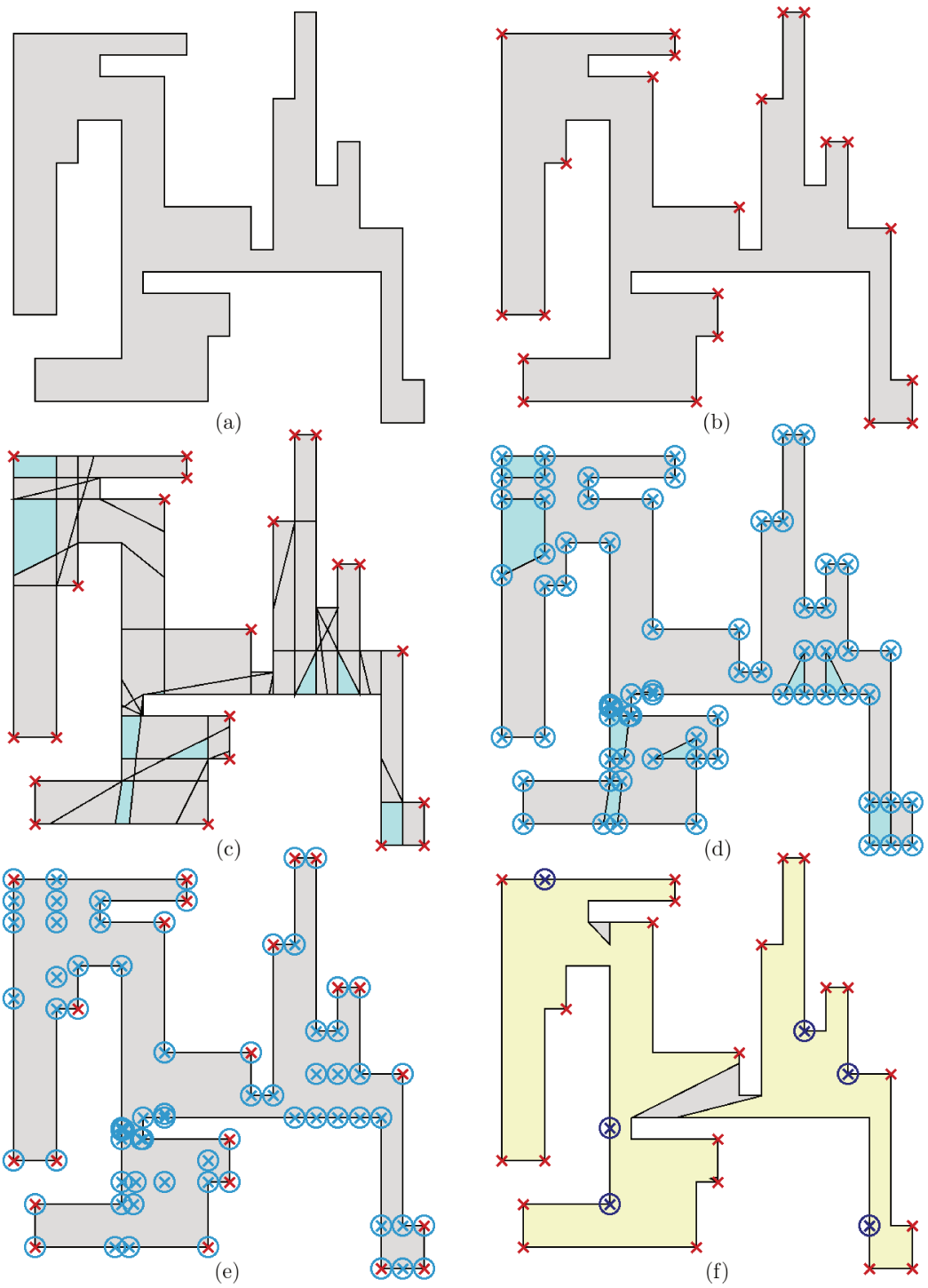


Figure 3.10: **Solving the AGPW (lower bound):** (a) An orthogonal polygon; (b) the initial witness set D (Convex-Vertices); (c) the arrangement Arr(D) and the light AVPs; (d) the guard candidate set C; (e) witnesses and guard candidates forming an instance of AGPWFC(D,C); (f) the solution to AGPW(D).

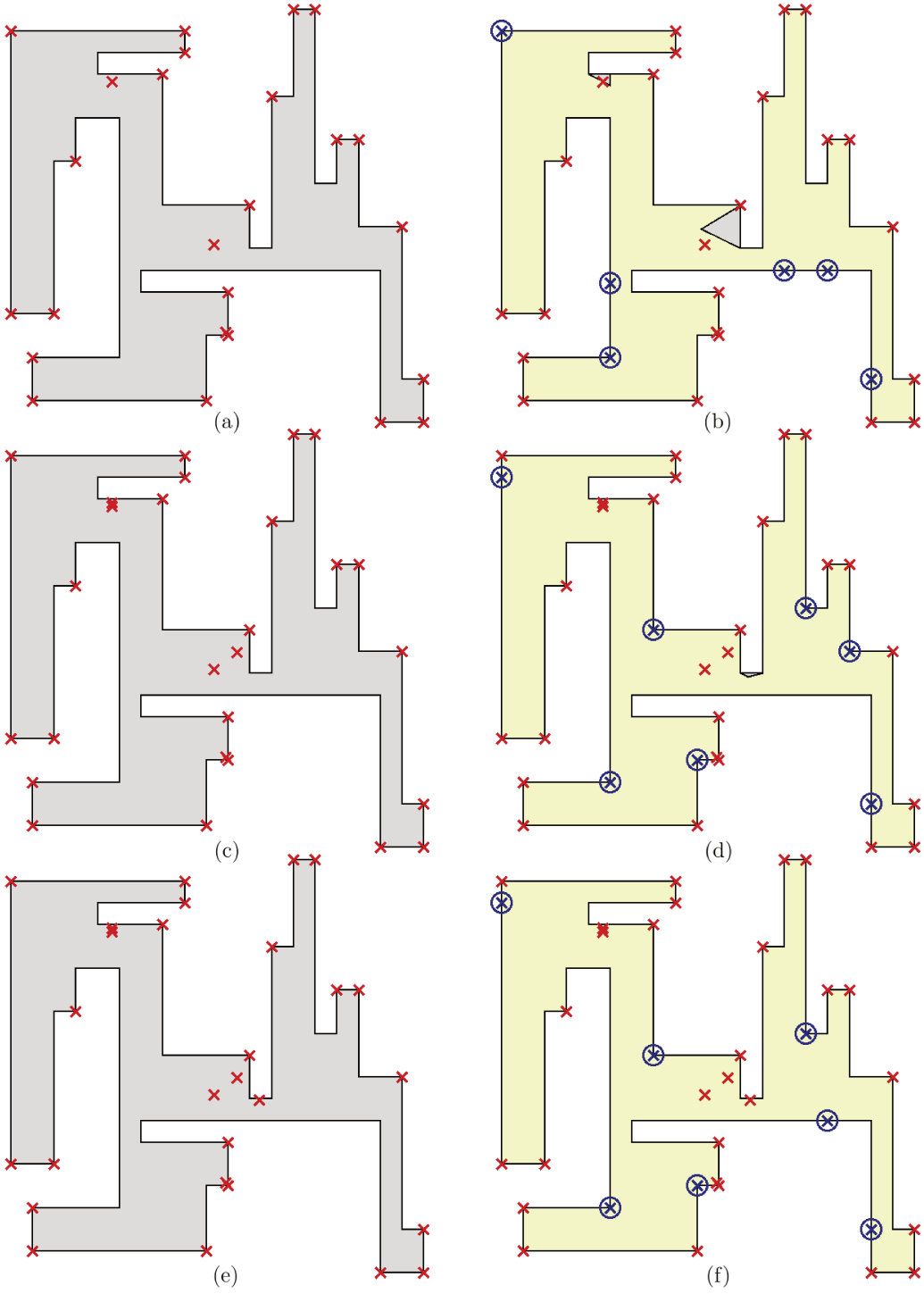


Figure 3.11: **Solving the AGPFC (upper bound):** Iterations of AGPFC(C) resolution. (a,c,e) Updated witness set D_f; (b,d,f) the optimal solution to AGPWFC($D_f, V_{\mathcal{L}}(D) \cup V$); (f) a viable solution to the AGP.

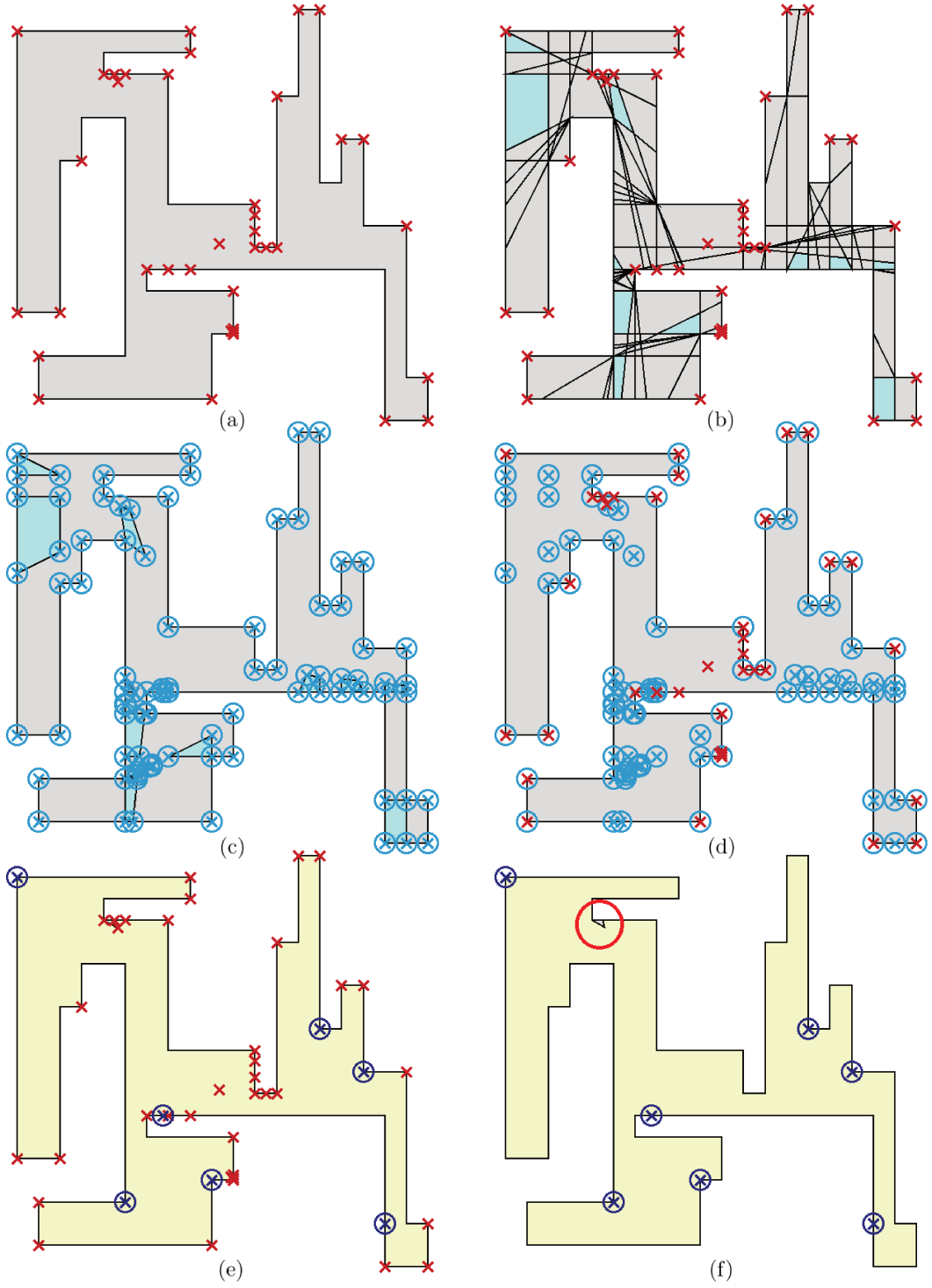


Figure 3.12: **Solving the AGPW (lower bound):** (a) The new witness set D; (b) the arrangement Arr(D) and the light AVPs; (c) the guard candidate set C; (d) witnesses and guard candidates forming an instance of AGPWFC(D,C); (e) the solution to AGPW(D); (f) region not covered by current AGPW solution.

Chapter 4

Implementation and Computational Results

The algorithm described in the last chapter was implemented, enabling us to measure its quality. In this chapter, we explain how the implementation was done and how the experiments were conducted. Following this, we show a summary and an analysis of our results, including a comparison with other state-of-the-art techniques.

4.1 Implementation

Our implementation of the algorithm presented in Chapter 3 went through several changes since its first release. Such modifications, which included the employment of new routines, data structures and decisions, substantially improved the performance of our software. In this dissertation, we highlight the three versions that were the most widely tested and whose results were part of papers we submitted for publication. An analysis of these versions is important to show different stages of the project and how the implementation has matured.

Our first version was completed in late 2012 and was reported on a conference paper [34]. Shortly after that, in the first half of 2013, a second version was produced and described in our full paper [33]. The third (and latest) version was developed during a two-month internship in which I visited the Technische Universität Braunschweig (TUBS), in Germany, under the supervision of Professor Dr. Alexander Kröller. During this visit, I collaborated with researchers from the Algorithms Group (ALG) headed by Professor Dr. Sándor P. Fekete. Their contributions to the research on the AGP include [5, 26, 19] (see Section 1.2 for a brief description). Our new implementation is described in a survey on algorithms for Art Gallery Problems submitted for publication in 2013 [17], as a joint work between researchers from UNICAMP and TUBS. In the subsections that follow, we

describe each of these three implementations.

4.1.1 I1: Implementation for the first paper [34]

Our implementation tested for paper [34], here called I1, was straightforward, effective and reliable, even though the code was not deeply investigated for possible optimizations. The major drawback of I1 was that it was not capable of solving the AGP on polygons with holes. Despite this, it was tested in a great variety of polygon classes, where we were able to prove its robustness.

This version, as well as the subsequent two, was coded in the `C++` programming language and used the Computational Geometry Algorithms Library (CGAL) [11] to benefit from visibility operations, arrangement constructions and other geometric tasks. In addition, I1 employed the XPRESS Optimization Suite [36] library to solve the integer programs that model the AGPWFC (SCP) instances, as discussed in Section 3.4.

Since the algorithm is iterative, some care was necessary to avoid recomputation of certain types of structures, especially those related to the CGAL library, which normally have considerable computational cost. This situation happens, for example, when computing visibility polygons of witnesses or candidates. Firstly, notice that, besides being important for computing the arrangement and verifying the coverage of P, in our code, visibility polygons are also employed in the construction process of the ILP matrix to test visibility between guard candidates and witnesses. This task is performed by executing point location routines from CGAL to verify if a given witness is (or not) inside the visibility polygon of a guard candidate. Knowing this, to be able to properly construct the ILP matrix, our technique needs to obtain the visibility polygons of all current guard candidates. However, during the iterations, the set C of guard candidates is variable and not necessarily incremental, changing according to $\mathrm{Arr}(D)$. This means that points may be removed from C and then, in a future iteration, be included again. Thus, to avoid recomputing visibility polygons, a hash table containing those already calculated was implemented. Furthermore, as this table also stores visibility data of witnesses, our code could take advantage of cases where points belong, at the same time, to sets D and C.

Another situation where the management of computed structures is of great importance is in the arrangement construction. Recall that our algorithm computes optimal solutions for the AGPW using a guard candidate selection procedure based on $\mathrm{Arr}(D)$. At each iteration of our algorithm, this arrangement must be updated following the changes in the witness set D (see Section 3.5). Since D is affected only by the addition of new witnesses, in order to obtain the current arrangement, our code just needs to compute the overlay between the previous and the one induced solely by the new witnesses. This idea saves a lot of time because it avoids the reconstruction of the whole arrangement, which

gets more and more complex over the iterations.

Moreover, an implementation detail that is worth mentioning is the use of exact arithmetic as provided by CGAL. In this exact mode, the coordinates of all points are represented as quotients of integers. While the vertices of the input polygon usually have small numerators and denominators, the same can not be ensured about the points computed during the algorithm. These new points, which are derived from operations like intersection of lines or centroid calculations, have representations that keep growing larger. As a consequence, any arithmetic operation with such numbers becomes very time consuming, severely deteriorating the algorithm's overall performance. To brake or even cancel the continuous increase of these representations, our solution was to modify our implementation of the witness update method. Our initial strategy, which consisted in including the centroids of triangles or midpoints of diagonals from uncovered regions, was amended by choosing simpler witnesses. Since (for the correctness of our algorithm) it is irrelevant which point within an uncovered region is chosen, the new idea was to truncate the representation of these points as much as possible, while ensuring that they remain within the given region. Preliminary experiments showed that this technique is able to dramatically reduce the size of representations and to ensure a higher stability of our code over the iterations. Figure 4.1 presents an example of this simplification procedure.

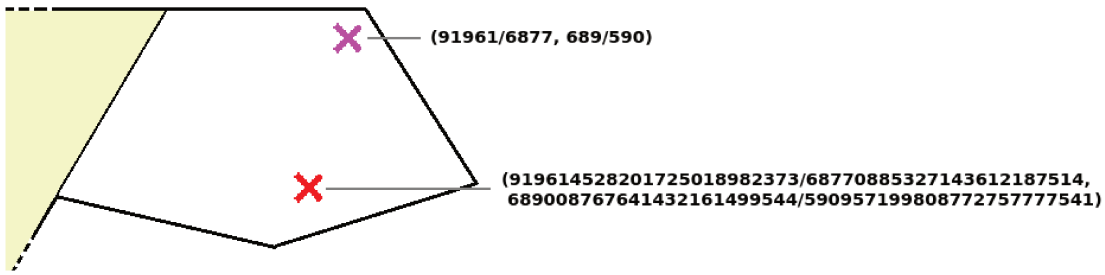


Figure 4.1: Example of the fraction simplification procedure where an initial witness (red) placed in the interior of an uncovered region is turned into another with a much smaller representation (violet).

In [34], I1 was tested on more than 1400 instances with up to 1000 vertices and obtained optimal solutions for all of them, something not achieved by any other AGP solver presented in literature at that time.

4.1.2 I2: Implementation for the second paper [33]

After I1, we focused our attention in producing a code capable of solving polygons with holes. Furthermore, important optimizations were made to the code, greatly improving

the performance of the solver and leading to a second version I2, which was reported in [33]. Some of the most effective changes consisted in:

1. cutting short the AGPFC procedure (Algorithm 3) whenever the general upper bound found for the AGP had the same cardinality as the current AGPWFC solution, allowing it to skip to the next iteration of Algorithm 5;

2. reversing the point of view of visibility testing from the perspective of the guards to that of the witnesses (fewer in number, in our approach), avoiding the computation of many visibility polygons and reusing those already required for setting up the arrangement. Notice that the ILP matrix construction remained the same and changed little on how the algorithm works;

3. caching visibility information based on pairs of points already tested, reducing the number of point location operations. To achieve this, we implemented a new hash table where the key corresponds to a pair of points and the value is true if the two points are visible to each other and false otherwise. As a result, in cases of recomputation, a simple table search became enough to verify visibility between pairs of points, instead of being necessary to repeat a point location operation between a visibility polygon and a point;

4. removing redundant lines and columns from the original SCP matrix (see Section 3.4 for details);

5. applying Lagrangian Heuristic to find viable solutions for SCP instances, helping the ILP solver to find optimal solutions faster (see Section 3.4 for details);

6. reusing information from previous iterations. For instance, if a lower bound LB for the AGP instance has been established, Algorithm 2 can be halted when solving a new AGPW(D) instance as soon as a primal solution with cardinality LB is found for the corresponding SCP instance in line 4 (for instance, using the Lagrangian Heuristic). This follows from the fact that LB was obtained by solving an AGPW(S) instance, for some $S \subset D$. Another interesting situation happens when solving an AGPWFC instance inside the iterative algorithm for the AGPFC (Algorithm 3). Since, in this procedure, witnesses are added and never removed, we can guarantee that the solution of the previous AGPWFC instance is a lower bound for the next instance.

As seen later in Section 4.4.1, I2 represented a great improvement in performance when compared to I1. In [33], polygons with up to 2500 vertices were tested and solved to optimality in a matter of minutes.

I2 is currently freely available in the website of our project [16], to be used by the scientific community. However, as the ILP solver employed in I1 (XPRESS) is a proprietary software, we included in I2 the possibility of using a free solver to facilitate further tests and comparisons of our techniques with new algorithms. The free package of choice was the GLPK (GNU Linear Programming Kit) [24], a set of routines written in the `ANSI C` programming language and organized in the form of a callable library. Although switching to a non-commercial solver is expected to lead to some degradation in performance, it remains a handy means for benchmarking future research on the subject.

4.1.3 I3: Implementation for the third paper [17]

In the second half of 2013, I was invited by Professors Alexander Kröller and Sándor Fekete to study for two months at TUBS and assist in experimenting and writing a Survey on AGP algorithms. During this period, I took the opportunity to make changes in the code of I2 and improve the previous solver, giving rise to I3. In this context, the future visibility package of CGAL, developed during the Google Summer of Code of 2013 under the supervision of Dr. Michael Hemmer, which is also part of the research group of TUBS, was used in our implementation. Moreover, after learning more about the different kernels of CGAL, which defines how coordinates and structures are constructed and represented, the previous Cartesian kernel was replaced by the lazy-exact kernel. The lazy-exact kernel guarantees the viability of the AGP solutions obtained, but, whenever possible, avoids the computation with exact arithmetic, which saves time. To implement the improvements described in this paragraph, we benefited from the help by researchers from TUBS, who have vast knowledge and a long experience with the CGAL library.

Besides the changes on the geometric side, some other ideas were implemented. One of those was to postpone the computation of an upper bound (solving the AGPFC) to the time that a good lower bound and, consequently, a "good" set of guard candidates is obtained. This way, it is expected that the number of AGPFC instances solved during the execution is severely reduced, which also decreases the overall running time.

Furthermore, the hash table created in I2, with the results of visibility between pairs of points already tested, was modified (see Item 3 of the list of improvements presented in Section 4.1.2). In this new version, the key of the hash table is a guard candidate g and the associated value is a vector containing the visibility test result (true or false) between g and the t first witnesses included in D, where t is the size of D in the last iteration that g was a candidate. This new storage mode was only possible because, as explained before, a witness is never removed from D, making it easy to maintain an index to each one of them. As a result, a drastic decrease was observed in the number of hash operations executed during the construction of the ILP matrix.

Other features implemented in I3 included the possibility of using CPLEX as an ILP solver option and the creation of the guard discretization strategy called Center-Guards (CG), where a smaller number of candidates is initially chosen (see Section 3.6).

The implemented optimizations contributed to a more robust code, as it is possible to see in Section 4.4.1. For an illustration, I3 is now able to solve simple polygons with up to 5000 vertices in less than 20 minutes, while I2 had difficulty in solving polygons half that size.

4.2 Computational Environment

As shown in the previous section, three versions of our algorithm were developed during this Master's project. The first two of them, I1 and I2, were experimented for our conference [34] and full [33] papers, respectively. The tests were conducted in the Laboratory of Optimization and Combinatorics (LOCo) at UNICAMP. All the experiments were performed on standard PCs featuring an Intel® Core™ i7-2600 at 3.4 GHz, 8 GB of RAM and running `GNU/Linux 3.2.0`, where the solvers were linked with versions 3.9 of CGAL, 7.0 of XPRESS and 4.52 of GLPK. To achieve accurate time measurements, all tests were run in isolation, i.e., no other process was being executed concomitantly. Lastly, each process was given a time limit of 60 minutes, after which, the instance was considered unsolved and the program was terminated. To facilitate mentioning this set of configurations and characteristics throughout this chapter, we refer to this environment as M1.

On the other hand, for the experiments performed while writing the survey on algorithms for the AGP [17], the computational environment of the Algorithms Group (ALG) at TUBS was the one chosen. For this survey, all of our three implementations (I1, I2 and I3) were tested. This time, the programs ran on PCs with an Intel® Core™ i7-3770 at 3.4 GHz and 16 GB of RAM. The operating system was a `GNU/Linux` distribution with kernel version 3.11.0. and the AGP solvers were now linked with releases 4.0 of CGAL, 12.5 of CPLEX and again 7.0 of XPRESS. As in M1, all tests were done in isolation, but now each execution had a run time limit of only 20 minutes. The environment described in this paragraph is called M2.

4.3 Instances

A specific implementation may have different levels of difficulty in solving a polygon depending on its own characteristics or even on the input's particulars. Based on this, we sought to test our technique with an extended experimentation testbed, from various

sources, comprised of polygons of multiple classes and sizes, in order to be able to stress the algorithm's robustness. In total, more than 2800 instances were collected.

Below we present a detailed description of the 6 classes of polygons employed in our experimentation.

4.3.1 Simple

This class refers to random simple polygons (without holes). In our experiments, simple instances were selected from two sources: Couto et al. [14] and Bottino et al. [10].

Those from Couto et al. were generated by a special purpose procedure available in CGAL. Essentially, this procedure starts by distributing the vertices of the polygon uniformly in a given rectangle and applies the method of elimination of self-intersections using 2-opt moves. In total, we experimented 630 instances from this source, with polygons ranging from 20 to 5000 vertices. For each existing size, 30 instances were considered.

In contrast, the instances obtained from Bottino et al. are based on the Delaunay triangulation of a set of points randomly distributed in a square region. From this source, we tested all the 250 polygons available, which contain between 30 and 60 vertices. In addition, except for 170 polygons of 30 vertices used for preliminary tests in [10], all other instances are subdivided by size in groups of 20 polygons each.

4.3.2 Orthogonal

This class comprises random polygons generated respecting the property that all edges are parallel to the x-axis or y-axis. Once again, the works of Couto et al. [14] and Bottino et al. [10] were used as sources.

In both works, the instances were generated devoid of collinear edges on an $n^2 \times n^2$ unit square grid, in accordance to the method described in [32]. The difference between them lies in the code employed for generating the polygons and in the size range used. Bottino et al. resorted to a code supplied by the authors of [32] and produced 80 polygons with sizes 30, 40, 50 and 60 (20 polygons per size). Meanwhile, Couto et al. developed their own program and generated 630 polygons from 20 to 5000 vertices, 30 for each size.

4.3.3 Simple-simple

The simple-simple class comprises the instances where the boundary and all the holes are simple polygons generated with the method by Couto et al. described in Section 4.3.1. Two similar implementations were employed to produce these instances. In the first one, denoted here *GB*, the process works as follows: after creating the outer boundary for a random simple polygon with holes, consider that we are left with v vertices to be

distributed among h holes. After generating a random uniform partition of v into h parts, we iteratively generate the h holes in the following way. At each iteration, we randomly select a point in the interior of the polygon around which we center an isothetic square entirely contained inside the polygon. This square is then stretched in each of the four orthogonal directions, chosen in random order, by λD where λ is a stretch factor randomly picked from the interval $[0.25, 0.75]$ and D is the maximum elongation within the polygon in that direction. A hole is then created, within this placeholder rectangle, having its number of vertices chosen from one of the unused parts of the previously mentioned random partition of v. Here, for an instance with a total of n vertices, $n/4$ of them were assigned to the outer boundary and $3n/4$ of them distributed among $n/10$ holes. The range $[100, 500]$, with step size 100, was used for the number of vertices of the polygons and a total of 30 polygons of each size were produced.

On the other hand, in the second technique, denoted *GD*, a maximal placeholder rectangle is constructed at each iteration so that the chosen random interior point c is one of its vertices. After this, the algorithm randomly selects a new hole size and generates a simple polygon to be inserted in the randomly stretched placeholder. One peculiarity of this generator is that two holes are allowed to intersect, in which case, they merge into a single hole. Similarly to the previous generator, for an instance with a total of n vertices, the outer boundary has roughly $n/4$ of them, while the remainder are distributed among the resulting $n/10$ holes. With GD, 30 instances were created for each of the following sizes: 200, 500, 1000, 2000 and 5000.

4.3.4 Ortho-ortho

This class refers to instances where boundary and holes are product of the orthogonal generator developed by Couto et al. (see Section 4.3.2). To populate this class, we used *GD* technique, already employed in the generation of simple-simple polygons. This time, a total of 240 instances were generated and equally divided in 8 different sizes, ranging from 100 to 5000 vertices.

4.3.5 von Koch

This class of polygons was created in Couto et al.'s work [14] based on a modified version of the von Koch curve. The fractal is generated, starting with a square, by iteratively replacing randomly chosen edges as shown in Figure 4.2, where $\bar{ar} = \bar{st} = \bar{ub}$ and $\bar{sr} = \bar{tu} = 3/4\bar{ar}$. The operation is repeated until the number of vertices of the polygon reaches the desired size. In our experiments, we tested a total of 540 von Koch instances, from 20 to 5000 vertices. For each size considered, 30 instances were collected.

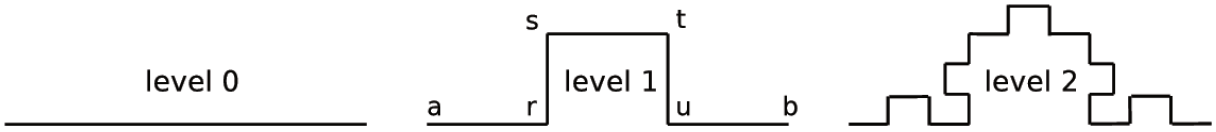


Figure 4.2: Levels of edges based on the modified von Koch curve.

4.3.6 Spike

The spike polygons were introduced in the work of Kröller et al. [26]. To generate them, star-shaped polygons are created consisting of a central octagon, to which random spikes that can be guarded from the octagonal center are added. Each spike instance is the overlay of several such star-shaped pieces. Due to its generation method, the optimal solution for a spike polygon is always known and consists in placing one optimal guard in the center of each octagonal created. Observe that the algorithm employed allows for the existence of holes, which may increase the challenges of solving this type of instance. From this class, 210 instances were selected and divided in subgroups by size, which vary between 60 and 5000 vertices. Each of these subgroups consists of 30 polygons.

Figure 4.3 presents an example of each one of the previously described classes of polygons. In addition, Table 4.1 summarizes how the collection of experimented instances is composed.

4.4 Results and Analysis

After describing our code, testing environments and instances, it is now time to discuss our results. In this section, we make an analysis on the performance and robustness of the algorithm outlined in Chapter 3 and show how the solution has evolved through time. In this context, we also present the parameters employed and how they affect the overall performance of our implementation.

Even though the latter paper [17] introduced the implementation of I3, which is an optimization of I2 and, consequently, of I1, results prepared for earlier papers [34, 33] (using only I1 or I2) are also shown in this section to characterize the behavior of the algorithm. The reason for this is the limitation on the number of experiments performed in [17], since the survey had to deal with multiple versions of algorithms. As an example of this limitation, only 5 out of the 21 existing sizes of polygons (200, 500, 1000, 2000, 5000) were experimented in the survey. Furthermore, it was not possible to test all 4 strategies for the initial witness set discretization (described in Section 3.5).

In this context, so as to facilitate our narration, we say that a set of results was obtained with the experimental configuration $E(Ix, My)$ if the corresponding experiments

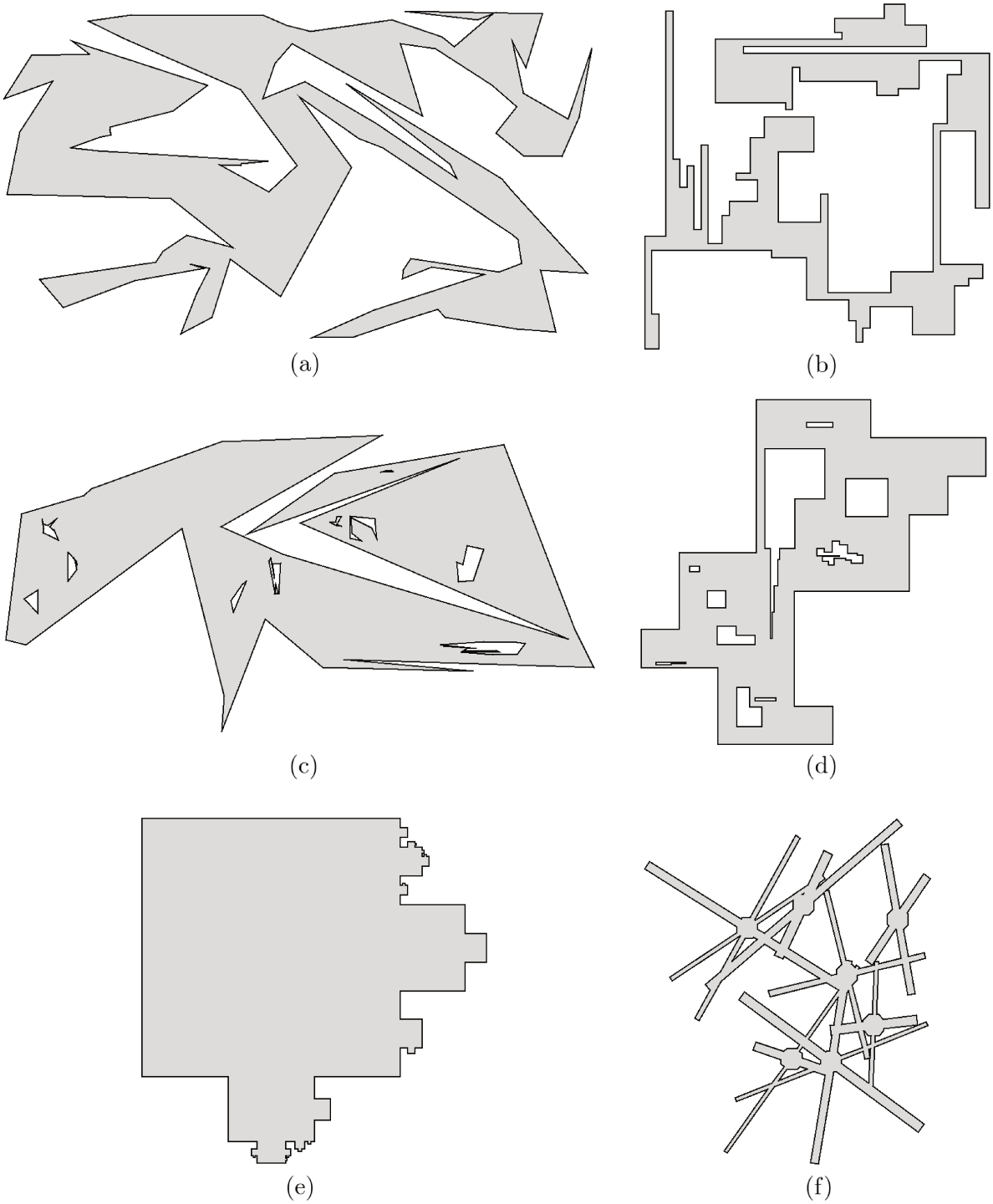


Figure 4.3: Examples of instances from different classes. (a) Simple; (b) Orthogonal; (c) Simple-simple; (d) Ortho-ortho; (e) von Koch; (f) Spike.

Class	Source	Sizes experimented	Instances per size	Number of instances
Simple	From [10] (preliminary)	30	170	170
	From [10]	30, 40, 50, 60	20	80
	From [14]	20, 40, 60, 80, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1250, 1500, 1750, 2000, 2250, 2500, 5000	30	630
Orthogonal	From [10]	30, 40, 50, 60	20	80
	From [14]	20, 40, 60, 80, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1250, 1500, 1750, 2000, 2250, 2500, 5000	30	630
Simple-simple	From [33] (GB)	100, 200, 300, 400, 500	30	150
	From [33] (GD)	200, 500, 1000, 2000, 5000	30	150
Ortho-ortho	From [33]	100, 200, 300, 400, 500 1000, 2000, 5000	30	240
von Koch	From [14]	20, 40, 60, 80, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1250, 1500, 2000, 5000	30	540
Spike	From [26]	60, 100, 200, 500, 1000, 2000, 5000	30	210
Total				2880

Table 4.1: Summary of instances

were performed with implementation Ix in environment My. Not all experimental configurations $E(Ix, My)$ were tested with the same groups of instances and only a small portion of the data collected is actually displayed in this dissertation.

4.4.1 General Results

As said before, our method was examined in three opportunities [34, 33, 17]. In [34], for example, $E(I1, M1)$ was tested and, for every one of the 1440 hole-free polygons (with hundreds of vertices) from various classes gathered from the literature, optimal solutions were attained in just a few minutes of computing time. In our second paper [33], a great improvement was seen in the algorithm implementation (I2), which proved being capable of solving polygons with more than 2000 vertices. In this opportunity, 2440 instances were tested with $E(I2, M1)$, including polygons with holes, and I2 achieved an *optimality rate* of more than 98%. Just to clarify, the optimality rate here means the percentage of instances from the total in which the optimality gap equaled zero and, therefore, a proven optimal solution was obtained.

Finally, in the AGP Survey [17], I1, I2 and a new version I3 were experimented on 900 instances in environment M2. The joint experimentation provided a direct comparison and verification of the improvement occurred during the Master's project. In this occasion, I3 showed to be very successful, being able to optimally solve 768 polygons, including instances with 5000 vertices, in runs of less than 20 minutes. Table 4.2 displays the optimality rates achieved by I1, I2 and I3.

The results in Table 4.2 evince two big steps in our headway. From I1 to I2, besides a considerable improvement in the optimality rate, we became able to solve polygons with holes, greatly increasing the range of treatable classes. Subsequently, from I2 to I3, a remarkable performance improvement was conquered, as evidenced by the resolution of polygons of 5000 vertices. These polygons have twice the size of the previous largest instances already treated by AGP solvers, fact achieved by I2 in [33].

In order to confirm this analysis, we collected information about the time necessary to find optimal solutions. Table 4.3 shows the average time needed to solve simple, orthogonal and von Koch polygons, considering only instances where optimal solutions were found by all three implementations. From this table, one can see that the average time of I2 can be about 5 times smaller than I1, as verified in results of von Koch polygons with 500 vertices. The difference is even greater when analyzing I2 against I3, which is capable of solving, on average, orthogonal polygons of size 1000 almost 22 times faster than I2. But which of the implementation upgrades (presented in Section 4.1.3) is responsible for this great improvement? To help answering this question, Figure 4.4 brings the time spent by the last two implementations in each of the major tasks of our algorithm, when

Class	**Source**	n	**Optimality Rate (%)**		
			I1	**I2**	**I3**
Simple	From [14]	200	100.00	100.00	100.00
		500	100.00	100.00	100.00
		1000	96.67	100.00	100.00
		2000	6.67	50.00	100.00
		5000	0.00	0.00	100.00
Orthogonal	From [14]	200	100.00	100.00	96.67
		500	100.00	96.67	93.33
		1000	100.00	100.00	100.00
		2000	70.00	90.00	100.00
		5000	0.00	0.00	93.33
Simple-simple	From [17] (GD)	200	-	100.00	100.00
		500	-	83.33	100.00
		1000	-	0.00	100.00
		2000	-	0.00	46.67
		5000	-	0.00	0.00
Ortho-ortho	From [33]	200	-	96.67	100.00
		500	-	83.33	100.00
		1000	-	3.33	96.67
		2000	-	0.00	33.33
		5000	-	0.00	0.00
von Koch	From [14]	200	100.00	100.00	100.00
		500	96.67	100.00	100.00
		1000	46.67	100.00	100.00
		2000	0.00	0.00	100.00
		5000	0.00	0.00	0.00
Spike	From [26]	200	-	100.00	100.00
		500	-	100.00	100.00
		1000	-	96.67	100.00
		2000	-	96.67	100.00
		5000	-	0.00	100.00

Table 4.2: Optimality Rates of I1, I2 and I3 in environment M2.

Class	Source	n	Average Time (sec)		
			I1	**I2**	**I3**
Simple	From [14]	200	7.31	3.63	0.75
		500	67.81	32.82	2.96
		1000	358.97	158.73	9.18
Orthogonal	From [14]	200	4.10	2.72	0.37
		500	30.06	19.61	1.49
		1000	189.41	111.40	5.22
von Koch	From [14]	200	11.12	3.54	1.20
		500	158.53	31.88	7.80
		1000	767.01	186.49	52.81

Table 4.3: Average Time of I1, I2 and I3 in environment M2.

dealing with simple polygons of 1000 vertices.

In the chart from Figure 4.4, it is easy to see that I3 has advantage in all of the verified stages, but mainly in three of them. From the visibility computation results, it was possible to prove the high quality of the new algorithm employed to execute this specific task, which will be part of future versions of CGAL. On the other hand, the enhancement in the ILP procedure can be credited possibly to the use of CPLEX instead of XPRESS but specially to the tactic designed to reduce the number of AGPFC resolutions to fewer iterations. Finally, the impressive difference verified in the Matrix setup time must be assigned to the new hash table with visibility information (see Section 4.1.3 for details), which greatly reduced the number of table searches.

After reviewing the evolution of our implementation, let us focus on the overall behavior of our algorithm on instances from all available classes. To perform this analysis, we display Table 4.4, which has information obtained with $E(I3, M2)$. In this table, we can find results of optimality rate, average number of guards in optimal solutions, average number of iterations required to achieve convergence and average time spent for each subgroup of polygons.

Clearly, some classes of polygons seem easier than others: orthogonal polygons tend to take less time than simple or von Koch polygons. The latter are clearly the hardest ones: within the average time taken to solve a von Koch instance of 1000 vertices, we were able to solve simple polygons of double that size. In the same vein, except for spike polygons, it is evident that polygons with holes are at least an order of magnitude harder than their hole-free counterpart.

Besides using average time information as a measurement of the difficulty in solving different groups of instances, we can also analyze performance in a broader manner using a boxplot chart, as presented in Figure 4.5. The chart provides run time information on polygons of 1000 vertices when using E(I3,M2). One can see that, in addition to usually

Class	Source	n	Optimality Rate (%)	Guards	Iterations	Time (s)
Simple	From [14]	200	100.00	25.97	3.83	0.75
		500	100.00	63.20	4.60	2.96
		1000	100.00	126.57	5.57	9.95
		2000	100.00	249.23	7.33	48.29
		5000	100.00	624.37	9.13	538.26
Orthogonal	From [14]	200	96.67	28.90	4.00	0.37
		500	93.33	73.54	5.00	1.51
		1000	100.00	147.90	5.73	5.22
		2000	100.00	296.37	7.10	21.18
		5000	93.33	743.39	7.71	123.07
Simple-simple	From [33] (GD)	200	100.00	24.90	5.07	6.58
		500	100.00	59.13	6.53	61.37
		1000	100.00	118.20	8.53	373.15
		2000	46.67	234.43	8.86	1133.13
		5000	-	-	-	-
Ortho-ortho	From [33]	200	100.00	24.97	5.90	9.37
		500	100.00	62.17	8.10	101.29
		1000	96.67	127.14	9.79	402.61
		2000	33.33	259.20	10.50	1182.93
		5000	-	-	-	-
von Koch	From [14]	200	100.00	14.00	2.97	1.20
		500	100.00	32.23	2.83	7.99
		1000	100.00	56.80	3.03	59.19
		2000	100.00	122.37	4.07	329.33
		5000	-	-	-	-
Spike	From [26]	200	100.00	6.87	2.03	0.47
		500	100.00	9.77	2.10	2.03
		1000	100.00	11.97	2.13	22.20
		2000	100.00	15.07	2.07	14.93
		5000	100.00	30.00	2.03	32.85

Table 4.4: Optimality rate, average cardinality of optimal guard sets, average number of iterations and average time spent results with E(I3,M2).

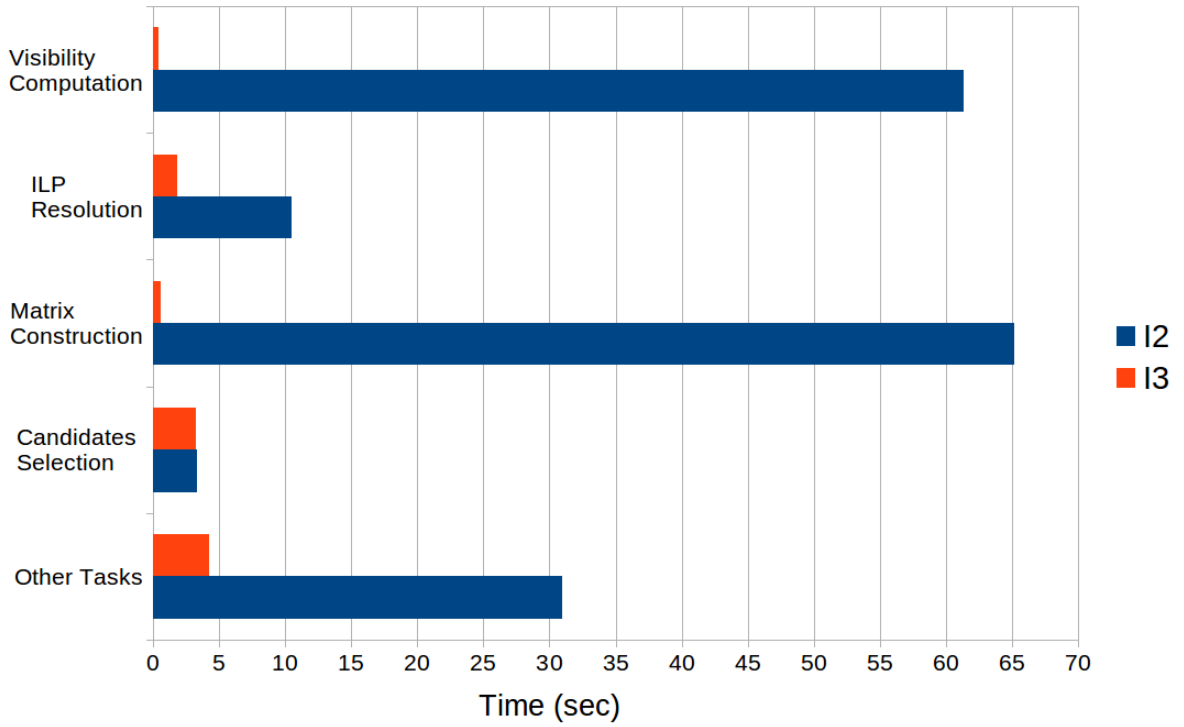


Figure 4.4: Average time spent in each of the major tasks of the technique when solving simple polygons of 1000 vertices.

having longer average time results (see Table 4.4), the resolution of simple-simple and ortho-ortho polygons also presents a larger time variance in comparison to other classes. As an example, while there is an ortho-ortho polygon of 1000 vertices that can be solved in about 50 seconds, there is another from the same group which practically requires the full 20 minutes available to achieve a zero optimality gap. In contrast, when solving simple and orthogonal instances, there is a small time range and, as a consequence, the average is closer to the median and also to the quartiles.

Other interesting conclusions can be drawn by analyzing Table 4.4. See that the number of guards in the optimal solutions seems to grow linearly with the size of the polygons. This somehow suggests that the generators worked well and were consistent, keeping the same relative complexity in instances regardless of their sizes. Also surprising is the sublinear growth of the number of iterations required to achieve optimal solutions. In the case of simple polygons, while the size increased 25 times (200 to 5000), the number of iterations less than tripled (3.83 to 9.13). This information attests to the quality of the algorithm, independently of its implementation, showing that it can quickly converge to optimal solutions.

In addition, we must highlight the special behavior of the algorithm when solving spike

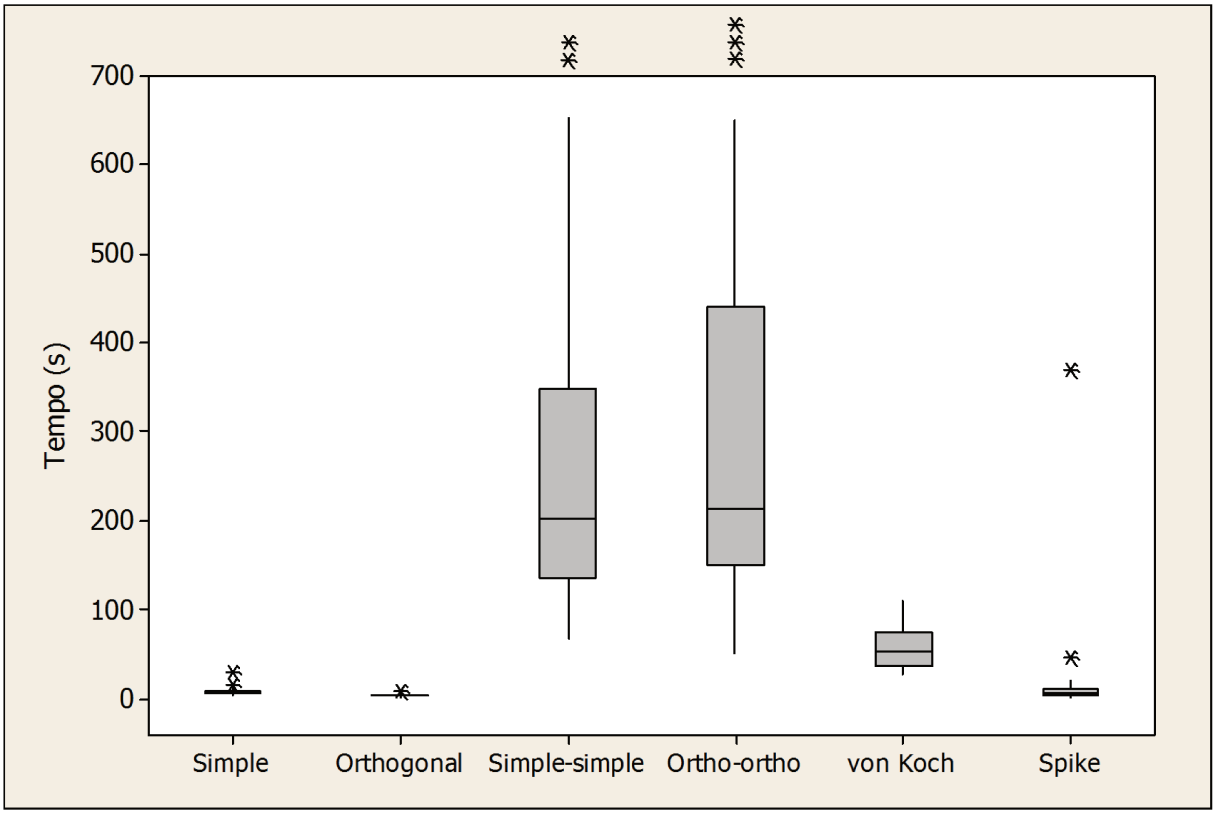


Figure 4.5: Run time information of E(I3,M2) for polygons of 1000 vertices.

polygons. If we increase the size of these instances, the number of iterations required remains constant. Moreover, it is clear that the resolution time of the spike instances grows very slowly. As a consequence, with the time used to solve the entire subgroup of spike polygons of 5000 vertices, we can not solve a single instance of the same size from classes ortho-ortho or simple-simple. The explanation for this situation lies in the generation method of these instances. As seen in section 4.3, a spike polygon is generated so that each guard of the optimal solution is within an octagon, which is then cut by several thin rectangles called spikes. In our resolution method, when using our main discretization strategies (whose results are discussed in Section 4.4.2), the light AVPs are normally located exactly inside these octagons, making it very easy to find the necessary set of guard candidates.

Even though our work focuses in obtaining proven optimal solutions to the AGP, in practical cases, solutions with slightly higher cardinality are considered good enough. In this context, we created Table 4.5 to help us understand how good is the quality of the AGP solutions when the maximum time limit is reached. Table 4.5 shows the percentage of instances where the final optimality gap was less than or equal to an integer d, which

varies from 0 to 4. In case $d = 0$, we again have the percentage of proven optimal solutions obtained by I3 in M2. As for the case of $d = 1$, for example, in addition to considering all instances with proven optimal solutions, we also included those whose final number of guards was only one unit above the lower bound computed.

To simplify our analysis of Table 4.5, cases where the value in column d is higher than in column $d - 1$ are marked in bold. We see that changes according to d happen in 3 different types of polygon: orthogonal, simple-simple and ortho-ortho. In the specific case of the orthogonal class, our method obtained solutions with gap less than or equal to 1 for all instances. Meanwhile, for simple-simple and ortho-ortho instances, we observe a gradual improvement when the optimality gap is continuously relaxed (up to $d = 4$).

Finally, it is worth mentioning that $d = 4$ actually corresponds to the maximum optimality gap found in our experiments if we consider only instances where there was enough time for computing at least one viable solution (and one upper bound). This set of data suggests that our method for discovering viable guard sets is able to achieve results quite close to the optimum of the AGP. At the same time, the large amount of instances with no upper bound obtained within 20 minutes also indicates that implementation I3 takes too much time to find such solutions, which can be partially attributed to our decision to postpone the AGPFC resolution and, consequently, the computation of upper bounds (see Section 4.1.3).

4.4.2 Witness Management

As explained in Section 3.5, four different discretization strategies were developed to construct the initial witness set D. In our first paper [34], all four strategies were tested with $E(I1, M1)$ and results revealed significant changes in performance depending on the option adopted. In Tables 4.6 and 4.7, the average number of iterations and time spent (respectively) are displayed.

Table 4.6 reports some relevant information. Among them, we can see that, for the Simple and Orthogonal classes, CP usually needs a lower number of iterations to find optimal solutions than other discretization options. The quality of CP was somewhat expected, since it corresponds to a placement of witnesses that ensures the whole coverage of a witnessable polygon (see Section 3.5). However, an unforeseen fact is that CP was even able to achieve better iteration results than CE, which constructs a superset of the one initially produced by CP. On the opposite side, CV, which chooses only convex vertices of P, obtained, in most cases, the worst results. When considering solely von Koch polygons, a lot changes and AV becomes the option that needs less iterations.

If we consider now the average time results displayed in Table 4.7, we are led to new conclusions. As expected, CP remains with the best results when treating simple

Class	**Source**	n	**Percentage of instances with optimality gap less than or equal to**				
			0	**1**	**2**	**3**	**4**
Simple	From [14]	200	100.00	100.00	100.00	100.00	100.00
		500	100.00	100.00	100.00	100.00	100.00
		1000	100.00	100.00	100.00	100.00	100.00
		2000	100.00	100.00	100.00	100.00	100.00
		5000	100.00	100.00	100.00	100.00	100.00
Orthogonal	From [14]	200	96.67	**100.00**	100.00	100.00	100.00
		500	93.33	**100.00**	100.00	100.00	100.00
		1000	100.00	100.00	100.00	100.00	100.00
		2000	100.00	100.00	100.00	100.00	100.00
		5000	93.33	**100.00**	100.00	100.00	100.00
Simple-simple	From [17] (GD)	200	100.00	100.00	100.00	100.00	100.00
		500	100.00	100.00	100.00	100.00	100.00
		1000	100.00	100.00	100.00	100.00	100.00
		2000	46.67	**60.00**	**70.00**	**76.67**	**80.00**
		5000	0.00	0.00	0.00	0.00	0.00
Ortho-ortho	From [33]	200	100.00	100.00	100.00	100.00	100.00
		500	100.00	100.00	100.00	100.00	100.00
		1000	96.67	**100.00**	100.00	100.00	100.00
		2000	33.33	**50.00**	**83.33**	**86.67**	86.67
		5000	0.00	0.00	0.00	0.00	0.00
von Koch	From [14]	200	100.00	100.00	100.00	100.00	100.00
		500	100.00	100.00	100.00	100.00	100.00
		1000	100.00	100.00	100.00	100.00	100.00
		2000	100.00	100.00	100.00	100.00	100.00
		5000	0.00	0.00	0.00	0.00	0.00
Spike	From [26]	200	100.00	100.00	100.00	100.00	100.00
		500	100.00	100.00	100.00	100.00	100.00
		1000	100.00	100.00	100.00	100.00	100.00
		2000	100.00	100.00	100.00	100.00	100.00
		5000	100.00	100.00	100.00	100.00	100.00

Table 4.5: Percentage of executions with I3 in M2 within different optimality gaps.

Class	**Source**	n	**Iterations**			
			AV	**CV**	**CP**	**CE**
Simple	From [10]	30	1.50	1.55	1.45	1.50
		40	1.25	1.40	1.15	1.10
		50	1.45	1.70	1.55	1.35
		60	1.55	1.80	1.20	1.40
	From [14]	100	2.53	2.63	1.80	1.87
		200	2.83	3.10	2.47	2.50
		500	3.83	3.93	3.97	3.80
		1000	4.70	4.67	4.47	4.57
Orthogonal	From [10]	30	1.38	1.38	1.14	1.10
		40	1.50	1.75	1.45	1.35
		50	1.55	1.65	1.45	1.45
		60	1.80	1.90	1.40	1.55
	From [14]	100	2.80	2.57	2.37	2.33
		200	3.33	3.30	2.93	2.97
		500	4.50	4.37	3.73	3.80
		1000	5.40	5.87	5.00	5.43
von Koch	From [14]	100	1.57	1.70	1.60	1.77
		200	2.13	2.13	2.33	2.00
		500	2.03	2.20	2.43	2.13

Table 4.6: Average number of iterations (main loop) until an optimal solution is found for each initial discretization strategy with E(I1,M1).

Class	**Source**	n	**Time (sec)**			
			AV	**CV**	**CP**	**CE**
Simple	From [10]	30	0.22	0.17	0.19	0.22
		40	0.32	0.25	0.23	0.29
		50	0.61	0.43	0.42	0.58
		60	0.91	0.79	0.54	0.84
	From [14]	100	3.03	2.29	1.72	2.12
		200	11.84	9.04	7.09	9.88
		500	114.51	78.62	65.64	103.60
		1000	926.39	554.24	408.71	718.93
Orthogonal	From [10]	30	0.14	0.12	0.12	0.13
		40	0.21	0.18	0.17	0.20
		50	0.28	0.26	0.23	0.28
		60	0.41	0.35	0.30	0.38
	From [14]	100	1.34	1.09	0.95	1.17
		200	5.99	4.99	3.95	4.86
		500	68.01	41.31	30.85	42.46
		1000	297.50	233.82	155.00	235.35
von Koch	From [14]	100	2.26	1.44	1.62	2.60
		200	30.90	17.21	25.32	32.86
		500	1064.08	256.77	595.89	1639.80

Table 4.7: Average running time until an optimal solution is found for each initial discretization strategy with E(I1,M1).

and orthogonal polygons, but, in contrast to Table 4.6, the CV strategy is the second best, despite it obtained some of the worst results in number of iterations. Even more impressive are the outcomes relative to von Koch polygons. In this case, even though AV has the lowest number of iterations, CV was the one with the best results, with a significant advantage over CP, the second best option.

From this analysis, we conclude that, despite being important, the number of iterations is not necessarily the decisive factor in the final performance of the method. The CV strategy was not able to achieve good iteration results, but it was usually the first or second best option in run time. The fact that CV also creates a set D of quality but with smaller size than the other options, reduces the number of visibility polygons to be calculated, the arrangement complexity, the number of point location operations and, not least, the complexity of SCP instances generated. All this consequences have a deep impact in the performance of the program and permits CV to compete with CP. This situation is better visualized in von Koch polygons, where the arrangements are usually more complex and require more computational time from geometric tasks.

In more recent experiments, performed for the survey [17], CV and CP were again put to the test. This time, we analyzed the performance of I3 instead of I1, what gave us the possibility of experimenting on polygons with holes. The results, displayed in Table 4.8, proved once again that no strategy can be the best in all cases. While CP has now obtained the best results solving ortho-ortho and simple-simple polygons, CV showed a great advantage when dealing with von Koch and spike classes. In the special case of spike polygons, the average time of CV was orders of magnitude smaller than its competitor's. As for testing with simple and orthogonal instances, the dispute seemed more equal, with a slight advantage in optimality rate for CV, which is apparently able to scale better.

These new performance results allow us to reach to some conclusions. From them, it can be argued that using only convex vertices as initial witnesses works better with our third implementation I3 than with I1. A possible reason for this is that I3 employed a new mode to decide whether to solve or not an AGPFC instance at a given iteration. In I3, we only solve an AGPFC instance if the resolution of the current AGPW was unable to improve the previous lower bound for the AGP (see more in Section 4.1.3). This idea avoids the computation of unnecessary AGPFCs, which are very common when using smaller witness sets, as proposed by CV.

Moreover, the great advantage in applying CV instead of CP to solve spike polygons is probably due to the shape of the holes in these instances. As shown in Figure 4.3, practically all vertices of holes are reflex, which implies in CP choosing a large number of midpoints of edges as initial witnesses, normally unnecessary to the resolution of spike polygons. For all these outcomes, the CV strategy is today considered the default option

Class	**Source**	n	**Optimality Rate (%)**		**Time (sec)**	
			CV	**CP**	**CV**	**CP**
Simple	From [14]	200	100.00	100.00	0.75	0.62
		500	100.00	100.00	2.96	2.68
		1000	100.00	100.00	9.95	10.19
		2000	100.00	100.00	48.29	51.92
		5000	100.00	93.33	506.63	498.56
Orthogonal	From [14]	200	96.67	96.67	0.37	0.34
		500	93.33	93.33	1.51	1.36
		1000	100.00	100.00	5.22	4.63
		2000	100.00	100.00	21.18	18.48
		5000	93.33	83.33	120.73	117.46
Simple-simple	From [33] (GD)	200	100.00	100.00	6.58	4.50
		500	100.00	100.00	61.37	42.21
		1000	100.00	100.00	373.15	408.47
		2000	46.67	46.67	1026.13	1019.32
		5000	0.00	0.00	-	-
Ortho-ortho	From [33]	200	100.00	100.00	9.37	6.33
		500	100.00	100.00	101.29	86.57
		1000	96.67	96.67	402.61	275.14
		2000	33.33	43.33	1083.98	930.56
		5000	0.00	0.00	-	-
von Koch	From [14]	200	100.00	100.00	1.20	1.54
		500	100.00	100.00	7.99	12.14
		1000	100.00	100.00	59.19	88.06
		2000	100.00	100.00	329.33	700.48
		5000	0.00	0.00	-	-
Spike	From [26]	200	100.00	100.00	0.47	1.09
		500	100.00	100.00	2.03	12.22
		1000	100.00	100.00	22.20	183.15
		2000	100.00	100.00	14.93	353.38
		5000	100.00	33.33	26.03	947.73

Table 4.8: Optimality rate and average time results for CV and CP discretization strategies with $E(I3, M2)$.

for our implementation.

4.4.3 Guard Candidate Management

Recall that, in Section 3.6, we discussed about two different strategies for constructing the guard candidate set C at each iteration of our algorithm. The first idea (BG), implemented in I1, consists in choosing all vertices from Light AVPs ($V_{\mathcal{L}}(D)$) and from P, while the second (CG), which was introduced in I3, constructs C by choosing one interior point from each light AVP ($C_{\mathcal{L}}(D)$) and again all vertices of P. Table 4.9 shows results that allow a performance comparison between the traditional method and the new one with $E(I3, M2)$.

Firstly, if we analyze solely the optimality rates displayed in Table 4.9, we can see that CG achieved equal or better results than BG in 5 out of 6 classes of polygons. The only exception occurred in Orthogonal polygons, where BG was clearly able to converge better and obtained optimal solutions for all the 150 instances considered.

The superior quality of CG strategy over BG is again verified when considering the run time data of the comparison table. Although both techniques achieved similar results when solving smaller instances, CG was capable of scaling better and had a material advantage in performance on the largest ones. The verified efficiency in solving larger polygons was certainly one decisive factor for CG obtaining a better optimality rate.

The reason for the good results of CG is directly linked to the smaller guard candidate set generated. This fact leads to a lower number of geometric operations and also to smaller ILP models, which saves time. The encouraging outcomes obtained turned this new approach into our default option for guard candidate discretization.

4.4.4 SCP Resolution

In the algorithm presented in Chapter 3, the resolution of SCP instances plays a very important role. In the technique, each AGPW or AGPFC instance suffers a reduction to the AGPWFC, which can be seen as a geometric interpretation of the SCP. Thus, reducing the time needed to solve these instances directly results in improving the program as a whole. Figure 4.6 illustrates the percentage of time spent solving ILPs with each of our implementations in M2.

In I1, the task of solving SCPs was entrusted only to ILP solvers. However, in some experiments, we observed complex cases where even modern solvers were having trouble in finding optimal solutions. After this, in I2, techniques for reducing the ILP matrix were implemented, along with a Lagrangian Heuristic (LH). This heuristic, based in a work by Beasley on Lagrangian Relaxation [6] (see Section 2.2 for details), was used with

Class	**Source**	n	**Optimality Rate (%)**		**Time (sec)**	
			CG	**BG**	**CG**	**BG**
Simple	From [14]	200	100.00	100.00	0.75	0.67
		500	100.00	100.00	2.96	2.96
		1000	100.00	100.00	9.95	11.10
		2000	100.00	100.00	48.29	53.03
		5000	100.00	93.33	504.63	633.12
Orthogonal	From [14]	200	96.67	100.00	0.37	0.41
		500	93.33	100.00	1.51	1.71
		1000	100.00	100.00	5.22	5.66
		2000	100.00	100.00	21.18	24.14
		5000	93.33	100.00	123.07	144.45
Simple-simple	From [33] (GD)	200	100.00	100.00	6.58	6.22
		500	100.00	100.00	61.37	54.16
		1000	100.00	96.67	285.47	285.35
		2000	46.67	13.33	598.82	967.43
		5000	0.00	0.00	-	-
Ortho-ortho	From [33]	200	100.00	100.00	9.37	8.23
		500	100.00	100.00	101.29	88.42
		1000	96.67	93.33	338.75	319.04
		2000	33.33	6.67	729.23	841.84
		5000	0.00	0.00	-	-
von Koch	From [14]	200	100.00	100.00	1.20	1.14
		500	100.00	100.00	7.99	9.57
		1000	100.00	100.00	59.19	76.49
		2000	100.00	100.00	329.33	483.34
		5000	0.00	0.00	-	-
Spike	From [26]	200	100.00	100.00	0.47	0.42
		500	100.00	100.00	2.03	2.15
		1000	100.00	100.00	22.20	22.04
		2000	100.00	100.00	14.93	16.92
		5000	100.00	100.00	32.85	47.00

Table 4.9: Optimality rate and average time results for CG and BG guard candidate selection strategies with $E(I3, M2)$.

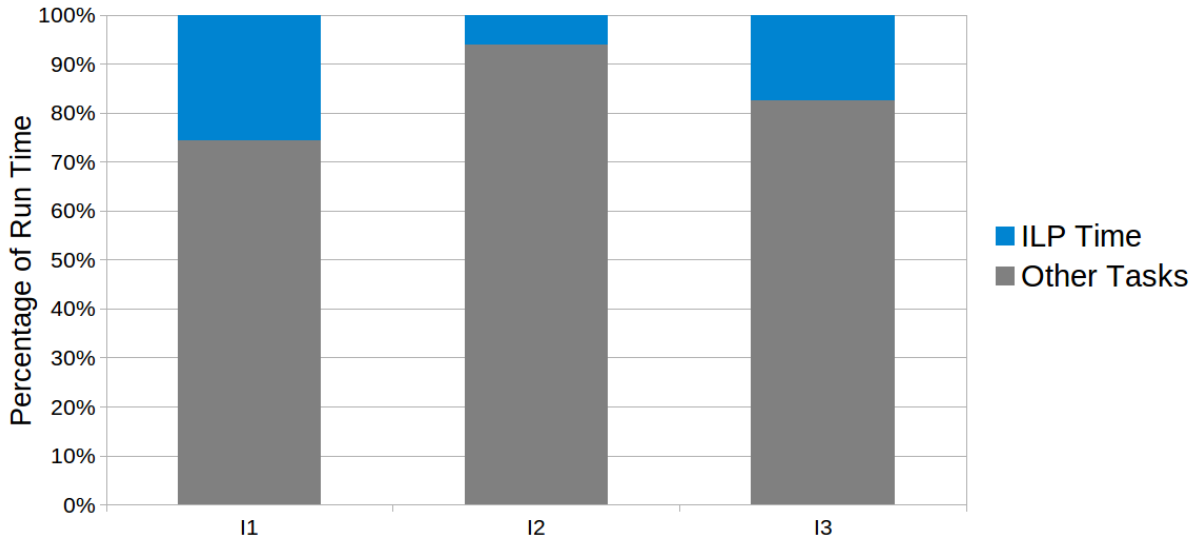


Figure 4.6: Percentage of time used in ILP resolution with each of our three implementations in M2, when solving simple polygons with 1000 vertices.

the purpose of replacing the ILP solver or at least helping it by providing a good initial viable solution.

To verify the heuristic contribution to implementation I2, a group of instances was tested in M1 in two different modes, one employing the Lagrangian Heuristic and the other without it. In addition, the same experiment was performed considering two different solvers: XPRESS and GLPK. The outcomes, recently published in [33], are summarized in Figures 4.7 and 4.8.

In the charts, a clear difference is observed in the benefit provided by the LH, depending on the ILP solver. From XPRESS results, we can see a considerable advantage in performance when using the Lagrangian Heuristic on simple and orthogonal polygons. However, when looking to results employing GLPK, it is harder to visualize the same effect, even though the version with LH won in 14 out of the 20 subgroups experimented. This difference in behavior between ILP solvers is possibly due to the fact that XPRESS takes more advantage from viable solutions provided by the LH than GLPK.

Another important issue to consider here is that changing the way SCPs are solved also implies that different solutions can be found for the same AGPWFC instance. When this happens, distinct uncovered regions are produced, permanently affecting the future decisions of the algorithm. In instances where this occurs, it is not unusual to see significant differences in performance. Consider, for example, the polygon of 900 vertices called “randsimple-900-4.pol”. For this instance, the resolution time using GLPK plus LH

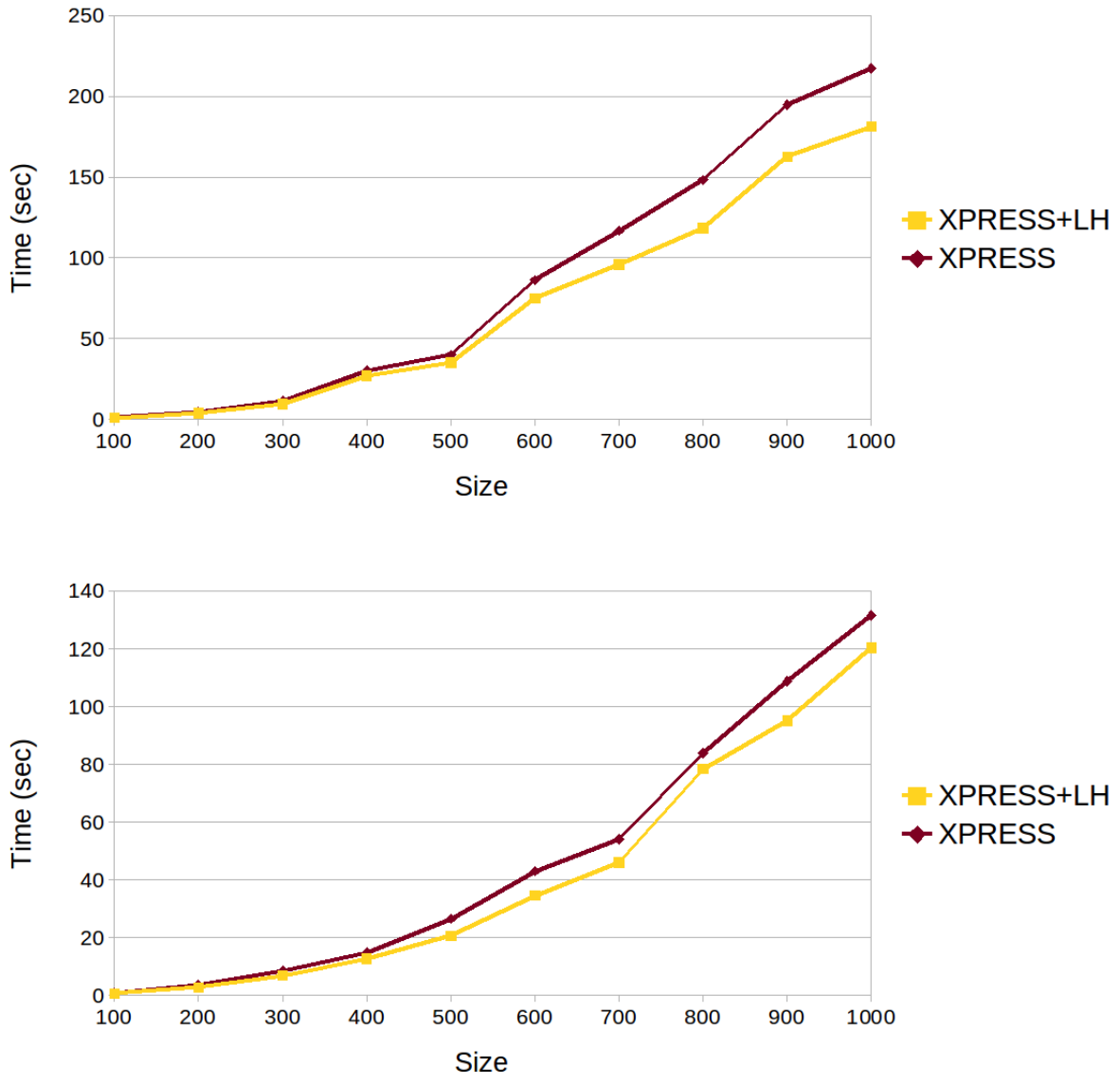


Figure 4.7: Comparison of average run time with $E(I2, M1)$ when using Lagrangian Heuristic or only XPRESS on simple (top) and orthogonal (bottom) polygons.

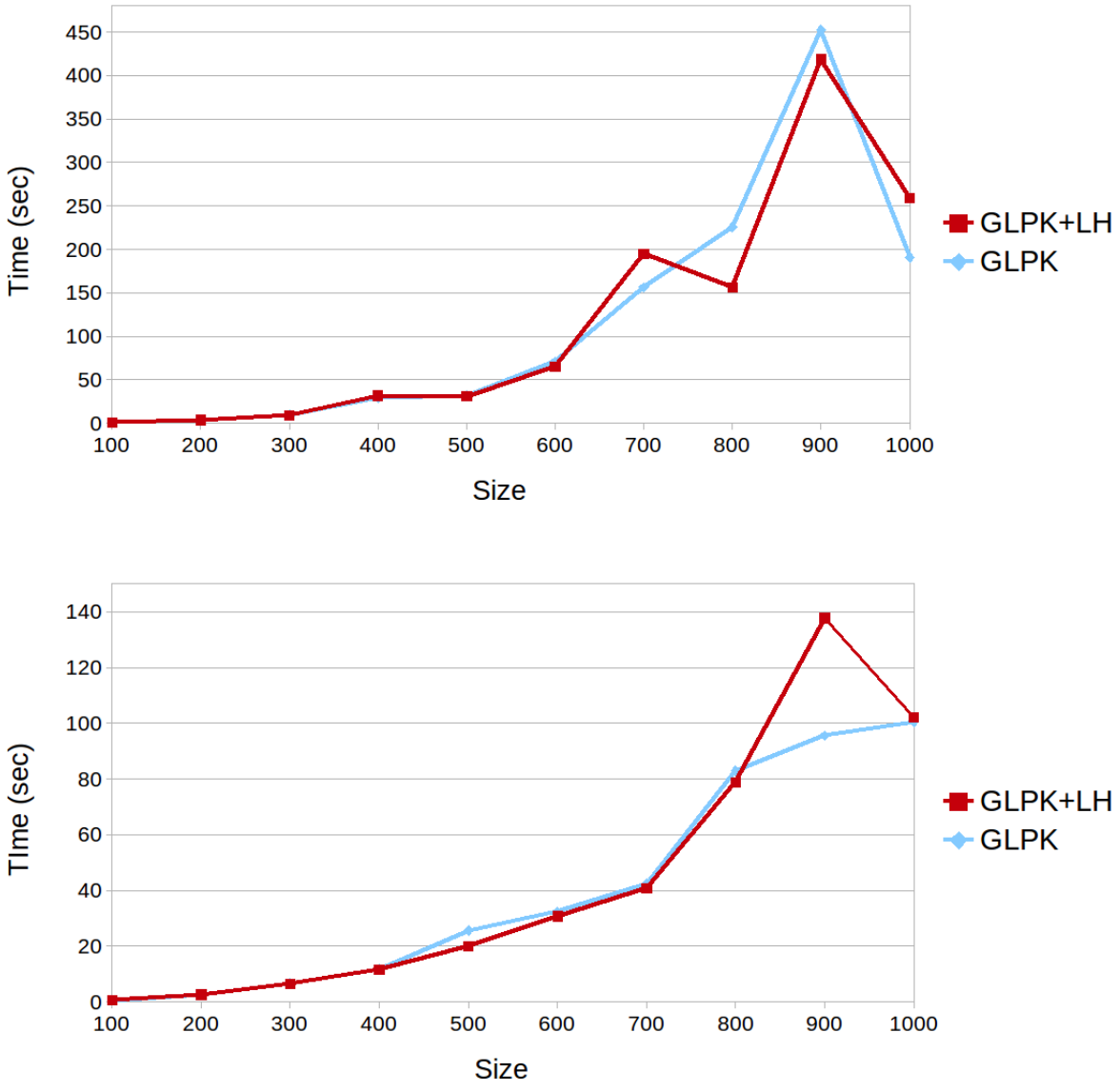


Figure 4.8: Comparison of average run time with $E(I2, M1)$ when using Lagrangian Heuristic or only GLPK on simple (top) and orthogonal (bottom) polygons.

is 3260.54 seconds, while the time employing only GLPK is 1559.54 seconds. However, the average time found for the subgroup of simple polygons with 900 vertices is lower for the version with LH (418.80 seconds against 452.29 seconds).

The same kind of experiment was repeated in the survey [17], using I3 with CPLEX. This time, however, it was evidenced that the heuristic has difficulty to scale. As illustrated in Table 4.10, in the largest subgroups of each class, the heuristic was only capable of (slightly) improving the performance on spike polygons, which, as discussed before, are part of a very specific set of instances. The likely reason for this is the difficulty in obtaining proven optimal solutions for SCP instances with thousands of variables and constraints. In these situations, the time spent by the heuristic is considerably higher and the solution found not good enough to help CPLEX.

As a final conclusion on the Lagrangian heuristic, we can argue that the advantage of using it depends on the size and the class of the instance and also on the ILP solver available to the user. However, since our objective is to solve instances increasingly larger and in the shortest amount of time, the current default version of our technique does not apply the LH.

4.5 Comparison With Other Techniques

In recent years, as discussed in Section 1.2, other algorithms were proposed for the AGP. During this Master's project, we compared our achievements with some of these techniques. In this section, we first analyze the differences between Bottino et al.'s method [10] and ours and then perform a complete comparison with the algorithm of Kröller et al. [26], based on results obtained from the survey on algorithms for the AGP [17].

4.5.1 Comparison with Bottino et al.'s technique [10]

In 2011, Bottino and Laurentini proposed a heuristic for the original AGP [10], aiming to produce good viable solutions with an efficient method. The technique was experimented and obtained promising results, including some optimal solutions. In the paper, the authors compared their technique with the one by Amit et al. [2] and claimed that their method was able to achieve better results.

Upon learning about this work, we decided to try our I2 version with exactly the same instances used by Bottino and Laurentini and compare our findings. The experiments were done using all simple and orthogonal instances from Bottino et al., which vary between 30 and 60 vertices (see Section 4.3 for details). Table 4.11 summarizes the results, showing two types of information: average number of guards and average run time.

In our tests, I2 was able to find proven optimal solutions for all instances, meaning that

Class	Source	n	Optimality Rate (%)		Time (sec)	
			CPLEX	**+LH**	CPLEX	**+LH**
Simple	From [14]	200	100.00	100.00	0.75	0.60
		500	100.00	100.00	2.96	2.66
		1000	100.00	100.00	9.95	9.78
		2000	100.00	100.00	48.29	59.75
		5000	100.00	33.33	415.64	937.62
Orthogonal	From [14]	200	96.67	96.67	0.37	0.32
		500	93.33	93.33	1.51	1.45
		1000	100.00	96.67	5.05	5.17
		2000	100.00	100.00	21.18	30.26
		5000	93.33	93.33	123.07	352.79
Simple-simple	From [33] (GD)	200	100.00	100.00	6.58	5.81
		500	100.00	100.00	61.37	57.00
		1000	100.00	96.67	285.47	499.82
		2000	46.67	6.67	820.79	1008.05
		5000	0.00	0.00	-	-
Ortho-ortho	From [33]	200	100.00	100.00	9.37	14.69
		500	100.00	100.00	101.29	146.37
		1000	96.67	90.00	282.12	357.71
		2000	33.33	0.00	-	-
		5000	0.00	0.00	-	-
von Koch	From [14]	200	100.00	100.00	1.20	1.03
		500	100.00	100.00	7.99	6.51
		1000	100.00	100.00	59.19	33.74
		2000	100.00	100.00	329.33	356.94
		5000	0.00	0.00	-	-
Spike	From [26]	200	100.00	100.00	0.47	0.41
		500	100.00	100.00	2.03	1.73
		1000	100.00	100.00	22.20	11.62
		2000	100.00	100.00	14.93	11.51
		5000	100.00	100.00	32.85	30.94

Table 4.10: Optimality rate and average time results when using or not the Lagrangian Heuristic to help CPLEX with $E(I3, M2)$.

Class	n	Number of Guards		Time (sec)	
		Method [10]	I2	Method [10]	I2
Simple	30	4.20	4.20	1.57	0.14
	40	5.60	5.55	2.97	0.10
	50	6.70	6.60	221.92	0.24
	60	8.60	8.35	271.50	0.27
Orthogonal	30	4.60	4.52	1.08	0.04
	40	6.10	6.00	9.30	0.07
	50	7.80	7.70	6.41	0.12
	60	9.30	9.10	81.95	0.16

Table 4.11: Comparison between the method of Bottino et al. [10] and I2.

the column with average number of guards found by our method actually contains optimal values. Knowing this, we can conclude that the heuristic by Bottino and Laurentini was able to find good solutions, but not always optimal. Except for simple polygons with 30 vertices, the heuristic did not manage to find the best possible solutions for all polygons of a subgroup. Looking more carefully, we can notice a growing gap between the average number of guards from both techniques as the size of the instances increases.

Besides comparing the quality of the solutions, it is also important to evaluate the time needed to find them. To this end, Table 4.11 exhibits the computing times for the two methods. It is important to notice though that the experiments were done in different environments, which invalidates a direct comparison of performance between the techniques. While our tests were conducted in environment M1, with machines featuring an Intel® Core™ i7-2600 at 3.40 GHz and 8 GB of RAM, the researchers of [10] performed their experiments on an Intel® Core2™ processor at 2.66 GHz and with 2 GB of RAM. Despite this, Table 4.11 shows that the average run time of our technique to compute proven optimal solutions for the AGP is orders of magnitude smaller than the time used by the heuristic. At least it seems safe to say that this large disparity in computing times can not be entirely attributed to hardware and software differences.

4.5.2 Comparison with Kröller et al.'s technique [26]

In 2009, a group of researchers from Germany proposed a new technique based on linear programming, called here BS1, to solve a fractional version of the AGP [5]. Over the years, the idea has evolved and was modified, becoming a practical option for computing high quality solutions to the original problem [19] (see Section 1.2 for a summary of the technique). From these changes, two new implementations emerged, one in 2012, called BS2, and the current one, named BS3. For this reason, we decided to compare their techniques with ours. This was made possible thanks to my internship in TUBS, where

we worked in straight collaboration with the authors of [5, 19].

In the experiments performed during this internship, the three implementations from UNICAMP (I1, I2 and I3) were tested along with three different versions of TUBS: BS1, BS2 and BS3. These tests were conducted in environment M2 using 900 polygons from the following classes: simple, orthogonal, von Koch, simple-simple, ortho-ortho and spike. It is important to remember that, in M2, each instance was allowed to run for a maximum of 20 minutes and, after this, was considered unsolved. Table 4.12 summarizes the optimality rate results obtained, where the columns are in chronological order of implementation. Note that columns I1, I2 and I3 are just copies of the ones already displayed in Table 4.2 and are repeated here to facilitate our analysis.

From Table 4.12, it is clear that both lines of thought have improved over time. As discussed in Section 4.4.1, our implementations became able to solve polygons with holes and can now solve instances with up to 5000 vertices. On the other hand, those from Kröller et al. managed to greatly improve the optimality rates. The initial version, which tried to solve the AGP using only LP, got a small number of proven optimal solutions, while the latest obtained more than 90% of optimality for hole-free polygons with 1000 vertices.

Moreover, if we analyze all techniques at once, we can see what appears to be an algorithms race, with the latest techniques overcoming the achievements of the previous ones. Our first implementation, for example, was able to resolve a far greater number of instances than TUBS' first (BS1) and had a close dispute with their second version. After this, our version I2, produced during the first semester of 2013, achieved great results in optimality rate, surpassing those obtained by the German release of 2012 (BS2). I2 was also capable of obtaining better optimality results than TUBS' current implementation (BS3) on instances of smaller size. However, when increasing the size to thousands of vertices, BS3 was distinctly superior.

Finally, when comparing the optimality rate obtained by the current versions of both research groups (I3 and BS3), one can conclude that I3 has a significant advantage, being far more robust than its opponent. While I3 was able to obtain 100% optimality in 21 out of 30 subgroups, the version from TUBS only achieve this for 4 subgroups.

To get a deeper insight into the differences in behavior of the latest two techniques, we also developed a running time comparison between them, using results of all polygon classes. This comparison is shown in Figure 4.9. For a fairer analysis, the average times in the charts only considered values of instances resolved by both I3 and BS3.

In Figure 4.9, it is easy to see that BS3 was faster in solving simple-simple, ortho-ortho and von Koch polygons. On the other hand, I3 was more efficient with simple polygons and meaningly better when dealing with orthogonal and spike instances. In the specific case of the spike class, I3 was about 20 times faster than BS3 to solve the instances with

Class	**Source**	n	**Optimality Rate (%)**					
			BS1	**BS2**	**I1**	**I2**	**BS3**	**I3**
Simple	From [14]	200	20.00	**100.00**	**100.00**	**100.00**	96.67	**100.00**
		500	3.33	76.67	**100.00**	**100.00**	96.67	**100.00**
		1000	0.00	70.00	96.67	**100.00**	90.00	**100.00**
		2000	0.00	36.67	6.67	50.00	60.00	**100.00**
		5000	0.00	0.00	0.00	0.00	26.67	**100.00**
Orthogonal	From [14]	200	16.67	96.67	**100.00**	**100.00**	96.67	96.67
		500	3.33	86.67	**100.00**	96.67	93.33	93.33
		1000	0.00	70.00	**100.00**	**100.00**	86.67	**100.00**
		2000	0.00	46.67	70.00	90.00	70.00	**100.00**
		5000	0.00	0.00	0.00	0.00	40.00	93.33
Simple-simple	From [33] (GD)	200	3.33	93.33	-	**100.00**	86.67	**100.00**
		500	0.00	76.67	-	83.33	60.00	**100.00**
		1000	0.00	3.33	-	0.00	13.33	**100.00**
		2000	0.00	0.00	-	0.00	0.00	46.67
		5000	0.00	0.00	-	0.00	0.00	0.00
Ortho-ortho	From [33]	200	10.00	83.33	-	96.67	86.67	**100.00**
		500	0.00	53.33	-	83.33	53.33	**100.00**
		1000	0.00	16.67	-	3.33	16.67	96.67
		2000	0.00	0.00	-	0.00	0.00	33.33
		5000	0.00	0.00	-	0.00	0.00	0.00
von Koch	From [14]	200	36.67	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**
		500	10.00	**100.00**	96.67	**100.00**	93.33	**100.00**
		1000	0.00	**100.00**	46.67	**100.00**	96.67	**100.00**
		2000	0.00	83.33	0.00	0.00	86.67	**100.00**
		5000	0.00	0.00	0.00	0.00	0.00	0.00
Spike	From [26]	200	70.00	**100.00**	-	**100.00**	96.67	**100.00**
		500	60.00	**100.00**	-	**100.00**	**100.00**	**100.00**
		1000	80.00	3.33	-	96.67	**100.00**	**100.00**
		2000	83.33	0.00	-	96.67	**100.00**	**100.00**
		5000	0.00	0.00	-	0.00	96.67	**100.00**

Table 4.12: Optimality Rate of our implementations and AGP Solvers from [26] in environment M2.

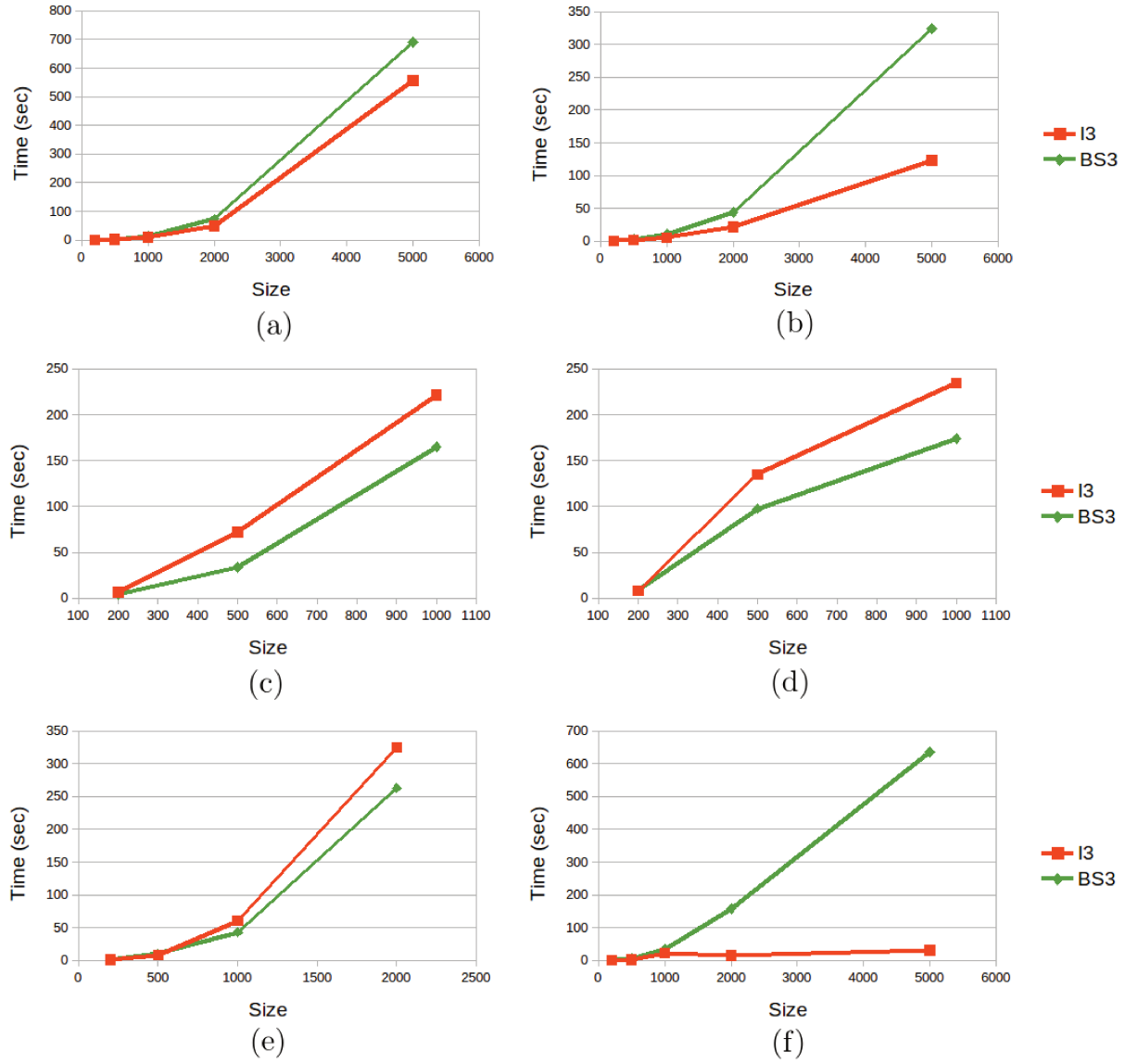


Figure 4.9: Performance comparison between I3 and BS3 in M2 when solving the following classes: (a) Simple; (b) Orthogonal; (c) Simple-simple; (d) Ortho-ortho; (e) von Koch; (f) Spike. Here, only fully solved instances are considered.

5000 vertices.

Through all results presented, one can conclude that the methods from TUBS have a natural difficulty in converging to a proven optimal solution and that this problem can not be totally assigned to the code performance. While some positive results where observed in run time, the optimality rate of BS3 was not able to follow it. For illustration, BS3 needed an average time of 164.65 seconds to solve simple-simple polygons with 1000 vertices (26% percent less than I3), but the optimality rate for this subgroup was only 13.33%, far below the results using I3, when **all** 30 instances were solved within the imposed time limit. In the case of our method, the evolution mostly occurred due to optimizations on routines and changes in some internal decisions of the algorithm. It seems that our technique, since its first release, tends to find the optimal solution in almost all cases and the low optimality observed in larger instances is directly related to the maximum run time imposed in the testing environment.

Chapter 5

Conclusions

In this dissertation, we studied applying ILP modeling to optimally solve the Art Gallery Problem, an $\mathbb{NP}$-hard problem in the Computational Geometry field. As a result, an algorithm was designed that iteratively discretizes the original problem to find lower and upper bounds while seeking an optimal solution for the AGP.

To allow its correct evaluation, our algorithm was coded and had its implementation modified and optimized through time. In total, we experimented our technique on more than 2800 instances from different sources and classes of polygons. Our methodology proved capable of optimally solving polygons with up to 5000 vertices in less than 20 minutes each, something not imagined a few years ago.

In order to demonstrate the quality of our solution, we also compared our results with those produced by other state-of-the-art techniques. These comparisons revealed a significant advantage when using our technique, which proved to be far more effective, faster and more robust than all the others. These results encouraged us to release an implementation of our algorithm that does not require any proprietary software library on the web page of our project [16]. By doing so, we expect to contribute to future research on the topic, since it is now possible for new techniques to be directly tested and compared to our software package.

Besides providing a robust code for solving the AGP, our work also lead to four papers on the subject, two of which have already been published [7, 34] and two recently submitted [33, 17]. Some of these studies provided a strong interaction with other researchers on the topic, as was the case of the survey on algorithms for the AGP [17], produced in partnership with a group from TUBS, in Germany. This interaction was important for two reasons: it introduced me to an internationally recognized research group and expanded the relationships between researchers from UNICAMP and TUBS, enabling other future joint works as well.

To conclude, despite the high quality results achieved by our technique, there is cur-

rently no proof of the method's convergence. It remains a future challenge to study additional geometric properties that may assist in designing a strategy that leads to proven convergence of the method. In addition, we hope that research on other variants of the AGP can also benefit from the work done in this project.

Bibliography

[1] A. Aggarwal, S. K. Ghosh, and R. K. Shyamasundar. Computational complexity of restricted polygon decompositions. In G. T. Toussaint, editor, *Computational Morphology*, pages 1–11. North-Holland, 1988.

[2] Y. Amit, J. S. B. Mitchell, and E. Packer. Locating guards for visibility coverage of polygons. In *ALENEX*, pages 1–15, New Orleans, Lousiana, January 2007. SIAM.

[3] D. L. Applegate, R. E. Bixby, V. Chvatal, and W. J. Cook. *The Traveling Salesman Problem: A Computational Study (Princeton Series in Applied Mathematics)*. Princeton University Press, Princeton, NJ, USA, 2007.

[4] Y. Bartal and L.-A. Gottlieb. A linear time approximation scheme for euclidean tsp. In *Foundations of Computer Science (FOCS), 2013 IEEE 54th Annual Symposium on*, pages 698–706, Oct 2013.

[5] T. Baumgartner, S. P. Fekete, A. Kröller, and C. Schmidt. Exact solutions and bounds for general art gallery problems. In *Proceedings of the SIAM-ACM Workshop on Algorithm Engineering and Experiments, ALENEX 2010*, pages 11–22. SIAM, 2010.

[6] J. E. Beasley. Lagrangian relaxation. In C. R. Reeves, editor, *Modern Heuristic Techniques for Combinatorial Problems*, pages 243–303. John Wiley & Sons, Inc., New York, NY, USA, 1993.

[7] D. Borrmann, P. J. de Rezende, C. C. de Souza, S. P. Fekete, S. Friedrichs, A. Kröller, A. Nüchter, C. Schmidt, and D. C. Tozoni. Point guards and point clouds: solving general art gallery problems. In *Proceedings of the twenty-ninth annual symposium on Computational geometry*, SoCG '13, pages 347–348, New York, NY, USA, 2013. ACM.

[8] P. Bose, A. Lubiw, and J. I. Munro. Efficient visibility queries in simple polygons. *Computational Geometry*, 23(3):313–335, 2002.

[9] A. Bottino and A. Laurentini. A nearly optimal sensor placement algorithm for boundary coverage. *Pattern Recognition*, 41(11):3343–3355, 2008.

[10] A. Bottino and A. Laurentini. A nearly optimal algorithm for covering the interior of an art gallery. *Pattern Recognition*, 44(5):1048–1056, 2011.

[11] CGAL. Computational Geometry Algorithms Library, 2012. www.cgal.org (last access January 2012).

[12] K.-Y. Chwa, B.-C. Jo, C. Knauer, E. Moet, R. van Oostrum, and C.-S. Shin. Guarding art galleries by guarding witnesses. *Intern. Journal of Computational Geometry And Applications*, 16(02n03):205–226, 2006.

[13] M. C. Couto, P. J. de Rezende, and C. C. de Souza. An exact algorithm for an art gallery problem. Technical Report IC-09-46, Institute of Computing, University of Campinas, Nov. 2009.

[14] M. C. Couto, P. J. de Rezende, and C. C. de Souza. An exact algorithm for minimizing vertex guards on art galleries. *International Transactions in Operational Research*, 18(4):425–448, 2011.

[15] M. C. Couto, C. C. de Souza, and P. J. de Rezende. An exact and efficient algorithm for the orthogonal art gallery problem. In *Proc. of the XX Brazilian Symp. on Comp. Graphics and Image Processing*, pages 87–94. IEEE Computer Society, 2007.

[16] P. J. de Rezende, C. C. de Souza, M. C. Couto, and D. C. Tozoni. The Art Gallery Problem Project (AGPPROJ), 2013. *www.ic.unicamp.br/∼cid/Problem-instances/Art-Gallery*.

[17] P. J. de Rezende, C. C. de Souza, S. Friedrichs, M. Hemmer, A. Kröller, and D. C. Tozoni. Engineering art galleries. 2014. Submitted.

[18] S. Eidenbenz. Approximation algorithms for terrain guarding. *Inf. Process. Lett.*, 82(2):99–105, 2002.

[19] S. P. Fekete, S. Friedrichs, A. Kröller, and C. Schmidt. Facets for art gallery problems. In D.-Z. Du and G. Zhang, editors, *Computing and Combinatorics*, volume 7936 of *Lecture Notes in Computer Science*, pages 208–220. Springer Berlin Heidelberg, June 2013.

[20] R. Fukasawa, H. Longo, J. Lysgaard, M. P. de Aragão, M. Reis, E. Uchoa, and R. F. Werneck. Robust branch-and-cut-and-price for the capacitated vehicle routing problem. *Mathematical Programming*, 106(3):491–511, 2006.

[21] S. K. Ghosh. Approximation algorithms for art gallery problems. In *Proc. Canadian Inform. Process. Soc. Congress*, pages 429–434, Mississauga, Ontario, Canada, 1987. Canadian Information Processing Society.

[22] S. K. Ghosh. *Visibility Algorithms in the Plane*. Cambridge University Press, New York, 2007.

[23] S. K. Ghosh. Approximation algorithms for art gallery problems in polygons. *Discrete Applied Mathematics*, 158(6):718–722, 2010.

[24] GLPK. *GNU Linear Programming Kit*. GNU, 2013. http://www.gnu.org/software/glpk/ (access December 2013).

[25] R. M. Karp. Reducibility among combinatorial problems. In R. Miller, J. Thatcher, and J. Bohlinger, editors, *Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103. Springer US, 1972.

[26] A. Kröller, T. Baumgartner, S. P. Fekete, and C. Schmidt. Exact solutions and bounds for general art gallery problems. *J. Exp. Algorithmics*, 17(1):2.3:2.1–2.3:2.23, May 2012.

[27] J. Laarhoven and J. Ohlmann. A randomized Delaunay triangulation heuristic for the euclidean Steiner tree problem in $\mathbb{R}^d$. *Journal of Heuristics*, 17(4):353–372, 2011.

[28] D. T. Lee and A. Lin. Computational complexity of art gallery problems. *Information Theory, IEEE Transactions on*, 32(2):276–282, March 1986.

[29] J. Mitchell. Approximating watchman routes. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 844–855, 2013.

[30] J. O'Rourke. *Art Gallery Theorems and Algorithms*. Oxford University Press, New York, NY, 1987.

[31] T. C. Shermer. Recent results in art galleries. *Proceedings of the IEEE*, 80(9):1384–1399, September 1992.

[32] A. P. Tomás and A. L. Bajuelos. Generating random orthogonal polygons. In *Current Topics in Artificial Intelligence*, volume 3040 of *LNCS*, pages 364–373. Springer, 2004.

[33] D. C. Tozoni, P. J. de Rezende, and C. C. de Souza. A practical iterative algorithm for the art gallery problem using integer linear programming. *Optimization Online*, Oct. 2013. www.optimization-online.org/DB_HTML/2013/11/4106.html.

[34] D. C. Tozoni, P. J. de Rezende, and C. C. de Souza. The quest for optimal solutions for the art gallery problem: A practical iterative algorithm. In V. Bonifaci, C. Demetrescu, and A. Marchetti-Spaccamela, editors, *Proceedings of the 12th International Symposium on Experimental Algorithms, SEA 2013*, volume 7933 of *Lecture Notes in Computer Science*, pages 320–336, Rome, Italy, 2013. Springer.

[35] J. Urrutia. Art gallery and illumination problems. In J. R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 973–1027, Amsterdam, 2000. Elsevier Science Publishers.

[36] XPRESS. *Xpress Optimization Suite*. FICO Solutions, 2009. http://www.fico.com/en/Products/DMTools/Pages/FICO-Xpress-Optimization-Suite.aspx (access January 2012).