Algorithms for Art Gallery Problems

Vasco José Rodrigues Cruz

Mestrado em Ciência de Computadores

Departamento de Ciência de Computadores

2022

Orientador

Ana Paula Tomás, Professora Associada,

Faculdade de Ciências da Universidade do Porto

U.PORTO

FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

Todas as correções determinadas pelo júri, e só essas, foram efetuadas.

O Presidente do Júri,

Porto, ______/______/_________

MSc

Abstract

The Art Gallery Problem is one of the most renowned problems in Computational Geometry. While its original formulation asks for the minimum number of guards that one needs for watching over the interior of a simple polygon, several variants have emerged through the years, and several results have revealed it to be a computationally hard problem in particular scenarios. In this thesis we study applications of parameterized, approximation and randomized algorithms to orthogonal variants of the Art Gallery Problem and characterize their computational and combinatorial complexity.

We present a new randomized algorithm for finding approximate solutions to the SET COVER problem and use it to determine minimum-cardinality vertex-guard sets on orthogonal polygons under straight-line visibility. We develop time- and memory-efficient data structures for implementing it and discuss kernelization strategies for reducing instance complexity. We study the performance of the algorithm against a benchmark dataset, over which it exhibits a very low approximation ratio, as well as satisfactory running times.

We define a new family of orthogonal polygons, the SCOTs, which are made up of rectangular rooms linked by rectangular corridors, mimicking properties of real-world buildings. We prove that, if a SCOT P is simple or r-independent, a minimum-cardinality guard set for guarding P under r-visibility can be computed in polynomial time. For that, we propose three methods. First, we describe a linear-time algorithm for simple SCOTs, with both vertex- and point-guards, based on a tree decomposition of the polygon. Then, we present one that runs in time $\mathcal{O}\left(c\sqrt{c}\right)$ with vertex-guards for a SCOT with c corridors. Finally, a third one is given for point-guards, running in time $\mathcal{O}\left(c^3 \log c\right)$. For the case where the SCOT has holes and is not r-independent, we show that the problem becomes NP-hard – indeed the decision problem is NP-complete for the case of point-guards – and give two approximation algorithms with factor $\mathcal{O}(\log c)$ for vertex-guards and point-guards.

At last, a dynamic programming algorithm for finding a lexicographically minimum optimal guard set for polyominoes under chess rook visibility is proposed, proving that the problem is fixed-parameter tractable parameterized by the length of the shortest side of the minimal unit grid that encloses the polyomino.

Keywords: Approximation Algorithms · Art Gallery Problem · Computational Complexity · Computational Geometry · Orthogonal Polygons · Parameterized Algorithms · Randomized Algorithms

Resumo

O Problema da Galeria de Arte é um dos problemas mais conceituados em Geometria Computacional. Enquanto que a sua formulação original pede o número mínimo de guardas necessários para vigiar o interior de um polígono simples, diversas variantes surgiram ao longo dos anos, e vários resultados revelaram ser um problema computacionalmente difícil em cenários particulares. Nesta tese estudamos aplicações de algoritmos parametrizados, de aproximação e aleatorizados a variantes ortogonais do Problema da Galeria de Arte e caracterizamos a sua complexidade computacional e combinatória.

Apresentamos um novo algoritmo aleatorizado para encontrar soluções aproximadas para o problema de cobertura de conjuntos (Set Cover) e usamo-lo para determinar conjuntos de guardas em vértices de cardinal mínimo em polígonos ortogonais segundo visibilidade em linha reta. Desenvolvemos estruturas de dados eficientes em tempo e memória para o implementar e discutimos estratégias de kernelização para reduzir a complexidade das instâncias. Estudamos o desempenho do algoritmo num conjunto de dados de referência, sobre o qual evidencia uma razão de aproximação muito baixa, bem como tempos de execução satisfatórios.

Definimos uma nova família de polígonos ortogonais, os SCOTs, que são formados por salas retangulares ligadas por corredores retangulares, imitando propriedades de edifícios do mundo real. Demonstramos que, se um SCOT P for simples ou r-independente, um conjunto mínimo de guardas para vigiar P segundo r-visibilidade pode ser calculado em tempo polinomial. Para tal, propomos três métodos. Primeiro descrevemos um algoritmo de tempo linear para SCOTs simples, com guardas quer em vértices, quer em pontos, baseado numa decomposição em árvore do polígono. De seguida, apresentamos um algoritmo que corre em tempo $\mathcal{O}(c\sqrt{c})$ com guardas em vértices para um SCOT com c corredores. Finalmente, um terceiro algoritmo é dado para guardas em pontos, com tempo de execução $\mathcal{O}(c^3 \log c)$. Para o caso em que o SCOT tem buracos e não é r-independente, mostramos que o problema se torna NP-difícil – de facto o problema de decisão fica NP-completo para guardas em pontos – e fornecemos dois algoritmos de aproximação com fator $\mathcal{O}(\log c)$ para guardas em vértices e em pontos.

Por fim, é proposto um algoritmo baseado em programação dinâmica que determina um conjunto ótimo de guardas lexicograficamente mínimo para poliominós segundo visibilidade de torres de xadrez, mostrando que o problema é tratável em parâmetro fixo quando parametrizado pelo comprimento do menor lado da grelha unitária minimal que envolve o poliominó.

Palavras-chave: Algoritmos Aleatorizados · Algoritmos de Aproximação · Algoritmos Parametrizados · Complexidade Computacional · Geometria Computacional · Polígonos Ortogonais · Problema da Galeria de Arte

To the three most important people in my life: my brother, my father and my mother.
If I am what I am, it is because of you.

To my grandparents.
You never, ever cease to replenish my strength.

Acknowledgements

I would like to express deep gratitude to Professor Ana Paula Tomás, who embraced me in this journey and made me see how beautiful guarding polygons can be. Thank you for having encouraged me, for having shown your persistent support even during off-hours and for having provided numerous helpful suggestions that have unquestionably improved the quality of the final product. Working with you has been a delight.

I thank my family for always being there in the nicest and hardest moments, without expecting anything in return. You keep inspiring me to believe in myself. I owe you everything.

Many thanks to my friends and colleagues for the unforgettable moments we shared and bonds we undoubtedly solidified for the rest of our lives. Thank you, Kevin, for the brilliant and insightful conversations we had, which have definitely propelled my work in the right direction.

And, of course, to Francisco and to Pedro, for having made FCUP a house of the family.

This work was partially supported by CMUP, under the research project with reference UIDB/00144/2020, whose opportunity I am very grateful for.

Contents

Abstract **i**

Resumo **iii**

Acknowledgements **vii**

Contents **xi**

List of Tables **xiii**

List of Figures **xxii**

List of Algorithms **xxiii**

Acronyms **xxv**

1 Introduction **1**

1.1 Background on the Art Gallery Problem 2

1.2 Contributions and thesis outline 6

2 Preliminaries **9**

2.1 Geometry 9

2.1.1 Polygons 9

2.1.2 Visibility 11

2.1.3 Arrangements 13

2.1.4 Partitions 15

2.2 Computational hardness . . . 16
2.2.1 Approximation algorithms . . . 16
2.2.2 Parameterized complexity . . . 18

3 Randomized Algorithm for Set Cover **19**

3.1 Background on Set Cover . . . 19
3.1.1 H_s-approximation algorithm . . . 20
3.1.2 f-approximation algorithm . . . 21
3.2 Polygon discretization and instance generation . . . 22
3.2.1 First approach . . . 23
3.2.2 Second approach . . . 24
3.3 Definition of Randomized-Set-Cover . . . 25
3.3.1 Implementation . . . 26
3.3.2 Probability of success with uniformly random instances . . . 27
3.3.3 Kernelization . . . 29
3.3.4 Algorithm reformulation . . . 33
3.3.5 Number of repetitions . . . 37
3.4 Data structures used . . . 39
3.4.1 Sampleable set . . . 39
3.4.2 Set . . . 41

4 Experimental Analysis of Randomized-Set-Cover **45**

4.1 Experimental setup . . . 45
4.2 Results . . . 47
4.2.1 Min-Area . . . 47
4.2.2 Fat . . . 48
4.2.3 Random . . . 50
4.2.4 Large-Random . . . 52
4.2.5 Discussion and general remarks . . . 54

4.3 Order of growth of $|\Pi_n|$ for grid n-ogons 55
4.3.1 Min-Area 56
4.3.2 Fat 56
4.3.3 Random 58

5 On r-Guarding SCOTs 59

5.1 Simple SCOTs 61
5.2 r-independent SCOTs 62
5.2.1 Super-corridors 62
5.2.2 3-approximation algorithm for vertex-guards and point-guards 66
5.2.3 Exact algorithm for vertex-guards 67
5.2.4 Exact algorithm for point-guards 69
5.3 NP-completeness with point-guards 78
5.3.1 Construction 79
5.3.2 Complexity 81
5.3.3 Correctness 82
5.3.4 NP-completeness 84
5.3.5 Approximation algorithms for vertex-guards and point-guards 86

6 Rook Vision on Polyominoes 89

6.1 Parameterized algorithm for Rook-AGP 90
6.1.1 Complexity 95

7 Conclusions and Future Work 97

Bibliography 99

List of Tables

3.1 Total number and percentage of Set Cover instances that admit an optimal cover of size 1, 2, 3 or more, or no feasible cover at all, for a fixed size of the universe $|\mathcal{U}| \in \{1, 2, 3, 4\}$. Notice that exactly half of the instances admit a cover of size 1, for any universe size, and the percentage of instances that admit an optimal cover of size 2 approaches 50%. 30

4.1 Statistics for Min-Area grid n-ogons. 48

4.2 Statistics for Fat grid n-ogons. 49

4.3 Statistics for Random grid n-ogons. 51

4.4 Statistics for Large-Random grid n-ogons. 53

List of Figures

1.1 **(a)** A convex polygon. **(b)** A star-shaped polygon. Each of them only requires one guard. 1

1.2 Two lamp placements on a polygonal region P under straight-line visibility. The left one does not fully cover P, while the one on the right is a feasible (and optimal) solution. 2

1.3 The example given by Chvátal to show that sometimes $\lfloor n/3 \rfloor$ guards are necessary. It consists of a comb-shaped polygon with $n = 3m$ vertices, for some integer $m \geq 1$. Notice that the points $1, 2, \ldots, m$ can only be seen by a guard placed in the shaded triangle that contains it. Since these triangles are disjoint, we need a guard for each of these m points separately, yielding the $m = \lfloor n/3 \rfloor$ bound. . . . 3

1.4 Illustration of the sufficiency proof by Fisk's method. The polygon is triangulated and a 3-coloring of its vertices is found. The green vertices form a valid guard set of size 2, which obeys the upper bound of $\lfloor n/3 \rfloor = \lfloor 10/3 \rfloor = 3$ given by the Art Gallery Theorem. 3

1.5 **(a)** Two vertex-guards are required for guarding this polygon. **(b)** A single point-guard would be enough. 4

1.6 Illustration of the sufficiency proof by the method of Kahn et al. [64], adapted from [99]. Similar in spirit to Fisk's proof, the authors prove Orthogonal Art Gallery Theorem (OAGT) by showing that any orthogonal polygon can be decomposed into convex quadrilaterals by adding non-intersecting line segments between vertices (convex quadrilateralization), which induces a 4-coloring of the vertices. 4

2.1 **(a)** Simple polygon. **(b)** Polygon with self-intersections. **(c)** Polygon with holes. **(d)** Not a polygon, because the chain of edges is not closed. **(e)** Not a polygon, because edges cannot be curve. 10

2.2 Orthogonal polygon. v_1 is a convex vertex and v_2 is a reflex vertex. 10

2.3 A 23-polyomino with a hole. 11

2.4 Which subset of an orthogonal polygon a guard at vertex v sees **(a)** under straight-line visibility and **(b)** under r-visibility. 12

2.5 **(a)** Non-regularized straight-line visibility. Point p is collinear with vertices v_1 and v_2 and sees the union of a polygon and a needle. **(b)** Regularized straight-line visibility. The visibility region of p does not include degeneracies and the needle is discarded. 12

2.6 Triangular expansion algorithm. Visibility is propagated to neighbour triangles through *windows* on shared edges. 13

2.7 **(a)** Information that **(a)** a vertex v, a half-edge e and **(b)** a face f keep in a DCEL structure. 14

2.8 Overlay of two arrangements $\mathcal{A}_1$, $\mathcal{A}_2$. 15

2.9 A polygon with $n = 12$ vertices, triangulated into $n - 2 = 10$ triangles by adding $n - 3 = 9$ internal diagonals (dashed in the figure). 15

2.10 **(a)** The grid partition $\Pi_{HV}(P)$ of an orthogonal polygon P. **(b)** The convex partition $\Pi(P)$ induced by vertex visibility regions (under straight-line visibility). 16

3.1 **(a)** A Set Cover instance $(\mathcal{U}, \mathcal{F})$ and **(b)** a corresponding minimum cover $\mathcal{C}^\star$. $\{S_2, S_3\}$ is another optimal cover for $\mathcal{U}$. 20

3.2 Reduction from Minimum Vertex Guard (MVG) to Set Cover. The convex partition of an orthogonal polygon with 8 vertices and 8 pieces is mapped to a Set Cover instance. 23

3.3 Estimated fraction of Set Cover instances admitting an optimal cover of cardinality 2 for each universe size $1 \leq |\mathcal{U}| \leq 14$, tending to 0.5. For each universe size, 10^5 instances were sampled uniformly at random and solved using Uniform-Randomized-Set-Cover. This trend suggests that the error probability of Uniform-Randomized-Set-Cover hastily approaches 0 as uniform random instances grow in size. 30

3.4 The function $2^x / \binom{x}{\lfloor x/2 \rfloor}$ asymptotically approaches $\sqrt{\frac{\pi x}{2}}$. 32

3.5 Valid set state transitions in Randomized-Set-Cover. 34

3.6 Solving the instance of Figure 3.1 using the improved version of RANDOMIZED-SET-COVER. The possible outcomes of the algorithm are shown. Elements in bold blue have frequency 1 and, therefore, belong to fixed sets. Removed elements $x \in \mathcal{U}$ appear in light gray instead of being omitted, so that it becomes clear which set is which. Elements that, in some iteration, belong to every remaining set are also shown in light gray, because they do not affect kernelization nor random choices in any way and can be ignored. The algorithm finds an optimal solution for this instance in a single run with probability 75%. After 3 runs, this probability increases to $1-(1-0.75)^3$, which is approximately 98.4%. 36

3.7 Two isomorphic SET COVER instances derived from the convex partition of an orthogonal polygon. One can convert the first instance to the second one through the permutation $\sigma = (6\ 2\ 7\ 8\ 3\ 4\ 5\ 1)$. On the original instance, $\mathsf{LMP}_{\mathsf{id}}(S_1, S_7) = 2$, while, for the transformed instance, $\mathsf{LMP}_{\sigma}(S_1, S_7) = 1$. $\sum \mathsf{LMP}_{\mathsf{id}}(S_i, S_j) = 75$ and $\sum \mathsf{LMP}_{\sigma}(S_i, S_j) = 85$. 38

3.8 **(a)** A bitset tree $\mathcal{T}_S$ for the set $S = \{2, 3, 8, 10, 16, 18, 19, 20, 21, 22, 23, 24\}$, with universe $\mathcal{U} = \{0, 1, 2, \ldots, 24\}$ and $W = 4$. Elements were inserted in increasing order of their value. Each node indicates a pair (b, w), where b is the identifier of the block and w is the bitmask word corresponding to the block. Nodes are colored black or red according to the color they would have in an underlying Red-Black tree. **(b)** Order of iteration in $\mathcal{T}_S$ of the elements of S, in increasing order of their values. 44

4.1 The unique (up to symmetry) FAT and MIN-AREA grid n-ogons for $n \in \{4, 6, 8, 10, 12\}$ (adapted from [94]). 46

4.2 Evolution of the partition and instance generation times for MIN-AREAS with number of vertices $n = |\mathcal{F}|$ **(left)** and partition size $|\Pi(P)|$ **(right)**. The time it takes to generate an instance from $\Pi(P)$ is insignificant when compared to the partition time, which hints that partition will be a bottleneck for our algorithm. 48

4.3 Preprocessing and solving times for MIN-AREA instances with $n = |\mathcal{F}|$ vertices **(left)** and partition size $|\Pi(P)|$ **(right)**. Generating a SET COVER instance from P takes consistently longer than solving it afterwards using RANDOMIZED-SET-COVER. 49

4.4 Evolution of the partition and instance generation times for FATS with number of vertices $n = |\mathcal{F}|$ **(left)** and partition size $|\Pi(P)|$ **(right)**. Just like for MIN-AREAS, mapping a partition to a SET COVER instance is faster than computing the partition itself. 50

4.5 Preprocessing and solving times for Fat instances with $n = |\mathcal{F}|$ vertices **(left)** and partition size $|\Pi(P)|$ **(right)**. Since the generated Set Cover instance has a size that is quadratic in n, both times grow faster than for a Min-Area with the same number of vertices. 50

4.6 Partition and instance generation times for Randoms, averaged over all the instances with a fixed number of vertices $n = |\mathcal{F}|$ **(left)** and a fixed partition size $|\Pi(P)|$ **(right)**. Similarly to Min-Areas and Fats, instance generation takes considerably less time than computing the partition. 51

4.7 Preprocessing and solving times for Randoms, averaged over all the instances with a fixed number of vertices $n = |\mathcal{F}|$ **(left)** and with a fixed partition size $|\Pi(P)|$ **(right)**. Clearly the preprocessing phase dominates the time taken by the algorithm. 52

4.8 The variation of the maximum value hit by APX when collectively considering all the Random instances with a fixed n **(left)** or $|\Pi(P)|$ **(right)**. There is no clear relationship between these parameters. 52

4.9 Growth of preprocessing and solving times for Large-Random polygons. Once again, the time it takes to solve a Set Cover instance remains moderately low with n and $|\Pi(P)|$, although the partition time explodes. 53

4.10 The variation of the maximum value hit by APX when collectively considering all the Large-Random instances with a fixed n **(left)** or $|\Pi(P)|$ **(right)**. As observed in Randoms, no evident relationship exists between the parameters. . 54

4.11 Comparison of the average number of pieces in $\Pi(P)$ for each grid n-ogon class, as the number of vertices n increases from 8 to 200. 55

4.12 From a Min-Area with n vertices to a Min-Area with $n+2$ vertices, the number of visibility regions increases by a square and two triangles. The case where $n = 4$ does not satisfy the property because the respective Min-Area is convex. . . . 56

4.13 The number of pieces in the partition $\Pi(P)$, induced by vertex visibility regions, of a Min-Area P with n vertices perfectly obeys the identity $|\Pi(P)| = \frac{3n}{2} - 4$ for every $n = 8, 10, \ldots, 200$. 56

4.14 The number of pieces in the partition $\Pi(P)$, induced by vertex visibility regions, of a Fat P with n vertices perfectly obeys the identity $|\Pi(P)| = 76 + \frac{17(n-18)}{2} + \left\lfloor \frac{(n-18)^2}{16} \right\rfloor + 2\left\lfloor \frac{(n-20)^2}{16} \right\rfloor$ for every $n = 16, 18, \ldots, 200$. 58

4.15 The average size of $\Pi(P)$ for a Random grid n-ogon P seems to grow linearly with n. The part of the plot with $14 \leq n \leq 200$ integrates results from the Random class, while for $250 \leq n \leq 1000$ the data used comes from Large-Random. These data, coming from two independent sessions, exhibit a smooth continuity. 58

5.1 **(a)** Simple $(\mathcal{R},\mathcal{C})$-SCOT, with $|\mathcal{R}| = 7$ and $|\mathcal{C}| = 6$. Rooms are represented as white rectangles and corridors as shaded rectangles. The corridor connecting rooms R_1 and R_2 is *horizontal* and the corridor connecting rooms R_1 and R_3 is *vertical*. **(b)** SCOT with a cycle involving rooms R_1, R_2, R_3 and R_4. **(c)** SCOT with multiple corridors between the same sides of rooms R_1 and R_2 and also between R_2 and R_3. 60

5.2 Convex vertices may be required in an optimal vertex-guard set for a general orthogonal polygon under r-visibility. For this polygon, $\{v_1\}$ is a minimum-cardinality guard set, but if we restrict ourselves to reflex vertices the only feasible solution is $\{v_2, v_3\}$, which is not optimal. 61

5.3 Blind spots on the corners of a room. These regions cannot be r-seen from outside the room. 62

5.4 Step-by-step execution of the greedy algorithm for determining a minimum-cardinality vertex-guard set in a simple SCOT P (with no room cycles and no multiple corridors). **(a)** Initially, a tree T representing room adjacencies in P is constructed. **(b-g)** We keep detaching leaf nodes from T until a single node remains. We break the ties arbitrarily; in this example, leaves are removed in increasing order of the node identifier. In each intermediate step, a guard is placed on a reflex vertex to r-see exactly one room and one corridor. **(h)** When the last leaf node remaining is removed, the entire polygon P becomes covered. The number of guards used is $|\mathcal{R}| = 7$. 63

5.5 **(a)** Simple SCOT P. **(b)** The stretches of the two horizontal corridors of P perfectly coincide, so they are aligned corridors. **(c)** The stretches of the two vertical corridors of P intersect, but do not coincide, so they are not aligned nor disjoint. 64

5.6 **(a)** r-independent SCOT. Notice that, even though the stretches of corridors C_1 and C_2 coincide, C_1 and C_2 are not adjacent. **(b)** A SCOT that is not r-independent. The stretches of corridors C_3 and C_4 intersect in a vertical line. A guard placed at the top-right corner of C_4 would r-see both corridors. 64

5.7 There are four super-corridors (r-equivalence classes of corridors) in this polygon: $\{C_1\}$, $\{C_2, C_3, C_4\}$, $\{C_5\}$ and $\{C_6\}$. Corridors C_1 and C_5 are not r-equivalent because they are not adjacent and do not belong to a succession of adjacent, aligned corridors. The same happens with corridors C_4 and C_6. 64

5.8 **(a)** If r-equivalent corridors were not compressed, a succession of $|S| - 1$ corridors connecting a subset of rooms S would contribute with $2(|S| - 1)$ pairs to $|Q|$, because each corridor is incident to exactly two rooms. **(b)** By compressing this maximal succession of aligned corridors into a super-corridor, Q now gets $|S|$ pairs instead: one for each room that is incident to the super-corridor. 66

5.9 **(a)** r-independent SCOT P, with four super-corridors: $C'_1 = \{C_1, C_2\}$, $C'_2 = \{C_3\}$, $C'_3 = \{C_4\}$ and $C'_4 = \{C_5, C_6, C_7\}$. A partial guard set watching over everything in P except for rooms R_5 and R_6 is presented. Each guard covers exactly one room and one super-corridor. **(b)** Bipartite graph H, whose subset of nodes A represents rooms in P and whose subset B represents super-corridors. The edges contained in a B-perfect matching $M^\star$, corresponding to the partial guard set given in **(a)**, are highlighted. The guard set could be extended with two guards, one in R_5 and another in R_6, to optimally cover the entirety of P. 68

5.10 **(a)** A vertex-guard that is chosen to r-see both the room R_2 and the super-corridor that crosses all three rooms can be placed in any of the four vertices v_1, v_2, v_3 or v_4, because they are r-equivalent. **(b)** If the matching $M^\star$ determines that a vertex-guard has to be placed seeing both the room R_2 and the super-corridor, there are two possibilities for the guard: either at v_1 or at v_2. 69

5.11 **(a)** r-independent SCOT P, showing a partial vertex-guard set covering every region except for room R_6 and super-corridor C'_2. It is a counterexample for the conjecture that $\max\{|\mathcal{R}|, |\mathcal{C}'|\}$ vertex-guards are sufficient for watching over any r-independent $(\mathcal{R}, \mathcal{C}, \mathcal{C}')$-SCOT P. **(b)** Bipartite graph $H = (A \cup B, E)$, along with a maximum cardinality matching $M^\star$, representing the partial guard placement in P. H admits no perfect matching and therefore two extra guards are needed for covering R_6 and C'_2. 70

5.12 **(a)** Guard g only r-sees a corridor. **(b-c)** By moving g to an incident room, it still sees the corridor, but now also sees a room. Therefore, the solution has not become worse. 71

5.13 **(a)** Network G with demands and capacities, an instance of Minimum Flow with Demands. Each edge (u, v) is annotated with a range $[d(u, v), c(u, v)]$. **(b)** Example of a feasible flow in G, where on each edge (u, v) it is indicated inside parenthesis the amount that flows through it, as $(f(u, v))$. 72

5.14 **(a)** Small original network G with edge capacities and demands. **(b)** New, transformed network G' obtained by reducing Minimum Flow with Demands to Maximum Flow. The value on an edge (u, v) refers to the transformed capacity $c'(u, v)$. 72

5.15 Example adapted from [44]. **(a)** Original network G with edge capacities and demands. **(b)** New network G' obtained by reducing Minimum Flow with Demands to Maximum Flow. The value on an edge (u, v) refers to the transformed capacity $c'(u, v)$. **(c)** A saturating flow f' in G', obtained by running a maximum flow algorithm over the transformed network. **(d)** A feasible flow f in G corresponding to f' by taking $f(u, v) = f'(u, v) + d(u, v)$ for every edge $(u, v) \in E$. Notice that $f'(t, s) = |f| = 11$. 74

5.16 **(a)** The same r-independent SCOT P of Figure 5.9a, with four super-corridors: $C'_1 = \{C_1, C_2\}$, $C'_2 = \{C_3\}$, $C'_3 = \{C_4\}$ and $C'_4 = \{C_5, C_6, C_7\}$. **(b)** The network G corresponding to P that is an instance of Minimum Flow with Demands. It has a total of 26 nodes and 41 edges (including those on the gadgets). We display nodes vertically split into five conceptual groups: source (s), gadgets for horizontal super-corridors, room gadgets, gadgets for vertical super-corridors and sink (t). Notice that pseudo-gadgets $\Gamma(C'_{hf})$ and $\Gamma(C'_{vf})$, for both horizontal and vertical super-corridors, are connected to every room to enable solutions where guards are placed in positions from where they cannot see any corridor in one or both directions. Every edge in G has capacity ∞. All the edges belonging to real gadgets have demand 1; every other edge has demand 0. 76

5.17 **(a)** Minimum feasible flow f in the network G, with $|f| = 6$. The flow has not been annotated on edges (u, v) with $f(u, v) = 0$ to avoid clutter. **(b)** SCOT P, with an optimal set of 6 point-guards that has been determined by flow f. . . . 79

5.18 **(a)** Example of a 25-polyomino P. **(b)** The 7×7 minimal unit grid $\Delta(P)$ that contains P. 80

5.19 Anchor gadget Γ. 81

5.20 Three anchor gadgets laying side by side and connected by horizontal corridors (only as an example). Notice that corridors linking two adjacent anchor gadgets are contained in the plane region $6 \leq y \leq N + 6$. 81

5.21 Example of a reduction from Minimum Polyomino r-Guard to Minimum SCOT r-Guard. SCOT B is not presented to scale so that the correspondence between the guard sets in both instances is visually clearer. 82

5.22 An orthogonal polygon P whose vertices have rational coordinates, reprinted from [1]. P can be covered under straight-line visibility with 9 point-guards if we allow irrational coordinates, but requires 10 point-guards if we do not. 85

5.23 The grid partition $\mathit{\Pi}_{HV}(P)$ of a SCOT P. 86

6.1 A 19-polyomino P (shaded region) within a 10×3 grid and a corresponding minimum-cardinality rook-guard set of 5 rooks, with coordinates $(0, 0)$, $(1, 2)$, $(4, 0)$, $(5, 1)$ and $(9, 0)$. P has a one-tile hole with coordinates $(2, 1)$. 89

6.2 Traversing the grid g in row-major order as we keep the current dynamic programming state $(r, c, h_l, h_r, V_u, V_d)$, while standing at cell (r, c). The yellow, striped tiles indicate for which row ($r - 1$ or r) each element $0 \leq x < C$ in V_u and V_d is carrying information from. 91

6.3 Certificate propagation. When we reach a dynamic programming state for this instance with $(r, c) = (3, 2)$, $h_l = \mathsf{False}$ and $V_u = \emptyset$, we must have $h_r = \mathsf{True}$ and $V_d = \{2\}$. Delaying the rook placement by propagating unchecked certificates from the top-right and bottom-left cells is the only way to achieve an optimal solution (1 rook) for this instance. 92

6.4 A run of the algorithm over a 5-polyomino P. The operations performed and the decisions taken at each state when obtaining one of the optimal solutions for P (in fact, the lexicographically minimum one) are shown. The bolder tile marks our current position at each step. 94

List of Algorithms

1 Greedy-Set-Cover($\mathcal{U}, \mathcal{F}$) 21
2 Randomized-Set-Cover($\mathcal{U}, \mathcal{F}$) 25
3 Uniform-Randomized-Set-Cover($\mathcal{U}, \mathcal{F}$) 28
4 Decide-$\subseteq(A, B)$ 32
5 Randomized-Set-Cover($\mathcal{U}, \mathcal{F}$), revised 35
6 Identify proper subsets in $\mathcal{F}$ 37
7 Insert(x) 39
8 Erase(x) 40
9 Choose-Random() 40

Acronyms

AGP Art Gallery Problem

AGT Art Gallery Theorem

ETH Exponential Time Hypothesis

FPT fixed-parameter tractable

MVG Minimum Vertex Guard

OAGT Orthogonal Art Gallery Theorem

PTAS polynomial-time approximation scheme

TOP thin orthogonal polygon

Chapter 1

Introduction

The warden of a museum wants to make sure that all the points in its interior are guarded. For that, he will place guards at strategic positions. Each guard is stationary, but is free to look around. How many guards does the warden need to hire?

One can see that, if the museum has the shape of a *convex polygon* (one whose internal angles are all less than or equal to 180°), a single guard is enough (Figure 1.1a). By definition, the same holds for *star-shaped polygons* – polygons that contain at least one point from which the entire boundary is visible (Figure 1.1b).

(a) **(b)**

Figure 1.1: **(a)** A convex polygon. **(b)** A star-shaped polygon. Each of them only requires one guard.

What about more general polygons? Despite the apparent simplicity of this question, it has nevertheless triggered decades of research by mathematicians and computer scientists. Traditionally known as the Art Gallery Problem (AGP), it is not only appealing, but also one of the most celebrated and thoroughly studied problems in Computational Geometry. The original formulation, posed in 1973 by Klee [81], asks about the minimum number of point-guards that are required to watch over the interior of any simple polygon P (that is, one without holes or self-intersections) with n vertices in the plane, where two points $p, q \in P$ see each other if the closed line segment pq lies completely within P.

Besides being of theoretical interest, it is not difficult to see how this problem is also practically useful: many real-life situations can be modeled by slight variations of AGP. How many 360º surveillance cameras are needed to watch over a university campus? How many lamps does one need to install in order to keep an entire basement illuminated (Figure 1.2)? How many Wi-Fi repeaters do we need to place in a building so that everyone has an Internet connection?

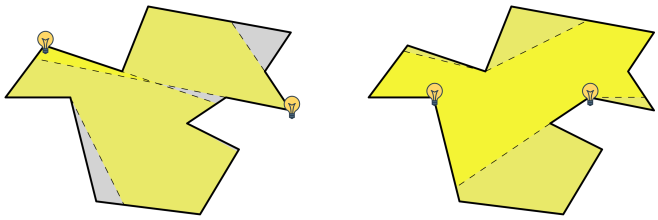


Figure 1.2: Two lamp placements on a polygonal region P under straight-line visibility. The left one does not fully cover P, while the one on the right is a feasible (and optimal) solution.

Many variants of the problem have been proposed through the years, mainly regarding the way guards see, which positions guards are allowed to be at, and which section of the polygon has to be covered.

In this thesis we focus on characterizing the computational and combinatorial complexity of guarding orthogonal polygons with guards either on vertices – Minimum Vertex Guard (MVG) – or anywhere in the polygon. We research the hardness of guarding specific polygon families under specific visibility models, improving known bounds. Different approaches commonly employed to attack NP-hard problems are explored, namely randomized algorithms, approximation algorithms and parameterized algorithms. On our journey to develop a new methodology, we meet the renowned Set Cover problem, whose solutions we approximate through a new randomized approach, and define new families of orthogonal polygons, for which we provide efficient polynomial algorithms for tractable cases and a hardness proof for the general case.

In Section 1.2 we describe how the subjects we deal with are organized in this thesis and hint at our contributions on them, but before that we believe an appropriate context must be given. Without providing a theoretical background on the current state of affairs, the goals of the present work would perhaps seem abrupt and their motivation hard to follow. As every research and experimentation is the product of knowledge accumulated over time, we begin by presenting in Section 1.1 an overview of the relevant history of the problem in an attempt to clarify the state of the art that this work launches from. Some of the technical terminology used in Section 1.1 will be addressed and formalized again in Chapter 2.

1.1 Background on the Art Gallery Problem

While much progress has been made, the Art Gallery Problem is not yet fully solved. Aside from the main classic results that have been established right after the question gained publicity, more and more fresh discoveries keep coming to light even in recent years. Chvátal [29] was the first to show, in 1975, that $\lfloor n/3 \rfloor$ point guards are always sufficient, and sometimes necessary (Figure 1.3), to guard any simple polygon with n vertices. This important result became known as the Art Gallery Theorem (AGT).

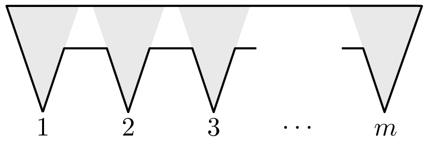


Figure 1.3: The example given by Chvátal to show that sometimes $\lfloor n/3 \rfloor$ guards are necessary. It consists of a comb-shaped polygon with $n = 3m$ vertices, for some integer $m \geq 1$. Notice that the points $1, 2, \ldots, m$ can only be seen by a guard placed in the shaded triangle that contains it. Since these triangles are disjoint, we need a guard for each of these m points separately, yielding the $m = \lfloor n/3 \rfloor$ bound.

Three years later, Fisk [49] published a more compact, one-paragraph demonstration of the same theorem, whose elegance and simplicity earned him a spot in the "*Proofs from THE BOOK*" [5]. His proof is based on 3-coloring the vertices of the dual graph of a polygon's triangulation and placing guards on all the vertices of the least frequent color (Figure 1.4). Every triangle has a vertex of each color, so it is definitely covered if we place a guard on each vertex of any fixed color. Since the least frequent color cannot occur more than $\lfloor n/3 \rfloor$ times, this gives a proof for the AGT.

In 1981, Avis and Toussaint [10] published a divide-and-conquer algorithm that, closely following Fisk's constructive proof, determines a concrete positioning of up to $\lfloor n/3 \rfloor$ guards in a simple polygon in $\mathcal{O}(n \log n)$ time. Later improvements have shown how to actually do it in linear-time, using Chazelle's revolutionary triangulation algorithm [27] and Kooshesh-Moret 3-coloring algorithm [71].

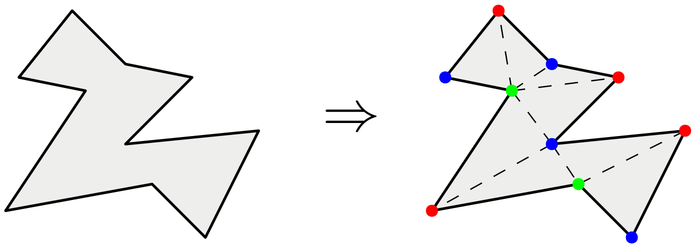


Figure 1.4: Illustration of the sufficiency proof by Fisk's method. The polygon is triangulated and a 3-coloring of its vertices is found. The green vertices form a valid guard set of size 2, which obeys the upper bound of $\lfloor n/3 \rfloor = \lfloor 10/3 \rfloor = 3$ given by the Art Gallery Theorem.

Through time, many authors contributed with an umbrella of interesting variants of the AGP. For example, in the original definition, guards can see in straight lines, with a 360° range; this is referred to as the *general* or *straight-line visibility model*. However, one may opt to replace this notion so that instead they have a limited range of vision (α-floodlights [46]) or only see axis-parallel rectangles contained in orthogonal polygons – those whose edges all meet at right angles (r-visibility).

One can also enforce specific locations where guards are allowed to be placed: anywhere in P (point-guards, Figure 1.5b), only at vertices (vertex-guards, Figure 1.5a) or only on edges (boundary-guards). We may not as well require the entire polygon P to be guarded: an exact region within P that we want to be watched over can also be specified. Usually, these variations are abbreviated with names of the form G-S ART GALLERY, where one wants to find the smallest subset of $G \in \{\text{VERTEX}, \text{EDGE}, \text{POINT}\}$ that covers all the points in $S \in \{\text{VERTEX}, \text{EDGE}, \text{POINT}\}$. The VERTEX-POINT variation of the problem is often called MVG in the literature. Given that we have more freedom regarding where to place point-guards, the size of a point-guard set for a polygon P is never greater than the size of a vertex-guard set for the same polygon P (and is often smaller). Note that a vertex-guard set is, by definition, also a valid point-guard set and a valid boundary-guard set.

(a) (b)

Figure 1.5: (a) Two vertex-guards are required for guarding this polygon. **(b)** A single point-guard would be enough.

Each variant of the AGP leads to different combinatorial results and complexity bounds, which may vary wildly according to the polygon class being considered. For instance, several alternate proofs have shown $\lfloor n/4 \rfloor$ guards to be sufficient, and in some cases necessary, for guarding orthogonal polygons with n vertices [64] (see Figure 1.6). This tighter bound is known as the Orthogonal Art Gallery Theorem (OAGT).

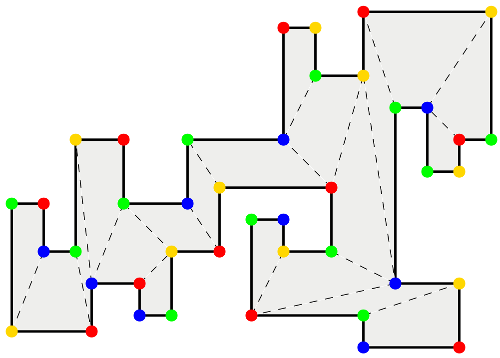


Figure 1.6: Illustration of the sufficiency proof by the method of Kahn et al. [64], adapted from [99]. Similar in spirit to Fisk's proof, the authors prove OAGT by showing that any orthogonal polygon can be decomposed into convex quadrilaterals by adding non-intersecting line segments between vertices (convex quadrilateralization), which induces a 4-coloring of the vertices.

NP-hardness results The Art Gallery Problem's innocent-looking description can be deceiving, though. In fact, it has been proved by Lee and Lin to be NP-hard [73], meaning that one cannot hope for a polynomial-time algorithm that guards an arbitrary polygon P using either vertex-guards, edge-guards or point-guards, unless $\mathsf{P} = \mathsf{NP}$. Perhaps surprisingly, the NP-hardness result still holds for vertex-guarding or point-guarding orthogonal polygons [83], even if we only want to cover their vertices [68]. This even holds for very restricted classes of polygons, such as simple *monotone polygons* [72] and *k-link polygons*, for $k \geq 2$ [23].

Parameterized algorithms Given that the problem is hard in a general setting, one of the ways of attacking it is to explore specific structural properties of instances and take advantage of them to develop *parameterized algorithms* that are very efficient in constrained scenarios. In other words, one would like to find some instance parameter that tends to be pragmatically small enough, so as to enable an algorithm to be very efficient for all practical purposes. Formally, if the instance size is n and the parameter chosen is k, the existence of an algorithm running in time $f(k)\, n^{\mathcal{O}(1)}$, for some computable function f, would ensure fixed-parameter tractability of the problem.

Following this reasoning, in 2020, Agrawal et al. [4] have shown that if, for example, one only needs to guard the vertices of P with vertex-guards, the problem becomes fixed-parameter tractable (FPT) with parameter the number of reflex vertices r. Concretely, this gives an approach running in time $r^{\mathcal{O}(r^2)}\, n^{\mathcal{O}(1)}$ that becomes efficient when the polygon P is "almost convex", regardless of the number of the vertices, n.

The Exponential Time Hypothesis (ETH) is the unproven assumption, conjectured by Impagliazzo and Paturi [60], that there is no $2^{o(n)}$-time algorithm for 3-SAT on n variables and, if true, it has the consequence that $\mathsf{P} \neq \mathsf{NP}$. In spite of recent positive developments, it has also been proved that for Point-Point AGP or Vertex-Point AGP there is no algorithm running in time $f(k)\, n^{o(k/\log k)}$, for any computable function f and allowed number of guards k, even for simple polygons, unless the ETH fails [20]. These results imply that the problem is $W[1]$-hard when parameterized by the number of guards, k.

Approximation algorithms Another possible line of attack is to focus on finding guard sets that do not need to be optimal as long as they are close to the minimum (in a mathematically precise sense). These *approximation algorithms* are useful if we do not require sticking to exact solutions, which is a perfectly reasonable assumption in many realistic settings. These ideas can even be combined with randomization to achieve expectedly good running times.

Ghosh proposed in 2010 a reduction from MVG to the Set Cover problem, by partitioning the polygon into vertex visibility regions that could be used to obtain a greedy $\mathcal{O}(\log n)$-approximation polynomial algorithm [53]. Krohn and Nilsson [72] gave in 2013 an $\mathcal{O}(OPT)$-approximation algorithm for guarding orthogonal polygons. This means that, if the smallest guard set for a given polygon P has size OPT, their algorithm assuredly finds one with size $\mathcal{O}(OPT^2)$. The strength of this algorithm is that the approximation factor is entirely independent of the number of vertices of the polygon. Three years later, the point-guard AGP was discovered to admit an $\mathcal{O}(\log OPT)$-approximation algorithm running in randomized polynomial time for simple polygons, assuming integer coordinates and a specific general position [21] which can be achieved with small random perturbations. Also, a polynomial-time algorithm with factor $\mathcal{O}(\log\log OPT)$ was published for guarding simple polygons with vertex- or edge-guards [69].

More recently, in 2017, Bhattacharya et al. [13] published allegedly 18- and 27-factor approximation algorithms for guarding the vertices, the boundary or the interior of simple polygons using vertex-guards, meaning that the size of the found guard set is guaranteed to be at most a multiplicative constant away from the optimum. These findings settle an old conjecture by Ghosh [53] that there exist constant-factor approximation algorithms for MVG.

Given the practicability of $\mathcal{O}(1)$-approximating a minimum solution to AGP, it makes sense to wonder whether one would also be able to approximate it with an arbitrary desired precision in polynomial time. In other words, given an instance of the problem and a parameter $\epsilon > 0$, we could be interested in finding a solution that is within a factor of $1+\epsilon$ of being optimal. This would imply AGP to admit what is called a polynomial-time approximation scheme (PTAS). For specific classes of polygons, such as *weakly-visible polygons* under Vertex-Vertex or Vertex-Edge guarding, there indeed exist PTASs [67].

Nevertheless, in 1998 and 2001 Eidenbenz et al. [40, 41] had published the first inapproximability results for AGP, proving that guarding an n-vertex polygon P with or without holes either with vertex-guards, edge-guards or point-guards is indeed APX-hard. This means that, for polygons with holes, unless $\mathsf{NP} \subseteq \mathsf{TIME}(n^{\mathcal{O}(\log\log n)})$ these problems cannot be approximated by a polynomial-time algorithm with ratio $\frac{1-\epsilon}{12}\ln n$, for any $\epsilon > 0$. For polygons without holes, on the other hand, there exists a constant $\epsilon > 0$ such that no polynomial-time algorithm can possibly guarantee an approximation ratio of $1+\epsilon$ for each of these problems.

Back to the realm of exact solutions, results under the notion of r-visibility have usually been quite encouraging. Worman and Keil [102] gave, in 2007, an exact algorithm to guard simple orthogonal polygons under r-visibility with complexity $\widetilde{\mathcal{O}}(n^{17})$, by determining a maximum independent set in a perfect graph as a sub-problem. Furthemore, despite guarding tree/thin orthogonal polygons (TOPs) under the standard visibility model having been proved NP-hard – in fact, APX-hard – by Tomás [91], Biedl and Mehrabi [14, 15] have shown how simple TOPs can be r-guarded in polynomial time. If they have an arbitrary number of holes the problem remains NP-hard, however. Simple orthogonal *path polygons* with dent edges can similarly be r-guarded in linear time [58].

1.2 Contributions and thesis outline

The previous section addressed several directions that the current thesis follows. In this work we explore applications of techniques commonly used to attack NP-hard problems to conceive new efficient algorithms for several AGP variants and describe their computational complexity. Including this Introduction, this thesis has 7 chapters, and its structure is as follows.

We start off by reviewing, in Chapter 2, geometric and computational hardness notions that are recurrently invoked throughout the thesis, some of which have already been mentioned in Section 1.1.

The remaining chapters naturally unfold into two main parts. We begin the first half in Chapter 3, where we seek solutions for the problem of finding a minimum-cardinality vertex-guard set for orthogonal polygons through a reduction to Set Cover. With that in mind we propose a new randomized algorithm for approximating Set Cover, study kernelization techniques to speed up the algorithm and improve its probability of success, and present new efficient data structures that have been developed for that purpose.

Chapter 4 describes the setup we adopted for empirically evaluating the performance of the randomized algorithm that is proposed in Chapter 3 against a dataset of orthogonal polygons. We discuss the results we obtained and, based on them, formulate a conjecture about the growth of polygon convex partition sizes.

The second half begins with Chapter 5, in which we present a new class of orthogonal polygons, the SCOTs, and study the complexity of guarding them under r-visibility. We provide efficient algorithms for the cases where the problem is polynomially solvable, and give a proof of NP-hardness and approximation algorithms for the remaining ones.

An exact algorithm for finding an optimal guard set for polyominoes under rook-vision is proposed in Chapter 6, and this proves a result on the fixed-parameter tractability of the problem.

Finally, in Chapter 7 we draw conclusions about the work done, highlight the main results we arrived at and discuss prospects of future work.

Chapter 2

Preliminaries

We dedicate the present chapter to defining terminology and formalizing recurrent concepts that are transversal to the entirety of this thesis. We begin by defining relevant geometry concepts, namely the central object of study in our work: polygons.

2.1 Geometry

2.1.1 Polygons

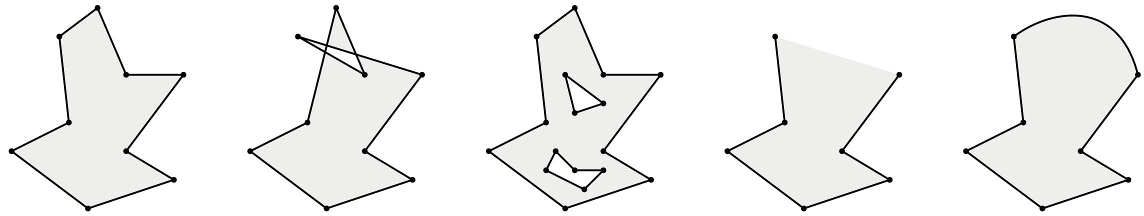
Definition 2.1 (Simple polygon)**.** A *simple polygon* P is a subset of the plane $\mathbb{R}^2$ that is enclosed by a finite, cyclic chain of $n \geq 3$ non-disjoint line segments. Only pairs of segments that are adjacent in the chain do intersect, and their intersections coincide with segment endpoints. The intersection points are the *vertices* of P and the line segments are its *edges*.

In some scenarios, the constraint that only adjacent edges intersect at their common endpoints is relaxed, enabling the polygon to have self-intersections (Figure 2.1b). Other times, more than one cyclic chain is used to define the boundary of the polygon, introducing internal *holes* (Figure 2.1c). In these two cases, the polygon is said to be *non-simple* – but, if the polygon has no holes or self-intersections, it remains simple according to Definition 2.1 (Figure 2.1a). We will work with both simple and non-simple polygons in this thesis, making it clear whether we are referring to one type or another. All the non-simple polygons we treat have holes, but not self-intersections.

It is usual to denote the interior of a polygon P by $\mathsf{int}\, P$, its exterior by $\mathsf{ext}\, P$ and its boundary by ∂P. We consider ∂P as part of the polygon, so $P = \mathsf{int}\, P \cup \partial P$.

Each vertex v of P is often classified as *convex* or *reflex* according to its internal angle, that is, the angle formed by the two edges that are incident to v with the interior of P (Figure 2.2).

Definition 2.2 (Convex and reflex vertices)**.** A vertex v of a polygon is said to be *convex* if its



(a) (b) (c) (d) (e)

Figure 2.1: **(a)** Simple polygon. **(b)** Polygon with self-intersections. **(c)** Polygon with holes. **(d)** Not a polygon, because the chain of edges is not closed. **(e)** Not a polygon, because edges cannot be curve.

internal angle measures 180° or less. On the contrary, v is said to be *reflex* if its internal angle measures strictly more than 180°.

The main class of polygons our work is based upon is that of orthogonal polygons, which we now formalize.

Definition 2.3 (Orthogonal polygon)**.** A polygon P is *orthogonal* if each one of its convex vertices has an internal angle of 90° and each one of its reflex vertices has an internal angle of 270°.

It is useful to view every edge of an orthogonal polygon as being parallel to a pair of orthogonal coordinate axes (which we can take as the x- or y-coordinate axes, without loss of generality). In this representation, edges alternate between horizontal and vertical.

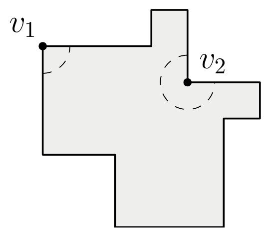


Figure 2.2: Orthogonal polygon. v_1 is a convex vertex and v_2 is a reflex vertex.

Theorem 2.1. *A simple orthogonal polygon P of n vertices has exactly $r = \frac{n-4}{2}$ reflex vertices and $c = \frac{n+4}{2}$ convex vertices.*

Proof. Each of the r reflex vertices of P has an internal angle of 270° and each of the c convex vertices of P has an internal angle of 90°. Since the sum of the internal angles of a simple polygon with n vertices is $180(n-2)°$, we have the system of equations

$$\begin{cases} n = r + c \\ 180(n-2) = 270r + 90c \end{cases}$$

which solves to $r = \frac{n-4}{2}$ and $c = \frac{n+4}{2}$. $\square$

Definition 2.4 (Polyomino)**.** An *m-polyomino* P_m is a polyform that results from the (finite) union of m squares of equal size edge to edge. More formally, an m-polyomino is a finite subset of a regular square tiling of the plane, consisting of m closed unit squares whose interior is connected [16, 18]. We may fairly interchangeably refer to the unit squares as *pixels*, *cells* or *tiles*. One can construct any polyomino by starting with a pixel and then repeatedly gluing new pixels to the already constructed set, edge to edge.

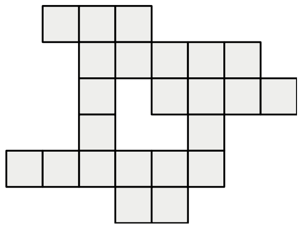


Figure 2.3: A 23-polyomino with a hole.

2.1.2 Visibility

This thesis is about watching over art galleries, and we could not discuss any ideas without first making it clear how guards may actually see.

Definition 2.5 (Straight-line visibility)**.** Let P be a simple polygon. Two points $p, q \in P$ *see* each other (or *are visible* from each other) under *straight-line* (or *unrestricted*) visibility if the closed line segment pq does not intersect $\mathsf{ext}\, P$.

Definition 2.6 (Rectangular visibility, r-visibility)**.** Let P be a simple orthogonal polygon. Two points $p, q \in P$ *see* each other (or *r-see* each other, or are *r-visible* from each other) under *r-visibility* if the minimal axis-aligned rectangle containing p and q does not intersect $\mathsf{ext}\, P$. Note that the rectangle degenerates to a line segment if p and q have the same x- or y-coordinate.

Definition 2.7 (Visibility polygon of a point)**.** Let P be a simple polygon. The *visibility region* of a point $p \in P$, which we denote by $\mathcal{V}(p)$, is the set of all the points of P that are visible to p. We may write $\mathcal{V}_r(p)$ if the adopted model is r-visibility.

When the visibility notion we assume is r-visibility and that can be clearly inferred from the context, we might simply use the term *see* instead of r-see. Figure 2.4 compares the visibility regions of the same vertex under different models.

Clearly $\mathcal{V}_r(p) \subseteq \mathcal{V}(p)$ for any point $p \in P$. This implies that any feasible guard set in the r-visibility model is also a feasible guard set in the straight-line visibility model (in general, the same does not hold in reverse).

It may happen, in some polygons P, that the visibility region of a point is the union of a subpolygon $Q \subseteq P$ and a set of line segments, called *antennas* or *needles* (Figure 2.5a). However,

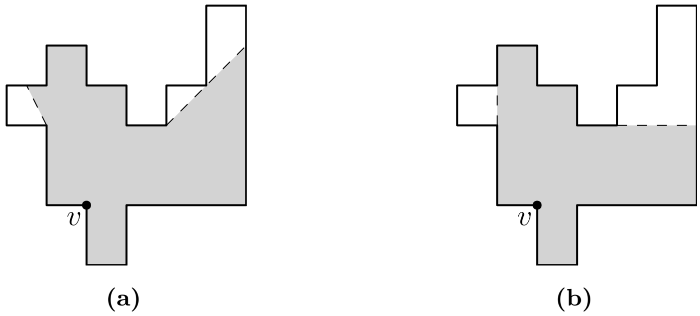


Figure 2.4: Which subset of an orthogonal polygon a guard at vertex v sees **(a)** under straight-line visibility and **(b)** under r-visibility.

in this work we restrict $\mathcal{V}(p)$ to Q, under a visibility assumption that is called *regularized visibility* [95] (Figure 2.5b). The regularized region seen by p is therefore

$$\mathcal{V}^*(p) = \mathsf{closure}(\mathsf{int}(\mathcal{V}(p))) = (\mathcal{V}(p) \setminus \partial\mathcal{V}(p)) \cup \partial\,(\mathcal{V}(p) \setminus \partial\mathcal{V}(p))\,.$$

We will use $\mathcal{V}(p)$ to mean $\mathcal{V}^*(p)$. This assumption allows us to disregard degeneracies and, as we argue in due course, does not spoil the correctness nor optimality of our algorithms. The regularized visibility region $\mathcal{V}(p)$ is a star-shaped polygon, so we also refer to it as a *visibility polygon*.

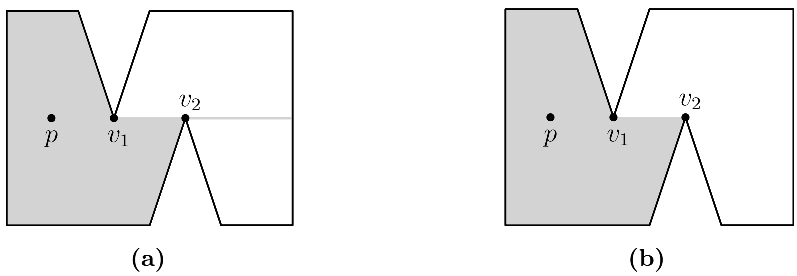


Figure 2.5: **(a)** Non-regularized straight-line visibility. Point p is collinear with vertices v_1 and v_2 and sees the union of a polygon and a needle. **(b)** Regularized straight-line visibility. The visibility region of p does not include degeneracies and the needle is discarded.

2.1.2.1 Computing visibility

Efficient algorithms are known for computing $\mathcal{V}(v)$ for a vertex v within a polygon P of n vertices under straight-line visibility – but some of them were proven faulty.

Shamos' algorithm [85] allegedly runs in $\mathcal{O}(n)$, but was shown to be incorrect by El-Gindy [42]. El-Gindy and Avis [43] published another linear-time algorithm that updates three stacks while examining vertices in clockwise order, which in turn was also proven incorrect by Joe and Simpson. Lee [74] proposed an algorithm that only uses one stack and is simpler than El-Gindy and Avis' – however Joe and Simpson have also shown that it fails for some polygons. Joe-Simpson's algorithm [62] was then published as an attempt to correct Lee's algorithm, but corner-case instances were found through random testing during the organization of a geometric programming contest for university students ("Art Gallery Competition") [84].

Instead of applying these algorithms, in this work we compute visibility regions using the *triangular expansion* algorithm. We privileged it due to its robustness, efficiency and simplicity. The technique was first described in Tomás et al. [90, 94] as an adaptation of a work by Aronov et al. [8], and was later integrated in CGAL [25].

As a preprocessing phase, the triangular expansion algorithm starts by computing a triangulation of the polygon P (see Definition 2.10). A query asking for the visibility region $\mathcal{V}(p)$ of a given point p is answered by performing a breadth-first search over the triangulation. We start from the triangle that contains p and successively propagate visibility cones to adjacent triangles, through the sections of the shared edges that are seen by p (Figure 2.6). Empirical tests [25] demonstrate that triangular expansion is much faster than previous algorithms – about two orders of magnitude on large inputs. This is because it is in some sense output-sensitive: the search only involves relevant triangles that are totally or partially seen by p.

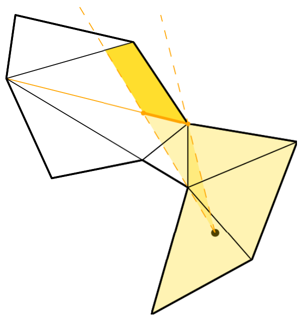


Figure 2.6: Triangular expansion algorithm. Visibility is propagated to neighbour triangles through *windows* on shared edges.

2.1.3 Arrangements

Definition 2.8 (Arrangement)**.** A planar *arrangement* $\mathcal{A}(\mathcal{C})$ is the subdivision of the $\mathbb{R}^2$ plane induced by a set $\mathcal{C}$ of geometric objects, such as curves.

We will only consider arrangements in $\mathbb{R}^2$ induced by line segments. Such an arrangement can be described by three types of topological objects: vertices (0-dimensional), edges (1-dimensional) and faces (2-dimensional). Each face is a maximal connected component of the $\mathbb{R}^2$ plane that does not contain any vertex or edge (in this sense, it is an open set). The boundary of each face consists of several disjoint edge cycles, one of which is external and may delimit an unbounded face (Figure 2.7b).

Data structures such as DCEL (*doubly-connected edge list*) [79] and *quad-edge* [55] can be used to efficiently represent these entities using linear space, maintain the incidence relations among them, obtain query results (such as point location) and perform operations such as computing the overlay of two arrangements. A DCEL keeps a list of interconnected vertices, faces and half-edges (splitting each edge into two *twin* directed edges with opposite directions) conveniently updated (Figure 2.7). Each half-edge keeps pointers to its target endpoint, the incident face to its left and its twin, next and previous half-edges. A vertex stores its coordinates and one arbitrary half-edge originating from it. Finally, each face in the DCEL stores an arbitrary half-edge enclosing its outer border and a linked-list of half-edges enclosing borders of inner faces that may eventually exist.

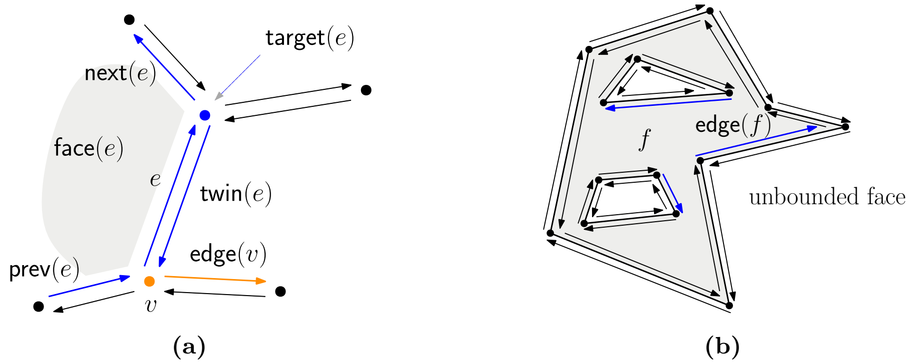


Figure 2.7: **(a)** Information that **(a)** a vertex v, a half-edge e and **(b)** a face f keep in a DCEL structure.

2.1.3.1 Overlays

Arrangement components can also be extended to store additional data, which proves convenient when performing arrangement overlays. Overlaying two arrangements with extra data attached to faces allows performing operations on polygons, such as Boolean set operations (union, intersection, difference, symmetric difference) (Figure 2.8).

Definition 2.9 (Overlay, [100])**.** The *overlay* of two arrangements $\mathcal{A}_1$, $\mathcal{A}_2$ is another arrangement $\mathcal{A}$ such that there are faces f_1 and f_2 in $\mathcal{A}_1$ and $\mathcal{A}_2$, respectively, if and only if there is a face f in $\mathcal{A}$ that is a maximal connected component of $f_1 \cap f_2$.

The overlay of two arrangements $\mathcal{A}_1$, $\mathcal{A}_2$ can be computed in $\mathcal{O}\left((n+k)\log n\right)$ time by Bentley-Ottmann's sweep-line algorithm [12], where n is the total number of edges of $\mathcal{A}_1$ and $\mathcal{A}_2$ and k is the number of edge intersections between $\mathcal{A}_1$ and $\mathcal{A}_2$. Indeed, it can be computed in $\mathcal{O}(n+k)$ time [48] if both $\mathcal{A}_1$ and $\mathcal{A}_2$ form simply connected subdivisions – which is the case for visibility regions.

It is useful in this work to naturally extend the definition to more than two arrangements, in order to enable the notion of *cumulative overlay*. For example, for computing the overlay between three arrangements $\mathcal{A}_1$, $\mathcal{A}_2$, $\mathcal{A}_3$, we first compute the overlay $\mathcal{A}'$ of $\mathcal{A}_1$ and $\mathcal{A}_2$ and then compute the overlay $\mathcal{A}''$ of $\mathcal{A}'$ and $\mathcal{A}_3$.

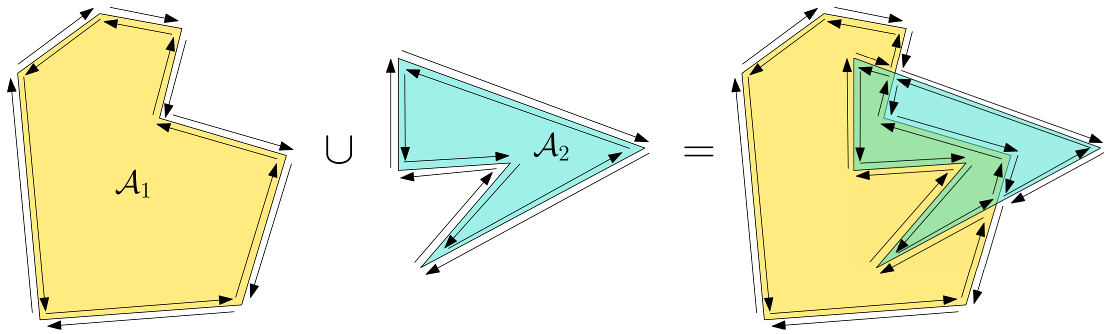


Figure 2.8: Overlay of two arrangements $\mathcal{A}_1$, $\mathcal{A}_2$.

2.1.4 Partitions

Many art gallery algorithms are made efficient – or even possible – by first decomposing input polygons into a set of smaller, disjoint subpolygons according to some well-defined rule.

Definition 2.10 (Triangulation)**.** A triangulation of a polygon P is a subdivision of P into $n-2$ triangles, by adding $n-3$ internal, non-crossing diagonals between vertices of P (Figure 2.9).

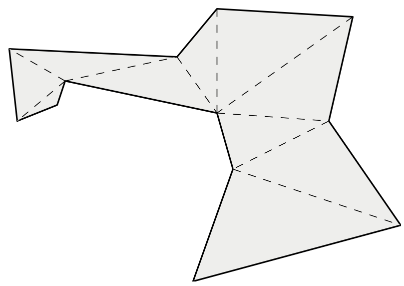


Figure 2.9: A polygon with $n = 12$ vertices, triangulated into $n - 2 = 10$ triangles by adding $n - 3 = 9$ internal diagonals (dashed in the figure).

There are several algorithms for computing polygon triangulations. A noteworthy one is Chazelle's algorithm [27], which allows us to triangulate a simple polygon with n vertices in optimal $\Theta(n)$ time.

Definition 2.11 (Convex partition)**.** The *convex partition* [53] of a polygon P is denoted by $\Pi(P)$ and corresponds to a partition of P induced by vertex visibility regions into convex pieces (also called *convex components*). It can be represented as the overlay of the arrangements defined by the boundaries of all the vertex visibility polygons (Figure 2.10b).

Because of the way a convex partition is produced, every piece in $\Pi(P)$ is bounded by line segments that connect two vertices of the polygon – either polygon edges or constructed edges –, and is included in the visibility region of some vertex. Each vertex of a piece is either a vertex of P or a point called *Steiner point*. For any polygon P with n vertices, $|\Pi(P)| \in \mathcal{O}(n^4)$ because $\mathcal{O}(n^2)$ diagonals are drawn to compute the partition. However, this bound is lowered to $|\Pi(P)| \in \mathcal{O}(n^3)$ if P is guaranteed to be simple [53].

Definition 2.12 (Grid partition)**.** The *grid partition* [91] of an orthogonal polygon P (also called *r-cut* [11] or *pixelation* [14]) is denoted by $\Pi_{HV}(P)$ and corresponds to a partition of P into rectangular pieces called *r-pieces* (also called *pixels* [14] or *basic regions* [102]), obtained by extending every horizontal and vertical edge incident to a reflex vertex towards the interior of P, until it touches another edge (Figure 2.10a).

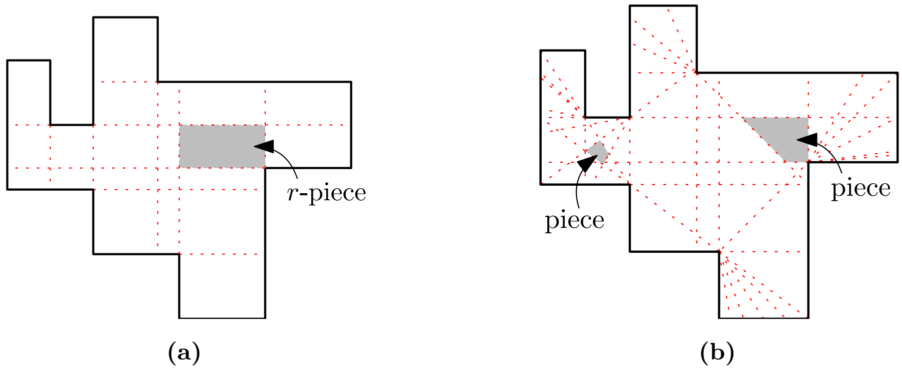


Figure 2.10: **(a)** The grid partition $\Pi_{HV}(P)$ of an orthogonal polygon P. **(b)** The convex partition $\Pi(P)$ induced by vertex visibility regions (under straight-line visibility).

Observe that, for any orthogonal polygon P, the convex partition $\Pi(P)$ is more refined than the grid partition $\Pi_{HV}(P)$. For a fixed number of vertices $n = 2r + 4$, the size of $\Pi_{HV}(P)$ is $\Omega(n)$ and $\mathcal{O}(n^2)$. These bounds are tight for polygons that we call Thin and Fat [11], respectively, and we address these again in Chapter 3. The partition $\Pi_{HV}(P)$ can be computed in $\mathcal{O}(n^2)$ time using a sweep-line algorithm, as shown in [90]: first, a horizontal sweep locates the intersection points of the horizontal edge extensions with ∂P; these points are marked on vertical edges and the polygon is then swept vertically to cut the r-pieces.

2.2 Computational hardness

2.2.1 Approximation algorithms

Approximation algorithms are one of the ways to attack NP-hard problems, in particular when one does not require exact solutions and an estimate with guarantees of being close to optimum is acceptable. We present a few concepts about approximability, partially adapted from [37], that are relevant for later chapters.

Definition 2.13 (NPO problem)**.** An NP-optimization problem (NPO) $\mathcal{P}$ is a quadruple $\langle I, \mathsf{sol}, c, \mathsf{goal}\rangle$ such that:

- I is the set of instances of $\mathcal{P}$ and is recognizable in polynomial time.
- given an instance $x \in I$, $\mathsf{sol}(x)$ is the set of feasible solutions of x and, for any $y \in \mathsf{sol}(x)$, the size of the solution, $|y|$, is polynomial in the size of the instance, $|x|$.

- given an instance x and a feasible solution y of x, $c(x, y) \in \mathbb{R}$ denotes the cost of y; c is often called the *objective function*.
- the goal function refers to the type of the optimization problem: min or max.

Given some instance $x \in I$, the goal is to find an *optimal solution*, that is, a feasible solution y such that

$$c(x, y) = \mathsf{goal}\{c(x, y') \mid y' \in \mathsf{sol}(x)\}$$

The cost of an optimal solution for an instance $x \in I$ is denoted by $OPT(x)$.

The class NPO contains all the NP-optimization problems. Every problem in NPO has a corresponding decision problem in NP that asks, for a given instance x, whether there is a feasible solution y that obeys a certain threshold on the cost $c(x, y)$. For instance, Minimum Vertex Cover is an NP-optimization problem which, given a graph G, asks for the smallest subset of nodes that covers every edge. Its corresponding decision problem in NP is Vertex Cover, which, given a graph G and a positive integer k, asks whether there is a subset of at most k nodes that covers all the edges of G.

Definition 2.14 (α-approximation algorithm)**.** Let $\mathcal{P} = \langle I, \mathsf{sol}, c, \min\rangle$ be an NPO minimization problem, let $\mathcal{A}$ be a polynomial-time algorithm for $\mathcal{P}$ and let $\alpha(\cdot) : \mathbb{Z}^+ \to [1, +\infty]$. Then, $\mathcal{A}$ is an α-approximation algorithm for $\mathcal{P}$ if, for any instance x of $\mathcal{P}$, the solution y computed by $\mathcal{A}$ satisfies

$$c(x, y) \leq \alpha(|x|) \cdot OPT(x)$$

In this case, we refer to the function α as the *approximation factor* or *approximation ratio* of $\mathcal{A}$. If α is a constant function, that is, it does not depend on the size of the input $|x|$, we say that $\mathcal{A}$ is a constant-factor approximation algorithm.

Definition 2.15 (APX class)**.** An NPO problem $\mathcal{P}$ belongs to the class APX if it admits a constant-factor approximation algorithm.

Definition 2.16 (APX-hard)**.** An NPO problem is APX-hard if there is a constant $\epsilon > 0$ such that no polynomial-time algorithm can guarantee an approximation ratio of $1 + \epsilon$ for $\mathcal{P}$, unless P = NP.

Definition 2.17 (PTAS)**.** An NPO problem $\mathcal{P}$ belongs to the class PTAS if an algorithm $\mathcal{A}$ exists such that, for any instance x of $\mathcal{P}$ and any fixed, constant rational $\epsilon \geq 0$ given as input, $\mathcal{A}$ is a $(1 + \epsilon)$-approximation algorithm for $\mathcal{P}$.

By these definitions, it is clear that PTAS $\subseteq$ APX $\subseteq$ NPO, but it is not known whether these inclusions are strict. If there is a PTAS for some APX-hard problem, then P = NP [9].

2.2.2 Parameterized complexity

Another way to cope with the intractability of NP-hard problems while seeking exact solutions is to confine the combinatorial explosion to instance parameters that, in practice, may be much smaller than the instance size. The goal of parameterized complexity is to exploit structural properties of instances that, in realistic scenarios, may lead to very efficient algorithms.

Definition 2.18 (Fixed-parameter tractability)**.** Formally, a *parameterization* of a problem maps each instance x to a pair (x, k), where k is a non-negative integer (the parameter). A parameterized problem is said to be *fixed-parameter tractable (FPT)*, if it is decidable in $f(k)\ n^{\mathcal{O}(1)}$ time, where $n = |(x, k)|$ and f is a computable function that only depends on k (likely running in time exponential in k or even slower).

In this sense, the combinatorial hardness of the problem becomes condensed into a (hopefully small) parameter instead of the whole input size. The complexity class of fixed-parameter tractable problems is called FPT. *Kernelization* techniques are often employed as a preprocessing phase to reduce an instance to an equivalent instance (*kernel*) that has a size bounded by a function of k and is, thus, *easier* to solve.

Chapter 3

Randomized Algorithm for Set Cover

In this chapter, we solve Minimum Vertex Guard (MVG) for orthogonal polygons under straight-line visibility by a reduction to the Set Cover problem. We present a new polynomial-time algorithm we developed for approximating Set Cover and call it Randomized-Set-Cover. It is *randomized* in the sense that some of the decisions it takes are based on a degree of randomness – that is, a stream of uniformly random bits is also provided to the algorithm along with the input. Randomized-Set-Cover is a Monte Carlo algorithm, so its running time can be precisely characterized, but the solution it outputs for a given instance may not always be optimal.

We first present known background on the approximability of Set Cover and discuss techniques for reducing MVG to Set Cover. We describe how a simple pre-processing step (kernelization) can help significantly reduce the size of the instance being solved while still preserving optimality, and how this step can be generalized to run after every iteration. We also show that, for uniformly random inputs of Set Cover, one can deterministically find optimal solutions with very high probability. Finally, we develop time- and memory-efficient data structures for implementing Randomized-Set-Cover.

3.1 Background on Set Cover

Set Cover is one of the most fundamental optimization problems in Computational Complexity Theory and Combinatorics. Its corresponding decision version integrates the "Karp's 21 NP-complete problems" [66].

Definition 3.1 (Set Cover, decision version)**.** Given a pair $(\mathcal{U}, \mathcal{F})$ consisting of a finite set $\mathcal{U}$ (universe) and a family $\mathcal{F} = \{S_1, \ldots, S_{|\mathcal{F}|}\}$ of subsets of $\mathcal{U}$ such that $\bigcup_{S \in \mathcal{F}} S \subseteq \mathcal{U}$, and an integer $k \in \mathbb{Z}^+$, Set Cover asks whether there exists a subset $\mathcal{C} \subseteq \mathcal{F}$ such that $\mathcal{U} = \bigcup_{S \in \mathcal{C}} S$ and $|\mathcal{C}| = k$. Such a set $\mathcal{C}$ is called a *cover* for $\mathcal{U}$.

Definition 3.1 defines Set Cover as a decision problem. However, it is also common to address it as an optimization problem – also called Minimum Set Cover –, in which the goal is to determine a cover $\mathcal{C}^\star \subseteq \mathcal{F}$ for $\mathcal{U}$ of minimum cardinality:

Definition 3.2 (Set Cover, optimization version)**.** Given a pair $(\mathcal{U}, \mathcal{F})$ consisting of a finite set $\mathcal{U}$ (universe) and a family $\mathcal{F} = \{S_1, \ldots, S_{|\mathcal{F}|}\}$ of subsets of $\mathcal{U}$ such that $\bigcup_{S\in\mathcal{F}} S \subseteq \mathcal{U}$, Set Cover asks for a minimum-cardinality subset $\mathcal{C} \subseteq \mathcal{F}$ such that $\mathcal{U} = \bigcup_{S\in\mathcal{C}} S$ or to report that no such $\mathcal{C}$ exists. Such a set $\mathcal{C}$ is called a *cover* for $\mathcal{U}$.

Throughout this thesis, we use Set Cover to denote the optimization problem. Note that, by convenience, we have slightly modified the usual definition of the problem, which often requires that $\bigcup_{S\in\mathcal{F}} S = \mathcal{U}$. Our definition allows $\bigcup_{S\in\mathcal{F}} S \subset \mathcal{U}$, for which case there is no feasible cover. Figure 3.1 shows an example of an instance and a corresponding optimal solution.

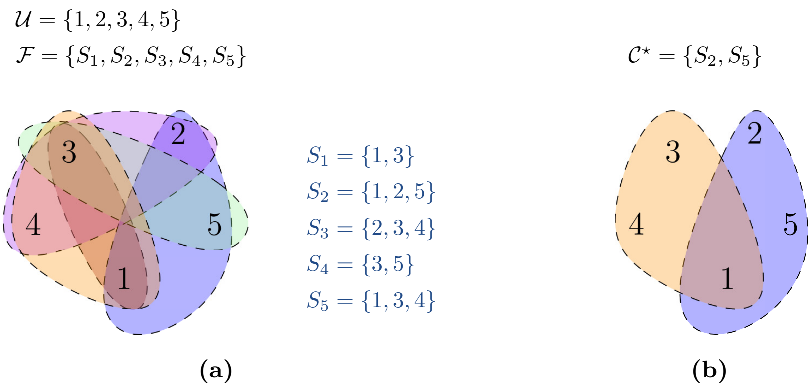


Figure 3.1: **(a)** A Set Cover instance $(\mathcal{U}, \mathcal{F})$ and **(b)** a corresponding minimum cover $\mathcal{C}^\star$. $\{S_2, S_3\}$ is another optimal cover for $\mathcal{U}$.

Its statement is so general that many other frequently arising optimization problems can be trivially encoded as instances of Set Cover, namely Vertex Cover, Dominating Set, Hitting Set, and even the Art Gallery Problem itself [53]. As a simple example, suppose that $\mathcal{U}$ represents a set of rooms in a university department and $\mathcal{F}$ represents a set of locations where a Wi-Fi repeater can be installed, each repeater being able to provide stable Internet access to some subset of rooms. Determining the minimum number of Wi-Fi repeaters that we need to install so that each room is covered by at least one repeater can be modeled as a Set Cover instance.

Its wide applicability makes Set Cover a very interesting object of study, both from theoretical and practical standpoints. However, being NP-hard generally makes the usage of exhaustive search algorithms prohibitive in practice. Unless the family of instances being treated has very specific, and exploitable, properties, the usual approach involves instead looking for approximate solutions that may still be very helpful in practice. We now mention some important, state-of-the-art results regarding the approximability of Set Cover.

3.1.1 H_s-approximation algorithm

Definition 3.3 (Harmonic number)**.** $H_k = \sum_{i=1}^{k} \frac{1}{i}$ denotes the k-th harmonic number, for $k \geq 1$.

Proposition 3.1 ([33], A.14)**.** *For any integer $k \geq 1$, $H_k \leq 1 + \ln k$.*

Theorem 3.2. *[63] Let $(\mathcal{U}, \mathcal{F})$ be a SET COVER instance. An optimal solution for $(\mathcal{U}, \mathcal{F})$ can be approximated with factor H_s, where $s = \max\{|S| : S \in \mathcal{F}\}$, in polynomial time.*

Chvátal [30] showed that the result of Theorem 3.2 extends to WEIGHTED SET COVER, a variation of SET COVER where each set $S \in \mathcal{F}$ is also assigned a cost and the goal is to find a cover for $\mathcal{U}$ of minimum total cost.

An example of an algorithm that achieves such an approximation ratio for SET COVER is GREEDY-SET-COVER. Its greedy idea is very intuitive: at any step, choose the subset $S \in \mathcal{F}$ that covers as many elements as possible among the ones that are still uncovered. Pseudocode for the algorithm, based on the presentation of [33], is given in Algorithm 1.

Algorithm 1 GREEDY-SET-COVER$(\mathcal{U}, \mathcal{F})$

$\mathcal{X} \leftarrow \mathcal{U}$ ▷ uncovered elements
$\mathcal{C} \leftarrow \emptyset$
While $\mathcal{X} \neq \emptyset$:
    $S \leftarrow$ choose set from $\mathcal{F}$ that maximizes $|S \cap \mathcal{X}|$
    $\mathcal{X} \leftarrow \mathcal{X} \setminus S$
    $\mathcal{C} \leftarrow \mathcal{C} \cup \{S\}$
Return $\mathcal{C}$

GREEDY-SET-COVER can be implemented to run in linear time, $\mathcal{O}\left(\sum_{S \in \mathcal{F}} |S|\right)$, using a *bucket queue* data structure [33]. Because of Proposition 3.1, the approximation factor H_s is also often presented in a weaker form as $1 + \ln |\mathcal{U}|$, or even simply as $\mathcal{O}(\log |\mathcal{U}|)$. In fact, a tighter analysis [87] proves that GREEDY-SET-COVER achieves an approximation ratio of $\ln |\mathcal{U}| - \ln \ln |\mathcal{U}| + \Theta(1)$. A series of works [39, 47, 76] gives successively tighter inapproximability results and culminates in showing that SET COVER cannot be approximated to within $(1 - o(1)) \cdot \ln |\mathcal{U}|$, unless $\mathsf{P} = \mathsf{NP}$ (Theorem 3.3). This implies that GREEDY-SET-COVER essentially performs almost as good as we can possibly achieve, under plausible complexity-theoretic assumptions.

Theorem 3.3 ([39], Corollary 1.5)**.** *For any constant $\epsilon > 0$, it is NP-hard to approximate SET COVER with factor $(1 - \epsilon) \cdot \ln |\mathcal{U}|$.*

For instances whose duals have bounded VC-dimension, specialized algorithms have been proposed with significantly better approximation ratios [24]. For more about the VC theory, refer to the original papers by Vapnik and Chervonenkis [97].

3.1.2 f-approximation algorithm

One may be able to improve the H_s approximation factor provided further constraints on the SET COVER instance are ensured. Namely, if every element $x \in \mathcal{U}$ is *infrequent* in $\mathcal{F}$, we can achieve a better ratio, as Theorem 3.4 formalizes.

Theorem 3.4 ([57]). *Let $(\mathcal{U}, \mathcal{F})$ be a Set Cover instance such that any element $x \in \mathcal{U}$ occurs in at most f sets of $\mathcal{F}$, for some $f \geq 1$. An optimal solution for $(\mathcal{U}, \mathcal{F})$ can be approximated with factor f in polynomial time.*

Notice that we can view Vertex Cover as a special case of Set Cover with $f = 2$ (every edge is incident to two vertices). A 2-approximation algorithm, attributed to Gavril and Yannakakis in [82], is also known for Vertex Cover, based on greedily finding a *maximal* matching M and adding all the nodes covered by M to the vertex cover. This result, in fact, generalizes for hypergraphs whose hyperedges connect up to f nodes, from which an f-approximation algorithm for Set Cover follows [52].

There are alternative methods to obtain an f-approximation in polynomial time, for some constant f. For example, *LP-rounding* techniques formulate the problem as an integer linear program (ILP) and solve the instance where integrality constraints are relaxed. Continuous variables in the solution are then rounded back to feasible, integral values, in a way that guarantees the approximation factor – either according to an established threshold criterion (*deterministic rounding*) or by being interpreted as probabilities (*randomized rounding*) [98, 101].

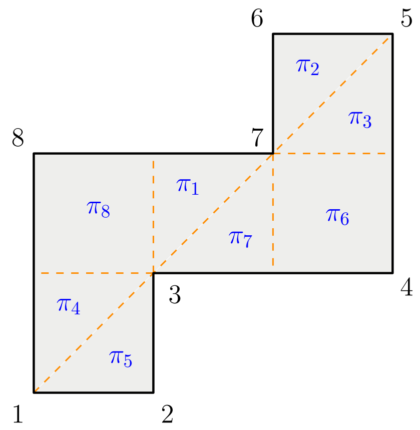
We note that H_s and f are only loose upper bounds for the approximation factors provided by Greedy-Set-Cover and LP-rounding methods; it could be that the ratio that they obtain for a concrete instance is smaller than those.

To conclude our review on Set Cover, there exists a pair of *L-reductions* between Set Cover and Dominating Set [65], implying that, if a polynomial-time α-approximation algorithm exists for one of them, for some α, then a polynomial-time α-approximation algorithm also exists for the other. This establishes a deep connection between Set Cover and Dominating Set in terms of (in)approximability.

3.2 Polygon discretization and instance generation

In this chapter, we are interested in finding minimum-cardinality vertex-guard sets for orthogonal polygons under straight-line visibility (MVG). For that we reduce the problem to Set Cover. But a polygon is a continuous set of points, while Set Cover deals with finite, discrete sets, so we cannot hope for such a reduction without first discretizing the instance in some way. In this section we detail the steps taken in that direction.

A general strategy that works for any simple polygon P involves a partition of P into convex pieces induced by vertex visibility regions (Definition 2.11). This induces an arrangement defined by the boundaries of all the vertex visibility polygons, and provides a simple way to make up a Set Cover instance $(\mathcal{U}, \mathcal{F})$: convex pieces in $\Pi(P)$ correspond to elements in the universe $\mathcal{U}$ and all the pieces that some vertex of P sees are turned into a set in $\mathcal{F}$ (Figure 3.2).



$$
\begin{aligned}
S_1 &= \{x_1, x_4, x_5, x_8\}\\
S_2 &= \{x_4, x_5, x_8\}\\
S_3 &= \{x_1, x_3, x_4, x_5, x_6, x_7, x_8\}\\
S_4 &= \{x_1, x_2, x_3, x_6, x_7, x_8\}\\
S_5 &= \{x_2, x_3, x_6, x_7\}\\
S_6 &= \{x_2, x_3, x_6\}\\
S_7 &= \{x_1, x_2, x_3, x_4, x_6, x_7, x_8\}\\
S_8 &= \{x_1, x_4, x_5, x_6, x_7, x_8\}
\end{aligned}
$$

Figure 3.2: Reduction from MVG to SET COVER. The convex partition of an orthogonal polygon with 8 vertices and 8 pieces is mapped to a SET COVER instance.

A useful property of convex partitions is that no piece in $\mathit{\Pi}(P)$ can be partially seen by a vertex. That is, for any piece $\pi \in \mathit{\Pi}(P)$ and any vertex v, either v fully sees $\mathsf{int}\,\pi$ or v sees nothing from $\mathsf{int}\,\pi$.

Theorem 3.5. *A vertex v of P completely sees the interior of a piece $\pi \in \mathit{\Pi}(P)$ if and only if v sees some point in the interior of π.*

Proof. Of course, if v completely sees $\mathsf{int}\,\pi$, then v sees some point in $\mathsf{int}\,\pi$. For the other direction, suppose, by contradiction, that v sees a point $q \in \mathsf{int}\,\pi$ but does not see another point $q' \in \mathsf{int}\,\pi$ (that is, v sees π partially). Then, π can be split into two sub-pieces, one containing q and the other containing q', according to $\mathcal{V}(v)$. This contradicts the fact that the partition π belongs to is induced by vertex visibility regions. Hence, either both q and q' are seen by v or none of them are, which means that, if v sees some point in $\mathsf{int}\,\pi$, then v also sees every other point in $\mathsf{int}\,\pi$. □

For computing regularized vertex visibility polygons we employed the triangular expansion algorithm (Figure 2.5b). The reason why we adopted regularized visibility is that needles have an empty interior and their boundary will be naturally guarded anyway by a subset of guards that cover their immediate neighbourhood.

3.2.1 First approach

One way of generating the SET COVER instance after partitioning the polygon could be to elect a representative point π_p for each piece π, calculate its visibility polygon $\mathcal{V}(\pi_p)$, and determine which vertices of P are contained in $\mathcal{V}(\pi_p)$. By inverse construction this tells us which arrangement faces are seen by each vertex $v \in P$. Point-in-polygon tests of the form $v \in \mathcal{V}(\pi_p)$ can be performed in $\mathcal{O}(n)$ time per query using the *even-odd* ray-casting algorithm [86] or the *winding* *number* algorithm [6]. Queries can also be answered offline in $\mathcal{O}((|\mathcal{V}(\pi_p)| + n) \log(|\mathcal{V}(\pi_p)| + n))$ time overall, using a sweep-line approach that processes polygon edges and query points at the same time [95]. Theorem 3.5 enables us to select any point from $\operatorname{int} \pi$ as the representative of piece π. The centroid $c_\pi = (c_{\pi_x}, c_{\pi_y})$ of π is often a useful choice, because it assuredly belongs to the interior of π (by convexity of the piece) and can be computed in time linear in the number of vertices of the piece. Assuming that the vertices of π are $(x_0, y_0), (x_1, y_1), \ldots, (x_{|\pi|-1}, y_{|\pi|-1})$ in counter-clockwise order and that $(x_{|\pi|}, y_{|\pi|}) = (x_0, y_0)$, the coordinates of the centroid are given [80] by

$$c_{\pi_x} = \frac{1}{6A} \sum_{i=0}^{|\pi|-1} (x_i + x_{i+1})(x_i y_{i+1} - x_{i+1} y_i) \tag{3.1}$$

$$c_{\pi_y} = \frac{1}{6A} \sum_{i=0}^{|\pi|-1} (y_i + y_{i+1})(x_i y_{i+1} - x_{i+1} y_i)\,, \tag{3.2}$$

where $|\pi|$ is number of vertices of π and A is π's signed area:

$$A = \frac{1}{2} \sum_{i=0}^{|\pi|-1} (x_i y_{i+1} - x_{i+1} y_i) \tag{3.3}$$

Note that, in the place of explicitly computing a centroid visibility region $\mathcal{V}(c_\pi)$ just to check if a vertex $v \in \mathcal{V}(c_\pi)$, we could instead check whether the line segment vc_π is fully contained in P. Such a segment-in-polygon test $vc_\pi \subset P$ could be performed in $\mathcal{O}(n)$ time by testing whether vc_π intersects any edge of P and, if so, whether it passes through some vertex of P.

A classic result known as Carathéodory's theorem [26] implies that any point in a convex polygon Q can be expressed as the convex combination of any three vertices of Q. From this it follows that, as an alternative to the centroid, we can instead take as the representative of piece π the point

$$\left(\frac{x_0 + x_1 + x_2}{3}, \frac{y_0 + y_1 + y_2}{3} \right)\,, \tag{3.4}$$

which has the advantage of being computable in constant time.

3.2.2 Second approach

Although the previous method is concise, it is not clever in the sense that it fails to take advantage of having already computed vertex visibility regions initially. It ends up performing inclusion tests just to recover which vertices had contributed to producing each arrangement face in the first place. Better would have been if we had kept this information about vertices all along from the start.

In this regard, the optimization we actually perform in this work to speed up the instance generation phase is done by bookkeeping information at the arrangement level. As we are incrementally building the overlay of arrangements, we augment each face to additionally store the set of vertices that see it. Every time two faces overlap in a given area after we extend the current overlay with a new vertex visibility region $\mathcal{V}(v)$, we merge the sets kept by both faces. This means that, after considering every vertex in succession, the final overlay will represent the convex partition $\Pi(P)$ and each face (other than the unlimited, outer face) will be associated with the set of vertices that (fully) see the corresponding piece $\pi \in \Pi(P)$. At the expense of a slightly higher space consumption, we are now able to directly extract the SET COVER instance from the overlay, without need for extra computations.

3.3 Definition of Randomized-Set-Cover

We now describe our RANDOMIZED-SET-COVER algorithm. It is a Monte Carlo algorithm, so it always terminates within a given polynomial time bound, but there is a non-zero probability that the cover it outputs for a given instance is not optimal.

For any finite set S, assume that S.CHOOSE-RANDOM() returns an element $x \in S$ uniformly at random in constant time. Check out Section 3.4.1 for the way a set supporting this method can be efficiently implemented in practice. If no cover exists, the RANDOMIZED-SET-COVER algorithm reports that the instance has no solution by returning $\perp$ ("impossible"). Otherwise, the algorithm starts with a cover $\mathcal{C} = \mathcal{F}$ and works by removing, at each step, a random set $S \in \mathcal{C}$ such that every one of its elements is also in at least one other set $S' \in \mathcal{C}$.

Algorithm 2 RANDOMIZED-SET-COVER($\mathcal{U}, \mathcal{F}$)

If $\bigcup_{S \in \mathcal{F}} S \neq \mathcal{U}$: **Return** $\perp$
$\mathcal{C} \leftarrow \mathcal{F}$
While True:
    CANDIDATES $\leftarrow \{S \in \mathcal{C} : \forall x \in S, \exists S' \in \mathcal{C}, S' \neq S, x \in S'\}$
    If CANDIDATES $= \emptyset$: **Break**
    $S \leftarrow$ CANDIDATES.CHOOSE-RANDOM()
    $\mathcal{C} \leftarrow \mathcal{C} \setminus \{S\}$
Return $\mathcal{C}$

Notice that CANDIDATES contains the sets whose elements all have frequency at least 2 in $\mathcal{C}$. It is easy to see that the algorithm always finds a valid, minimal cover for $\mathcal{U}$, although its cardinality may not necessarily be minimum. $\mathcal{C}$ is minimal because, when the loop ends, every set $S \in \mathcal{C}$ has at least one element which only S contains, so no set in $\mathcal{C}$ can be further removed without breaking feasibility. The following invariant remains true when the termination condition is tested at the beginning of every iteration of the **While** loop:

Invariant. *Every element $x \in \mathcal{U}$ is covered by at least one set $S \in \mathcal{C}$.*

Proof. This invariant is true when the termination condition is tested for the first time, as $\mathcal{C} = \mathcal{F}$ covers $\mathcal{U}$ (otherwise, the algorithm would have returned $\perp$ at line 1). At lines 4–7, a random set $S \in \mathcal{C}$ is removed such that every single element $x \in S$ remains covered by at least one other set $S' \in \mathcal{C}$ – if such a set S exists – and so $\mathcal{U}$ remains covered after this iteration and the termination condition is tested again. If S does not exist, the loop breaks and the set $\mathcal{C}$ at that point, which is a cover for $\mathcal{U}$, is returned. □

3.3.1 Implementation

A possible implementation of Randomized-Set-Cover follows:

1: **If** $\bigcup_{S \in \mathcal{F}} S \neq \mathcal{U}$: **Return** $\perp$
2: $\mathcal{C} \leftarrow \mathcal{F}$ ▷ cover to be produced
3: Candidates $\leftarrow \emptyset$ ▷ indices i of sets $S_i \in \mathcal{F}$ that may be removed in the next iteration
4: **For each** $x \in \mathcal{U}$:
5: $\quad S^{-1}[x] \leftarrow \emptyset$
6: **For each** $S_i \in \mathcal{F}$:
7: $\quad$ **For each** $x \in S_i$:
8: $\quad\quad S^{-1}[x] \leftarrow S^{-1}[x] \cup \{i\}$ ▷ $S^{-1}[x] = \{i \in \{1, \ldots, |\mathcal{F}|\} : x \in S_i \in \mathcal{F}\}$
9: **For each** $S_i \in \mathcal{F}$:
10: $\quad$ **If** $\forall x \in S_i, |S^{-1}[x]| > 1$:
11: $\quad\quad$ Candidates $\leftarrow$ Candidates $\cup \{i\}$
12: **While** Candidates $\neq \emptyset$:
13: $\quad i \leftarrow$ Candidates.Choose-Random()
14: $\quad$ Candidates $\leftarrow$ Candidates $\setminus \{i\}$
15: $\quad \mathcal{C} \leftarrow \mathcal{C} \setminus \{S_i\}$
16: $\quad$ **For each** $x \in S_i$:
17: $\quad\quad S^{-1}[x] \leftarrow S^{-1}[x] \setminus \{i\}$
18: $\quad\quad$ **If** $|S^{-1}[x]| = 1$:
19: $\quad\quad\quad$ Candidates $\leftarrow$ Candidates $\setminus \{j\}$, where $S^{-1}[x] = \{j\}$
20: **Return** $\mathcal{C}$

We now analyze the performance of the algorithm when given an instance $(\mathcal{U}, \mathcal{F})$. Line 1 performs $\Theta(\sum_{S \in \mathcal{F}} |S|)$ operations. The number of operations performed in lines 4–5 is $\Theta(|\mathcal{U}|)$. Also, in lines 6–11 we process $\Theta(\sum_{S \in \mathcal{F}} |S|)$ elements. The **While** loop on lines 12–19 runs at most $|\mathcal{F}| - |\mathcal{C}^\star|$ times ($|\mathcal{C}^\star| \leq |\mathcal{F}|$) and in each time $\mathcal{O}(\max_{S \in \mathcal{F}} |S|)$ elements are processed. However, notice that each set S can be removed from Candidates at most once, and only then we do consider the elements in S. Therefore, a tighter analysis shows that the number of operations performed inside the **While** loop is in fact $\mathcal{O}(|\mathcal{F}| - |\mathcal{C}^\star| + \sum_{S \in \mathcal{F}} |S|) \subseteq \mathcal{O}(|\mathcal{F}| + \sum_{S \in \mathcal{F}} |S|)$.

As such, the algorithm performs $\mathcal{O}\left(|\mathcal{U}| + |\mathcal{F}| + \sum_{S\in\mathcal{F}} |S|\right)$ operations overall, which is linear in the size of the instance. Observe that the term $|\mathcal{C}^\star|$ does not appear in the expression of the complexity class because $|\mathcal{F}|$ (with hidden constants) dominates $|\mathcal{C}^\star|$.

As usually happens with any Monte Carlo algorithm, in order to improve the probability of success in practice, we shall perform several runs over the same instance and output the best cover we find among those trials. If we let $R(\mathcal{U}, \mathcal{F})$ denote the number of repetitions over the instance $(\mathcal{U}, \mathcal{F})$ – which can be regarded as a parameter of the algorithm –, a full characterization of the number of operations becomes:

$$\mathcal{O}\left(R(\mathcal{U},\mathcal{F}) \cdot \left(|\mathcal{U}| + |\mathcal{F}| + \sum_{S\in\mathcal{F}} |S|\right)\right)$$

3.3.2 Probability of success with uniformly random instances

First, it is helpful to understand why the probability of the algorithm finding a minimum cover $\mathcal{C}^\star$ for $\mathcal{U}$ is non-zero, that is, why any minimum cover can be obtained by successively removing from $\mathcal{F}$ one of the sets S for which each element is also covered by some other set, $S' \in \mathcal{F}$. This is because such a set S is trivially redundant and does not belong to any minimum cover: if some cover $\mathcal{C}$ had such a set S, we could make $\mathcal{C}$ smaller by removing it, and thus the cover would not be optimal in the first place. Since the algorithm assigns each of these redundant sets a uniform (non-zero) probability of being chosen at each step, there is at least one way of having it make the overall right sequence of choices.

For any given instance $(\mathcal{U}, \mathcal{F})$, let $n = R(\mathcal{U}, \mathcal{F})$ be the number of trials and let p be the probability that Randomized-Set-Cover determines an optimal cover for it in an individual trial. The number of times the algorithm determines a minimum cover among those trials is a random variable $X \sim \mathrm{B}(n, p)$, that is, X follows a binomial distribution with parameters n and p. The expected number of times the algorithm actually determines a minimum cover for that instance is $\mathbb{E}[X] = np$ and its variance is $np(1-p)$.

Now, the optimality of Randomized-Set-Cover is clearly conditioned by the number of times it makes a random guess in a single trial: the *more deterministic* it is, the better the guarantees we have that it is moving in the right direction. In this section, we study how a simple sanity test performed before running Randomized-Set-Cover can dramatically improve the probability that the algorithm finds an optimal cover when applied to uniformly random Set Cover instances.

A uniformly random instance $(\mathcal{U}, \mathcal{F})$ of Set Cover is one where each element of $2^{\mathcal{U}}$ is in $\mathcal{F}$ with probability $\frac{1}{2}$. Recall that, according to our definition of Set Cover, an instance where $\bigcup_{S\in\mathcal{F}} S \subset \mathcal{U}$ is also valid. Given that each of the $2^{|\mathcal{U}|}$ subsets of $\mathcal{U}$ may appear in $\mathcal{F}$ or not, there is a total of $2^{2^{|\mathcal{U}|}}$ different instances of Set Cover for a fixed cardinality $|\mathcal{U}|$. Therefore, another way of defining a uniformly random instance is that it is an instance which has probability $1/2^{2^{|\mathcal{U}|}}$ of being drawn from a discrete uniform distribution.

As discussed below, we observe that uniformly random instances are interesting in the sense that, for growing $|\mathcal{U}|$, they tend to admit optimal covers of size 1 or 2 with very high probability. A straightforward algorithm with an $\mathcal{O}(|\mathcal{F}|^2)$ number of operations preceding RANDOMIZED-SET-COVER would optimally solve the vast majority of these instances by simply testing, in turn:

1. whether $\bigcup_{S\in\mathcal{F}} S \subset \mathcal{U}$: if so, there is no solution.
2. if not, whether $\mathcal{U} \in \mathcal{F}$: if so, there is an optimal cover of size 1 (which is $\{\mathcal{U}\}$).
3. if not, whether any pair of sets $(S, S') \in \mathcal{F}^2$ satisfies $S \cup S' = \mathcal{U}$; if so, there is an optimal cover of size 2 (which is $\{S, S'\}$).
4. if not, return an arbitrary cover, such as one that is computed by RANDOMIZED-SET-COVER (which, in this case, may not be optimal).

Algorithm 3 formalizes this reasoning. Note that this method can be made deterministic by having it return any valid cover in step 4 – say, $\mathcal{F}$ itself.

Algorithm 3 UNIFORM-RANDOMIZED-SET-COVER($\mathcal{U}, \mathcal{F}$)

```
1: If ⋃_{S∈F} S ⊂ U: Return ⊥
2: If U ∈ F: Return {U}
3: For each pair (S, S') ∈ F²:
4:     If S ∪ S' = U: Return {S, S'}
5: Return Randomized-Set-Cover(U, F)
```

We now argue that uniformly random SET COVER instances admit small covers with high probability.

Optimal cover of size 1 Any given uniformly random instance admits a cover of size 1 if and only if $\mathcal{U} \in \mathcal{F}$ and this happens with probability $\frac{1}{2}$.

Optimal cover of size 2 Let's now focus on the case where $\mathcal{U} \notin \mathcal{F}$, which also happens with probability $\frac{1}{2}$. For every $S \in \mathcal{F}$, we define an indicator random variable χ_S as:

$$\chi_S = \begin{cases} 1 & \text{if there is } \widehat{S} \in \mathcal{F} \text{ such that } S \cup \widehat{S} = \mathcal{U} \\ 0 & \text{otherwise} \end{cases}$$

Notice that such an $\widehat{S}$ cannot be S, otherwise $S = \mathcal{U} \in \mathcal{F}$, which we assumed to be false. For such an $\widehat{S}$ to exist, it has to contain every element of $\mathcal{U}$ that S is missing (i.e., $\widehat{S} \supseteq \mathcal{U} \setminus S$). This means that there are $|\mathcal{U}| - |S|$ fixed elements that must necessarily appear in $\widehat{S}$ and $|S|$ elements that may or may not appear, so there are $2^{|S|}$ possible candidates for $\widehat{S}$ in $2^{\mathcal{U}}$. As each of these candidates occurs in $\mathcal{F}$ with probability $\frac{1}{2}$, the probability that at least one of them does indeed is $\mathbb{P}[\chi_S = 1] = 1 - \left(\frac{1}{2}\right)^{2^{|S|}}$.

As such, the probability that at least one $S \in \mathcal{F}$ admits a corresponding $\widehat{S} \in \mathcal{F}$ such that $S \cup \widehat{S} = \mathcal{U}$ is $1 - \mathbb{P}\left[\bigwedge_{S\in\mathcal{F}} (\chi_S = 0)\right] \geq 1 - \min_{S\in\mathcal{F}} \mathbb{P}[\chi_S = 0]$. Finally, the probability that the instance admits an optimal cover of size 2 is

$$\frac{1 - \mathbb{P}\left[\bigwedge_{S\in\mathcal{F}} (\chi_S = 0)\right]}{2} \tag{3.5}$$

because, in order for that to happen, it must as well not admit an optimal cover of size 1, which happens with probability $\frac{1}{2}$. Either way, since $\mathbb{E}[|\mathcal{F}|] = 2^{|\mathcal{U}|-1}$, it is intuitive that this probability approaches $\frac{1}{2}$ exponentially fast with the growth of $|\mathcal{U}|$; Table 3.1 and Figure 3.3 confirm this fact empirically.

In essence, as $|\mathcal{U}|$ grows, the probability that either (a) there is no cover or (b) there is an optimal cover of size 3 or more becomes extremely low: *almost all* uniformly random instances have an optimal cover of size 1 or 2. Hence, UNIFORM-RANDOMIZED-SET-COVER would very seldom reach the last **Return** instruction in practice, especially for large universe sizes of $|\mathcal{U}|$, and so it would likely find an optimal cover $\mathcal{C}^\star$ for $\mathcal{U}$. Table 3.1 shows how the total number of instances of each type evolves with the size of the universe for $|\mathcal{U}| \in \{1, 2, 3, 4\}$.

Since the total number of instances grows extremely fast with the universe size, exact results for $|\mathcal{U}| \geq 5$ were not computed. For instance, there exist $2^{2^5} \approx 4.29 \times 10^9$ instances with $|\mathcal{U}| = 5$ and $2^{2^6} \approx 1.84 \times 10^{19}$ instances with $|\mathcal{U}| = 6$, which would take an unfeasible amount of time to generate and solve optimally. Instead, for each universe size $1 \leq |\mathcal{U}| \leq 14$, 10^5 instances were sampled uniformly at random without replacement from the set of all instances and solved using Algorithm 3 to provide an estimate of the fraction of instances that admit an optimal cover of size 2. The plot in Figure 3.3 summarizes these results and confirms our theoretical analysis.

Remark. *For simplifying the previous analysis and making the space of valid instances more symmetric, our definition of* SET COVER *allows the situation where $\emptyset \in \mathcal{F}$ or $\mathcal{F} = \emptyset$, which is not very usual as $\emptyset$ could be trivially removed from the instance. If we had decided to ignore $\emptyset$ from the count, the fraction $\dfrac{2^{(2^{|U|}-2)}}{2^{(2^{|U|}-1)} - 1}$ of instances which admit an optimal cover of size 1 would not be exactly $\frac{1}{2}$ anymore, although this ratio would still very quickly converge to $\frac{1}{2}$. Therefore, this simplification does not compromise our conclusions and even makes them more intuitive.*

3.3.3 Kernelization

We recall that a binary relation R over a set S is a *partial order* if it is *reflexive* ($\forall x \in S, xRx$), *antisymmetric* ($\forall x, y \in S, xRy \wedge yRx \Rightarrow x = y$) and *transitive* ($\forall x, y, z \in S, xRy \wedge yRz \Rightarrow xRz$). The set S, along with R, is called a *partially ordered set* (or *poset*) and is denoted by (S, R).

$\lvert\mathcal{U}\rvert$	$OPT = 1$	$OPT = 2$	$OPT \geq 3$	**Unfeasible instances**	**Total**
1	**2** (50%)	**0** (0%)	**0** (0%)	**2** (50%)	**4** $= 2^{2^1}$
2	**8** (50%)	**2** (12.5%)	**0** (0%)	**6** (37.5%)	**16** $= 2^{2^2}$
3	**128** (50%)	**88** (34.375%)	**2** (0.78125%)	**38** (14.8438%)	**256** $= 2^{2^3}$
4	**32768** (50%)	**31392** (47.9004%)	**434** (0.662231%)	**942** (1.43738%)	**65536** $= 2^{2^4}$

Table 3.1: Total number and percentage of Set Cover instances that admit an optimal cover of size 1, 2, 3 or more, or no feasible cover at all, for a fixed size of the universe $|\mathcal{U}| \in \{1, 2, 3, 4\}$. Notice that exactly half of the instances admit a cover of size 1, for any universe size, and the percentage of instances that admit an optimal cover of size 2 approaches 50%.

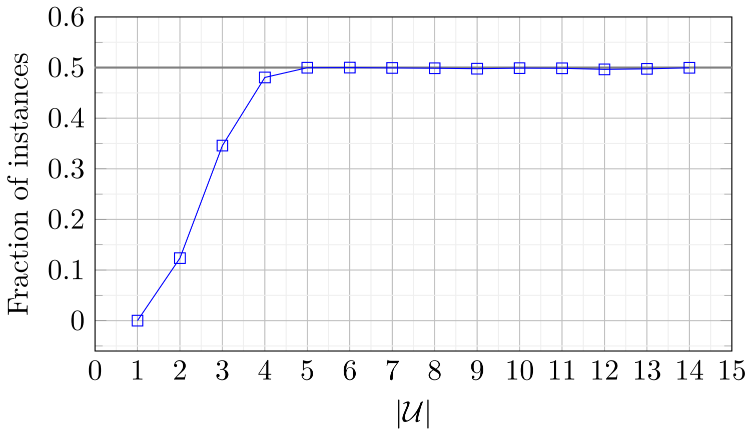


Figure 3.3: Estimated fraction of Set Cover instances admitting an optimal cover of cardinality 2 for each universe size $1 \leq |\mathcal{U}| \leq 14$, tending to 0.5. For each universe size, 10^5 instances were sampled uniformly at random and solved using Uniform-Randomized-Set-Cover. This trend suggests that the error probability of Uniform-Randomized-Set-Cover hastily approaches 0 as uniform random instances grow in size.

Definition 3.4 (Antichain)**.** Let (S, R) be a poset. An antichain in S is a subset of S in which no two distinct elements are comparable by R.

For instance, set inclusion $\subseteq$ is a partial order and, given a set S, an antichain in the power set 2^S is a subset $A \subseteq 2^S$ such that, for all $X, Y \in A$, neither $X \subseteq Y$ nor $Y \subseteq X$.

Definition 3.5 (Partial order width)**.** Let (S, R) be a poset. The width of R is the cardinality of the largest antichain in S.

Definition 3.6 (Sperner family)**.** A Sperner family is a family of sets $\mathcal{F} \subseteq 2^S$, for some set S, in which none of the sets in $\mathcal{F}$ is a subset of another. That is, a Sperner family is an antichain in the inclusion partial order over 2^S.

Theorem 3.6 (Sperner's theorem, [88]). *The width of the partial order* $\subseteq$ *on* 2^S*, for a given set* S*, is* $\binom{|S|}{\lfloor |S|/2 \rfloor}$ *and the only antichains in* S *that have this cardinality are* $\{X \subseteq S : |X| = \lfloor |S|/2 \rfloor\}$ *and* $\{X \subseteq S : |X| = \lceil |S|/2 \rceil\}$.

These results suggest a simple pre-processing step (*kernelization step*) which allows any instance of Set Cover to potentially be reduced in size, in such a way that it is made *faster* to solve while still preserving the optimum. Concretely, while there are any $S, S' \in \mathcal{F}$ such that $S \subset S'$, remove S from $\mathcal{F}$ – because S can always be replaced by S' in any optimal solution. Note that, after this transformation, only sets from an antichain in $(\mathcal{F}, \subseteq)$ remain and, hence, the transformed $\mathcal{F}'$ becomes a Sperner family. By Theorem 3.6, this transformation guarantees that the resulting instance will have $|\mathcal{F}'| \leq \binom{|\mathcal{U}|}{\lfloor |\mathcal{U}|/2 \rfloor}$.

Theorem 3.7 (Stirling's approximation, [89]). $n! \sim \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$*, where* e *is the Euler's number.*

The notation $f(n) \sim g(n)$ stands for $\lim\limits_{n \to +\infty} \dfrac{f(n)}{g(n)} = 1$. Theorem 3.7 gives a useful asymptote for the central binomial coefficient:

$$\binom{2n}{n} = \frac{(2n)!}{(n!)^2} \sim \frac{2\sqrt{\pi n}\left(\frac{2n}{e}\right)^{2n}}{2\pi n\left(\frac{n}{e}\right)^{2n}} = \frac{4^n}{\sqrt{\pi n}} \tag{3.6}$$

This corresponds to the following ratio between the worst-case cardinalities of $\mathcal{F}$ and $\mathcal{F}'$, respectively before and after the kernelization:

$$\frac{|\mathcal{F}|}{|\mathcal{F}'|} = \frac{2^{|\mathcal{U}|}}{\binom{|\mathcal{U}|}{\lfloor |\mathcal{U}|/2 \rfloor}} \sim \sqrt{\frac{\pi |\mathcal{U}|}{2}} \tag{3.7}$$

Also see Figure 3.4. Note that this ratio does not give a lower nor an upper bound on the performance of the transformation. Indeed, the kernelization step may not always have an effect (such as in instances which are already given pre-kernelized in the input). However, in some cases the reduction can be massive (such as in the case where $\mathcal{F} = 2^{\mathcal{U}}$, which leads to $\mathcal{F}' = \{\mathcal{U}\}$ and therefore to a reduction factor of $\frac{2^{|\mathcal{U}|}}{1} = 2^{|\mathcal{U}|}$).

In terms of implementation, we can, for each pair of sets $S, S' \in \mathcal{F}$, test inclusions $S \subseteq S'$ and $S' \subseteq S$ using the two-pointers technique presented in Algorithm 4, which is linear in $|S| + |S'|$. We assume that the elements in $\mathcal{U}$ have an order and that each set $S \in \mathcal{F}$ is given sorted in the input – so the i-th element in S can be indexed by S_i.

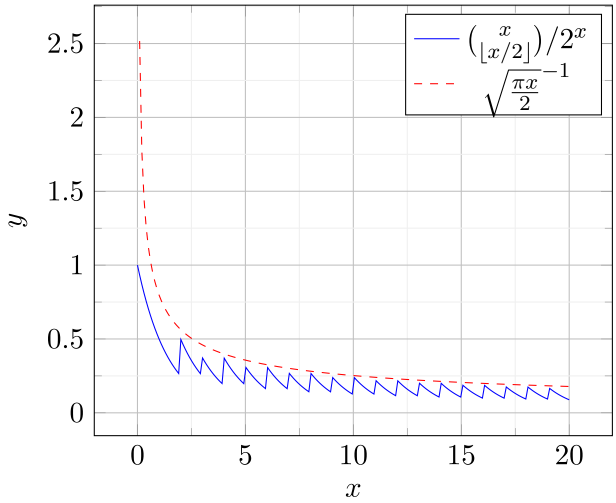


Figure 3.4: The function $2^x/\binom{x}{\lfloor x/2 \rfloor}$ asymptotically approaches $\sqrt{\frac{\pi x}{2}}$.

Algorithm 4 DECIDE-$\subseteq(A, B)$

```
i ← 1
j ← 1
While i ≤ |A|:
    If j > |B| ∨ A_i < B_j:
        Return False
    If A_i = B_j:
        i ← i + 1
    j ← j + 1
Return True
```

This pre-processing step can be performed in time $\mathcal{O}\left(|\mathcal{F}|^2|\mathcal{U}|\right)$ – in fact $\mathcal{O}\left(|\mathcal{F}| \cdot \sum_{S\in\mathcal{F}} |S|\right)$, if we conduct a careful analysis:

$$\sum_{S,S'\in\mathcal{F}} (|S| + |S'|) = \sum_{S\in\mathcal{F}} \sum_{S'\in\mathcal{F}} (|S| + |S'|) \tag{3.8}$$

$$= \sum_{S\in\mathcal{F}} \left(|\mathcal{F}| \cdot |S| + \sum_{S'\in\mathcal{F}} |S'| \right) \tag{3.9}$$

$$= \sum_{S\in\mathcal{F}} (|\mathcal{F}| \cdot |S| + \Sigma) \tag{3.10}$$

$$= |\mathcal{F}| \cdot \Sigma + \sum_{S\in\mathcal{F}} (|\mathcal{F}| \cdot |S|) \tag{3.11}$$

$$= |\mathcal{F}| \cdot \Sigma + |\mathcal{F}| \cdot \Sigma \tag{3.12}$$

$$= 2 \cdot |\mathcal{F}| \cdot \Sigma \tag{3.13}$$

where $\Sigma = \sum_{S\in\mathcal{F}} |S|$. Note that the expression $2|\mathcal{F}|\Sigma$ coincides with the bound $|\mathcal{F}|^2|\mathcal{U}|$ when

$\mathcal{F} = 2^{\mathcal{U}}$, since in that case $2|\mathcal{F}|\Sigma = 2 \cdot 2^{|\mathcal{U}|} \cdot |\mathcal{U}| \cdot 2^{|\mathcal{U}|-1} = |\mathcal{F}|^2|\mathcal{U}|$. For this, we used the fact that

$$\sum_{X \subseteq S} |X| = \sum_{k=1}^{|S|} k \binom{|S|}{k} = |S| \cdot 2^{|S|-1}. \tag{3.14}$$

Identity (3.14) holds because each element of S appears in exactly half of its $2^{|S|}$ subsets, so it contributes with $2^{|S|-1}$ to the sum.

Hence, we have:

Theorem 3.8. *Any instance $(\mathcal{U}, \mathcal{F})$ of* SET COVER *can be reduced to an instance $(\mathcal{U}, \mathcal{F}')$ having the same set of optimal solutions and $|\mathcal{F}'| \leq \binom{|\mathcal{U}|}{\lfloor |\mathcal{U}|/2 \rfloor}$, in time $\mathcal{O}\left(|\mathcal{F}| \cdot \sum_{S \in \mathcal{F}} |S|\right)$.*

3.3.4 Algorithm reformulation

The hardness of SET COVER lies in the difficulty of breaking ties when deciding on which sets to include or not in the final cover. Each valid choice forks into a series of possible paths, inducing an implicit *decision tree* of progressively simpler instances. Randomized removals from CANDIDATES essentially provide a way of randomly breaking these ties, which avoids time being spent in exhaustively examining every possible path in the decision tree.

A crucial observation is that we do not need to restrict kernelization to the beginning of the algorithm. In fact, after some set $S \in \mathcal{F}$ is chosen to be removed, the structure of the instance may radically change and become kernelizable again. So kernelization can prove to be even more useful if we keep performing it after *every* iteration of the algorithm. In this sense, we can reframe the algorithm as alternating between two main phases: randomized removals (pushing towards efficiency) and deterministic kernelizations (pushing towards optimality). This interleaving of two phases helps significantly reduce the number of random choices made throughout the whole execution, which has a beneficial impact on the approximation ratio. While randomly removing a set $S \in \mathcal{F}$ transforms the instance into another instance that may not be completely equivalent, every kernelization step is guaranteed to preserve an optimal solution.

Another aspect to note is that, when we repeatedly run kernelization over and over again, removing proper subsets is not the only thing we can do to simplify the SET COVER instance. For example, if a set S has at least one element that only it contains, then S must necessarily appear in the final cover, so we can remove S from the "equation" by just forcing it to be included in $\mathcal{C}$. And, after S has been permanently placed in the cover, all the elements it contains will definitely be covered in the end anyway, regardless of other future decisions we make. Hence, these elements are not required anymore to be explicitly covered later and can be ignored – that is, subtracted from $\mathcal{U}$ and from every set in $\mathcal{F}$. Moreover, by removing these elements from every set, it may happen that some set now becomes empty and can be safely removed because it no longer contributes to covering $\mathcal{U}$. Or maybe some two sets now become identical to each other, and including both in $\mathcal{C}$ would be redundant – so one of them may also be ignored. Furthermore, we could also choose to ignore elements $x \in \mathcal{U}$ that, in some iteration, belong to every remaining set $S \in \mathcal{F}$, because any of these sets would clearly be covering them anyway.

Hereupon we revise the kernelization process – and, with it, Randomized-Set-Cover itself –, making it more complete and robust. Whenever we use "Randomized-Set-Cover" in the remainder of this chapter we will be referring to this new version. In the new description of the algorithm that follows, each set originally in $\mathcal{F}$ may at any time be in one of three states: *fixed*, *candidate* or *removed*. A set that is fixed has at least one element with frequency 1 and, thus, must definitely be placed in the cover (and can be safely removed from the instance in the next kernelization iteration). Candidate sets are the redundant ones that are randomly removed from the instance in every iteration – but at some point they may also become (permanently) fixed, if after a subtraction operation they become the only sets containing some element. After a set S is removed, either because it is superfluous or because it is not, S will not matter ever again during the remainder of the execution. In this logic, the only possible state transitions are the ones in Figure 3.5. Note that fixed sets and candidate sets may both be removed from the instance, but for opposite reasons: a fixed set is (deterministically) removed from the instance to integrate the final cover $\mathcal{C}$, while a candidate set is (randomly) removed from the instance to *not* be placed in $\mathcal{C}$.

$$\begin{tikzcd}
\text{candidate} \arrow[rr] \arrow[dr] & & \text{fixed} \arrow[dl] \\
& \text{removed} &
\end{tikzcd}$$

Figure 3.5: Valid set state transitions in Randomized-Set-Cover.

Let us define Fixed and Candidates as the sets of all fixed and candidate sets in $\mathcal{F}$, respectively. Randomized-Set-Cover keeps running while Candidates $\neq \emptyset$, alternately kernelizing the instance and removing one of the sets $S \in$ Candidates at random. As soon as Candidates becomes empty, it terminates and the cover $\mathcal{C}$ is returned. Before finally describing the algorithm, we define the following functions, each of them corresponding to a kernelization stage:

Kernelize-Recompute-Fixed-Sets(): Identifies all the sets $S \in$ Candidates for which some element x has frequency 1 (that is, $|S^{-1}[x] = 1|$), removes them from Candidates and inserts them into Fixed.

Kernelize-Fixed-Sets(): For each fixed set S, subtracts S from every fixed or candidate set and moves it into $\mathcal{C}$ (thus, incrementing $|\mathcal{C}|$ in 1). Also calls Kernelize-Recompute-Fixed-Sets().

Kernelize-Equal-Sets(): Removes duplicate sets from Candidates, which may be caused by set subtractions. (It may be noted that Candidates is a set and thus cannot contain duplicate elements, making this operation theoretically redundant. However, from the perspective of implementation we may need to explicitly enforce this uniqueness property, and that is why we define Kernelize-Equal-Sets.) We may (arbitrarily) assume that the first occurrence of a set is the one to keep; the following occurrences are removed. This operation also calls Kernelize-Recompute-Fixed-Sets(). Note that, by definition, no fixed set can be equal to any other set in $\mathcal{F}$ (that is, if two sets in $\mathcal{F}$ are equal, then they must both be candidate sets). Therefore, this operation only needs to be performed over Candidates. Also note that the choice of the subset to be removed has a direct effect in the content of the final cover, although all of these possible covers have equal cardinality.

Kernelize-Proper-Subsets(): Removes from Candidates any set that is a proper subset of another and calls Kernelize-Recompute-Fixed-Sets(). Again, by definition, only a candidate set may be a proper subset of another set (fixed or candidate).

Kernelize(): Calls Kernelize-Fixed-Sets(), Kernelize-Equal-Sets() and Kernelize-Proper-Subsets() in this order and returns True if and only if at least one of these three phases has modified the instance (and False otherwise).

Algorithm 5 shows the more refined version of Randomized-Set-Cover, which we adopt from this moment on.

Algorithm 5 Randomized-Set-Cover($\mathcal{U}, \mathcal{F}$), revised

```
Compute S^{-1}[x] for every x ∈ U
Split F into Fixed and Candidates based on S^{-1}[·]
C ← ∅
While True:
    While Kernelize(): {}            ▷ keep kernelizing until a fixed-point is reached
    If Candidates = ∅: Break
    S ← Candidates.Choose-Random()
    Remove S from Candidates, erase every x ∈ S and update S^{-1}[x] accordingly
    Kernelize-Recompute-Fixed-Sets()
Return C
```

Example 3.1. Figure 3.6 exemplifies an execution of Randomized-Set-Cover and the possible sequences of actions it may branch into, which we describe now. The very first thing the algorithm does is split $\mathcal{F}$ into Fixed and Candidates. Since every element $x \in \mathcal{U}$ has frequency greater than 1, every set becomes a candidate. Then the main "**While** Kernelize()" loop begins and the set $\{1, 3\} \subseteq \{1, 3, 4\}$ is removed by Kernelize-Proper-Subsets(). Since there remain no fixed sets, equal sets or proper subsets, this kernelization iteration ends and the randomized phase begins. Each of the four candidate sets is equally likely to be chosen, with probability 1/4. Let us analyze what happens in each case.

- If the algorithm chooses to remove $\{1, 2, 5\}$, the frequency of 1, 2 and 5 becomes 1. Hence, all three sets $\{\mathbf{2}, 3, 4\}$, $\{3, \mathbf{5}\}$ and $\{\mathbf{1}, 3, 4\}$ become fixed, and the algorithm terminates with a suboptimal cover of size 3.

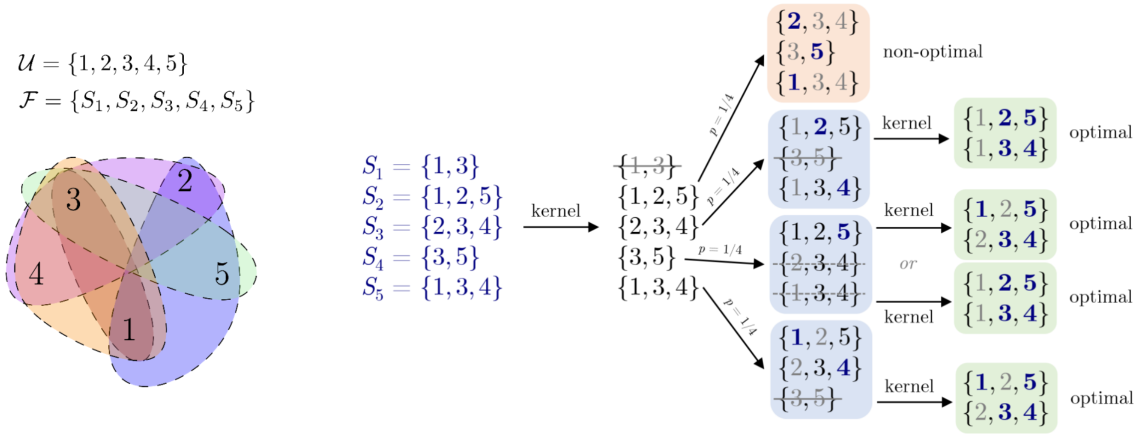


Figure 3.6: Solving the instance of Figure 3.1 using the improved version of Randomized-Set-Cover. The possible outcomes of the algorithm are shown. Elements in bold blue have frequency 1 and, therefore, belong to fixed sets. Removed elements $x \in \mathcal{U}$ appear in light gray instead of being omitted, so that it becomes clear which set is which. Elements that, in some iteration, belong to every remaining set are also shown in light gray, because they do not affect kernelization nor random choices in any way and can be ignored. The algorithm finds an optimal solution for this instance in a single run with probability 75%. After 3 runs, this probability increases to $1-(1-0.75)^3$, which is approximately 98.4%.

- If $\{2,3,4\}$ is chosen, then the frequency of elements 2 and 4 becomes 1. Given that $\{1,\mathbf{2},5\}$ and $\{1,3,\mathbf{4}\}$ become fixed, they are both moved into $\mathcal{C}$ and their union $\{1,2,3,4,5\}$ gets subtracted from every remaining set, due to Kernelize-Fixed-Sets(). Therefore, $\{3,5\}$ becomes empty and is removed due to Kernelize-Proper-Subsets(), which leaves an optimal cover of size 2.

- If, on the other hand, $\{3,5\}$ is removed initially, $\{1,2,5\}$ becomes fixed because it is the only set containing element 5. $\{1,2,5\}$ then gets subtracted from every set, leaving two equal sets $\{3,4\}$, both candidates. Kernelize-Equal-Sets() then removes one of them arbitrarily; either way, the frequency of elements 3, 4 and 5 becomes 1 and, thus, the two remaining sets become fixed. A cover of size 2 is then returned.

- Finally, if the set $\{1,3,4\}$ is selected instead, elements 1 and 4 make sets $\{\mathbf{1},2,5\}$ and $\{2,3,\mathbf{4}\}$ become fixed. This in turn makes their union be subtracted from every set, emptying $\{3,5\}$. This empty set is then removed by Kernelize-Proper-Subsets() and an optimal cover is produced.

Additional optimization For implementing Randomized-Set-Cover, we give each convex piece a unique integer identifier, from 1 to $|\Pi(P)| = |\mathcal{U}|$. These identifiers are perfectly arbitrary in the sense that, if one were to permute them, the resulting Set Cover instance would become different, but would still admit an equivalent solution set. What is subtler in this is that the running time of Randomized-Set-Cover is in fact sensitive to the naming of the pieces: by permuting piece identifiers, we obtain an isomorphic Set Cover instance which may take more or less time to solve. This seemingly odd event is directly related with the way set operations are implemented. We assume $\mathcal{F} = \{S_1, S_2, \ldots, S_{|\mathcal{F}|}\}$. To understand whether there are any sets $S_i, S_j \in \mathcal{F}$ such that $S_i \subset S_j$, we do something analogous to Algorithm 6.

Algorithm 6 Identify proper subsets in $\mathcal{F}$

```
For i ∈ {1, 2, ..., |F|}:
    For j ∈ {1, 2, ..., |F|} \ {i}:
        If |S_i| < |S_j| ∧ DECIDE-⊆ (S_i, S_j)
            ...                                  ▷ proper subset found
```

In the case $S_i \not\subset S_j$ for some $S_i, S_j \in \mathcal{F}$, the sooner an element x such that $x \in S_i$ and $x \notin S_j$ is found, the sooner the predicate $\forall x \in S_i, x \in S_j$ gets evaluated to False by short-circuiting (Boolean lazy evaluation). This corresponds to early pruning the evaluation in case a counterexample $x \notin S_j$ is found.

Suppose $S_i = \{x'_1, x'_2, \ldots, x'_{|S_i|}\}$, where $x'_1 < x'_2 < \cdots < x'_{|S_i|}$. We say that a prefix $x'_1, x'_2, \ldots, x'_k$ of S_i matches with S_j if $x'_1, x'_2, \ldots, x'_k \in S_j$. When testing if $S_i \subset S_j$ by traversing elements of S_i in sorted order, the length of the longest matching prefix (LMP) of S_i and S_j directly affects the time it takes to evaluate the predicate. For minimizing the time it takes to kernelize the SET COVER instance, we are interested in a permutation σ of the piece identifiers that minimizes the sum of lengths of the longest matching prefixes among all pairs $S_i, S_j \in \mathcal{F}$:

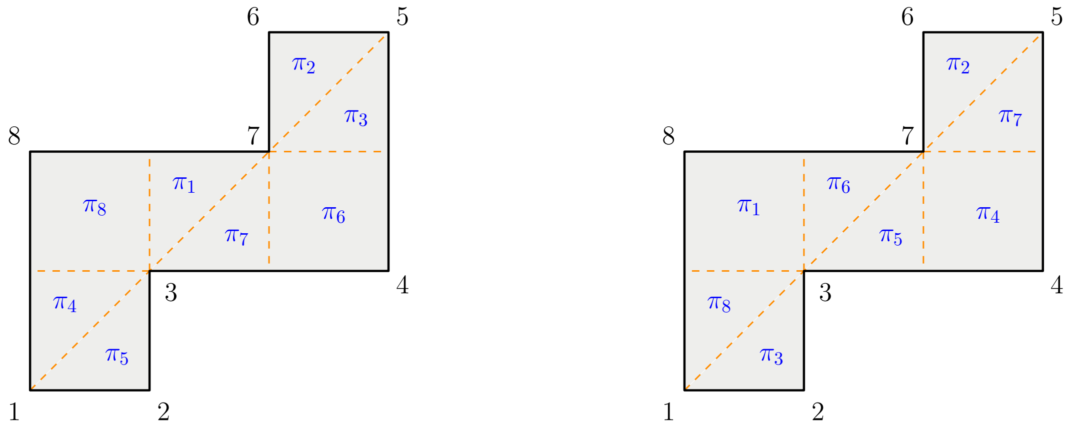
$$\min_{\sigma} \sum_{\substack{i \in \{1,2,\ldots,|\mathcal{F}|\} \\ j \in \{1,2,\ldots,|\mathcal{F}|\} \setminus \{i\}}} \mathsf{LMP}_{\sigma}(S_i, S_j) \tag{3.15}$$

This subproblem is interesting by itself and may admit a nice exact solution. However, given that at least one kernelization round happens in every iteration of RANDOMIZED-SET-COVER, different subproblems of this kind would need to be constantly re-solved to optimality for an ideal σ to be found, and that would make us spend an enormous amount of time. So, instead, this subproblem was solved here by simply generating a permutation of the piece identifiers uniformly at random immediately after the convex partition $\Pi(P)$ is computed (Figure 3.7).

For testing an inclusion $S_i \subset S_j$, elements of S_i are now traversed in increasing order of $x'_{\sigma(1)}, x'_{\sigma(2)}, \ldots, x'_{\sigma(k)}$. This is a clean and efficient way of dealing with the problem, because the worst case sum of lengths of longest matching prefixes is unlikely to be consistently hit when considering a random permutation of the elements. The instance solving times we obtained in practice were very satisfactory, as results in Section 4.2 tell. This transformation has the advantage of being very straightforward and only needing to be performed once at the beginning.

3.3.5 Number of repetitions

It is undesirable that the algorithm performs random choices to remove sets from $\mathcal{C}$ very often. The fewer times it has to choose randomly, the more *deterministic* it is, so the greater the probability



$$
\begin{aligned}
S_1 &= \{x_1, x_4, x_5, x_8\} \\
S_2 &= \{x_4, x_5, x_8\} \\
S_3 &= \{x_1, x_3, x_4, x_5, x_6, x_7, x_8\} \\
S_4 &= \{x_1, x_2, x_3, x_6, x_7, x_8\} \\
S_5 &= \{x_2, x_3, x_6, x_7\} \\
S_6 &= \{x_2, x_3, x_6\} \\
S_7 &= \{x_1, x_2, x_3, x_4, x_6, x_7, x_8\} \\
S_8 &= \{x_1, x_4, x_5, x_6, x_7, x_8\}
\end{aligned}
$$

$$
\begin{aligned}
S_1 &= \{x_1, x_3, x_6, x_8\} \\
S_2 &= \{x_1, x_3, x_8\} \\
S_3 &= \{x_1, x_3, x_4, x_5, x_6, x_7, x_8\} \\
S_4 &= \{x_1, x_2, x_4, x_5, x_6, x_7\} \\
S_5 &= \{x_2, x_4, x_5, x_7\} \\
S_6 &= \{x_2, x_4, x_7\} \\
S_7 &= \{x_1, x_2, x_4, x_5, x_6, x_7, x_8\} \\
S_8 &= \{x_1, x_3, x_4, x_5, x_6, x_8\}
\end{aligned}
$$

Figure 3.7: Two isomorphic SET COVER instances derived from the convex partition of an orthogonal polygon. One can convert the first instance to the second one through the permutation $\sigma = (6\ 2\ 7\ 8\ 3\ 4\ 5\ 1)$. On the original instance, $\mathsf{LMP}_{\mathsf{id}}(S_1, S_7) = 2$, while, for the transformed instance, $\mathsf{LMP}_{\sigma}(S_1, S_7) = 1$. $\sum \mathsf{LMP}_{\mathsf{id}}(S_i, S_j) = 75$ and $\sum \mathsf{LMP}_{\sigma}(S_i, S_j) = 85$.

of finding an optimal solution becomes. If an instance, after being kernelized, suddenly turns out to be easily solvable with few random choices, the probability of an individual run to locate an optimal cover will increase, so not many repetitions will be required to achieve optimality. However, how the kernelization is conducted depends very much on the structure of the sets in $\mathcal{F}$. Moreover, the accumulated outcome of the several kernelization steps itself becomes in a sense unpredictable after a sequence of random choices have been made. It is therefore very difficult to recommend a specific number of repetitions $R(\mathcal{U}, \mathcal{F})$ that work well for any instance $(\mathcal{U}, \mathcal{F})$ – not falling short of what is needed, nor overshooting a reasonable amount.

Nevertheless, we propose an informal heuristic for choosing $R(\mathcal{U}, \mathcal{F})$ when $|\mathcal{F}|$ is not very large. It serves as an advisable lower bound.

Theorem 3.9 (Coupon collector's problem, [22]). *Let A be a set of n distinct elements and $B = \emptyset$. Consider the operation of extracting an element uniformly at random from A, inserting a copy into B and then replacing it back into A. The expected number of independent operations that are performed until $A = B$ is $n \cdot H_n$ and its variance is given by $n^2 \cdot \sum_{k=1}^{n} \frac{1}{k^2} - n \cdot H_n$.*

Suppose that $\mathcal{F}' \subseteq \mathcal{F}$ results from the first kernelization of $\mathcal{F}$. Given that the kernelization ensures that $\mathcal{F}$ is *irreducible* (in the sense that there is no immediate way to reduce it by further kernelizing it), any of the sets in $\mathcal{F}'$ has, from the current state of the algorithm, equal potential to lead to an optimal solution. Each of these sets may give origin to several sequences of choices, each corresponding to a path in a decision tree. In order not to miss any of those starting points, Theorem 3.9 suggests a number of repetitions in the order of $\Omega(|\mathcal{F}| \cdot H_{|\mathcal{F}|})$. Notice that this heuristic is only probabilistic and in no way guarantees success in finding an optimal solution, nor even that each of the $|\mathcal{F}'|$ starting points will be attempted. It also works better for smaller values of $|\mathcal{F}|$, since, by Theorem 3.9, the variability of the expected number of repetitions needed is significant and grows quadratically with n.

3.4 Data structures used

We describe data structures that we have developed and used in our implementation of Randomized-Set-Cover. Alternatives have been carefully weighted so as to make the algorithm as efficient as possible – both in time and in memory.

3.4.1 Sampleable set

A new set data structure, which we call *sampleable set*, was devised for supporting the operation Choose-Random() of the Candidates set that has been mentioned several times so far. Besides Choose-Random(), a sampleable set also supports the more usual operations Insert(x) and Erase(x). As an internal state, it maintains a vector `items` for storing the elements in a set S, in any order, and a reverse map `item_position` for storing the index that each element $x \in S$ occupies in `items`. We assume that vector `items` supports the operations PushBack(x) and PopBack(), which, respectively, append a new element x to the end of the container and remove from it the element on the last position (Back()). On the other hand, map `item_position` provides methods Contains(x) and Erase(x), which, respectively, return a Boolean indicating whether element x is being mapped and stop mapping x (if it is currently doing so). Random(a, b) returns an integer in range $[a, b]$ uniformly at random in constant time.

Algorithm 7 Insert(x)

```
1: If item_position.Contains(x):
2:     Return
3: items.PushBack(x)
4: item_position[x] ← |items| − 1
```

Algorithm 8 Erase(x)

```
If ¬item_position.Contains(x):
    Return
position ← item_position[x]
item_position.Erase(x)
last_item ← items.Back()
items.PopBack()
If position ≠ |items|:
    items[position] ← last_item
    item_position[last_item] ← position
```

Algorithm 9 Choose-Random()

```
Return items[Random(0, |items| − 1)]
```

Example 3.2. Suppose that the following sequence of insertions was performed over an empty sampleable set:

$$\textsc{Insert}(3),\quad \textsc{Insert}(5),\quad \textsc{Insert}(3),\quad \textsc{Insert}(0),\quad \textsc{Insert}(2),\quad \textsc{Insert}(4)$$

The state is then:

$$\begin{aligned} \texttt{items} &= [3, 5, 0, 2, 4] \\ \texttt{item\_position} &= [(3:0), (5:1), (0:2), (2:3), (4:4)] \end{aligned}$$

If Erase(6) is called, nothing changes. If Erase(4) is called, we have:

$$\begin{aligned} \texttt{items} &= [3, 5, 0, 2, \cancel{4}] & &\to [3, 5, 0, 2] \\ \texttt{item\_position} &= [(3:0), (5:1), (0:2), (2:3), \cancel{(4:4)}] & &\to [(3:0), (5:1), (0:2), (2:3)] \end{aligned}$$

After Erase(5), the state becomes:

$$\begin{aligned} \texttt{items} &= [3, \cancel{5}, 0, 2] & &\to [3, 2, 0] \\ \texttt{item\_position} &= [(3:0), \cancel{(5:1)}, (0:2), (2:3)] & &\to [(3:0), (2:1), (0:2)] \end{aligned}$$

Although keeping the elements of S organized in a tree structure could seem more plausible at first, the difficulty of implementing a Choose-Random() method would increase, because we would then have to extend each tree node with bookkeeping information, namely its subtree size, in order to ensure that the random selection process is uniform (such a structure is often called an *augmented tree*). This is why we ended up storing elements of S linearly in a vector container, `items`. In that sense, we took advantage of the fact that the order of the elements in `items` does not affect the probability of each one being selected by Choose-Random() to simplify the implementation of all three functions. While Choose-Random() clearly takes constant time, Insert(x) and Erase(x) may take either $\mathcal{O}(1)$ or $\mathcal{O}(\log|S|)$ time, depending on whether the reverse map `item_position` is implemented as a vector (for bounded values $x \in S$) or as a tree-based dictionary (for large values).

3.4.2 Set

Apart from a sampleable set that efficiently supports the Choose-Random() operation, we are interested in implementing a more general data type for sets $S \in \mathcal{F}$. A set data type should abstractly represent a collection of distinct elements, without any order. Often enough, any element of a special domain can be mapped to a non-negative integer identifier that uniquely encodes it, and therefore in this subsection we only discuss sets of integers. We require sets $S \in \mathcal{F}$ to be dynamic or mutable, meaning that one may request insertions and deletions of elements on demand. Additional methods are also helpful, such as for finding the cardinality of the set, locating its minimum element and iterating through all the elements in some order. Because of that, we do also impose a total order on the elements in the universe $\mathcal{U}$, corresponding to the natural order of their identifiers.

Many ways exist for implementing a set, each providing different guarantees in terms of performance and different time and space trade-offs. We require deterministic sets, so probabilistic data structures such as Bloom filters [19] are out of question in our application. A trivial representation of a set is an array or a list that explicitly contains its elements in some order, without duplicates. In most cases, this is very inefficient because operations such as membership tests, insertions and removals take linear time in the set size. Another commonly used data structure for dictionary-based sets, the hash table, has a constant-time performance per operation on the average case, but collisions still make it linear in the table size on the worst case. The most generic implementation of a dynamic set is often internally supported by some variation of a self-balancing binary search tree, such as an AVL tree [3] or a Red-Black tree [54]. These data structures typically achieve linear memory and logarithmic time per operation, which are strong worst-case guarantees in general.

A trie [51] is another efficient data structure used for implementing sets, usually of strings. It provides insertions, removals and membership tests of a given string s in time linear in the length of the string, $|s|$. A trie can also be used for storing non-negative integers if one treats them as strings of bits (or digits), but, since an integer $0 < x < n$ has a binary representation with $1 + \lfloor \log_2 x \rfloor$ bits, each of these operations also takes $\mathcal{O}(\log n)$ time.

It is in fact possible to improve the running time of each operation and the total space that the representation of the set takes in memory even further by a significant constant factor. The idea is to take advantage of the fact that a modern CPU can efficiently operate with 32- or 64-bit words and execute bitwise operations that take very few clock cycles to complete. If we let $\mathcal{U} = \{0, 1, \dots\}$ be a universe of non-negative integers and W denote the word size (that is, the length of the natural unit of data processed by a CPU, which we take as $W = 64$ bits), a common approach is then to represent any set $S \subseteq \mathcal{U}$ as a bitmask $S_{(2)}$ of $W \cdot \left\lceil \frac{|\mathcal{U}|}{W} \right\rceil$ bits, whose i-th bit ($0 \le i < W \cdot \left\lceil \frac{|\mathcal{U}|}{W} \right\rceil$) is 1 if and only if $i \in S$.

This representation is very lightweight in terms of memory and achieves good performance if the universe is not very large. However, regardless of whether S is dense or sparse, storing such a representation will still always require the same amount of bits. For example, both sets $\emptyset$ and $\mathcal{U}$ will each occupy $W \cdot \left\lceil \frac{|\mathcal{U}|}{W} \right\rceil$ bits in memory and computing their intersection will still take time proportional to that value. This disadvantage motivates us to develop a new data structure, which we are ready to describe.

3.4.2.1 Bitset tree

We now propose an optimized data structure for representing and manipulating sets of non-negative integers that circumvents the problem of sparsity while still taking advantage of the CPU's bit-level parallelism. The magnitude of memory used by this structure for representing a set of n elements taken from a universe $\mathcal{U}$ is not greater than $\min\left(n, \left\lceil \frac{|\mathcal{U}|}{W} \right\rceil\right)$.

Let us partition $\mathcal{U}$ into $\left\lceil \frac{|\mathcal{U}|}{W} \right\rceil$ blocks of one W-bit word each. We number the blocks 0 through $\left\lceil \frac{|\mathcal{U}|}{W} \right\rceil - 1$ in order, and number each position 0 through $W - 1$ within a block. The *global position* of an element in $\mathcal{U}$ is described by a pair (b, p), which indicates the number b of the block and its position p within the block. Concretely, for an element $x \in \mathcal{U}$, its global position is $\left(\left\lfloor \frac{x}{W} \right\rfloor, x \bmod W\right)$. Also, given the global position (b, p) of an element x, we can recover x as $x = b \cdot W + p$.

Remark. *Although computing divisions and remainders is assumed to be performable in $\mathcal{O}(1)$ time, the hidden constants are often very large. In terms of implementation, a much faster way to compute the global position of an element $x \in \mathcal{U}$ that we have decided to employ makes use of bitwise operators: $(b, p) = (x \gg \log_2(W), x \,\&\, (W - 1))$. Similarly, we can recover the element x given its global position (b, p) as $x = (b \ll \log_2(W)) \mid p$. We assume we have precomputed the value $\log_2(W)$, which is always an integer given that W is a power of 2. In this work, we take $W = 64$ and $\log_2(W) = 6$.*

We define a *bitset tree* $\mathcal{T}$ as a self-balancing binary search tree of pairs of integers. Given any set $S \subseteq \mathcal{U}$, we construct a bitset tree representation for S, $\mathcal{T}_S$, as follows. Denote by $S_{(2)}$ the bitwise representation of S with $W \cdot \left\lceil \frac{|\mathcal{U}|}{W} \right\rceil$ bits. For each block b $\left(0 \leq b < \left\lceil \frac{|\mathcal{U}|}{W} \right\rceil\right)$ in $S_{(2)}$ that corresponds to a W-sized word w that is not empty (i.e., $w \neq \underbrace{00\ldots0}_{W}$), insert the pair (b, w) into $\mathcal{T}_S$. See Figure 3.8a for an example of a bitset tree built for a given set. We denote by $|\mathcal{T}_S|$ the number of nodes in a bitset tree $\mathcal{T}_S$.

Proposition 3.10. *If $\mathcal{T}_S$ is the bitset tree built for a given set $S \subseteq \mathcal{U}$, then $|\mathcal{T}_S| \leq \min\left(|S|, \left\lceil \frac{|\mathcal{U}|}{W} \right\rceil\right)$.*

Proof. The number of non-empty blocks in $S_{(2)}$ of size $W \geq 1$ cannot exceed $|S|$. Besides, we add at most one node in $\mathcal{T}_S$ for each block in $S_{(2)}$, of which there are $\left\lceil \frac{|\mathcal{U}|}{W} \right\rceil$ many. □

It can also be seen that, the denser the set is, the more efficient its bitset tree representation comparatively becomes, because the number of blocks stops increasing after it hits the threshold of $\left\lceil \frac{|\mathcal{U}|}{W} \right\rceil$, even if we keep inserting more elements.

A few operations that are useful for the algorithm are described now.

Forward iterator A forward iterator is an object that allows iterating through the elements of S in sorted order, while possibly performing some other operation in parallel (for instance, printing them). It also refers to the "iterator" design pattern in object-oriented programming. A naïve way of implementing a forward iterator for S would be to keep as the iterator state the global position (b, p) we are currently at, and use that to loop through all the nodes (b, w) in $\mathcal{T}_S$ and, for each one, loop through all the bits of the word w. In the worst case, this takes time $\mathcal{O}(|\mathcal{U}|)$, which is unacceptable if blocks in $S_{(2)}$ are very sparse. For instance, if $\mathcal{T}_S$ has $\left\lceil \frac{|\mathcal{U}|}{W} \right\rceil$ nodes, each with a single on bit in its word, we need to consider $|\mathcal{U}|$ bits in total.

Our implementation of a forward iterator is more efficient in that it uses bitwise tricks to *jump* directly to the next on bit of the current word w being processed without having to linearly look for it. As such, if for example a word only has one bit on, we process it and simply jump to the next node in $\mathcal{T}_S$ (using an iterator for the tree structure itself) without even needing to consider the remaining $W - 1$ off bits. This enables us to iterate over S in time linear in its size, $\mathcal{O}(|S|)$, while still maintaining the convenient tree structure we described. The idea is that most modern hardware architectures provide fast, dedicated instructions to count the number of trailing zeroes on the binary representation of a positive integer, with a very small constant factor. We used the function `__builtin_ctzll` provided by the GCC compiler for C++; refer to [32] for more on the hardware support of related operations. The number of trailing zeroes on the binary representation of a positive integer x, $\mathtt{tz}(x)$, coincides with the position of its least significant on bit (LSB). So the way we find every one of the k on bits in a word w with a number of operations that, for all practical purposes, is $\Theta(k)$ is simple. Make a copy $w' \leftarrow w$; then, while $w' \neq 00\ldots0$, compute the position of the LSB, which is $\mathtt{tz}(w')$, and turn it off: $w' \leftarrow w' \;\&\; \left(\sim 2^{\mathtt{tz}(w')}\right)$.

Figure 3.8b shows the order of iteration in $\mathcal{T}_S$ of the elements of a given set S.

Test containment We can test whether a given element x, with global position (b, p), is in S by checking if there is a node of the form (b, w) in $\mathcal{T}_S$ and, if so, checking whether its mask w has the p-th bit on: $(w \;\&\; (2^p)) \neq 0$. This is performed in $\mathcal{O}(\log |\mathcal{T}_S|) = \mathcal{O}\left(\log \min\left(|S|, \left\lceil \frac{|\mathcal{U}|}{W} \right\rceil\right)\right)$ time, because performing a lookup for the node (b, w) takes logarithmic time, given that $\mathcal{T}_S$ is self-balancing, and then accessing its word w takes constant time.

Test inclusion, proper inclusion and equality For testing if S is a subset of another set $O \subseteq \mathcal{U}$, with bitset tree $\mathcal{T}_O$, we iterate over the pairs (b, w_S) in $\mathcal{T}_S$ in order. For each one, we check if a node of the form (b, w_O) does also occur in $\mathcal{T}_O$. If not, then definitely $S \nsubseteq O$; otherwise, for $S \subseteq O$ it must hold that $w_S \;\&\; w_O = w_S$. This approach takes $\mathcal{O}(|\mathcal{T}_S| \log |\mathcal{T}_O|) = \mathcal{O}\left(\min\left(|S|, \left\lceil \frac{|\mathcal{U}|}{W} \right\rceil\right) \log \min\left(|O|, \left\lceil \frac{|\mathcal{U}|}{W} \right\rceil\right)\right)$ time.

Remark. *Note that this is, in general, better than the more common way of testing inclusion* *which performs a bitwise AND between the entire bitmasks $S_{(2)}$ and $O_{(2)}$, each with exactly $W \cdot \left\lceil \frac{|\mathcal{U}|}{W} \right\rceil$ bits, because it takes advantage of the sparsity of sets S and O.*

The correctness of our method directly stems from the following property:

Proposition 3.11. *Let $A, B \subseteq \mathcal{U}$. Then, $A \subseteq B$ if and only if for any $C \subseteq \mathcal{U}$ we have $A \cap C \subseteq B \cap C$.*

If we wanted to test whether $S \subset O$ or $S = O$, we could additionally check if $|S| < |O|$ or if $|S| = |O|$, respectively.

Insert element For inserting an element $x \in \mathcal{U}$ into S, we first calculate its global position (b, p) and check if a node (b, w) already exists in $\mathcal{T}_S$. If not, we create it, having $w = 00\ldots0$. Then, we turn on the p-th bit of w: $w \leftarrow w \mid (2^p)$. The time complexity of insertion is $\mathcal{O}(\log |\mathcal{T}_S|)$.

Erase element For erasing a given $x \in \mathcal{U}$ from S, we first ensure that $x \in S$ by testing containment. In the positive case, we locate the node (b, w) in $\mathcal{T}_S$ corresponding to x and turn off w's p-th bit: $w \leftarrow w \ \& \ \sim(2^p)$. If w becomes $00\ldots0$, we erase the node (b, w) from $\mathcal{T}_S$. The time complexity for erasing is also $\mathcal{O}(\log |\mathcal{T}_S|)$.

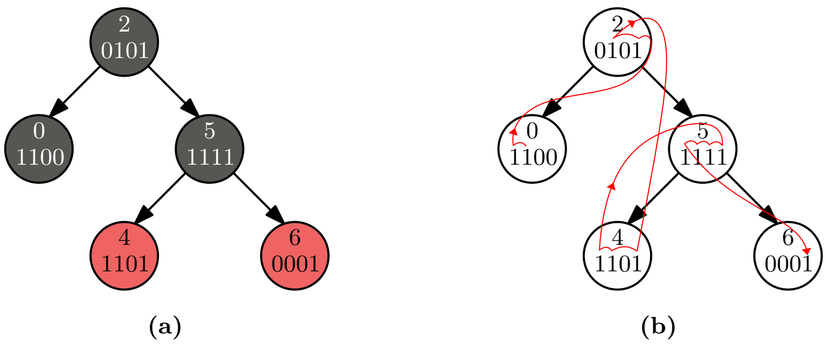


Figure 3.8: **(a)** A bitset tree $\mathcal{T}_S$ for the set $S = \{2, 3, 8, 10, 16, 18, 19, 20, 21, 22, 23, 24\}$, with universe $\mathcal{U} = \{0, 1, 2, \ldots, 24\}$ and $W = 4$. Elements were inserted in increasing order of their value. Each node indicates a pair (b, w), where b is the identifier of the block and w is the bitmask word corresponding to the block. Nodes are colored black or red according to the color they would have in an underlying Red-Black tree. **(b)** Order of iteration in $\mathcal{T}_S$ of the elements of S, in increasing order of their values.

Chapter 4

Experimental Analysis of Randomized-Set-Cover

This chapter describes the experimental setup we adopted for evaluating the performance of RANDOMIZED-SET-COVER in terms of efficiency and optimality. An empirical analysis of the algorithm is performed, in which we evaluate metrics such as running time and approximation ratio by running RANDOMIZED-SET-COVER against a public dataset of orthogonal polygons. Finally, we analyse the number of pieces that vertex visibility polygons induce in the tested classes and propose a conjecture.

4.1 Experimental setup

The algorithm and a supplementary geometry library for partitioning input polygons were written in C++17. The library is supported by CGAL 5.4 [50] and Boost 1.78.0. For generating an integer uniform distribution over which random choices are made, the Mersenne Twister [78] pseudo-random number generator (`mt19937`) was employed.

RANDOMIZED-SET-COVER was tested against a large benchmark dataset (`AGP2007`) that was adapted from the ones provided accompanying [34] and [35]. These can be found in [36]. The tests were performed in an ASUS Vivobook Pro 15 N580 running Ubuntu 18.04.6 LTS with a 2.20GHz Intel® Core™ i7-8750H CPU (12 cores) and 16 GiB of RAM.

Three main sessions of tests were conducted, one for each of three orthogonal polygon classes: FAT, MIN-AREA and RANDOM. These classes of instances are referred to as *grid n-ogons* and correspond to n-vertex orthogonal polygons free of collinear edges that may be placed in an $n/2 \times n/2$ square grid. Bounds on their area and their structure have been studied in [11]. According to Definition 4.1, for a fixed number of vertices, FAT polygons are the ones for which the number of pieces in their grid partition is maximum, while for MIN-AREAs this number is minimum. On the other hand, RANDOM instances consist of a sample of randomly generated orthogonal polygons that have been created using the INFLATE-CUT method, proposed in [92].

Afterwards, we also briefly describe a fourth session of tests that was performed with an extended dataset of larger random instances to evaluate Randomized-Set-Cover's scalability. The dataset we used is a subset of the original dataset `AGP2008a`, which can also be found in [36].

Throughout this chapter, we denote by $\Pi_{HV}(P)$ the set of r-pieces in the grid partition of a grid n-ogon P, and by $\Pi(P)$ the set of pieces in the partition of P induced by vertex visibility polygons, which is more granular than $\Pi_{HV}(P)$. Already next we retrieve main definitions and results from [11] that back up our experimental analysis of Fats and Min-Areas.

Definition 4.1 ([11], Definitions 3 and 4)**.** A grid n-ogon Q is called Fat if $|\Pi_{HV}(Q)| \geq |\Pi_{HV}(P)|$ for all grid n-ogons P. Similarly, a grid n-ogon Q is called Thin if $|\Pi_{HV}(Q)| \leq |\Pi_{HV}(P)|$ for all grid n-ogons P. Finally, a Thin grid n-ogon Q is called Min-Area if it has area $n-3$.

Proposition 4.1 ([11], Propositions 2 and 4)**.** *Up to symmetry, there exists a unique Fat grid n-ogon and a unique Min-Area grid n-ogon with a fixed number of vertices, n.*

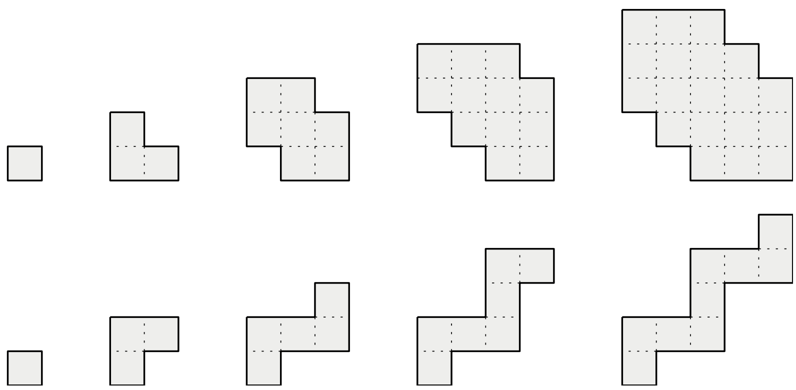


Figure 4.1: The unique (up to symmetry) Fat and Min-Area grid n-ogons for $n \in \{4, 6, 8, 10, 12\}$ (adapted from [94]).

Theorem 4.2 ([11], Theorem 1)**.** *Let P be a grid n-ogon with $r = \frac{n-4}{2}$ reflex vertices. If P is Thin, then $|\Pi_{HV}(P)| = 2r + 1 = n - 3$, and, if P is Fat, then*

$$|\Pi_{HV}(P)| = \begin{cases} \dfrac{3r^2 + 6r + 4}{4} = \dfrac{3n^2}{16} - \dfrac{3n}{4} + 1 & \text{for } r \text{ even} \\[2ex] \dfrac{3(r+1)^2}{4} = \dfrac{3n^2}{16} - \dfrac{3n}{4} + \dfrac{3}{4} & \text{for } r \text{ odd} \end{cases}$$

In each of the four sessions, multiple instances of the corresponding class were solved. The general approach we followed is what one would expect: for each instance P, its partition $\Pi(P)$ induced by vertex visibility polygons was computed and mapped to a Set Cover instance $(\mathcal{U}, \mathcal{F})$ to be solved with Randomized-Set-Cover. Elements of the universe $\mathcal{U}$ correspond to pieces in $\Pi(P)$ and each set in $\mathcal{F}$ contains all the pieces that some vertex of P sees. We have used the CGAL kernel `Exact_predicates_exact_constructions_kernel` for ensuring robust geometric computations. Randomized-Set-Cover was run 20 times on each input for deriving more accurate and meaningful results. We also took advantage of the fact that every execution of the algorithm is independent by making each repetition run in a parallel thread.

4.2 Results

We now describe the constitution of the dataset pertaining to each session and present the results obtained from each experiment. Several measures were gathered or computed in each test: the number of vertices n of the polygon, the number of pieces in its partition $\Pi(P)$, the approximation ratio APX given by RANDOMIZED-SET-COVER and also, for comparison purposes, the approximation factors H_s and f of the GREEDY-SET-COVER and LP-rounding methods that we have discussed in Sections 3.1.1 and 3.1.2. The sizes of optimal vertex-guard sets were provided by the authors of the dataset, and had been determined through an Integer Programming (IP) solver. The time it took for partitioning each polygon, generating the corresponding SET COVER instance and solving it have as well been measured. For reference, the original running times obtained by the dataset authors are provided too; these were obtained in a machine featuring an Intel® Pentium® IV at 2.66 GHz, 1 GB of RAM, CGAL 3.2.1 and FICO Xpress-Optimizer 17.01.02.

In the tables and figures below, by total preprocessing time we mean the sum of the polygon partition time and the SET COVER instance generation time, and the total time refers to the sum of the total preprocessing time and the effective solving time of RANDOMIZED-SET-COVER. These times are measured per input individually.

4.2.1 Min-Area

There are 97 MIN-AREA polygons in the dataset: one for each even n from 8 to 200. Table 4.1 contains collected statistics, while Figures 4.2 and 4.3 show how running times grow in average with n and with $|\Pi(P)|$.

RANDOMIZED-SET-COVER has found an optimal solution for 100% of the MIN-AREAS. It is interesting that $f = 6$ for all instances, which means that every piece in a MIN-AREA is seen by no more than 6 different vertices. Between 42.93% and 97.90% (in average, 92.36%) of the total time was spent in the preprocessing phase, and between 97.99% and 99.85% (in average, 99.62%) of the preprocessing phase time was spent for partitioning the polygon. Curiously, every MIN-AREA instance was fully solved using pure kernelization – no random choice had the chance to be made at all. This once again shows the power that kernelization can have in rendering instances flat simple.

We remark that it had been shown in [77] that the size of an optimal guard set for a MIN-AREA of n vertices is exactly $\lceil \frac{n}{6} \rceil = \lceil \frac{r+2}{3} \rceil$. Our results in Table 4.1 conform to this bound; for instance, the average number of guards in all the 97 instances is given by $\frac{1}{97}\sum_{r=2}^{98} \left\lceil \frac{r+2}{3} \right\rceil \approx 17.67$.

The similarity in the shapes of both left and right plots of Figures 4.2 and 4.3 cannot go unnoticed. Indeed, in Section 4.3.1 we observe that the numbers on the horizontal axes of the right plots are obtained from a linear transformation $f(n) = \frac{3n}{2} - 4$ of the ones on the left plots, and we prove that this holds for any MIN-AREA.

Parameter	Min.	Median	Mean	Max.
$\lvert\mathcal{F}\rvert = n$	8	104	104	200
$\lvert\mathcal{U}\rvert = \lvert\Pi(P)\rvert$	8	152	152	296
OPT	2	18	17.67	34
APX	1	1	1	1
H_s	2.593	2.829	2.827	2.829
f	6	6	6	6
Partition time (s)	0.000882	0.088503	0.112070	0.303239
Instance generation time (s)	1.409×10^{-5}	2.236×10^{-4}	2.343×10^{-4}	4.820×10^{-4}
Total preprocessing time (s)	0.0009001	0.0887157	0.1123040	0.3037210
Solving time (s)	0.0004246	0.0034440	0.0040966	0.0189626
Total time (s)	0.001672	0.091917	0.116401	0.313446
Original total time (s)	0.020	1.270	1.729	5.030

Table 4.1: Statistics for MIN-AREA grid n-ogons.

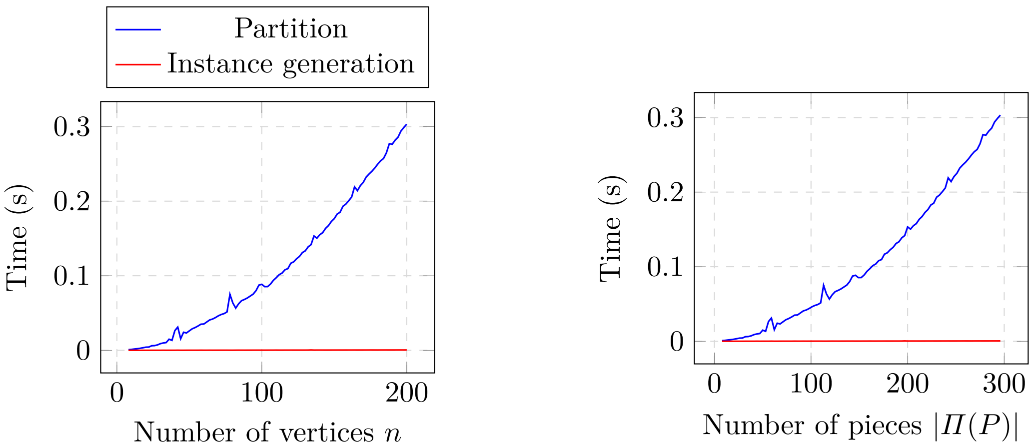


Figure 4.2: Evolution of the partition and instance generation times for MIN-AREAs with number of vertices $n = \lvert\mathcal{F}\rvert$ **(left)** and partition size $\lvert\Pi(P)\rvert$ **(right)**. The time it takes to generate an instance from $\Pi(P)$ is insignificant when compared to the partition time, which hints that partition will be a bottleneck for our algorithm.

4.2.2 Fat

Similarly to MIN-AREAs, the dataset contains 97 FAT polygons, one for each even n from 8 to 200. The statistics gathered are summarized in Table 4.2, and running times are presented in Figures 4.4 and 4.5.

Between 47.65% and 86.88% (in average, 59.00%) of the total time was spent preprocessing, and between 88.69% and 99.40% (in average, 92.68%) of the preprocessing phase time was spent partitioning the polygon. RANDOMIZED-SET-COVER found an optimal vertex-guard set for every FAT in the dataset. It had been observed in [94] that a FAT polygon requires two

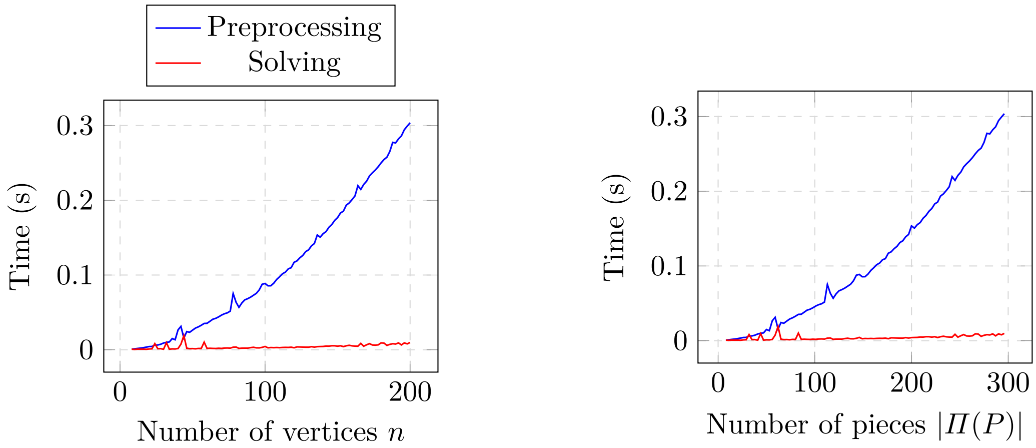


Figure 4.3: Preprocessing and solving times for Min-Area instances with $n = |\mathcal{F}|$ vertices **(left)** and partition size $|\Pi(P)|$ **(right)**. Generating a Set Cover instance from P takes consistently longer than solving it afterwards using Randomized-Set-Cover.

Parameter	**Min.**	**Median**	**Mean**	**Max.**
$\lvert\mathcal{F}\rvert = n$	8	104	104	200
$\lvert\mathcal{U}\rvert = \lvert\Pi(P)\rvert$	11	2151	2739	7743
OPT	1	2	1.979	2
APX	1	1	1	1
H_s	3.020	8.207	7.784	9.507
f	8	104	104	200
Partition time (s)	0.001333	0.670605	1.327012	5.242800
Instance generation time (s)	0.0000207	0.0518434	0.1394528	0.6115390
Total preprocessing time (s)	0.001353	0.720657	1.466465	5.854330
Solving time (s)	0.000993	0.631434	1.502703	6.271260
Total time (s)	0.002347	1.342260	2.969167	12.125600
Original total time (s)	0.03	8.97	21.93	95.19

Table 4.2: Statistics for Fat grid n-ogons.

vertex-guards for $n \geq 12$ vertices, and only one for $4 \leq n \leq 10$. Our algorithm did in fact always find a vertex-guard set of such cardinality for every tested input – and because of that Kernelize-Proper-Subsets() completely solved the instances with $n \leq 10$ in a single iteration all by itself (with zero random choices). It is very interesting that, for $n \geq 12$, Randomized-Set-Cover is able to solve Fat instances down to optimality 100% of the times, even when random selections are made. It seems that the kernelization phase that takes place after each random set is removed in a way compensates for possible mistakes, always leading the algorithm towards the optimal solution, no matter the choices made.

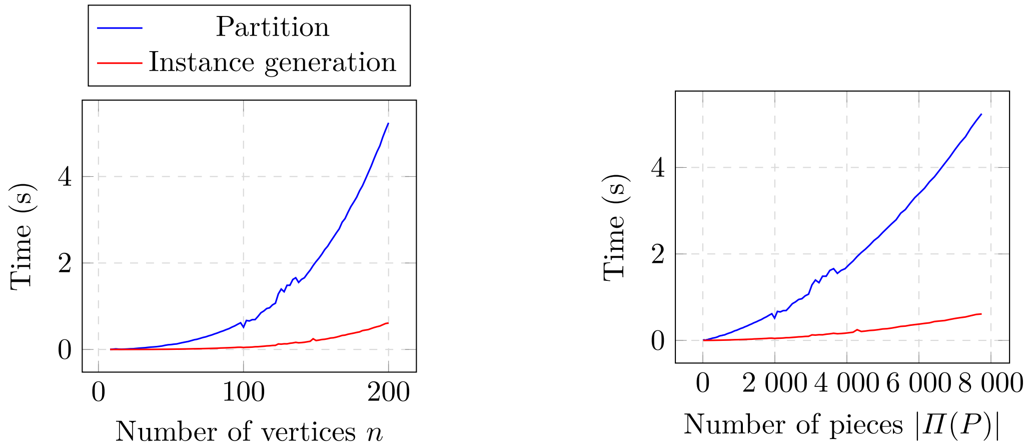


Figure 4.4: Evolution of the partition and instance generation times for Fats with number of vertices $n = |\mathcal{F}|$ **(left)** and partition size $|\Pi(P)|$ **(right)**. Just like for Min-Areas, mapping a partition to a Set Cover instance is faster than computing the partition itself.

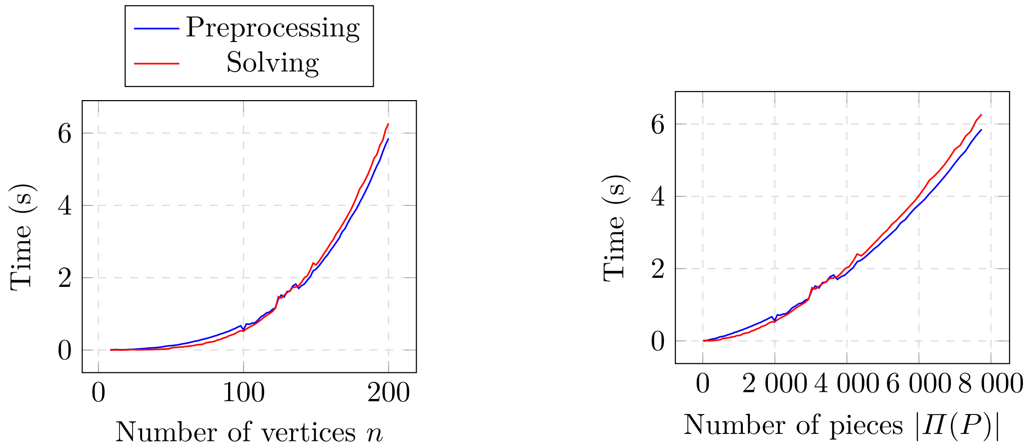


Figure 4.5: Preprocessing and solving times for Fat instances with $n = |\mathcal{F}|$ vertices **(left)** and partition size $|\Pi(P)|$ **(right)**. Since the generated Set Cover instance has a size that is quadratic in n, both times grow faster than for a Min-Area with the same number of vertices.

4.2.3 Random

In the third session, we tested Randomized-Set-Cover with a larger amount of polygons: 10 058 Random orthogonal instances, each having between 14 and 200 vertices. For each $n = 14, 16, \ldots, 200$ in steps of 2, the dataset contains n Random grid n-ogons. Once more, Table 4.3 gives an overview of the main results achieved by the algorithm and Figures 4.6 and 4.7 show how the instance size affects the time it takes to preprocess and solve an instance.

The number of pieces of Random polygons in the dataset ranges approximately from minimum (Min-Area) to maximum (Fat) (see Figure 4.11), so there is high variance on the number of pieces among the instances with the same number of vertices n. Because of that, the right-hand-side plots on Figures 4.6 and 4.7 appear fuzzier.

For every random orthogonal polygon with $14 \leq n \leq 70$ an optimal guard set was found,

Parameter	**Min.**	**Median**	**Mean**	**Max.**
$\lvert\mathcal{F}\rvert = n$	14	142	134.5	200
$\lvert\mathcal{U}\rvert = \lvert\Pi(P)\rvert$	24	1618	1610	5039
OPT	1	23	21.84	38
APX	1	1	1.003	1.143
H_s	3.548	6.726	6.696	8.407
f	8	20	20.65	37
Partition time (s)	0.002699	1.873770	2.029742	9.210860
Instance generation time (s)	3.964×10^{-5}	4.995×10^{-3}	5.387×10^{-3}	2.762×10^{-2}
Total preprocessing time (s)	0.002739	1.879375	2.035129	9.238020
Solving time (s)	0.0008115	0.0614234	0.0636090	0.3057130
Total time (s)	0.00396	1.94698	2.09874	9.54373
Original total time (s)	0.030	4.120	4.334	12.110

Table 4.3: Statistics for RANDOM grid n-ogons.

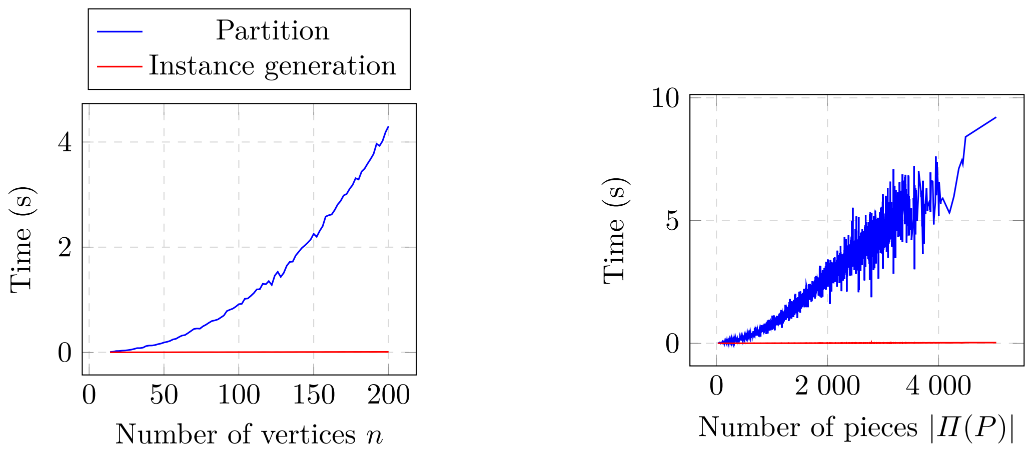


Figure 4.6: Partition and instance generation times for RANDOMS, averaged over all the instances with a fixed number of vertices $n = \lvert\mathcal{F}\rvert$ **(left)** and a fixed partition size $\lvert\Pi(P)\rvert$ **(right)**. Similarly to MIN-AREAS and FATS, instance generation takes considerably less time than computing the partition.

and for $72 \leq n \leq 200$ the algorithm sporadically returned suboptimal solutions. Concretely, an optimal vertex-guard set has been found for 9 396 instances (93.418% of the RANDOM polygons). For the instance on which the obtained APX was highest, the approximation ratio was only 1.143; the polygon had $n = 140$ vertices, the optimal cover had size 21 and the algorithm found one with size 24. Moreover, the approximation ratio was strictly less than H_s and f for 100% of the instances, having been achieved a consistently better performance than what is theoretically guaranteed by GREEDY-SET-COVER and LP-rounding methods.

Finally, between 50.12% and 98.73% (in average, 96.53%) of the total time was spent in the preprocessing phase, while between 96.47% and 99.86% (in average, 99.65%) of the preprocessing

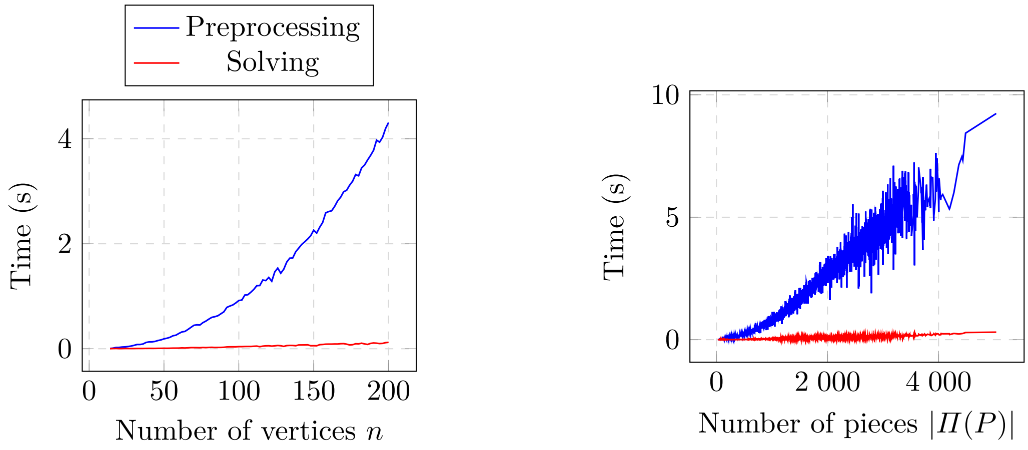


Figure 4.7: Preprocessing and solving times for Randoms, averaged over all the instances with a fixed number of vertices $n = |\mathcal{F}|$ **(left)** and with a fixed partition size $|\Pi(P)|$ **(right)**. Clearly the preprocessing phase dominates the time taken by the algorithm.

phase time was spent for partitioning the polygon.

Despite a slight raise of APX with n, the collected data does not evidence a clear relationship between the size of the instance $(\mathcal{U}, \mathcal{F})$ and the approximation ratio (Figure 4.8).

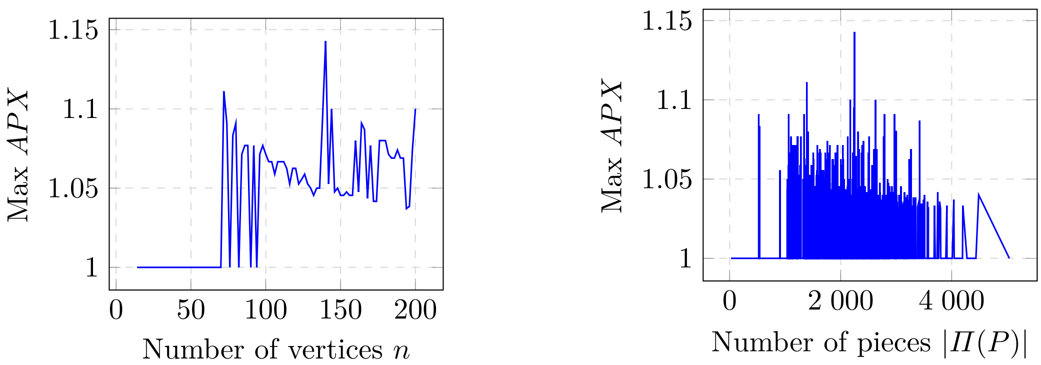


Figure 4.8: The variation of the maximum value hit by APX when collectively considering all the Random instances with a fixed n **(left)** or $|\Pi(P)|$ **(right)**. There is no clear relationship between these parameters.

4.2.4 Large-Random

A fourth session of tests was conducted for evaluating the scalability of Randomized-Set-Cover. We used progressively larger instances of randomly generated orthogonal polygons, with $n = 250$ to 1000 vertices, which we call Large-Random. From the original AGP2008a dataset, the first 5 instances provided for each $n \in \{250, 300, 350, 400, 450, 500, 600, 700, 800, 900, 1000\}$ have been used in this session. See Table 4.4 and Figures 4.9 and 4.10 for the main results.

Parameter	Min.	Median	Mean	Max.
$\|\mathcal{F}\| = n$	250	500	568.2	1000
$\|\mathcal{U}\| = \|\Pi(P)\|$	2385	6671	7329	15481
OPT	37	81	93.82	171
APX	1	1.014	1.016	1.054
H_s	6.445	7.228	7.280	8.557
f	18	25	24.96	34
Partition time (s)	4.739	32.759	58.342	206.674
Instance generation time (s)	0.006147	0.027755	0.034431	0.114595
Total preprocessing time (s)	4.745	32.784	58.376	206.789
Solving time (s)	0.1003	0.7821	1.2679	4.5575
Total time (s)	4.887	33.566	59.644	210.669

Table 4.4: Statistics for LARGE-RANDOM grid n-ogons.

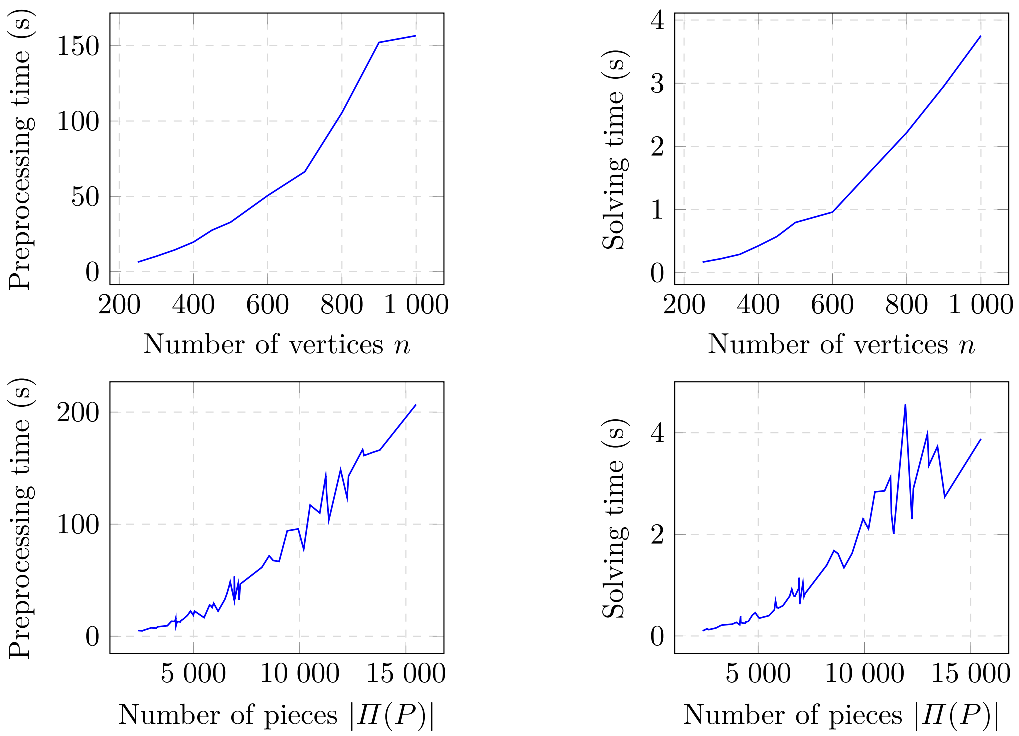


Figure 4.9: Growth of preprocessing and solving times for LARGE-RANDOM polygons. Once again, the time it takes to solve a SET COVER instance remains moderately low with n and $|\Pi(P)|$, although the partition time explodes.

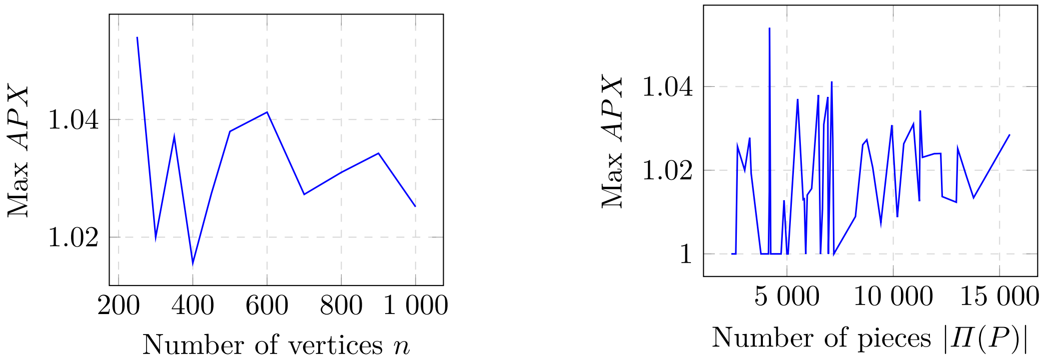


Figure 4.10: The variation of the maximum value hit by APX when collectively considering all the LARGE-RANDOM instances with a fixed n **(left)** or $|\Pi(P)|$ **(right)**. As observed in RANDOMs, no evident relationship exists between the parameters.

4.2.5 Discussion and general remarks

According to our tests, RANDOMIZED-SET-COVER demonstrated a good performance overall. It found a minimum-cardinality vertex-guard set for every FAT and MIN-AREA polygons, making it clear that kernelization has a remarkable effect when performed repeatedly in alternation with random choices. The approximation ratio was at most 1.143 for a RANDOM polygon, corresponding to an excess of only 3 guards from an optimal solution, and it remained low on larger instances (LARGE-RANDOM). We used a constant number of trials throughout the entire experiment; by increasing this number, the error rate could be improved in exchange for higher running times. We also believe that the geometric provenance of SET COVER instances imposes on them a very symmetric structure that gives the algorithm an extra advantage.

Our experiments assert that the time spent solving SET COVER instances was low. However, polygon partitioning constitutes a major burden for RANDOMIZED-SET-COVER, especially when the resulting number of pieces is large, as observed with FATs and LARGE-RANDOMs. Preprocessing, which overwhelmingly dominates the obtained running times, could likely be improved by following some ideas of the literature about lazy decomposition. Usually they involve starting with a grosser decomposition of the polygon and lazily postponing the calculation of the full partition until the algorithm really needs it. For example, several strategies are studied in [35] for efficiently building partial discretizations of orthogonal polygons which are computed on demand, as an alternative to performing an entire partition to get $\Pi(P)$ right from the start. In [93, 94] an anytime algorithm is proposed to compute successively better approximations of a vertex-guard set, which performs a series of refinements of the partition and exploits dominance of visibility regions. Finally, in [96] methods are devised for building initial discretizations, one of which (*Chwa-Points*) explores known results on *polygon witnessability* [31].

We used no specific heuristics to drive the selection of a random set $S \in$ CANDIDATES, every set being assigned the same weight. Alternatives to uniform weighting in the sampleable set could perhaps have a positive impact on the reduction of the approximation factor of the algorithm for RANDOM instances. We believe it would be worthwhile to devise more refined heuristics, such as dynamic re-weighting, that could take into account structural properties of the polygon and exploit local dominance relations between vertices and pieces to guide the selection of the most promising candidate set to remove. Some of our preliminary ideas involve assigning greater weights (that is, probabilities of being chosen) to smaller sets $S \in$ CANDIDATES or to sets that cover *fewer* low-frequency elements.

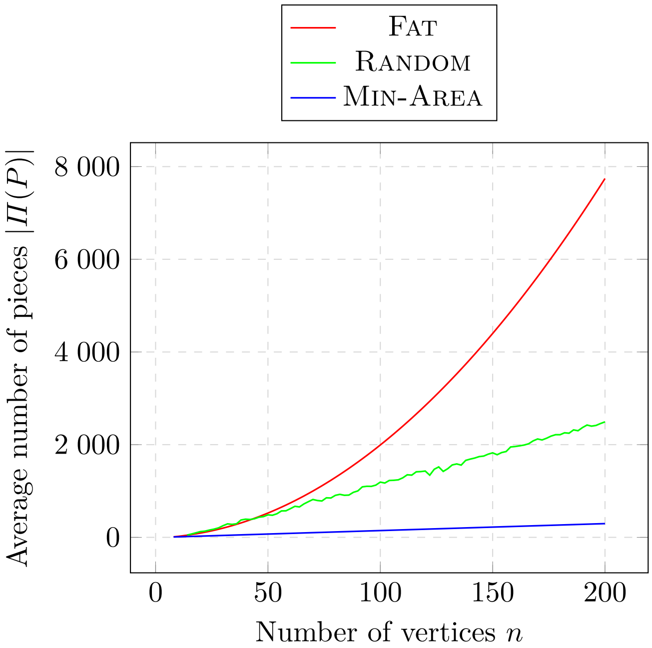


Figure 4.11: Comparison of the average number of pieces in $\Pi(P)$ for each grid n-ogon class, as the number of vertices n increases from 8 to 200.

Figure 4.11 confirms once more that FAT polygons indeed have the maximum number of pieces for a fixed n among all the polygons that make up the dataset, while MIN-AREAs have the minimum and RANDOMs lie approximately in between. Although the number of pieces of a FAT exhibits a quadratic behavior, curiously enough it seems that the size of the partition of a RANDOM polygon only grows linearly, just like MIN-AREAs. In the next section we look into this behaviour and formulate a conjecture that attempts to exactly characterize these growths.

We conclude with a remark that, although this work focused on orthogonal polygons under straight-line visibility, the ideas of RANDOMIZED-SET-COVER naturally extend to any generic polygon, under any visibility notion that one wishes to adopt.

4.3 Order of growth of $|\Pi_n|$ for grid n-ogons

In this section, we derive a proposition and a conjecture regarding MIN-AREAs and FATs with n vertices, based on the results obtained from our experiments of Section 4.2. These refer to closed formulas that relate the number of vertices, n, with the number of pieces in the convex partition of these polygons, which we denote by $|\Pi_n|$. We remark that $|\Pi_n|$ refers to $|\Pi(P)|$ for a polygon P with n vertices, not to the number of r-pieces in its grid partition, $|\Pi_{HV}(P)|$ (for which closed formulas were already known, Theorem 4.2). We explain how the formulas were guessed and gather evidence that supports them.

4.3.1 Min-Area

Proposition 4.3. *Let $|\Pi_n|$ be the number of pieces in the partition of a* MIN-AREA *with $n \geq 6$ vertices induced by vertex visibility regions. Then, $|\Pi_n| = \frac{3n}{2} - 4$.*

The proposition can be deduced from studying how the contribution of the number of squares and triangles to the total partition size evolves with the number of vertices n (Figure 4.12), and is validated by the plot in Figure 4.13.

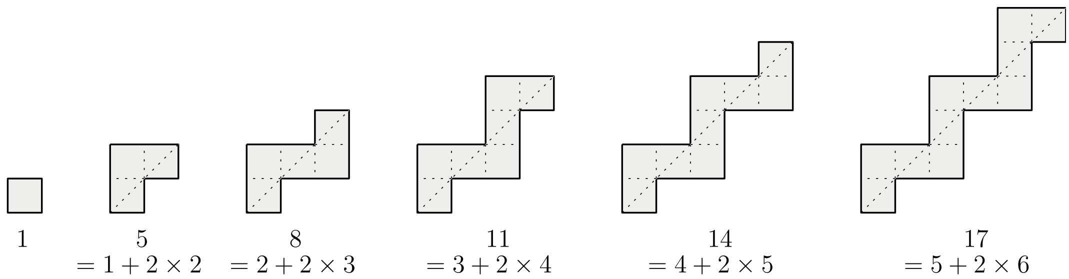


Figure 4.12: From a MIN-AREA with n vertices to a MIN-AREA with $n+2$ vertices, the number of visibility regions increases by a square and two triangles. The case where $n = 4$ does not satisfy the property because the respective MIN-AREA is convex.

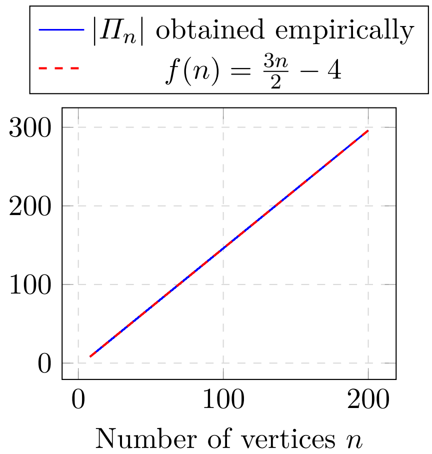


Figure 4.13: The number of pieces in the partition $\Pi(P)$, induced by vertex visibility regions, of a MIN-AREA P with n vertices perfectly obeys the identity $|\Pi(P)| = \frac{3n}{2} - 4$ for every $n = 8, 10, \ldots, 200$.

4.3.2 Fat

Conjecture 4.4. *Let $|\Pi_n|$ be the number of pieces in the partition of a* FAT *with $n \geq 16$ vertices induced by vertex visibility regions. Then, $|\Pi_n| = 76 + \frac{17(n-18)}{2} + \left\lfloor \frac{(n-18)^2}{16} \right\rfloor + 2\left\lfloor \frac{(n-20)^2}{16} \right\rfloor$.*

We now explain how this formula was deduced. Observe the sequence $|\Pi_n|$ for $n \geq 8$ and consider differences between consecutive terms. We recall that $|\Pi_{HV}(P)| \in \Theta(n^2)$ for a FAT grid n-ogon P. Since the partition $\Pi(P)$ is finer than $\Pi_{HV}(P)$, we find the class $\Omega(n^2)$ to suit $|\Pi_n|$. Therefore, let us analyze differences between *consecutive differences* of terms in the sequence:

$$
\begin{array}{lll}
|\Pi_8| = 11 & & \\
|\Pi_{10}| = 21 & |\Pi_{10}| - |\Pi_8| = 10 & \\
|\Pi_{12}| = 33 & |\Pi_{12}| - |\Pi_{10}| = 12 & 12 - 10 = 2 \\
|\Pi_{14}| = 46 & |\Pi_{14}| - |\Pi_{12}| = 13 & 13 - 12 = 1 \\
|\Pi_{16}| = 61 & |\Pi_{16}| - |\Pi_{14}| = 15 & 15 - 13 = 2 \\
|\Pi_{18}| = 76 & |\Pi_{18}| - |\Pi_{16}| = 15 & 15 - 15 = \mathbf{0} \\
|\Pi_{20}| = 93 & |\Pi_{20}| - |\Pi_{18}| = 17 & 17 - 15 = 2 \\
|\Pi_{22}| = 111 & |\Pi_{22}| - |\Pi_{20}| = 18 & 18 - 17 = 1 \\
|\Pi_{24}| = 131 & |\Pi_{24}| - |\Pi_{22}| = 20 & 20 - 18 = 2 \\
|\Pi_{26}| = 152 & |\Pi_{26}| - |\Pi_{24}| = 21 & 21 - 20 = 1 \\
|\Pi_{28}| = 175 & |\Pi_{28}| - |\Pi_{26}| = 23 & 23 - 21 = 2 \\
|\Pi_{30}| = 199 & |\Pi_{30}| - |\Pi_{28}| = 24 & 24 - 23 = 1 \\
\vdots & \vdots & \vdots
\end{array}
$$

We can observe an alternating pattern $2, 1, 2, 1, 2, \ldots$ which only seems to break at $n = 18$: $(|\Pi_{18}| - |\Pi_{16}|) - (|\Pi_{16}| - |\Pi_{14}|) = 15 - 15 = 0$. If we keep analyzing the sequence from $n = 20$, it appears that the pattern does not break anymore. Let us conjecture that this is true and write:

$$
\begin{array}{ll}
|\Pi_{20}| = 76 + 17 & = 76 + 1 \cdot 17 + 1 \cdot \mathbf{0} + 2 \cdot 0 \\
|\Pi_{22}| = |\Pi_{20}| + (17 + 1) & = 76 + 2 \cdot 17 + 1 \cdot \mathbf{1} + 2 \cdot 0 \\
|\Pi_{24}| = |\Pi_{22}| + (17 + 1 + 2) & = 76 + 3 \cdot 17 + 1 \cdot \mathbf{2} + 2 \cdot 1 \\
|\Pi_{26}| = |\Pi_{24}| + (17 + 1 + 2 + 1) & = 76 + 4 \cdot 17 + 1 \cdot \mathbf{4} + 2 \cdot 2 \\
|\Pi_{28}| = |\Pi_{26}| + (17 + 1 + 2 + 1 + 2) & = 76 + 5 \cdot 17 + 1 \cdot \mathbf{6} + 2 \cdot 4 \\
|\Pi_{30}| = |\Pi_{28}| + (17 + 1 + 2 + 1 + 2 + 1) & = 76 + 6 \cdot 17 + 1 \cdot \mathbf{9} + 2 \cdot 6 \\
|\Pi_{32}| = |\Pi_{30}| + (17 + 1 + 2 + 1 + 2 + 1 + 2) & = 76 + 7 \cdot 17 + 1 \cdot \mathbf{12} + 2 \cdot 9 \\
\vdots &
\end{array}
$$

The sequence of multiplicities $0, 1, 2, 4, 6, 9, 12, \ldots$ seems to be modeled by the general term $u_n = \lfloor \frac{n}{2} \rfloor \cdot \lceil \frac{n}{2} \rceil = \left\lfloor \frac{n^2}{4} \right\rfloor$, so we finally arrive at:

$$
\begin{aligned}
|\Pi_n| &= 76 + \frac{17(n-18)}{2} + \left\lfloor \frac{\left(\frac{n-18}{2}\right)^2}{4} \right\rfloor + 2 \left\lfloor \frac{\left(\frac{n-20}{2}\right)^2}{4} \right\rfloor \\
&= 76 + \frac{17(n-18)}{2} + \left\lfloor \frac{(n-18)^2}{16} \right\rfloor + 2 \left\lfloor \frac{(n-20)^2}{16} \right\rfloor
\end{aligned}
$$

By inspection, the equality still holds for $n = 16$ and $n = 18$ and appears to be false only for $n \in \{8, 10, 12, 14\}$. Conjecture 4.4 then follows and is corroborated by Figure 4.14.

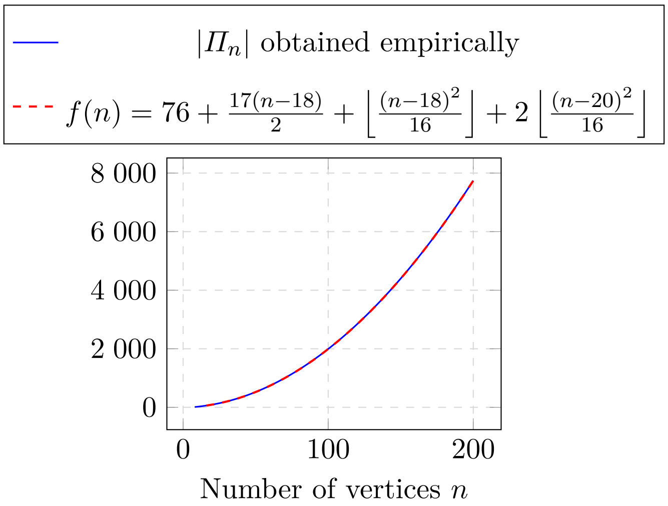


Figure 4.14: The number of pieces in the partition $\Pi(P)$, induced by vertex visibility regions, of a FAT P with n vertices perfectly obeys the identity $|\Pi(P)| = 76 + \frac{17(n-18)}{2} + \left\lfloor \frac{(n-18)^2}{16} \right\rfloor + 2 \left\lfloor \frac{(n-20)^2}{16} \right\rfloor$ for every $n = 16, 18, \ldots, 200$.

4.3.3 Random

Since RANDOM (and LARGE-RANDOM) polygons lead to very variable partitions, the measure $|\Pi_n|$ is not well-defined for them. Nonetheless, because $|\Pi(P)|$ visually appears to follow a linear trend for these polygon classes, we performed a linear regression by the least-squares method between n and $|\Pi(P)|$, averaged over all LARGE-RANDOM grid n-ogons P present in the dataset. Figure 4.15 confirms this almost linear growth of the average size of $\Pi(P)$ for a LARGE-RANDOM grid n-ogon P, with a Pearson correlation coefficient of 0.992, and signals that it can be approximated by the function $f(n) = 13.4n - 282.5$. The curve's behavior extrapolates for RANDOM polygons on the observed range $14 \leq n \leq 200$, for which the correlation is 0.999.

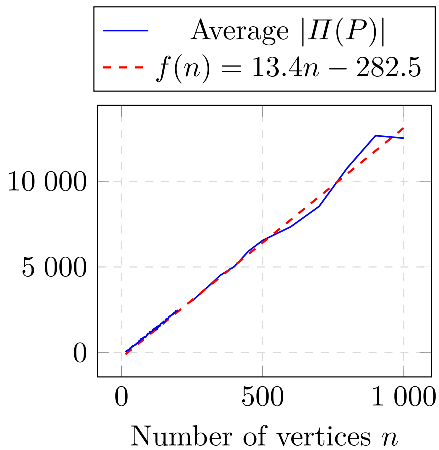


Figure 4.15: The average size of $\Pi(P)$ for a RANDOM grid n-ogon P seems to grow linearly with n. The part of the plot with $14 \leq n \leq 200$ integrates results from the RANDOM class, while for $250 \leq n \leq 1000$ the data used comes from LARGE-RANDOM. These data, coming from two independent sessions, exhibit a smooth continuity.

Chapter 5

On r-Guarding SCOTs

Although the Art Gallery Problem is hard for generic polygon instances, buildings and art galleries that we find in real life are not *random*: they show a very specific structure and a spatial organization that is characteristic of the human way of architecting. In this chapter, we define a new family of orthogonal polygons, the SCOTs, which prove to be useful models for mimicking properties that are close to those one can find in real-world rectangular galleries. We study the algorithmic complexity of guarding SCOTs under the r-visibility model – which we denominate Minimum SCOT r-Guard –, describe which of their subfamilies can be solved in polynomial time by providing algorithms for them and also prove hardness results for the remaining ones.

A SCOT is a connected orthogonal polygon that is made up of rectangular rooms linked by rectangular corridors. After all, that is how real buildings tend to be arranged. A corridor C links two rooms R_1 and R_2 in a SCOT if a side of C is strictly contained in an edge of R_1 and the opposite side is strictly contained in an edge of R_2. By “strictly contained” we mean that a corridor C is always narrower than the rooms R_1, R_2 that are incident to it and none of the four vertices of C coincide with either of the four corners of R_1 or R_2. A SCOT has no “dangling” corridors, so every corridor is incident to exactly two rooms. Each corridor has an implicit direction, *horizontal* or *vertical*, depending on whether it connects two rooms to its right/left or above/below it, respectively.

There may exist cycles that involve four or more rooms, in which case the SCOT is called *cyclic* (Figures 5.1b and 5.1c). The same pair of rooms may be linked by multiple, non-intersecting corridors (Figure 5.1c). If there exist *room cycles* or *multiple corridors*, the SCOT has holes; otherwise it is said to be *simple* (Figure 5.1a).

For convenience we denote a SCOT with set of rooms $\mathcal{R}$ and set of corridors $\mathcal{C}$ as $(\mathcal{R}, \mathcal{C})$-SCOT. The number of vertices in an $(\mathcal{R}, \mathcal{C})$-SCOT is equal to $n = 4|\mathcal{R}| + 4|\mathcal{C}|$, since each room and each corridor contribute with four new vertices to the total count. Because SCOTs are connected, $|\mathcal{C}| \geq |\mathcal{R}| - 1$ and $|\mathcal{R}| + |\mathcal{C}| \in \Theta(|\mathcal{C}|)$.

An obvious way of guarding a SCOT is placing an individual guard in each room and in each corridor, so we have a trivial upper bound of $|\mathcal{R}| + |\mathcal{C}| = n/4$ for the number of guards needed

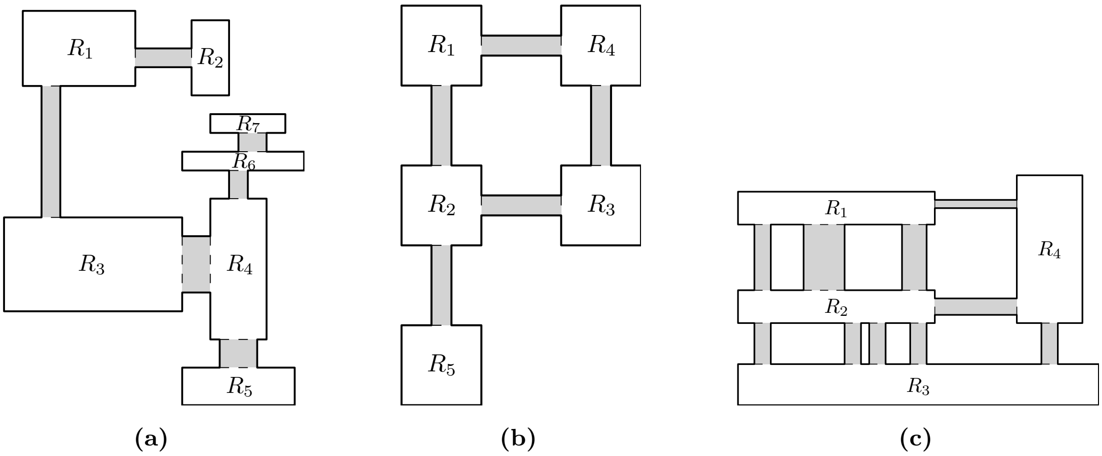


Figure 5.1: **(a)** Simple $(\mathcal{R},\mathcal{C})$-SCOT, with $|\mathcal{R}| = 7$ and $|\mathcal{C}| = 6$. Rooms are represented as white rectangles and corridors as shaded rectangles. The corridor connecting rooms R_1 and R_2 is *horizontal* and the corridor connecting rooms R_1 and R_3 is *vertical.* **(b)** SCOT with a cycle involving rooms R_1, R_2, R_3 and R_4. **(c)** SCOT with multiple corridors between the same sides of rooms R_1 and R_2 and also between R_2 and R_3.

under r-visibility (both for vertex-guards and for point-guards). This solution, however, may in general not be optimal. An immediate question one should pose is whether Minimum SCOT r-Guard can be solved in polynomial time at all. Theorem 5.1 establishes that this is true for *simple* SCOTs.

Theorem 5.1. *[102] A minimum-cardinality* r*-guard set for a simple orthogonal polygon with n vertices can be found in time $\widetilde{\mathcal{O}}\left(n^{17}\right)$, where $\widetilde{\mathcal{O}}(\cdot)$ hides a polylogarithmic factor.*

Despite the interest of this result, which sets Minimum SCOT r-Guard for simple SCOTs in P, $\widetilde{\mathcal{O}}\left(n^{17}\right)$ is not a very gracious running time; this is the penalty that we get for the algorithm to be applicable to a so broad polygon family. We are thus interested in exploiting the structure of SCOTs for developing more efficient algorithms that are specific for this class of instances, and understanding whether the problem involving SCOTs with holes remains polynomial.

A first observation towards a better algorithm is given by Lemma 5.2. It states that reflex vertices are always better choices to place guards at: for the purposes of finding a minimum-cardinality vertex-guard set for a SCOT, one could opt to ignore its convex vertices altogether without risking losing optimality.

Lemma 5.2. *Let P be an $(\mathcal{R},\mathcal{C})$-SCOT with $|\mathcal{R}| \geq 2$. For any convex vertex u of P, there is a reflex vertex v of P such that v* r*-sees more than u (that is, v* r*-sees a superset of what u* r*-sees).*

Proof. Since P has at least two rooms and is connected, every room has at least two reflex vertices. The only convex vertices in P are the four corners of each room and only r-see the room they belong to, given that corridors are narrower than incident rooms. A reflex vertex is shared between a room and a corridor; therefore, it r-sees that room and (at least) that corridor (possibly more, if there are multiple corridors along the same direction). □

Remark. *It is not true for orthogonal polygons in general that, under* r*-visibility, an optimal vertex-guard set exists consisting only of reflex vertices. For example, take the staircase polygon in Figure 5.2.*

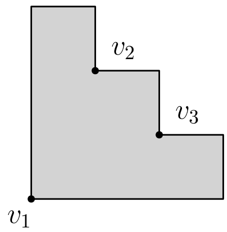


Figure 5.2: Convex vertices may be required in an optimal vertex-guard set for a general orthogonal polygon under r-visibility. For this polygon, $\{v_1\}$ is a minimum-cardinality guard set, but if we restrict ourselves to reflex vertices the only feasible solution is $\{v_2, v_3\}$, which is not optimal.

5.1 Simple SCOTs

In this section, we show that simple SCOTs can indeed be r-guarded in linear time by means of a greedy algorithm. The algorithm we present works for vertex-guards and for point-guards. It processes the SCOT as a tree graph induced by its rooms and corridors and detaches its leaves one by one, until a complete guard set has been determined.

Theorem 5.3. *Let P be a simple $(\mathcal{R}, \mathcal{C})$-SCOT. Exactly $|\mathcal{R}|$ point-guards (and, indeed, vertex-guards) are needed for* r*-guarding P.*

Proof. We prove that $|\mathcal{R}|$ point-guards (and, indeed, vertex-guards) are necessary and sufficient for covering P.

Necessity

For (fully) r-guarding some room R, it is mandatory that a guard is placed in the interior of R: even if one placed a guard on every corridor incident to R, but outside of R, the room's corners would not be covered (Figure 5.3). As we need a guard per room, we cannot use less than $|\mathcal{R}|$ guards in total.

Sufficiency

We provide a greedy algorithm as a constructive proof that $|\mathcal{R}|$ guards are sufficient. Figure 5.4 provides a run of this algorithm for the simple SCOT of Figure 5.1a. Build an undirected graph $T = (V, E)$, with $|V| = |\mathcal{R}|$ and $|E| = |\mathcal{C}|$. Each node in V represents a room of P and there is an edge in E connecting two nodes if the corresponding rooms in P are connected by a corridor. As P has no holes, T is a tree and has at least one leaf (a node with degree 0 or 1). While T is not empty, select an arbitrary leaf u corresponding to a room R. If u has degree 0, place a guard anywhere in R (namely, at one of the corners), remove u from T and terminate. Otherwise (if u has degree 1), place the guard at one of the reflex vertices that are shared between R and one of its corridors C and remove both u and the incident edge corresponding to C. The tree T will become empty (i.e., P will become r-guarded) after exactly $|\mathcal{R}|$ steps and in each step we have placed a vertex-guard, so $|\mathcal{R}|$ guards are enough.

□

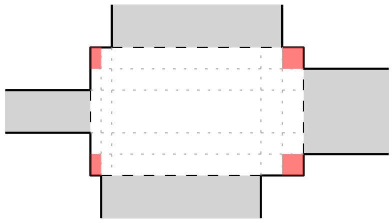


Figure 5.3: Blind spots on the corners of a room. These regions cannot be r-seen from outside the room.

5.2 r-independent SCOTs

In this section, we define a subfamily of SCOTs, the r-independent SCOTs, and prove that they can also be guarded in polynomial time, though by means of conceptually different algorithms regarding vertex-guards or point-guards.

Definition 5.1 (Corridor stretch)**.** Let C be a corridor in a SCOT P. We define the *stretch* of C as the infinite extension of C along its direction (horizontal or vertical) (Figure 5.5). That is: if C is horizontal, the stretch of C is a horizontally unbounded region in the plane that is bounded above and below by the lines that contain the top and bottom edges of C, respectively; if C is vertical, its stretch is a vertically unbounded region in the plane that is bounded on the left and right sides by the lines that contain the left- and right-side edges of C.

Definition 5.2 (Disjoint corridors and aligned corridors)**.** Let C_1 and C_2 be two corridors that are incident to the same room R in a SCOT P and have the same direction (either both horizontal or both vertical). We say that C_1 and C_2 are *disjoint* if the stretch of C_1 does not intersect the stretch of C_2 and that C_1 and C_2 are *aligned* if the stretch of C_1 exactly coincides with the stretch of C_2 (Figure 5.5b).

Definition 5.3 (r-independent SCOT)**.** An r-independent SCOT is a SCOT in which every pair of adjacent corridors with the same direction is either disjoint or aligned (Figure 5.6a). Note that P may have cycles over rooms and multiple corridors between the same pair of rooms.

5.2.1 Super-corridors

Let P be an r-independent $(\mathcal{R},\mathcal{C})$-SCOT. Notice that, for every pair (C_1, C_2) of adjacent, aligned corridors along the same direction, C_1 is r-seen if and only if C_2 is r-seen. This induces an

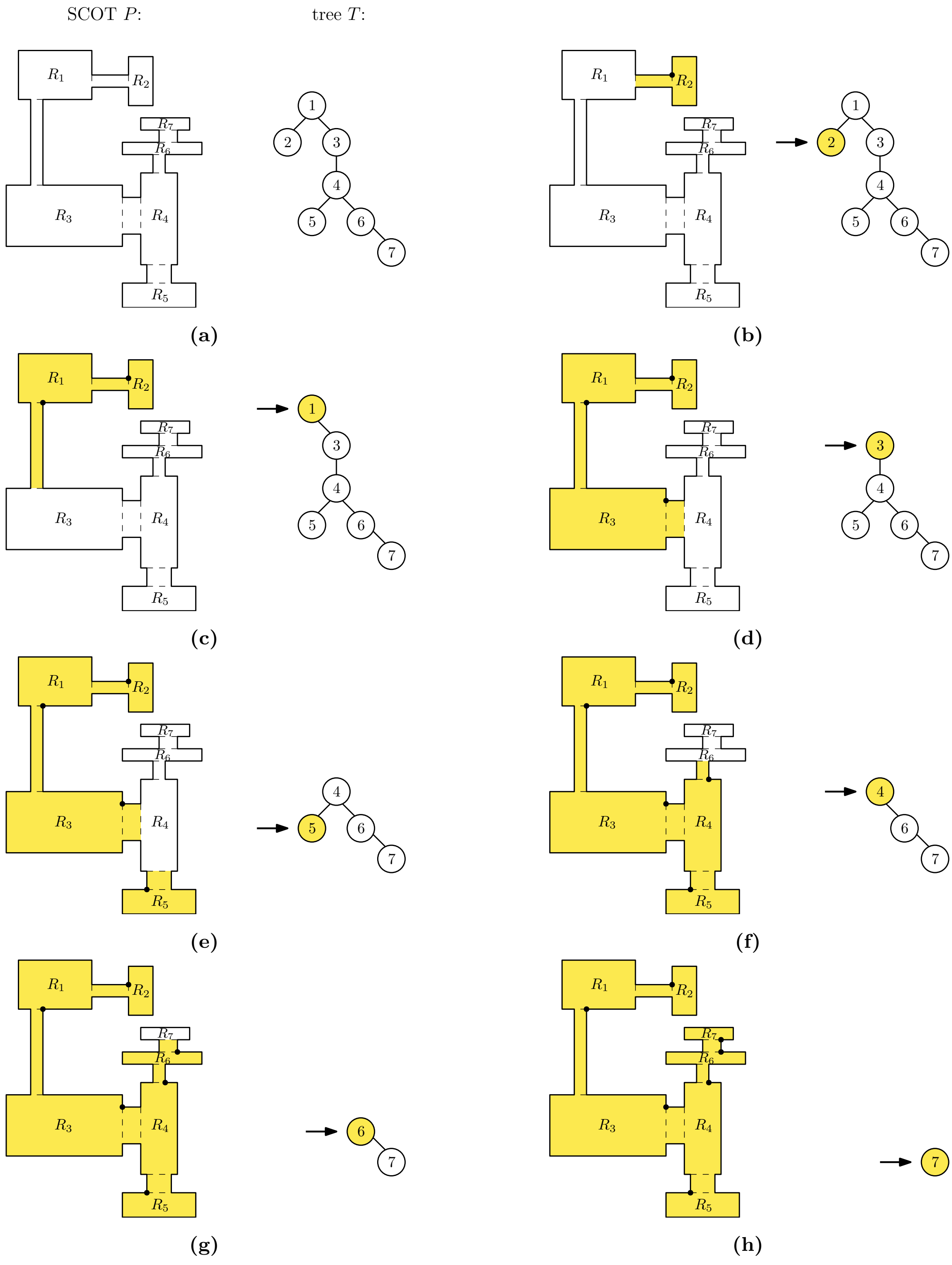


Figure 5.4: Step-by-step execution of the greedy algorithm for determining a minimum-cardinality vertex-guard set in a simple SCOT P (with no room cycles and no multiple corridors). **(a)** Initially, a tree T representing room adjacencies in P is constructed. **(b-g)** We keep detaching leaf nodes from T until a single node remains. We break the ties arbitrarily; in this example, leaves are removed in increasing order of the node identifier. In each intermediate step, a guard is placed on a reflex vertex to r-see exactly one room and one corridor. **(h)** When the last leaf node remaining is removed, the entire polygon P becomes covered. The number of guards used is $|\mathcal{R}| = 7$.

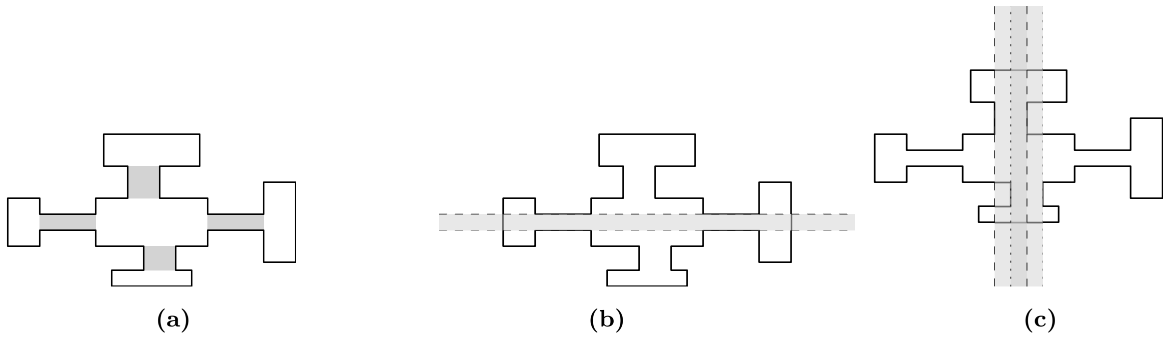


Figure 5.5: **(a)** Simple SCOT P. **(b)** The stretches of the two horizontal corridors of P perfectly coincide, so they are aligned corridors. **(c)** The stretches of the two vertical corridors of P intersect, but do not coincide, so they are not aligned nor disjoint.

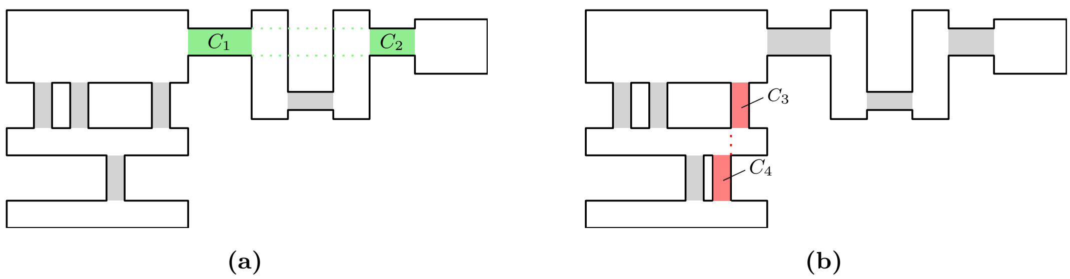


Figure 5.6: **(a)** r-independent SCOT. Notice that, even though the stretches of corridors C_1 and C_2 coincide, C_1 and C_2 are not adjacent. **(b)** A SCOT that is not r-independent. The stretches of corridors C_3 and C_4 intersect in a vertical line. A guard placed at the top-right corner of C_4 would r-see both corridors.

equivalence relation between corridors (r-equivalence), according to which two adjacent corridors are equivalent if and only if they are aligned. By transitivity, two nonadjacent corridors that belong to the same succession of adjacent, aligned corridors along the same direction are also r-equivalent (such as corridors C_2 and C_4 of the r-independent SCOT in Figure 5.7).

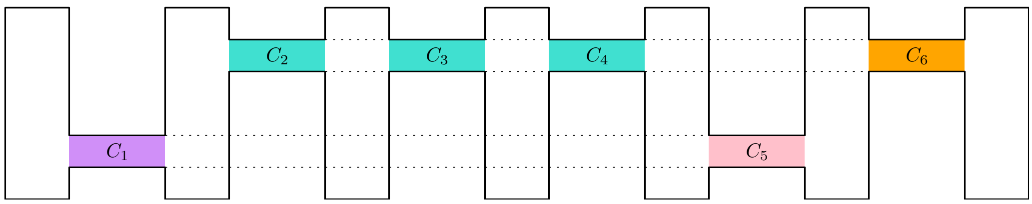


Figure 5.7: There are four super-corridors (r-equivalence classes of corridors) in this polygon: $\{C_1\}$, $\{C_2, C_3, C_4\}$, $\{C_5\}$ and $\{C_6\}$. Corridors C_1 and C_5 are not r-equivalent because they are not adjacent and do not belong to a succession of adjacent, aligned corridors. The same happens with corridors C_4 and C_6.

Compression of aligned corridors into super-corridors We now describe a general strategy that we will employ several times for dealing with r-independent SCOTs. Replace each maximal succession of adjacent corridors $C^{(1)}, C^{(2)}, \ldots$ that are aligned along the same direction by a *super-corridor* that may cross multiple rooms. In other words, conceptually represent all of these r-equivalent corridors by a unique, long corridor that acts as their equivalence class, $C' = \{C^{(1)}, C^{(2)}, \dots\}$. After this transformation, the problem has been reduced to the case where all pairs of adjacent (super-)corridors are disjoint. We denote by $\mathcal{C}'$ the set of all super-corridors in P, that is, $|\mathcal{C}'|$ is equal to the number of equivalence classes of corridors. In the specific case where all corridors in P were already pairwise disjoint, $|\mathcal{C}| = |\mathcal{C}'|$ and $|C'| = 1$ for every $C' \in \mathcal{C}'$. We say that a room R is incident to a super-corridor C' if and only if R is incident to some corridor C that belongs to C'.

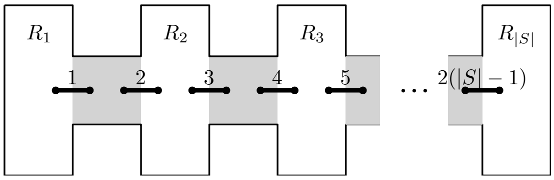
Because the decomposition into super-corridors is so important for developing algorithms specific to r-independent SCOTs, we will often denote by $(\mathcal{R}, \mathcal{C}, \mathcal{C}')$-SCOT an r-independent $(\mathcal{R}, \mathcal{C})$-SCOT with a set of super-corridors $\mathcal{C}'$.

Lemma 5.4. *Let P be an* r*-independent $(\mathcal{R}, \mathcal{C}, \mathcal{C}')$-SCOT. The number of pairs of rooms and super-corridors (R, C') such that $R \in \mathcal{R}$ and $C' \in \mathcal{C}'$ are incident is bounded above by $2|\mathcal{C}|$.*

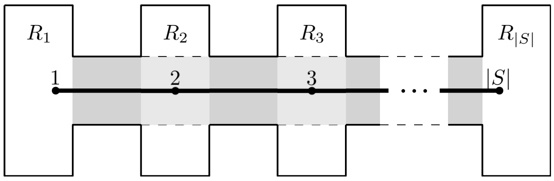
Proof. Let $Q = \{(R, C') : R \in \mathcal{R},\ C' \in \mathcal{C}',\ R \text{ is incident to } C'\}$. If all pairs of adjacent corridors in P are disjoint, the bound $|Q| \leq 2|\mathcal{C}|$ clearly holds: in this case, the amount of pairs in Q is equal to the number of pairs of rooms and corridors (R, C) such that R and C are incident to each other in P. Given that each corridor, connecting two rooms in P, introduces two new pairs in Q (between the corridor and both rooms), this leads to $|Q| = 2|\mathcal{C}|$.

However, the existence of r-equivalent corridors affects the cardinality of Q. Consider a subset S of rooms in P and a maximal succession of $|S| - 1$ aligned corridors connecting these rooms (Figure 5.8a). If these corridors were not compressed, they would contribute in total with $2(|S| - 1)$ pairs to the set Q. But compressing them into a single super-corridor contributes with $|S|$ pairs to Q: each one linking the super-corridor to a different room in S (Figure 5.8b). Since the inequality $x \leq 2(x - 1)$ holds for all $x \geq 2$ and indeed $|S| \geq 2$ (otherwise, there would be no corridor to consider at all), Q cannot exceed the size it would have without the transformation; in particular, $|Q|$ cannot be greater than $2|\mathcal{C}|$. $\square$

Compressing the corridors of an r-independent SCOT into super-corridors can be done in $\mathcal{O}(|\mathcal{C}| \log |\mathcal{C}|)$ time. For that, compute, for each room R, a list of all horizontal corridors incident to R sorted by the y-coordinate of their top edges. Pick any horizontal corridor C and launch a depth-first search from it. Transition from a corridor to an adjacent one if it is also horizontal and both share the same top edge y-coordinate. Valid neighbours can be binary searched in $\mathcal{O}(\log |\mathcal{C}|)$ time in the lists of the incident rooms. As soon as the current search terminates, add to $\mathcal{C}'$ a new super-corridor formed by all the corridors that have been visited during this search. Repeat while there are unvisited horizontal corridors. Do the same for vertical corridors, but now basing the search on the x-coordinate of their left edges.



(a)



(b)

Figure 5.8: (a) If r-equivalent corridors were not compressed, a succession of $|S|-1$ corridors connecting a subset of rooms S would contribute with $2(|S|-1)$ pairs to $|Q|$, because each corridor is incident to exactly two rooms. **(b)** By compressing this maximal succession of aligned corridors into a super-corridor, Q now gets $|S|$ pairs instead: one for each room that is incident to the super-corridor.

5.2.2 3-approximation algorithm for vertex-guards and point-guards

In the following Sections 5.2.3 and 5.2.4 we provide exact, polynomial-time algorithms for finding minimum-cardinality vertex-guard sets and point-guard sets for an r-independent SCOT. However, before describing these, we draw the attention to the fact that the trivial algorithm that places a guard per room and per super-corridor already gives us a linear-time 3-approximation algorithm for r-independent SCOTs, using either vertex-guards or point-guards, if we assume that corridors have already been compressed to super-corridors. We formalize this in Theorem 5.5.

Theorem 5.5. *Let P be an* r*-independent $(\mathcal{R}, \mathcal{C}, \mathcal{C}')$-SCOT. A minimum-cardinality point-guard set (and, indeed, vertex-guard set) for P can be approximated with factor* 3 *in time linear in* $|\mathcal{R}| + |\mathcal{C}'|$.

Proof. Let $\mathcal{C}'_h \subseteq \mathcal{C}'$ and $\mathcal{C}'_v \subseteq \mathcal{C}'$ be the sets of horizontal and vertical super-corridors in P, respectively. Consider the guard set $\mathcal{G}$ obtained by, for each room of P, placing a guard at one of its vertices and, for each super-corridor, placing as well a guard at one of its vertices. $\mathcal{G}$ is a vertex-guard set, so it is also a point-guard set, and has size at most $|\mathcal{R}| + |\mathcal{C}'| = |\mathcal{R}| + |\mathcal{C}'_h| + |\mathcal{C}'_v|$. Since at least $\max\{|\mathcal{R}|, |\mathcal{C}'_h|, |\mathcal{C}'_v|\}$ guards are required for guarding P, the approximation factor $\frac{|\mathcal{G}|}{OPT(P)}$ is at most

$$\frac{|\mathcal{R}| + |\mathcal{C}'_h| + |\mathcal{C}'_v|}{\max\{|\mathcal{R}|, |\mathcal{C}'_h|, |\mathcal{C}'_v|\}} \leq \frac{3\max\{|\mathcal{R}|, |\mathcal{C}'_h|, |\mathcal{C}'_v|\}}{\max\{|\mathcal{R}|, |\mathcal{C}'_h|, |\mathcal{C}'_v|\}} = 3.$$

□

Another linear-time 3-approximation algorithm for r-guarding general orthogonal polygons was already known [75], but, contrary to our result in Theorem 5.5, it is only applicable for covering simple polygons with point-guards.

5.2.3 Exact algorithm for vertex-guards

We now explain how an optimal vertex-guard set can be found for an r-independent $(\mathcal{R}, \mathcal{C}, \mathcal{C}')$-SCOT in polynomial time. Once more, we need one guard per room, otherwise the corners of some room will not be watched over. Notice that a vertex-guard can r-see at most one room and one super-corridor of P. It is intuitive that, when placing a guard g in a room R, it is best to try to also cover one of the incident super-corridors C', by placing g at one of the reflex vertices shared by R and C', so as to avoid wasting extra guards for C' later. Informally, we would like to maximize the number of guards that see two *objects* in P at once, so that the remaining guards that, individually, only see one *object* are minimum.

We thus consider two phases for the positioning of the guards in P: first, we place one guard in every room, also covering exactly one of the (not yet covered) incident super-corridors in each; then, and only after that, we spend a guard for separately watching over each of the remaining uncovered rooms and/or super-corridors. Of course, the first phase cannot be done in an arbitrary way, because a non-optimal placement may lead to more guards being required in the second phase.

The strategy we follow to ensure that this placement is done optimally involves finding a maximum matching between rooms and incident super-corridors. For that, we build a bipartite graph $H = (V, E)$ with node parts A and B. The set A contains $|\mathcal{R}|$ nodes corresponding to rooms in P and B contains $|\mathcal{C}'|$ nodes corresponding to super-corridors. We call a node in A a *room node* and a node in B a *super-corridor node*, and we add an edge between a room node and a super-corridor node in H if and only if their corresponding room and super-corridor are incident in P. Note that a separate edge should be added for each room that the super-corridor crosses (a super-corridor crosses a room if at least one of the corridors it contains is incident to that room).

A concrete selection of pairs of rooms and super-corridors that may be chosen to be r-seen collectively with a single guard can be modeled as a matching in graph H; in particular, by Lemma 5.6, an optimal selection of guards corresponds to a maximum cardinality matching (Figure 5.9). Let $M^\star$ be any maximum matching in H and let $A' \subseteq A$ and $B' \subseteq B$ be the sets of room nodes and super-corridor nodes in G that are covered up by $M^\star$. Since, by definition, no two edges in a matching are incident to the same node, we have that $|A'| = |B'| = |M^\star|$. The proof of Lemma 5.6 explains how we can extract a vertex-guard set for P from the obtained matching $M^\star$.

Lemma 5.6. *The graph H has a matching M of size $k \in \mathbb{Z}_0^+$ if and only if the SCOT P has a vertex-guard set $\mathcal{G}$ of size $|\mathcal{R}| + |\mathcal{C}'| - k$.*

Proof.

($\Rightarrow$) Let M be a matching in H of cardinality k and $\mathcal{G}$ be an empty vertex-guard set. For each edge (u, v) in M, add to $\mathcal{G}$ a guard at a reflex vertex that is shared between the room R

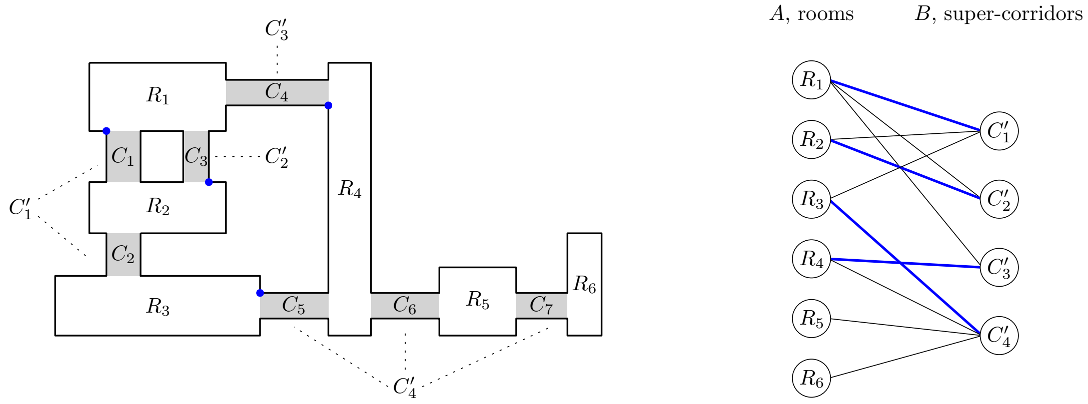
corresponding to u and a corridor that the super-corridor C' corresponding to v represents; there are either two or four possibilities for choosing that vertex (depending on whether C' contains one or two corridors incident to R, respectively), but it can in fact be picked arbitrarily, because all these shared vertices r-see the exact same region in P (Figure 5.10). Finally, place a guard in each of the rooms and/or super-corridors that may eventually still be left unguarded. $\mathcal{G}$ is feasible and has size $|M|+(|A|-|M|)+(|B|-|M|) = |\mathcal{R}|+|\mathcal{C}'|-k$.

($\Leftarrow$) Let $\mathcal{G}$ be a vertex-guard set for P, with $|\mathcal{G}| = |\mathcal{R}| + |\mathcal{C}'| - k$, and let $\mathcal{G}'$ be a subset of $\mathcal{G}$ with $|\mathcal{G}'| = |\mathcal{C}'|$ that covers all the super-corridors. $\mathcal{G}'$ exists because each super-corridor requires a different vertex-guard. By hypothesis, $\mathcal{G}'$ leaves at most $|\mathcal{R}| - k$ rooms uncovered and, thus, covers at least k different rooms. As we have placed each guard in a different super-corridor, there is a matching of size k between room and super-corridor nodes in H.

□

$M^\star$ may, for instance, be found using Hopcroft-Karp's algorithm [59], which runs in time $\mathcal{O}\left(\sqrt{|V|}|E|\right)$ – indeed $\mathcal{O}\left(\sqrt{\min\{|A|,|B|\}}|E|\right)$. In this case, we have $\min\{|A|,|B|\} = \min\{|\mathcal{R}|,|\mathcal{C}'|\}$ and, by Lemma 5.4, $|E| \leq 2|\mathcal{C}|$. Theorem 5.7 then follows.

Theorem 5.7. *Let P be an* r*-independent $(\mathcal{R},\mathcal{C},\mathcal{C}')$-SCOT. A minimum-cardinality vertex-guard set for P can be determined in time $\mathcal{O}\left(|\mathcal{C}|\sqrt{\min\{|\mathcal{R}|,|\mathcal{C}'|\}}\right)$.*



(a) **(b)**

Figure 5.9: **(a)** r-independent SCOT P, with four super-corridors: $C'_1 = \{C_1, C_2\}$, $C'_2 = \{C_3\}$, $C'_3 = \{C_4\}$ and $C'_4 = \{C_5, C_6, C_7\}$. A partial guard set watching over everything in P except for rooms R_5 and R_6 is presented. Each guard covers exactly one room and one super-corridor. **(b)** Bipartite graph H, whose subset of nodes A represents rooms in P and whose subset B represents super-corridors. The edges contained in a B-perfect matching $M^\star$, corresponding to the partial guard set given in **(a)**, are highlighted. The guard set could be extended with two guards, one in R_5 and another in R_6, to optimally cover the entirety of P.

Remark. *One could guess that exactly* $\max(|\mathcal{R}|,|\mathcal{C}'|)$ *vertex-guards are needed for* r*-guarding any* r*-independent $(\mathcal{R},\mathcal{C},\mathcal{C}')$-SCOT P. That is, it would be tempting to think that a A-perfect matching and/or a B-perfect matching in H always exists and that exactly* $\min(|\mathcal{R}|,|\mathcal{C}'|)$ *guards*

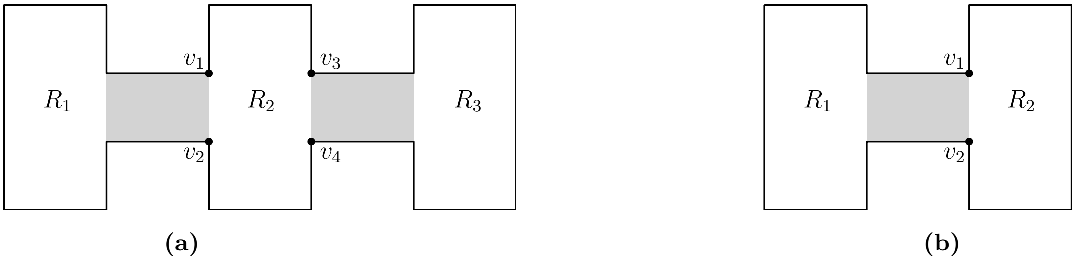


Figure 5.10: **(a)** A vertex-guard that is chosen to r-see both the room R_2 and the super-corridor that crosses all three rooms can be placed in any of the four vertices v_1, v_2, v_3 or v_4, because they are r-equivalent. **(b)** If the matching $M^\star$ determines that a vertex-guard has to be placed seeing both the room R_2 and the super-corridor, there are two possibilities for the guard: either at v_1 or at v_2.

could be placed on the first phase, covering either all the rooms or all the super-corridors in P (or both). However, this is not necessarily true and a counterexample of a SCOT P whose corresponding bipartite graph H admits no perfect matching is presented in Figure 5.11a. We prove it by invoking Hall's theorem, which asserts that a bipartite graph $G = (A \cup B, E)$ admits an A-perfect matching if and only if every subset of nodes $X \subseteq A$ has at least as many neighbours in B:

Theorem 5.8 (Hall's marriage theorem)**.** *[56] Let $G = (A \cup B, E)$ be a bipartite graph and, for some subset $X \subseteq A$, let $N_G(X)$ denote the* neighbourhood *of X, that is, the set of all the nodes in B that are adjacent to some element of X. Then, G admits an A-perfect matching if and only if, for any subset $X \subseteq A$, $|X| \leq |N_G(X)|$.*

We can now argue that the matching in graph H represented in Figure 5.11b is of maximum cardinality. For the set $X = \{R_4, R_6\}$ we have $N_H(X) = \{C'_5\}$ and thus $|X| > |N_H(X)|$, which then implies that H admits no perfect matching. Since $|M^\star| = |A| - 1 = |B| - 1$, $M^\star$ is maximum. Thus, the smallest vertex-guard set for P has $|M^\star| + 2 = 7 > \max\{|\mathcal{R}|, |\mathcal{C}'|\}$ guards.

5.2.4 Exact algorithm for point-guards

The proof of Theorem 5.7 shows how we can solve the problem with vertex-guards in time that is polynomial in the instance size. It would be interesting to know if the same algorithm can also be applied to optimally solve the version with point-guards. After some careful thinking, it becomes clear that it cannot. The reason for this is that a point-guard placed in a room R which is incident to a horizontal and a vertical super-corridor, respectively C'_h and C'_v, can actually r-see both super-corridors if it is placed in the intersection of the room and the stretches of C'_h and C'_v. Therefore, the matching would not account for the possibility of the same guard to cover three different objects in P at once. As we have seen, this possibility does not extend to vertex-guards, which are bound to only r-see at most one super-corridor.

Nevertheless, the problem of watching over r-independent SCOTs with point-guards is still polynomial-time solvable, although by a conceptually different algorithm – which we present

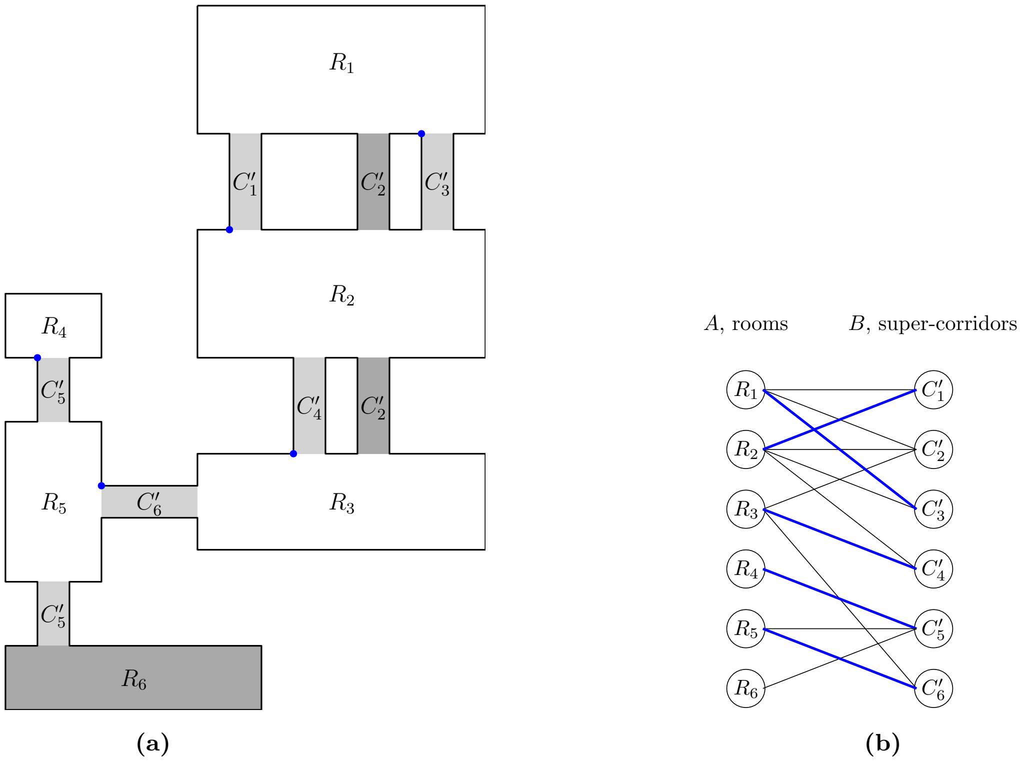


Figure 5.11: **(a)** r-independent SCOT P, showing a partial vertex-guard set covering every region except for room R_6 and super-corridor C'_2. It is a counterexample for the conjecture that $\max\{|\mathcal{R}|, |\mathcal{C}'|\}$ vertex-guards are sufficient for watching over any r-independent $(\mathcal{R}, \mathcal{C}, \mathcal{C}')$-SCOT P. **(b)** Bipartite graph $H = (A \cup B, E)$, along with a maximum cardinality matching $M^\star$, representing the partial guard placement in P. H admits no perfect matching and therefore two extra guards are needed for covering R_6 and C'_2.

next, along with some useful lemmas.

Lemma 5.9. *For any* r*-independent SCOT P, there is a minimum-cardinality point-guard set for P in which every guard lies inside a room.*

Proof. Suppose P admits a minimum point-guard set $\mathcal{G}$ with at least one guard, g, having been placed strictly inside a corridor C. What g (completely) r-sees is exactly the super-corridor corresponding to C and nothing else. Shift g along the direction of C until it becomes strictly contained in a room (Figure 5.12), call g' its new position and let $\mathcal{G}' = (\mathcal{G} \setminus \{g\}) \cup \{g'\}$ be the new guard set, which satisfies $|\mathcal{G}'| = |\mathcal{G}|$. Now g' still sees the super-corridor corresponding to C, but it does also see a room. Therefore, the new position for the guard g' has not made $\mathcal{G}'$ worse than $\mathcal{G}$ in terms of covered area. Repeat the same argument while there are guards strictly outside rooms in $\mathcal{G}'$. ☐

Lemma 5.9 says that, if we only consider solutions with guards placed in rooms, we do not risk losing optimality. This observation proves useful for developing an algorithm that finds a minimum-cardinality point-guard set by placing each guard strategically inside a room.

For that, we transform the problem into one of network flows, called Minimum Flow with

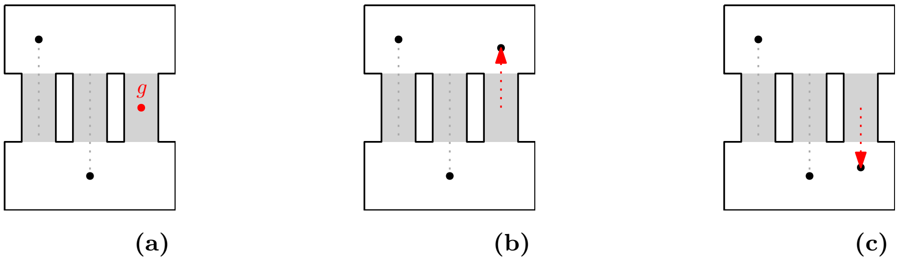


Figure 5.12: **(a)** Guard g only r-sees a corridor. **(b-c)** By moving g to an incident room, it still sees the corridor, but now also sees a room. Therefore, the solution has not become worse.

Demands. This problem, sometimes referred to as *circulation problem with lower bounds* [70], is a generalization of Maximum Flow in a network and can be solved in polynomial time. Before delving into the reduction, we take a short detour so that we can focus on what the problem of Minimum Flow with Demands is about, redefine some relevant concepts and detail an approach for solving it.

5.2.4.1 Minimum flow with demands

Like in the standard setting, a *flow network* is a directed graph $G = (V, E)$ with two distinguished nodes, a *source* s and a *sink* t. We extend the usual definition so that each edge $(u, v) \in E$ is now characterized by two functions, $d(u, v)$ and $c(u, v)$, where $0 \leq d(u, v) \leq c(u, v)$. The function $d(u, v)$ is called the *demand* of the edge (u, v) and $c(u, v)$ is called its *capacity*.

A *flow* in G is a function $f : V \times V \to \mathbb{R}$ which we want to satisfy the following properties:

- Flow conservation: for all $u \in V \setminus \{s, t\}$, it holds that $\sum\limits_{v \in V} f(u, v) = \sum\limits_{v \in V} f(v, u)$.
- Edge constraints: for all $u, v \in V$, it holds that $d(u, v) \leq f(u, v) \leq c(u, v)$.

So each edge in E has a lower bound for the value of the flow that must pass through it, in addition to the upper bound imposed by its capacity. A flow that satisfies these properties is called *feasible*.

Contrary to what happens in a regular flow network, a network with demands does not necessarily admit a feasible flow. Minimum Flow with Demands asks for a feasible flow of minimum value for the given network G – or to report that no feasible flow exists. The standard Maximum Flow problem corresponds to setting $d(u, v) = 0$ for every edge $(u, v) \in E$ and in this case finding a minimum feasible flow is trivial: the zero flow, which sets $f(u, v) = 0$ for every edge $(u, v) \in E$, is both feasible and minimum. However, in the more general setting of Minimum Flow with Demands, finding *some* feasible flow suddenly becomes not obvious at all, let alone minimizing it (Figure 5.13).

Let us first describe how we can find a feasible flow in G and afterwards we explain how to minimize it.

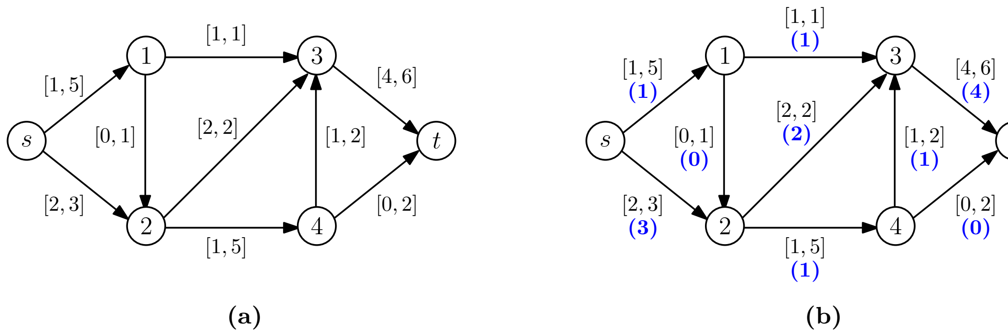


Figure 5.13: (a) Network G with demands and capacities, an instance of Minimum Flow with Demands. Each edge (u, v) is annotated with a range $[d(u, v), c(u, v)]$. **(b)** Example of a feasible flow in G, where on each edge (u, v) it is indicated inside parenthesis the amount that flows through it, as $(f(u, v))$.

Finding a feasible flow We reduce Minimum Flow with Demands to Maximum Flow, closely following the method described in [44]. For that, we construct a new network $G' = (V', E')$ (without demands on edges) that has the same nodes as G and, in addition, a new source s' and target t'. We replace each edge $(u, v) \in E$ with three edges in E': an edge (u, v) with capacity $c(u, v) - d(u, v)$, an edge (s', v) with capacity $d(u, v)$ and finally an edge (u, t') with capacity $d(u, v)$ as well. In the case where multiple edges leave s' to the same node v (or from the same node u to t'), we merge them into a single edge and add up their capacities. Furthermore, we add an extra edge from t to s having capacity ∞. See Figure 5.14 for a small example.

In other words, if we let $c' : E' \to \mathbb{R}$ denote the resulting capacity function in G', we define the capacity of each edge in E' as:

- $c'(s', v) = \sum_{u \in V} d(u, v)$ and $c'(v, t') = \sum_{w \in V} d(v, w)$ for every vertex $v \in V$.
- $c'(u, v) = c(u, v) - d(u, v)$ for every edge $(u, v) \in E$.
- $c'(t, s) = \infty$.

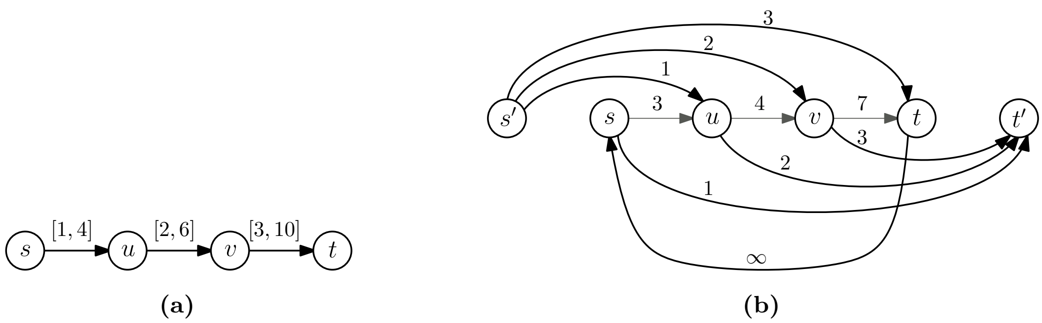


Figure 5.14: (a) Small original network G with edge capacities and demands. **(b)** New, transformed network G' obtained by reducing Minimum Flow with Demands to Maximum Flow. The value on an edge (u, v) refers to the transformed capacity $c'(u, v)$.

From this reduction, we have that $|V'| = |V| + 2$. Also, $|E'| \leq |E| + 2(|V| - 1) + 1 = 2|V| + |E| - 1$

because we have added at most one edge from s' to each node in V (other than s, since s has in-degree 0 in G), at most one edge to t' from each node in V (other than t, since t has out-degree 0 in G) and the edge (t, s). Hence this reduction takes time linear in $|G| = |V| + |E|$.

A flow in G' is *saturating* if its value equals the sum of the capacities of the edges leaving s' (or, equivalently by construction, all the edges entering t'). Every saturating flow is a maximum flow, because we cannot increase it further without overflowing the edge capacities. Also, if some maximum flow is saturating, all the maximum flows are as well (and, if some maximum flow is not saturating, then none is) [45].

Lemma 5.10. *[44] G has a feasible flow from s to t if and only if G' has a saturating flow from s' to t'.*

Let $f : E \to \mathbb{R}$ be a feasible flow in G. From the proof of Lemma 5.10, there is a saturating flow $f' : E' \to \mathbb{R}$ in G' that satisfies $f'(u, v) = f(u, v) - d(u, v)$. Conversely, for any saturating flow f' in G', the function $f(u, v) = f'(u, v) + d(u, v)$ is a feasible flow in G as well (when restricted to the original set of edges, E). Moreover, we have that $f'(t, s) = |f|$. See Figure 5.15 for an example. A significant consequence is that we can find some feasible flow in G – if one exists – by running any maximum flow algorithm on the new network G' and checking whether the determined flow $f'^{\star}$ is saturating. If it is, then $f'^{\star}(t, s)$ indicates the value of $|f|$ and we can in effect recover the actual flow f from $f'^{\star}$. If it isn't, no feasible flow exists in G.

Finding a minimum feasible flow Now that we are able to find a feasible flow, we would like to compute a *minimum* feasible flow on G, which we denote by $f^{\star}$. The main observation is that the entire flow of the old network G flows along the edge (t, s) in G', that is, $f'^{\star}(t, s) = |f|$. This edge was defined to have capacity ∞, which essentially corresponds to the maximum flow in G' found by any maximum flow algorithm being unlimited from above (apart from the remaining edge capacities themselves). By continuously limiting this edge with a smaller and smaller capacity $c'(t, s)$, the maximum flow value $|f'^{\star}|$ will keep decreasing until there is a moment when the network G' will stop admitting a saturated flow – and so the corresponding flow on the original network G will no longer satisfy all the edge demand constraints. Thus, we can binary search the lowest value k such that, by setting $c'(t, s) = k$, the new network G' still admits a saturating flow (which we can test by running a maximum flow algorithm on G'). The found value k gives us the value of the minimum feasible flow $f^{\star}$ on the original network, G.

This method runs in time $\mathcal{O}(T(G') \log |f^{\star}|)$, where $T(G')$ is the time complexity of the algorithm used for computing a maximum flow in G'. For instance, we get an overall running time of $\mathcal{O}(|V'|^2|E'| \log |f^{\star}|) = \mathcal{O}((|V|^3 + |V|^2|E|) \log |f^{\star}|) = \mathcal{O}(|V|^2|E| \log |f^{\star}|)$ if Dinic's algorithm [38] is used. We remark that a new paper [28], very recent by the time of writing of this thesis, claims that Maximum Flow in a network $H = (V_H, E_H)$ can be solved in almost-linear time $|E_H|^{1+o(1)}$. Should this result be confirmed, the algorithm we described for solving Minimum Flow with Demands can be sped up to get an overall time complexity of $|E'|^{1+o(1)} \log |f^{\star}| \leq (2|V| + |E| - 1)^{1+o(1)} \log |f^{\star}|$.

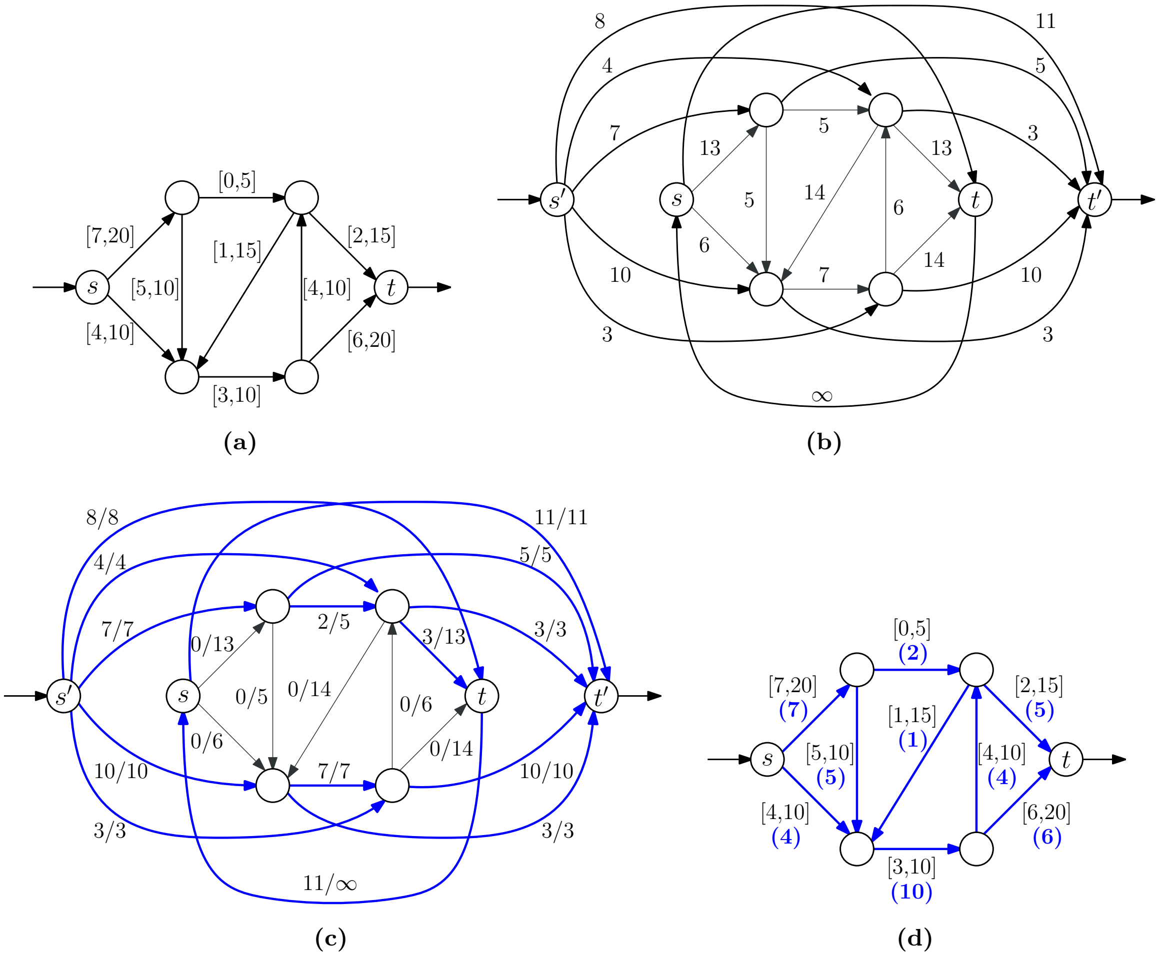


Figure 5.15: Example adapted from [44]. **(a)** Original network G with edge capacities and demands. **(b)** New network G' obtained by reducing Minimum Flow with Demands to Maximum Flow. The value on an edge (u, v) refers to the transformed capacity $c'(u, v)$. **(c)** A saturating flow f' in G', obtained by running a maximum flow algorithm over the transformed network. **(d)** A feasible flow f in G corresponding to f' by taking $f(u, v) = f'(u, v) + d(u, v)$ for every edge $(u, v) \in E$. Notice that $f'(t, s) = |f| = 11$.

5.2.4.2 Network construction

Having defined the problem Minimum Flow with Demands, we are ready to describe the reduction from Minimum SCOT r-Guard for r-independent SCOTs with point-guards, that is, how a network G can be built from a given r-independent $(\mathcal{R}, \mathcal{C}, \mathcal{C}')$-SCOT P. The idea is that, in our construction, every guard will be placed in a room (which is plausible by Lemma 5.9) and will r-see *exactly* one room, one horizontal super-corridor and one vertical super-corridor – perhaps artificial, as we explain below.

First, we define two types of gadgets. A *real-gadget* is a set of two nodes u and v that are connected by a directed edge (u, v) with demand 1 and capacity ∞. We call node u the *in-node* of the gadget and the other one, v, its *out-node*. A *pseudo-gadget* is defined identically to a real-gadget, but the edge linking the in-node to the out-node has demand 0. A real-gadget and a pseudo-gadget are illustrated on Figure 5.16b (above and below the network, respectively). The edges in real-gadgets are the only ones in G that have positive demand. Every edge in G has capacity ∞.

We will represent each room, horizontal super-corridor and vertical super-corridor in P by a corresponding real-gadget in G. We denote the gadgets corresponding to a given room R, to a horizontal super-corridor C'_h and to a vertical super-corridor C'_v as $\Gamma(R)$, $\Gamma(C'_h)$ and $\Gamma(C'_v)$, respectively. Also, add to G two pseudo-gadgets, $\Gamma(C'_{hf})$ and $\Gamma(C'_{vf})$, representing two mock super-corridors, one horizontal and one vertical, respectively.

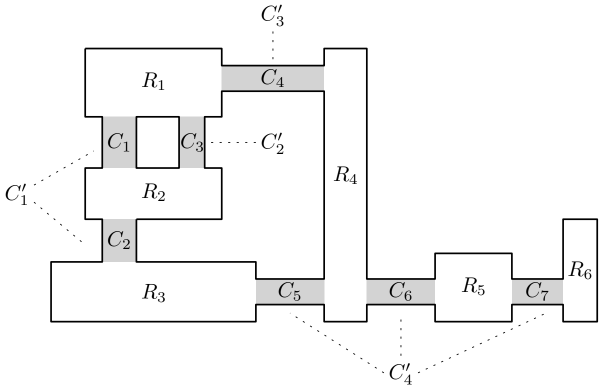
Connect the out-node of $\Gamma(C'_{hf})$ to the in-node of the real-gadget $\Gamma(R)$ of every room R with demand 0 and capacity ∞ and in the same way connect the out-node of the real-gadget $\Gamma(R)$ of every room to the in-node of $\Gamma(C'_{vf})$. Make sure to connect s to $\Gamma(C'_{hf})$ and $\Gamma(C'_{vf})$ to t with demands 0 and capacities ∞. For each pair of horizontal super-corridor and room (C'_h, R) that are incident in P, connect the out-node of $\Gamma(C'_h)$ to the in-node of $\Gamma(R)$ with an edge of demand 0 and capacity ∞. For each pair of room and vertical super-corridor (R, C'_v) that are incident in P, connect the out-node of $\Gamma(R)$ to the in-node of $\Gamma(C'_v)$, also with demand 0 and capacity ∞. Connect the source node s of G to the in-node of the real-gadget $\Gamma(C'_h)$ of each horizontal super-corridor with a demand of 0 and a capacity of ∞. Finally, connect the out-node of every real-gadget $\Gamma(C'_v)$ corresponding to a vertical super-corridor to the sink t, again with demand 0 and capacity ∞. See Figure 5.16 for an illustration of the whole reduction from a SCOT to a network of flows with demands.

Lemma 5.11. *The size of the network G and the time it takes to construct it are linear in the SCOT size, $|\mathcal{R}| + |\mathcal{C}| \in \Theta(|\mathcal{C}|)$. More specifically, $|V| \leq 2|\mathcal{R}| + 2|\mathcal{C}| + 6$ and $|E| \leq 3|\mathcal{R}| + 4|\mathcal{C}| + 4$.*

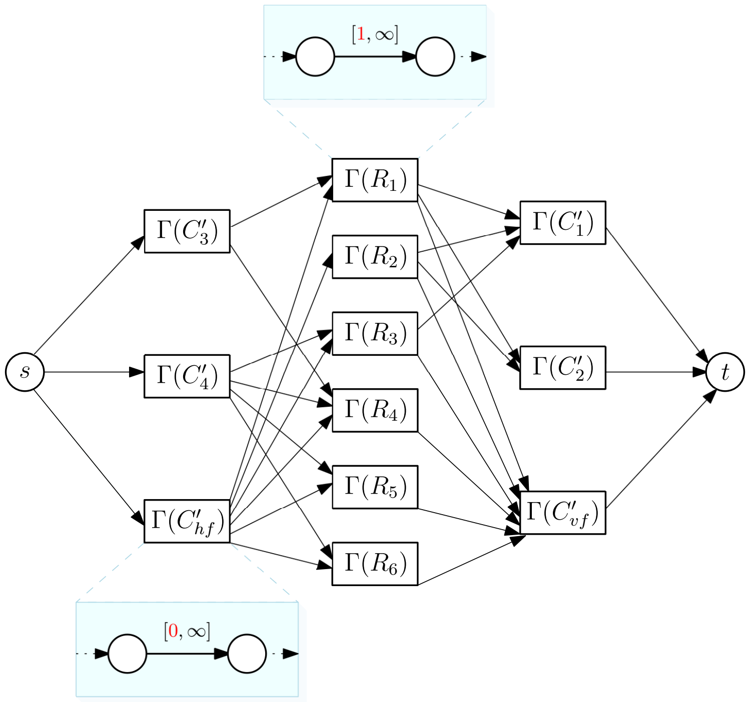
Proof. Let $\mathcal{C}'_h$ and $\mathcal{C}'_v$ be the sets of horizontal and vertical super-corridors in P. We have added two nodes in G for each real-gadget, for each pseudo-gadget and for the source s and sink t. We have added an edge from s to the in-node of every gadget corresponding to a (fictitious or not) horizontal super-corridor ($|\mathcal{C}'_h| + 1$), an edge from $\Gamma(C'_{hf})$ to every room ($|\mathcal{R}|$), an edge from every room to $\Gamma(C'_{vf})$ ($|\mathcal{R}|$) and an edge from the out-edge of the gadget corresponding to every (fictitious or not) vertical super-corridor to t ($|\mathcal{C}'_v|+1$). Also, we have added edges connecting rooms to incident super-corridors whose amount is bounded above by $2|\mathcal{C}|$, by Lemma 5.4, and edges linking in-nodes to out-nodes in gadgets ($|\mathcal{R}|+|\mathcal{C}'_h|+1+|\mathcal{C}'_v|+1$). Therefore, $|V| = 2(|\mathcal{R}|+|\mathcal{C}'|)+2\times 2+2 \leq 2(|\mathcal{R}|+|\mathcal{C}|)+6$ and $|E| \leq 2(|\mathcal{C}'_h| + 1) + 2(|\mathcal{C}'_v| + 1) + 3|\mathcal{R}| + 2|\mathcal{C}| \leq 3|\mathcal{R}| + 4|\mathcal{C}| + 4$. □

The meaning that we assign to one unit of flow on G *is* a point-guard in P; for each unit that flows from s to t, one guard is placed at some room in P. Recall that every edge in G has capacity ∞. This implies that G always admits a feasible flow; in particular, one with $|f^\star| = \infty$. This also means that the number of guards allowed for watching over any room or corridor of P is essentially unbounded: we do not care if multiple guards all see the same region in P, as long as the resulting solution is still optimal. In practice, we can replace ∞ by an upper bound for the number of guards needed for covering P entirely, such as $|\mathcal{R}| + |\mathcal{C}|$ or even $|\mathcal{R}| + |\mathcal{C}'|$.

Intuitively, the edges in real-gadgets, which were assigned a positive demand, require any



(a)



(b)

Figure 5.16: (a) The same r-independent SCOT P of Figure 5.9a, with four super-corridors: $C'_1 = \{C_1, C_2\}$, $C'_2 = \{C_3\}$, $C'_3 = \{C_4\}$ and $C'_4 = \{C_5, C_6, C_7\}$. **(b)** The network G corresponding to P that is an instance of Minimum Flow with Demands. It has a total of 26 nodes and 41 edges (including those on the gadgets). We display nodes vertically split into five conceptual groups: source (s), gadgets for horizontal super-corridors, room gadgets, gadgets for vertical super-corridors and sink (t). Notice that pseudo-gadgets $\Gamma(C'_{hf})$ and $\Gamma(C'_{vf})$, for both horizontal and vertical super-corridors, are connected to every room to enable solutions where guards are placed in positions from where they cannot see any corridor in one or both directions. Every edge in G has capacity ∞. All the edges belonging to real gadgets have demand 1; every other edge has demand 0.

feasible flow on G to pass through them and, thus, force every room and (real) super-corridor to have at least one guard covering it. Edges in pseudo-gadgets have demand 0 because we do not explicitly require any guard to see them; their only importance is to assure a correspondence between the network construction and the polygon. The same way a point-guard can see at most one room, one horizontal super-corridor and one vertical super-corridor in P, one unit of flow through the network G can satisfy the demands of at most three edges with positive demand, covering a path of the form $s \to \Gamma(C'_h) \to \Gamma(R) \to \Gamma(C'_v) \to t$. If a placed guard only sees a room and at most one super-corridor, the corresponding unit of flow will still cover up a path from s to t in G – but that path will pass through artificial super-corridor nodes.

Remark. *Notice how the order $\Gamma(C'_h) \to \Gamma(R) \to \Gamma(C'_v)$ of the gadgets in a path from s to t matters. Suppose that, alternatively, we had (incorrectly) decided it would be best for the gadget of a vertical super-corridor C'_v to immediately follow the gadget of an adjacent, horizontal super-corridor C'_h (or the other way around) in a path of the form $s \to \Gamma(R) \to \Gamma(C'_h) \to \Gamma(C'_v) \to t$. The subtle issue here is that the fact that C'_h and C'_v are adjacent (i.e., incident to the same room R') does not necessarily imply that $R' = R$. In such case we could be inadvertently enabling a guard that is positioned in room R to also see a vertical super-corridor C'_v that is attached to another room R', neighbour of R. On the other hand, the order we originally defined correctly forces the guard to* r*-see both horizontal and vertical super-corridors in coherence with the room it has been placed at, because consistency is enforced between the room and a horizontal super-corridor* and *also between the room and a vertical super-corridor.*

According to the meaning we have assigned to f, the value of the minimum feasible flow on G is equal to the minimum number of point-guards that are needed for guarding P, which is what Lemma 5.12 proves. Lemma 5.12 also explains how we can extract the actual guard set from the resulting flow configuration found by the algorithm for Minimum Flow with Demands that was proposed in Section 5.2.4.1. Recall that the value of the flow on G at any iteration of the binary search, in particular $|f^\star|$, is bounded above by the capacity of the edge (t, s) in G', for which we have argued that an appropriate value could be $|\mathcal{R}| + |\mathcal{C}| \in \Theta(|\mathcal{C}|)$.

The reduction to Minimum Flow with Demands we have presented then implies a method for determining a minimum-cardinality point-guard set for r-independent SCOTs under r-visibility. Its overall time complexity depends on the exact Maximum Flow algorithm that is used as subroutine. If Dinic's algorithm [38] is used, by the proof of Lemma 5.11 the running time that one obtains is $\mathcal{O}\left(|V|^2|E|\log|f^\star|\right) = \mathcal{O}\left((|\mathcal{R}| + |\mathcal{C}|)^3 \log\left(|\mathcal{R}| + |\mathcal{C}|\right)\right) = \mathcal{O}\left(|\mathcal{C}|^3 \log|\mathcal{C}|\right)$. If, on the other hand, the algorithm reported in [28] is used, one can achieve a time complexity of $(5|\mathcal{R}| + 6|\mathcal{C}| + 5)^{1+o(1)} \log|\mathcal{C}|$. Figure 5.17 exemplifies the parallelism between the minimum feasible flow in G and the optimal point-guard set in P.

Remark. *The upper bound $5|\mathcal{R}| + 6|\mathcal{C}| + 5$ that we obtained for $|E'|$ comes from the fact that only edges in the real-gadgets of G have positive demand and, thus, contribute with positive capacities to G'. When building the transformed network G' from G, we add an edge from s' to each real-gadget out-node, an edge from each real-gadget in-node to t' and an edge from t to s. Hence, we have $|E'| = |E| + 2(|\mathcal{R}| + |\mathcal{C}'|) + 1 \leq 5|\mathcal{R}| + 6|\mathcal{C}| + 5$. Note that the bounds*

$|V| \leq 2|\mathcal{R}| + 2|\mathcal{C}| + 6$, $|E| \leq 3|\mathcal{R}| + 4|\mathcal{C}| + 4$ and $|E'| \leq 2|V| + |E| - 1$ hold for general networks, but blindly applying these would yield a looser bound of $|E'| \leq 7|\mathcal{R}| + 8|\mathcal{C}| + 15$.

Lemma 5.12. *The $(\mathcal{R}, \mathcal{C}, \mathcal{C}')$-SCOT P has a point-guard set $\mathcal{G}$ of size $k \in \mathbb{Z}^+$ if and only if the network with demands G has a feasible flow f of value $|f| = k$.*

Proof.

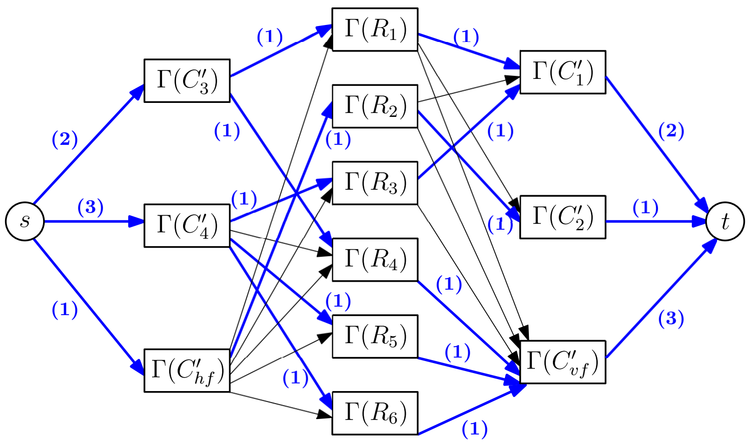
($\Rightarrow$) Let $\mathcal{G}$ be a point-guard set for P with $|\mathcal{G}| = k$. By Lemma 5.9, assume every guard in $\mathcal{G}$ is placed inside a room. For each guard $g \in \mathcal{G}$, let $R \in \mathcal{R}$, $C'_h \in \mathcal{C}'$ and $C'_v \in \mathcal{C}'$ be the room, horizontal super-corridor and vertical super-corridor that g r-sees; if it does not see any horizontal and/or vertical super-corridor, take C'_h and/or C'_v as fictitious super-corridors. Send 1 unit of flow from s to t through the path $s \to \Gamma(C'_h) \to \Gamma(R) \to \Gamma(C'_v) \to t$. This can always be done because all the edge capacities are ∞. This yields a flow of value k. Since the guard set $\mathcal{G}$ covers every room, every horizontal super-corridor and every vertical super-corridor in P, the demand of every edge in G will be satisfied and therefore the flow is feasible.

($\Leftarrow$) Let $\mathcal{G} = \emptyset$. Do what follows while in G there is some path from s to t that passes only through edges (u, v) with $f(u, v) > 0$. Let $\gamma = s \to \Gamma(C'_h) \to \Gamma(R) \to \Gamma(C'_v) \to t$ be such a path, which can be found by a breadth-first search from s. Place a guard anywhere in the intersection of R, the stretch of C'_h and the stretch of C'_v; if C'_h and/or C'_v are fictitious super-corridors, place the guard in an arbitrary y-coordinate and/or an arbitrary x-coordinate, respectively, within room R. Insert this guard into $\mathcal{G}$. Decrement in 1 the value of $f(u, v)$ for every edge (u, v) along γ. Since the flow f satisfied every edge demand in G, for any room and corridor in P there will be at least one guard in $\mathcal{G}$ watching over it and $|\mathcal{G}| = k$. The number of processed paths γ is equal to $|f|$ and, since every path γ considered has 4 edges, the cost per iteration is $\mathcal{O}(1)$, leading to an overall linear-time reduction in practice, where $|f| \leq |\mathcal{R}| + |\mathcal{C}|$.

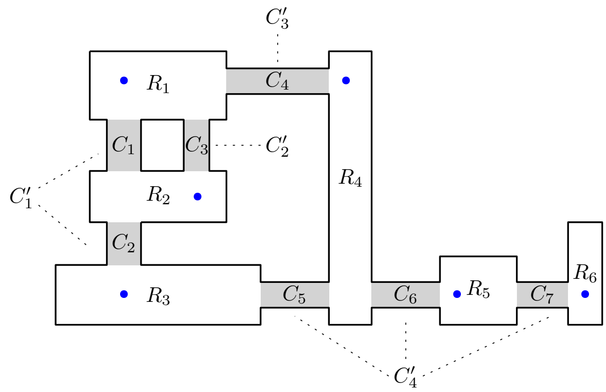
□

5.3 NP-completeness with point-guards

In Sections 5.1 and 5.2, we have presented efficient, polynomial-time algorithms for solving the cases where the SCOT instance has no holes or is r-independent. These were majorly based upon the specific structure of the polygon, which allowed for determining guard placements that we can combinatorially prove optimal. It remains for us to question the hardness of the general version of the problem – the one where the SCOT simultaneously has holes and is not r-independent – and whether or not we can extend our ideas to also solve it in polynomial time. As it turns out, the general version is intrinsically harder.



(a)



(b)

Figure 5.17: **(a)** Minimum feasible flow f in the network G, with $|f| = 6$. The flow has not been annotated on edges (u, v) with $f(u, v) = 0$ to avoid clutter. **(b)** SCOT P, with an optimal set of 6 point-guards that has been determined by flow f.

Theorem 5.13. *Minimum SCOT r-Guard with point-guards is* NP*-hard.*

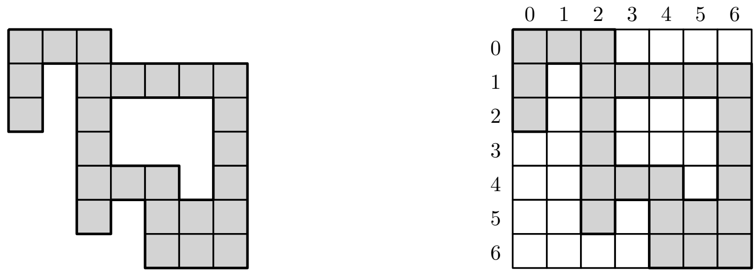
For the proof, we present a polynomial reduction from Minimum Polyomino *r*-Guard, which is the problem of finding a minimum-cardinality r-guard set for a polyomino. Deciding whether a polyomino with m cells can be r-guarded with up to k point-guards, for a positive integer k, is NP-hard [61].

5.3.1 Construction

At a high level, the SCOT B we construct for a given m-polyomino P is composed of a sequence of identical gadgets replicated side by side, horizontally. Each two consecutive gadgets in the sequence are connected by one or more horizontal corridors as we will describe in due course. For a visual preview of the final reduction, see Figure 5.21.

We define as $\Delta(P)$ the minimal unit grid that contains the polyomino P (Figure 5.18).

Assume its dimensions are N rows $\times$ M columns and its top-left corner has coordinates $(0, 0)$.



(a) (b)

Figure 5.18: (a) Example of a 25-polyomino P. **(b)** The 7×7 minimal unit grid $\Delta(P)$ that contains P.

5.3.1.1 Anchor gadget

We begin by defining an *anchor gadget* Γ that will be instanced several times in the construction of our SCOT. Figure 5.19 exemplifies the gadget's construction. It consists of an $(N+7) \times 5$ room (R_c) whose top and bottom walls are contained, respectively, in the lines $y = 0$ and $y = N + 7$ (for convenience, we assume a Cartesian coordinate system $(O, \vec{i}, -\vec{j})$, that is, the y-axis grows downwards). Two tiny rooms are appended onto the walls of R_c by means of corridors. On the left there is a 5×1 room (R_l), whose top edge also satisfies $y = 0$. R_l is connected to the big room R_c by means of two 1×1 corridors: one of the corridors (C_γ) has its top edge contained in $y = 1$ and the other corridor (C_δ) has its own contained in $y = 3$. Similarly, above R_c there is a 1×5 room (R_t) whose top wall is contained in $y = -2$. R_t is also connected to R_c through two 1×1 corridors whose left walls obey $x = 1$ (C_α) and $x = 3$ (C_β), respectively.

The coordinate system used to describe the anchor gadget is relative to the top-left corner of its big room R_c. Every instance of the gadget has a similar description, but their absolute coordinates will differ from each other when they actually get placed as components on the SCOT.

5.3.1.2 Instance transformation

SCOT B is formed by $M + 1$ identical anchor gadgets $\Gamma_0, \Gamma_1, \ldots, \Gamma_M$ arranged horizontally side by side, numbered 0 through M from left to right. See Figure 5.20 for an example. The big rooms of each two consecutive anchor gadgets Γ_i and Γ_{i+1} $(0 \leq i < M)$ are separated by a horizontal space of 3 units and the line $y = 0$ passes through the top edges of all the big rooms. All the corridors connecting two consecutive anchor gadgets in B have fixed length 3 but variable width.

For placing corridors in B, sweep through the grid $\Delta(P)$, column by column, from left to right. When processing the i-th column $(0 \leq i < M)$, consider all the vertical maximal connected components of cells that belong to that column in P. For each connected component spanning rows $a, a+1, \ldots, b$ of the i-th column, add a corridor with length 3 and width $b - a + 1$ connecting

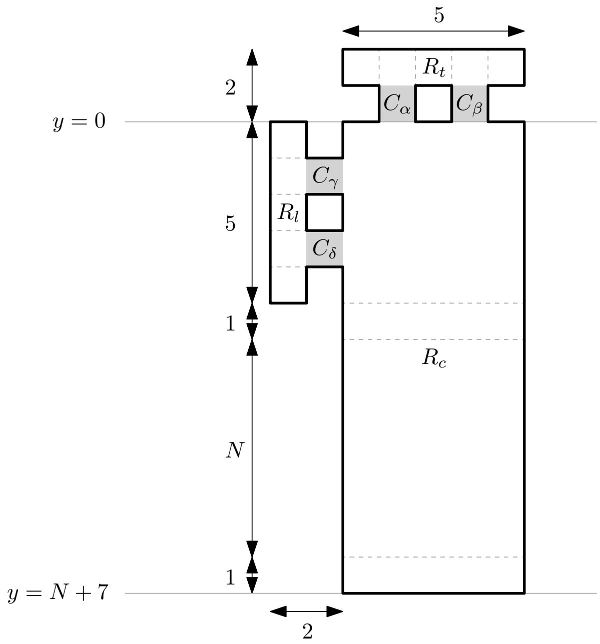


Figure 5.19: Anchor gadget Γ.

big rooms i and $i+1$ of B. That corridor has to be placed so that its top and bottom edges are contained, respectively, in the lines $y = a + 6$ and $y = (b+1) + 6 = b + 7$.

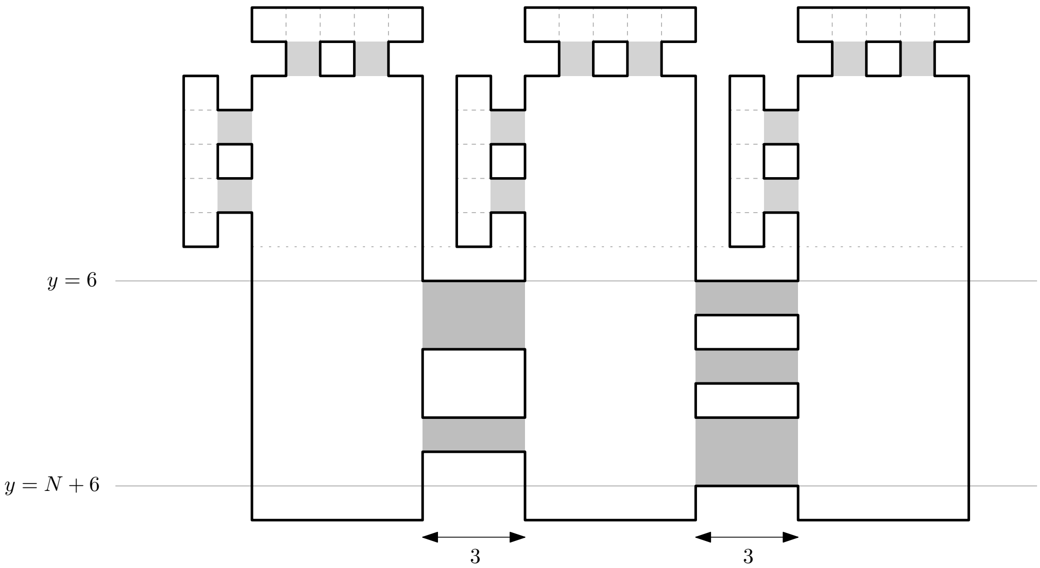


Figure 5.20: Three anchor gadgets laying side by side and connected by horizontal corridors (only as an example). Notice that corridors linking two adjacent anchor gadgets are contained in the plane region $6 \le y \le N + 6$.

5.3.2 Complexity

Lemma 5.14. *The reduction reported is polynomial on m, the number of cells in polyomino P.*

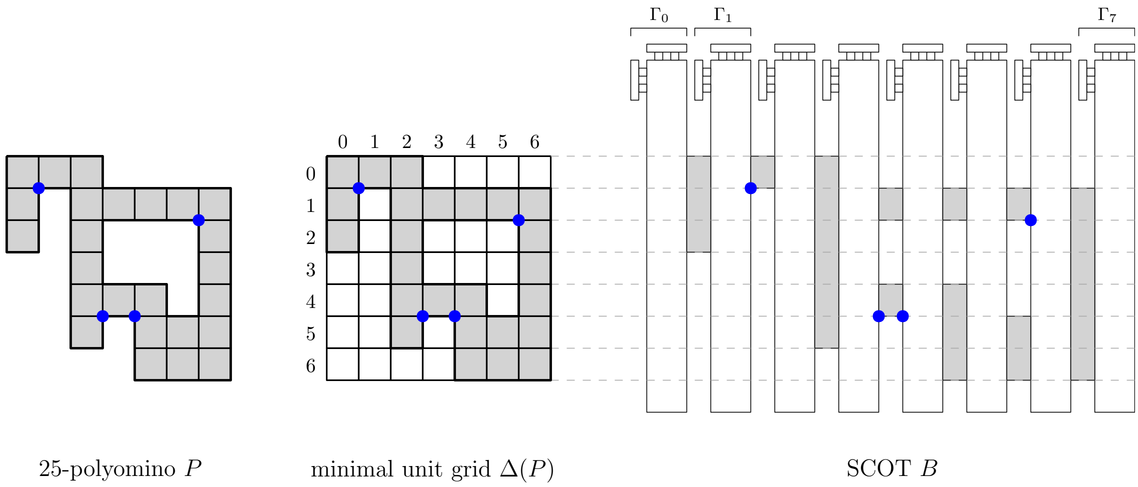


Figure 5.21: Example of a reduction from Minimum Polyomino r-Guard to Minimum SCOT r-Guard. SCOT B is not presented to scale so that the correspondence between the guard sets in both instances is visually clearer.

Proof. $N, M \leq m$ is a suitable upper bound for the dimensions $N \times M$ of the minimal unit grid $\Delta(P)$ containing P, so building $M+1$ anchor gadgets is done in $\mathcal{O}(m)$ time. Sweeping through the columns of the grid $\Delta(P)$, determining maximal vertical connected components and adding the corresponding corridors to the SCOT B can be done in time linear on the grid size, $\mathcal{O}(m^2)$. Every anchor gadget takes 7 horizontal and $N+9$ vertical units of space, each two consecutive anchor gadgets are connected by horizontal corridors of length 3 and all the coordinates of vertices in B are defined by integer numbers bounded by a linear function of m. As such, the reduction is polynomial on the size of the instance P. □

5.3.3 Correctness

Lemma 5.15. *Let Γ be an anchor gadget, possibly with extra horizontal corridors attached to its left and right walls that result from the presented construction. Exactly 3 point-guards are required for guarding Γ, ignoring the extra corridors. Moreover, we can construct such a point-guard set in which every guard satisfies $y \leq 5$.*

Proof. We first show that exactly 3 point-guards are required for guarding Γ.

Necessity

The tiny rooms R_t and R_l of Γ lie in the half-plane defined by $y \leq 5$. Given that, for r-guarding some room, it is mandatory that a guard is placed in its interior, we cannot cover the big room and the two tiny rooms with less than 3 guards.

Sufficiency

First, we choose one corridor among C_α and C_β and also choose one corridor among C_γ and C_δ. Suppose, without loss of generality, that we picked C_α and C_γ. Place a guard on one of the reflex vertices shared by R_t and C_α and another guard on one of the reflex vertices shared by R_l and C_γ. So far we have covered R_t, R_l, C_α and C_γ. The remaining parts of Γ – the big room, R_c, and the corridors, C_β and C_δ – can be covered with a third guard: simply place it somewhere in the intersection of the stretches of C_β and C_δ.

We now show that there is always an optimal solution for guarding Γ for which every guard satisfies $y \leq 5$. Suppose, by contradiction, that there is a better strategy, that is, one could instead choose to place the third guard in the intersection of the big room R_c and the stretch of another horizontal corridor $C^\star$ (other than C_γ and C_δ) incident in the big room to try to r-see $C^\star$. We would then still need another fourth guard somewhere in the stretch of C_δ to cover up C_δ: since all the remaining horizontal corridors incident in Γ are, by construction, positioned in the half-plane $y \geq 6$, they are r-independent from C_δ and therefore could not possibly be exploited to guard C_δ using fewer guards. Given that C_α has already been covered, we have nothing to lose by also placing the fourth guard in the stretch of C_β (because that way we can cover up C_δ and C_β in one go); so the fourth guard could be placed in the intersection of the stretch of C_β and the stretch of C_δ. But then one could swap the third and fourth guards to obtain precisely the solution described before. Therefore, there is no better strategy and a minimal guard set of 3 point-guards with $y \leq 5$ always exists for watching over Γ. Any horizontal corridor that remains in $6 \leq y \leq N+6$ can then be guarded independently from the gadget with extra guards. □

Before proving that the reduction works, let us explain the intuitive idea behind it. By Lemma 5.15, we can always construct an optimal solution for which all the rooms in B must be covered by exactly $3(M+1)$ guards that, together, do not see any extra corridor other than those that already belong to the gadget by definition. Also, it can be understood that a guard placed inside a room of B r-sees no less than a guard placed inside one of the incident corridors. This holds because every room is wider than all the incident corridors, so it does not obstruct r-visibility: only corridors do. Because of that, and given that distances in r-visibility are not important, rooms in B can entirely be ignored, as if they were not even there and adjacent corridors were directly connected to each other. This leaves us only corridors of B to cover – precisely those corridors that correspond to pixels in P.

Lemma 5.16. *[18] Under the* r*-visibility model, for any polyomino P there exists an optimal solution for guarding P whose guards are placed only at pixel corners.*

The proof of Lemma 5.16 goes through moving guards in a polyomino P into corners of the pixels that contain them and showing that this transformation preserves feasibility.

Lemma 5.17. *The polyomino P can be* r*-guarded by $k \in \mathbb{Z}^+$ point-guards if and only if the SCOT B can be* r*-guarded by $3(M+1)+k$ point-guards.*

Proof. We show that we can map any solution for P to an analogous one for B which has 3 extra guards per anchor gadget (and vice-versa).

($\Rightarrow$) Let $\mathcal{G}_B = \emptyset$ and let $\mathcal{G}_P$ be a guard set for P with $|\mathcal{G}_P| = k$ in which every guard is placed at a pixel corner by Lemma 5.16. For each guard $g \in \mathcal{G}_P$ that is placed at the intersection of the line $y = q$, for some $0 \leq q \leq N$, and the left (respectively, right) pixel corner of the i-th column of $\Delta(P)$ $(0 \leq i < M)$, place a guard in $\mathcal{G}_B$ at the intersection of $y = q + 6$ and the left (respectively, right) side of a corridor connecting gadgets i and $i + 1$ in B. Note that, if $i < M - 1$, the guard g also belongs to the $(i + 1)$-th column of $\Delta(P)$, but we do not place an extra, redundant guard in $\mathcal{G}_B$. Next, insert $3(M + 1)$ guards satisfying $y \leq 5$ into $\mathcal{G}_B$ to cover every gadget in P, according to Lemma 5.15. We have that $|\mathcal{G}_B| = 3(M + 1) + k$. By the assumption that $\mathcal{G}_P$ covers up the entirety of P, and since rooms in B are wider than the incident corridors and distances do not affect r-visibility, $\mathcal{G}_B$ covers up all the corridors connecting adjacent anchor gadgets in B and, thus, B is covered entirely as well.

($\Leftarrow$) Let $\mathcal{G}_P = \emptyset$ and let $\mathcal{G}_B$ be a guard set for SCOT B with $|\mathcal{G}_B| = 3(M + 1) + k$ in which every guard covering an anchor gadget satisfies $y \leq 5$ by Lemma 5.15. Assume that every guard in $\mathcal{G}_B$ other than those $3(M + 1)$ that see anchor gadgets are placed in corridors connecting consecutive anchor gadgets Γ_i and Γ_{i+1} $(0 \leq i < M)$. We can assume that because, by moving a guard g from a big room R_c of an anchor gadget to an incident corridor, the only region g stops seeing completely is R_c, which is still covered by one of the $3(M + 1)$ guards we have discriminated for watching over anchor gadgets. For each guard $g \in \mathcal{G}_B$ that is placed at the intersection of the line $y = q + 6$, for some $0 \leq q \leq N$, and a corridor connecting gadgets Γ_i and Γ_{i+1} in B $(0 \leq i < M)$, let d be the horizontal distance between g and Γ_i and place a guard in $\mathcal{G}_P$ at the point $(i + d, q)$. Big rooms R_c of anchor gadgets in B do not block r-visibility from incident corridors. Therefore, guards in $\mathcal{G}_B$ covering up all the corridors imply that the corresponding guards in $\mathcal{G}_P$ will also cover up all the pixels of P and, thus, $\mathcal{G}_P$ is a valid guard set for P, with $|\mathcal{G}_P| = k$.

$\square$

5.3.4 NP-completeness

Lemmas 5.14 and 5.17 prove that Minimum SCOT r-Guard for point-guards is NP-hard. Next we show that the decision problem is also NP-complete. However simple the invoked argument can be, we draw attention to the fact that proving NP-completeness results for Art Gallery Problem (AGP) variants requires some care and is not as *obvious* as it may seem. We now explain why.

The class $\exists\mathbb{R}$ contains the problems that are equivalent, under polynomial-time reductions, to deciding whether a system of polynomial equations with coefficients in $\mathbb{Z}$ and variables in $\mathbb{R}$ has a solution. As of now, it is known that NP $\subseteq \exists\mathbb{R}$, but whether $\exists\mathbb{R} \subseteq$ NP also holds remains an open question.

Until recently, the general AGP under straight-line visibility has been known to be NP-hard, but there was no clue whether it is also NP-complete. A possible way of proving that the point-guard AGP is in NP would be to argue that any instance admits an optimal guard set of rational coordinates with a polynomial number of bits – and that would indeed work if, in 2017, Abrahamsen et al. [1] had not demonstrated the existence of polygons with integer coordinates for which any optimal point-guard set must contain a guard with irrational coordinates. This holds even for orthogonal polygons (Figure 5.22). In the next year, it was shown that guarding simple polygons where all vertices have integer coordinates is indeed $\exists\mathbb{R}$-complete [2]. This result places AGP outside of the class NP, ruling out its NP-completeness, unless NP $= \exists\mathbb{R}$. In this sense, while at first the standard argument that any solution of AGP can *obviously* be checked to be a feasible guard set in polynomial time may seem reasonable, there is actually a subtle issue in this logic.

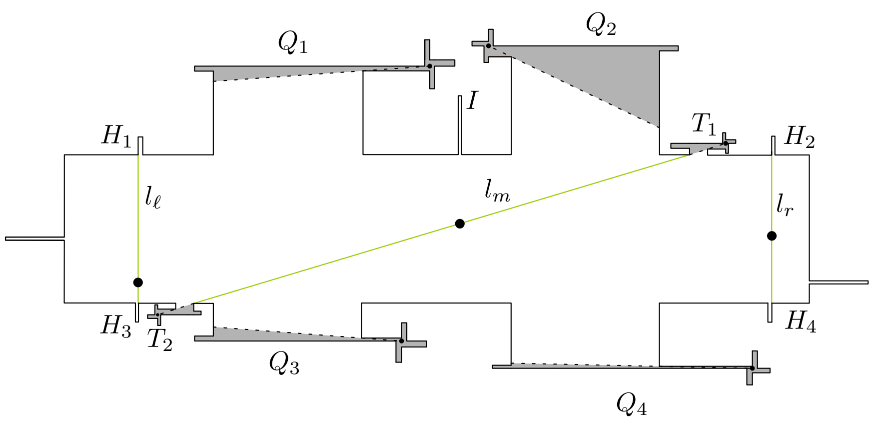


Figure 5.22: An orthogonal polygon P whose vertices have rational coordinates, reprinted from [1]. P can be covered under straight-line visibility with 9 point-guards if we allow irrational coordinates, but requires 10 point-guards if we do not.

To show that Minimum SCOT r-Guard is, nevertheless, still in NP, we must take advantage of the fact that we are adopting the notions of r-visibility on orthogonal polygons, which introduce a very specific structure to the problem. We invoke a result from [14, 17] that establishes an r-visibility dominance hierarchy between points in the interior, on the edges and at vertices of r-pieces in the grid partition $\Pi_{HV}(P)$:

Lemma 5.18. *([14], Lemmas A.1 and A.2) Let P be an orthogonal polygon. Let $\pi \in \Pi_{HV}(P)$ be an* r*-piece, let p be any point in the interior of π, let p_b be any point in the interior of a side e of π and let p_v be any endpoint of e. Under* r*-visibility, $\mathcal{V}_r(p) \subseteq \mathcal{V}_r(p_b) \subseteq \mathcal{V}_r(p_v)$, being $\mathcal{V}_r(p) \subseteq P$ the set of points that p* r*-sees.*

Lemma 5.18 guarantees that, for any SCOT P and positive integer k, if P can be watched over with k point-guards, then there is also a point-guard set of size k that is representable polynomially in the instance size, $|P|$. This enables us to state Theorem 5.19.

Theorem 5.19. *Minimum SCOT* r*-Guard with point-guards is in* NP.

Proof. Let P be a SCOT and $\Pi_{HV}(P)$ be its orthogonal grid partition. Lemma 5.18 implies that any feasible guard set for P can be transformed so that any guard lies at one of the vertices of an

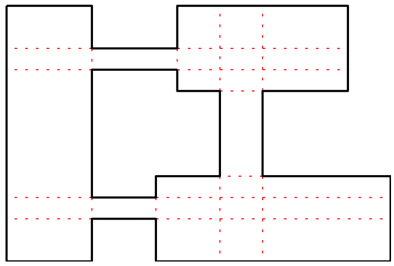


Figure 5.23: The grid partition $\Pi_{HV}(P)$ of a SCOT P.

r-piece $\pi \in \Pi_{HV}(P)$. A vertex of an r-piece in $\Pi_{HV}(P)$ either is a vertex of P or corresponds to the intersection of the extension of two or more edges of P. Therefore, if the coordinates of the vertices of P have a polynomial representation, then so have the coordinates of the guards. Such a solution is verifiable in polynomial time. $\square$

In accordance with Theorems 5.13 and 5.19, we are finally set to state our main result.

Theorem 5.20. *The decision version of* Minimum *SCOT* r-Guard *with point-guards is* NP*-complete.*

5.3.5 Approximation algorithms for vertex-guards and point-guards

We have proved that the unrestricted version of Minimum SCOT r-Guard for point-guards, where SCOTs may have holes and do not need to be r-independent, is NP-complete. We conclude this chapter by arguing that this problem can be approximated with logarithmic factor in the number of corridors, for both vertex-guards and point-guards.

Theorem 5.21. *Let P be an $(\mathcal{R}, \mathcal{C})$-SCOT. A minimum-cardinality vertex-guard set for P can be approximated with factor $\mathcal{O}(\log |\mathcal{C}|)$ in polynomial time.*

Proof. Compute the grid partition $\Pi_{HV}(P)$ in time $\mathcal{O}(n^2) = \mathcal{O}((|\mathcal{R}| + |\mathcal{C}|)^2) = \mathcal{O}(|\mathcal{C}|^2)$ by sweep-line (see Section 2.1.4). Reduce the problem to a Set Cover instance $(\mathcal{U}, \mathcal{F})$, where $\mathcal{U}$ contains an element for each r-piece $\pi \in \Pi_{HV}(P)$, and the subset of pieces seen by each vertex of P forms a set in $\mathcal{F}$. Finally, an optimal solution for this instance can be approximated with factor $\mathcal{O}(\log |\mathcal{U}|) = \mathcal{O}(\log |\Pi_{HV}(P)|) = \mathcal{O}\left(\log (|\mathcal{R}| + |\mathcal{C}|)^2\right) = \mathcal{O}(\log |\mathcal{C}|)$ using the Greedy-Set-Cover algorithm (Algorithm 1). $\square$

Theorem 5.22. *Let P be an $(\mathcal{R}, \mathcal{C})$-SCOT. A minimum-cardinality point-guard set for P can be approximated with factor $\mathcal{O}(\log |\mathcal{C}|)$ in polynomial time.*

Proof. Just as with vertex-guards, compute the grid partition $\Pi_{HV}(P)$ in time $\mathcal{O}(|\mathcal{C}|^2)$ by sweep-line. Reduce the problem to a Set Cover instance $(\mathcal{U}, \mathcal{F})$, where $\mathcal{U}$ contains an element for each r-piece $\pi \in \Pi_{HV}(P)$, and the subset of pieces seen by each r-piece corner forms a set in $\mathcal{F}$. Again, an optimal solution for $(\mathcal{U}, \mathcal{F})$ can be approximated with factor $\mathcal{O}(\log |\mathcal{U}|) = \mathcal{O}(\log |\mathcal{C}|)$

using the Greedy-Set-Cover algorithm (Algorithm 1), and, by Lemma 5.18, every position that is relevant for placing point-guards at is being taken into account. □

Chapter 6

Rook Vision on Polyominoes

In a 2021 publication by Alpert and Roldán [7], a new version of the Art Gallery Problem (AGP) for polyominoes is brought forward, in which guards are modeled as rook pieces from chess and can be placed on any tile of the polyomino. A rook standing in a pixel π *sees*, along rows and columns, exactly those pixels it would attack in a regular chess game: it covers both horizontal and vertical maximally connected segments of tiles that contain π. The rook's line of sight ends when it hits the polyomino's exterior. The authors call this visibility model *rook vision*. The goal is, again, to cover the whole interior of a polyomino by placing the minimum amount of rook guards on its tiles (Figure 6.1).

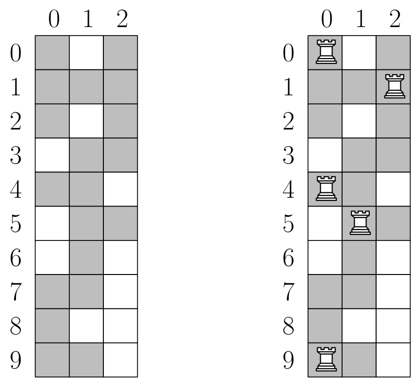


Figure 6.1: A 19-polyomino P (shaded region) within a 10×3 grid and a corresponding minimum-cardinality rook-guard set of 5 rooks, with coordinates $(0,0)$, $(1,2)$, $(4,0)$, $(5,1)$ and $(9,0)$. P has a one-tile hole with coordinates $(2,1)$.

Similarly to what happens in other versions of the AGP, two guards can see through each other, meaning that their visibility is not mutually blocked. Note that under rook vision this detail is irrelevant, given that the overall region that is guarded by two guards is the union of their visibility regions anyway.

We refer to this variant of the problem as Rook-AGP. Formally, an instance of Rook-AGP is a pair (P,k), where P is a polyomino and k is a positive integer. The problem asks whether k rooks can be placed on pixels of P to guard every pixel of P and is shown by the authors to be

NP-hard, so, unless P = NP, one cannot hope for a polynomial-time algorithm that solves every instance of Rook-AGP. Besides proving the hardness of Rook-AGP, the authors also give bounds on the number of rooks needed, similar in nature to the Art Gallery Theorem (AGT):

Theorem 6.1. *([7], Theorem 1) $\lfloor m/2 \rfloor$ rooks are sufficient and occasionally necessary to guard any m-polyomino P_m with $m \geq 2$ pixels.*

In the current chapter, we propose a new exact algorithm for determining a minimum cardinality rook-guard set for polyominoes. It is efficient for polyominoes P for which at least one of the side lengths of their minimal enclosing unit grid $\Delta(P)$ is bounded. More concretely, the algorithm is exponential parameterized by the smallest side length of the grid, $\min\{R, C\}$, where R and C are, respectively, the number of rows and columns of $\Delta(P)$, and becomes polynomial if the smallest of the dimensions of the grid is constant, even if the other one is arbitrarily large. Our algorithm works even if the input polyominoes have holes.

6.1 Parameterized algorithm for Rook-AGP

An m-polyomino P_m can be represented as the minimal unit grid $\Delta(P)$ that contains its pixels. Assume that the top-left corner pixel of $\Delta(P)$ is placed at the origin of the coordinate system, $(0, 0)$. In this context, we define R and C to be the number of rows and columns of such a minimal grid and g to denote an $R \times C$ Boolean matrix where $g_{r,c} = \mathsf{True}$ if and only if there is a pixel $\pi \in P_m$ with row and column coordinates (r, c) $(0 \leq r < R,\ 0 \leq c < C)$. A *dense* polyomino has a number of pixels $m \in \Theta(R \times C)$.

We describe an algorithm for efficiently solving Rook-AGP when either dimension R or C is comparatively smaller than the other, even if the product $R \times C$ is arbitrarily large. The algorithm is parameterized by $\min\{R, C\}$, runs in time $\mathcal{O}\left(R \cdot C \cdot 3^{\min\{R,C\}}\right)$ and has a memory consumption of $\mathcal{O}\left(R \cdot C \cdot 4^{\min\{R,C\}}\right)$ – which can be reduced to $\mathcal{O}\left(R \cdot C \cdot 3^{\min\{R,C\}}\right)$ at the expense of an extra $\log\left(R \cdot C \cdot 3^{\min\{R,C\}}\right)$ factor in the time complexity. Note that, even for a dense polyomino with no further restriction, we already have that $3^{\min\{R,C\}} \leq 3^{\sqrt{R \cdot C}}$ given that at least one of R or C has to be at most $\sqrt{R \cdot C}$.

Without loss of generality, suppose $C \leq R$ (if not, transpose g, as this will not affect the optimum). Append an extra empty row and an extra empty column to g, also adjusting $R \leftarrow R+1$ and $C \leftarrow C + 1$ accordingly.

We now characterize the solution through a forward dynamic programming recurrence. Throughout the following description, we write the set $\{0, 1, \ldots, u\}$, for any non-negative integer u, abbreviated as $[u]$. First, we define the dynamic programming state. Observe Figure 6.2 for a visual interpretation of our idea. Let $G(r, c, h_l, h_r, V_u, V_d)$ be the minimum number of rooks we need to cover the subpolyomino of P_m spanning from cell (r, c) $(0 \leq r < R, 0 \leq c < C)$ to cell $(R - 1, C - 1)$ (both cells inclusive), when traversing the grid g in row-major order, given that:

- $h_l \in \mathbb{B}$ ("*horizontal-left*"), where $\mathbb{B} = \{\mathsf{True}, \mathsf{False}\}$. h_l is True if and only if a rook has been placed on a cell (r, x), for some $0 \leq x < c$, in the current solution;
- $h_r \in \mathbb{B}$ ("*horizontal-right*"). h_r is True if and only if a rook must be placed on a cell (r, x), for some $c \leq x < C$, for the current solution to be valid;
- $V_u \subseteq [C-1]$ ("*vertical-up*") is a set such that, for any $0 \leq x < c$, $x \in V_u$ if and only if a rook has been placed on a cell (y, x), for some $0 \leq y \leq r$, and, for any $c \leq x < C$, $x \in V_u$ if and only if a rook has been placed on a cell (y, x), for some $0 \leq y < r$;
- $V_d \subseteq [C-1]$ ("*vertical-down*") is a set such that, for any $0 \leq x < c$, $x \in V_d$ if and only if a rook must be placed on a cell (y, x), for some $r < y < R$, and, for any $c \leq x < C$, $x \in V_d$ if and only if a rook must be placed on a cell (y, x), for some $r \leq y < R$.

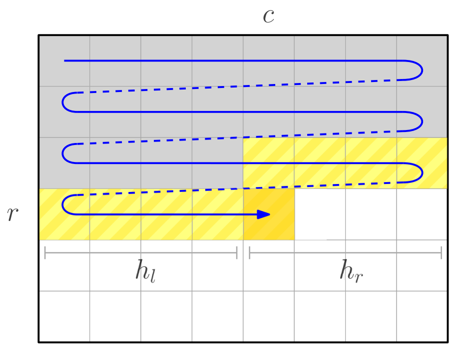


Figure 6.2: Traversing the grid g in row-major order as we keep the current dynamic programming state $(r, c, h_l, h_r, V_u, V_d)$, while standing at cell (r, c). The yellow, striped tiles indicate for which row ($r-1$ or r) each element $0 \leq x < C$ in V_u and V_d is carrying information from.

As can be understood from the previous definitions, if we are currently processing cell (r, c), the elements $0, 1, \ldots, c-1$ in V_u and V_d contain information from row r, while the most up-to-date information carried by elements $c, c+1, \ldots, C-1$ refers to row $r-1$ since these columns have not yet been visited on the current row. All the cells "before" our current position (in row-major order) – that is, all the cells in rows above r and all the cells in the current row to the left of (r, c) – have already been handled and the decision that was taken at them is deemed final.

Suppose we are currently processing cell (r, c) as the grid traversal takes place. When a rook guard is placed exactly on (r, c), above in the same column as (r, c) or to the left in the same row, it is clear that (r, c) is covered when we leave it. However, the cell (r, c) can also be guarded by a rook below or to the right of it; and we have no way of knowing whether we will opt to place a guard there in the future, given that we haven't reached those positions yet.

The idea behind introducing parameters h_r and V_d, which we call *certificate propagation*, provides a solution to this issue (Figure 6.3). Since we do not have a way of *knowing* our future decisions, we can *enforce* them by:

1. foretelling that a guard will be placed below or to the right;

2. propagating this requirement below or to the right;

3. either certifying the requirement by futurely placing a guard at an admissible spot, or failing (returning an impossibility flag, ∞) if we notice that the solution currently being built cannot possibly certify (guarantee) that requirement.

Also, certificate propagation is the reason why we began by appending an extra row and an extra column to the grid g. These extras act as sentinels, allowing us to keep validating the current solution, because every time we reach the end of a row or a column we always hit the boundary of the polyomino, which triggers the certificate validation as we will see.

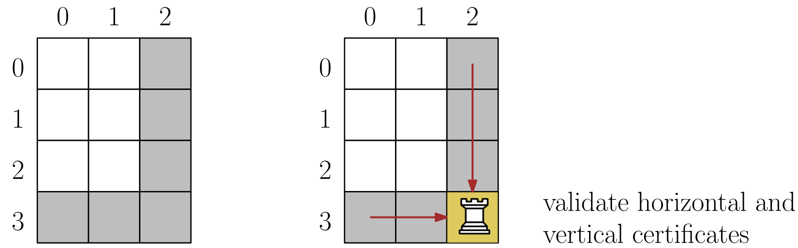


Figure 6.3: Certificate propagation. When we reach a dynamic programming state for this instance with $(r, c) = (3, 2)$, $h_l = \mathsf{False}$ and $V_u = \emptyset$, we must have $h_r = \mathsf{True}$ and $V_d = \{2\}$. Delaying the rook placement by propagating unchecked certificates from the top-right and bottom-left cells is the only way to achieve an optimal solution (1 rook) for this instance.

We now define the optimal solution recursively. The cases when $r = R$ or $c = C$ are boundary conditions and, respectively, either return a final answer for the base case (we need 0 guards to cover an empty polyomino) or transfer the subproblem to the next row (still adhering to the row-major order).

$$G(r, c, h_l, h_r, V_u, V_d) = \begin{cases} 0 & \text{if } r = R \\ G(r+1, 0, \mathsf{False}, \mathsf{False}, V_u, V_d) & \text{if } c = C \text{ (and } r \neq R) \end{cases} \tag{6.1}$$

If we are not at any of those boundary positions at the moment, we have two cases, according to the value of $g_{r,c}$:

- When $g_{r,c} = \mathsf{False}$ (we are outside of the polyomino), this is the right time to assert the validity of the current solution by ensuring that there are no pending certificates in the current row and column. If there are horizontal or vertical pending certificates, then we surely cannot validate them since we are outside the polyomino, and thus we return ∞ (signaling unfeasibility). Otherwise, we have no operation pending in this cell and simply move on to the next position in row-major order, also discontinuing the visibility of all the rooks that have been placed above or to the left of (r, c).

$$G(r, c, h_l, h_r, V_u, V_d) = \begin{cases} \infty & \text{if } h_r = \mathsf{True} \text{ or } c \in V_d \\ G(r, c+1, \mathsf{False}, \mathsf{False}, V_u \setminus \{c\}, V_d) & \text{otherwise} \end{cases} \tag{6.2}$$

- If $g_{r,c} = \mathsf{True}$ (we are inside of the polyomino), we define:

$$G_1(r, c, h_l, h_r, V_u, V_d) = G(r, c+1, \mathsf{True}, \mathsf{False}, V_u \cup \{c\}, V_d \setminus \{c\}) + 1 \tag{6.3}$$

(place a guard at (r, c) and validate certificates)

$$G_2(r, c, h_l, h_r, V_u, V_d) = G(r, c+1, h_l, h_r, V_u, V_d) \tag{6.4}$$

(safely and inconsequentially skip to the next column)

$$G_3(r, c, h_l, h_r, V_u, V_d) = G(r, c+1, h_l, \mathsf{True}, V_u, V_d) \tag{6.5}$$

(propagate a certificate to the right)

$$G_4(r, c, h_l, h_r, V_u, V_d) = G(r, c+1, h_l, h_r, V_u, V_d \cup \{c\}) \tag{6.6}$$

(propagate a certificate downwards)

$$G_5(r, c, h_l, h_r, V_u, V_d) = G(r, c+1, h_l, \mathsf{True}, V_u, V_d \cup \{c\}) \tag{6.7}$$

(propagate certificates downwards and to the right)

We are then left with two final cases. If the current tile (r, c) has already been guarded by some guard on a cell (r, x) (for $0 \leq x < c$) or on a cell (y, c) (for $0 \leq y < r$) (that is, $h_l = \mathsf{True}$ or $c \in V_u$), then we choose the best option between placing a guard in (r, c) or not:

$$G(r, c, h_l, h_r, V_u, V_d) = \min_{i \in \{1,2\}} G_i(r, c, h_l, h_r, V_u, V_d) \tag{6.8}$$

Otherwise, we must ensure that this tile is guarded by some guard on a cell (r, x) (for $c \leq x < C$) or on a cell (y, c) (for $r \leq y < R$):

$$G(r, c, h_l, h_r, V_u, V_d) = \min_{i \in \{1,3,4,5\}} G_i(r, c, h_l, h_r, V_u, V_d) \tag{6.9}$$

The optimal number of guards k needed to guard P_m that we are interested in computing is then given by $G(0, 0, \mathsf{False}, \mathsf{False}, \emptyset, \emptyset)$. An actual minimum-cardinality guard set can also be determined without any penalty in the time complexity by recording the sequence of optimal choices taken at each visited subproblem.

We assume that, in the presented definitions of Equations (6.8) and (6.9), the algorithm considers valid alternatives in the order G_1, G_2, G_3, G_4, G_5. Hence, in case of a tie for the best choice, recurrence transitions favor guard sets that have rooks with lexicographically smaller coordinates. This implies that, given a solution $S^\star = \{(r_1^\star, c_1^\star), \ldots, (r_k^\star, c_k^\star)\}$ produced by the algorithm – where $(r_1^\star, c_1^\star) < \cdots < (r_k^\star, c_k^\star)$ – and another optimal solution $S' = \{(r_1', c_1'), \ldots, (r_k', c_k')\}$ – where $(r_1', c_1') < \cdots < (r_k', c_k')$ –, there is some $1 \leq i \leq k$ such that, for all $1 \leq j < i$, $(r_j^\star, c_j^\star) = (r_j', c_j')$ and $(r_i^\star, c_i^\star) < (r_i', c_i')$.

Proposition 6.2. *The optimal guard set found by the algorithm is lexicographically minimum among all the optimal solutions for the given instance P_m.*

Figures 6.1 and 6.4 contain lexicographically minimum rook-guard sets found by our algorithm.

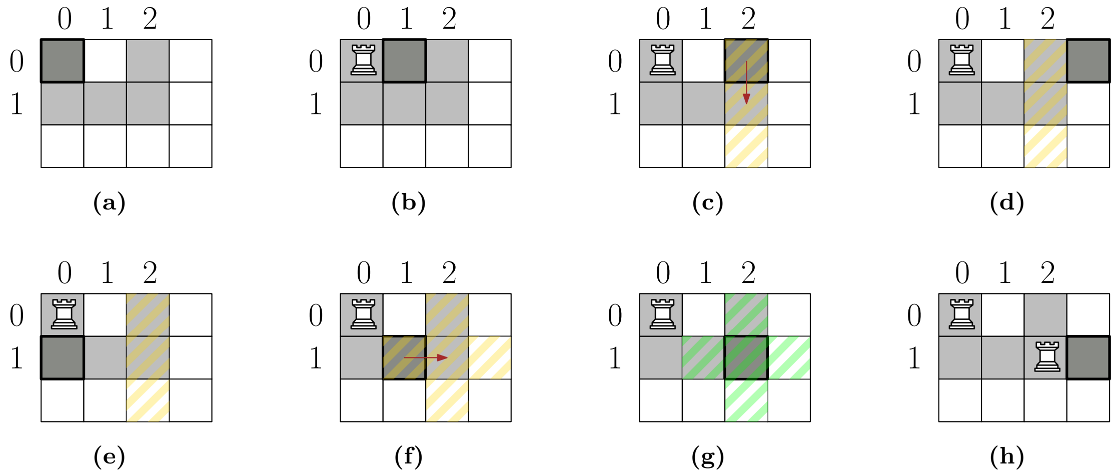


Figure 6.4: A run of the algorithm over a 5-polyomino P. The operations performed and the decisions taken at each state when obtaining one of the optimal solutions for P (in fact, the lexicographically minimum one) are shown. The bolder tile marks our current position at each step.

Example 6.1. Let us exemplify with Figure 6.4 how the algorithm runs for a given polyomino P. We sweep through the grid g in row-major order.

- **(a)** The recursive procedure is first called for the state $(0, 0, \textsf{False}, \textsf{False}, \emptyset, \emptyset)$. The algorithm chooses to place a guard at $(0, 0)$, thus we set $h_l = \textsf{True}$ and $V_u = \{0\}$ when transitioning to the next tile.

- **(b)** We move to the state $(0, 1, \textsf{True}, \textsf{False}, \{0\}, \emptyset)$, which corresponds to a tile outside of P. In this tile, we have nothing to do, so we simply move to the next position and break the visibility of the guard placed at $(0, 0)$ – by setting h_l back to $\textsf{False}$ – in order to prevent it from seeing the remaining cells of the row 0.

- **(c)** The next state to consider is $(0, 2, \textsf{False}, \textsf{False}, \{0\}, \emptyset)$. Here, we move back into P. In order to ensure that the current cell is seen, the algorithm can choose between placing a guard there or propagating a certificate downwards. In this case, it chooses to propagate a certificate, by inserting 2 into V_d; to represent this demand, column 2 appears striped.

- **(d)** We are now at $(0, 3, \textsf{False}, \textsf{False}, \{0\}, \{2\})$, which corresponds to the extra column we have appended to g before running the algorithm. Since tile $(0, 3)$ is outside of P, we move on to $(0, 4, \textsf{False}, \textsf{False}, \{0\}, \{2\})$, which gets transparently forwarded to the next row, $(1, 0, \textsf{False}, \textsf{False}, \{0\}, \{2\})$, by the base case of the recurrence.

- **(e)** The current cell $(1, 0)$ is already guarded by the rook above, so the algorithm decides to continue right.

- **(f)** The cell corresponding to $(1, 1, \textsf{False}, \textsf{False}, \{0\}, \{2\})$ is not guarded yet, but, instead of placing a guard at $(1, 1)$, the algorithm opts to propagate a certificate to the right (by setting $h_r = \textsf{True}$). It does this because it recursively determines that a single guard is able to optimally cover the uncovered section of P.

(g) We then arrive at the state $(1, 2, \mathsf{False}, \mathsf{True}, \{0\}, \{2\})$. By placing a guard at the cell $(1, 2)$, both vertical and horizontal pending certificates get validated, and so we can move right.

(h) The recursion hits the extra column again and confirms that $h_r = \mathsf{False}$, thus no pending certificates are left to be validated. The algorithm continues through the whole bottom row, looking for pending certificates. Since the entire polygon has already been guarded at this point, no pending certificates will be found and the algorithm will return the lexicographically minimum optimal solution $\{(0, 0), (1, 2)\}$.

6.1.1 Complexity

Theorem 6.3. *The algorithm has time complexity* $\widetilde{\mathcal{O}}\left(R \cdot C \cdot 3^{\min\{R,C\}}\right)$ *and space complexity* $\mathcal{O}\left(R \cdot C \cdot 3^{\min\{R,C\}}\right)$*, where* $\widetilde{\mathcal{O}}(\cdot)$ *hides a polylogarithmic factor.*

Proof. Again, assume that $C \leq R$ (otherwise, rotate the polyomino 90° before running the algorithm). Each dynamic programming transition is done in $\mathcal{O}(1)$ time and there seems to be an upper bound of $R \cdot C \cdot 2 \cdot 2 \cdot 2^C \cdot 2^C$ states. At first sight it may then seem that the time complexity of the algorithm is $\mathcal{O}\left(R \cdot C \cdot 4^C\right)$. While this analysis is not wrong, we can actually prove a tighter bound. In fact, observe that, at any moment, $V_d \subseteq \overline{V_u} = [C-1] \setminus V_u$, because we can only validate pending certificates for unguarded segments of columns. Specifically, every time we place a guard in a cell (r, c), we erase c from V_d (if it exists) and insert it into V_u (if it doesn't exist already). This property implies that the total number of states visited when considering a fixed position (r, c) is not actually $2^C \times 2^C = 4^C$, but instead

$$\sum_{s=0}^{C} \binom{C}{s} \cdot 2^{C-s} = \sum_{s=0}^{C} \binom{C}{s} \cdot 2^s = (1+2)^C = 3^C \tag{6.10}$$

by the binomial theorem and Pascal's triangle row symmetry. In other words, there is a bijection between each V_u and $\overline{V_u}$; for each $0 \leq s \leq C$, there are $\binom{C}{s}$ subsets of $[C-1]$ with size s and each of them has 2^s sub-subsets. As such, given that the dynamic programming transitions are performed in constant time, the algorithm's time complexity is $\mathcal{O}\left(R \cdot C \cdot 3^C\right)$.

The space taken as it is remains $\mathcal{O}\left(R \cdot C \cdot 4^C\right)$, though, if one explicitly initializes a full memoization table right from the start. For reducing it to $\mathcal{O}\left(R \cdot C \cdot 3^C\right)$, one can store only the solution for all the visited states using, for instance, a dictionary-like structure which allows for performing insertions and searches in time logarithmic in its size. This penalizes the time complexity with an extra factor of $\log\left(R \cdot C \cdot 3^C\right) = \log R + \log C + C \log 3$, but in turn greatly reduces the memory required for the algorithm to run. □

Taking these results into account, we may finally conclude that:

Theorem 6.4. *Rook-AGP is fixed-parameter tractable (FPT) parameterized by* $\min\{R, C\}$.

Chapter 7

Conclusions and Future Work

It all began with Klee making a simple question of how many guards would be required for watching over the interior of a polygonal art gallery with n vertices. As quickly as new algorithms for the Art Gallery Problem are invented, novel interesting variants of the problem do also appear, as well as results on their hardness and inapproximability – and we have carried on with this quest. We now summarize our contributions with this work and discuss our main results.

In the first main half of this thesis, we have presented a randomized algorithm for finding approximate solutions for SET COVER and used it to solve MINIMUM VERTEX GUARD (MVG) on orthogonal polygons under straight-line visibility. It has been observed how simple it is to solve uniformly random instances with probability approaching 1, and how interleaving kernelization and randomized phases contributes to improving both the approximation ratio and the running time of RANDOMIZED-SET-COVER. We developed time- and memory-efficient data structures, namely sampleable set and bitset tree, which consist, respectively, of a set that supports the operation CHOOSE-RANDOM (for implementing CANDIDATES) and a compact self-balancing binary search tree of bitmasks (for representing sets in $\mathcal{F}$). The ideas used in their construction allowed significant speed gains to be achieved by exploiting bit-level parallelism and maximizing cache usage when operating with sparse sets in memory.

Experimental tests against a dataset of three classes of grid n-ogons – MIN-AREAS, FATS and RANDOMS – proved RANDOMIZED-SET-COVER to be highly performant. The algorithm has found an optimal guard set for every MIN-AREA and FAT instance and exhibited low approximation ratio on every RANDOM instance from the dataset, having beaten the guarantees of both GREEDY-SET-COVER and LP-rounding methods. What the experiments suggest is that kernelization compensates for possible mistakes that random set removals may incur in, always pushing the algorithm towards optimality.

While the effective solving time was low, the execution time was observed to be mostly dominated by the preprocessing phase, specifically polygon discretization. A possible way of making up for this bottleneck in the future could be to explore lazy partitioning strategies, instead of building an entire decomposition right from the start. Moreover, we think it would be fruitful to experiment with refined heuristics for dynamically re-weighting candidate sets to be removed, as an alternative to the plain uniform selection. We believe that, by exploring these ideas, we could improve both the running time and approximation factor of RANDOMIZED-SET-COVER even further. Despite the focus having been given to orthogonal polygons, the techniques developed in this thesis for RANDOMIZED-SET-COVER naturally extrapolate to general polygons, under any visibility model. Based on the experimental results obtained, we observed a seemingly linear growth of the average number of pieces in the convex partition of a RANDOM grid n-ogon, proved an exact, also linear bound for MIN-AREAS and formulated a conjecture for FATS. Given that the combinatorial structure of these polygons is well behaved, we are convinced that there exists a very straightforward proof for this conjecture.

In the second half of this work, we have defined a new family of orthogonal polygons, the SCOTs. Their topology encompasses many properties that we usually see on real-world art galleries, which enable us to develop efficient algorithms for r-guarding them. We have proved that, if the SCOT is simple or r-independent, the MINIMUM SCOT r-GUARD problem is in P, and have given three methods for solving these cases. First, we described a linear-time algorithm for simple SCOTs, with both vertex- and point-guards, based on a tree decomposition of the polygon. We then presented one that runs in time $\mathcal{O}\left(|\mathcal{C}|\sqrt{\min\{|\mathcal{R}|, |\mathcal{C}'|\}}\right)$ for r-independent $(\mathcal{R}, \mathcal{C}, \mathcal{C}')$-SCOTs with vertex-guards, based on bipartite matchings. Finally, another one was given for r-independent SCOTs with point-guards, running in time $\mathcal{O}\left(T(G') \log |f^\star|\right)$, where $T(G')$ is the time that it takes to compute a maximum flow in a network G' that we have defined, and $|f^\star|$ is the optimal number of guards, based on a reduction to MINIMUM FLOW WITH DEMANDS.

On the contrary, should the SCOT have holes and not be r-independent, we have shown that the problem becomes NP-hard – indeed NP-complete for the case of point-guards – by reducing from MINIMUM POLYOMINO r-GUARD. MINIMUM SCOT r-GUARD with vertex-guards is also in NP, because any solution is a subset of the vertices of the SCOT, however whether it remains NP-hard is left as an open question. Finally, we have also given $\mathcal{O}(\log |\mathcal{C}|)$-factor approximation algorithms for unrestricted SCOTs with both vertex-guards and point-guards. We leave open whether the problem is hard to approximate with constant factor, either for point- or vertex-guards – and this seems as well a very promising road to continue work in the future. Another attractive direction is to understand the degree to which these ideas can be generalized to other polygon families, also formed by convex rooms and corridors, but not necessarily orthogonal.

At last, we have shown a result on the fixed-parameter tractability of the problem of guarding polyominoes, possibly having holes, with chess rooks, by giving a dynamic programming algorithm with complexity $\widetilde{\mathcal{O}}(R \cdot C \cdot 3^{\min\{R,C\}})$, where $R \times C$ are the dimensions of the minimal unit grid enclosing the polyomino. The solution returned by the algorithm is optimal and guaranteed to be lexicographically minimum among all the optimal solutions. It would be interesting to research alternative structural properties that could as well be useful parameters in practice and to extend these ideas on rook-guards to queen-guards.

Bibliography

[1] Mikkel Abrahamsen, Anna Adamaszek, and Tillmann Miltzow. Irrational Guards are Sometimes Needed. In *33rd International Symposium on Computational Geometry, SoCG 2017*, volume 77 of *LIPIcs*, pages 3:1–3:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.SoCG.2017.3.

[2] Mikkel Abrahamsen, Anna Adamaszek, and Tillmann Miltzow. The Art Gallery Problem is $\exists\mathbb{R}$-complete. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018*, pages 65–73. ACM, 2018. doi:10.1145/3188745.3188868.

[3] M. G. Adelson-Velskii and E. M. Landis. An Algorithm for the Organization of Information. Technical report, Joint Publications Research Service Washington DC, 1963.

[4] Akanksha Agrawal, Kristine V. K. Knudsen, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. The Parameterized Complexity of Guarding Almost Convex Polygons. In *36th International Symposium on Computational Geometry, SoCG 2020*, volume 164 of *LIPIcs*, pages 3:1–3:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.SoCG.2020.3.

[5] Martin Aigner and Günter M. Ziegler. *How to guard a museum*, pages 281–284. Springer Berlin Heidelberg, Berlin, Heidelberg, 2018. ISBN: 978-3-662-57265-8. doi:10.1007/978-3-662-57265-8_40.

[6] D. Alciatore and Rick Miranda. A Winding Number and Point-in-Polygon Algorithm. *Glaxo Virtual Anatomy Project Research Report, Department of Mechanical Engineering, Colorado State University*, 1995.

[7] Hannah Alpert and Érika Roldán. Art Gallery Problem with Rook and Queen Vision. *Graphs and Combinatorics*, 37(2):621–642, 2021. doi:10.1007/s00373-020-02272-8.

[8] Boris Aronov, Leonidas J. Guibas, Marek Teichmann, and Li Zhang. Visibility Queries and Maintenance in Simple Polygons. *Discrete and Computational Geometry*, 27(4):461–483, 2002. doi:10.1007/s00454-001-0089-9.

[9] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof Verification and Hardness of Approximation Problems. In *33rd Annual Symposium on*

Foundations of Computer Science, FOCS 1992, pages 14–23. IEEE Computer Society, 1992. doi:10.1109/SFCS.1992.267823.

[10] David Avis and Godfried T. Toussaint. An Efficient Algorithm for Decomposing a Polygon into Star-Shaped Polygons. *Pattern Recognition*, 13(6):395–398, 1981. doi:10.1016/0031-3203(81)90002-9.

[11] António Leslie Bajuelos, Ana Paula Tomás, and Fábio Marques. Partitioning Orthogonal Polygons by Extension of All Edges Incident to Reflex Vertices: Lower and Upper Bounds on the Number of Pieces. In *Computational Science and Its Applications - ICCSA 2004, International Conference*, volume 3045 of *LNCS*, pages 127–136. Springer, 2004. doi:10.1007/978-3-540-24767-8_14.

[12] Jon Louis Bentley and Thomas A. Ottmann. Algorithms for Reporting and Counting Geometric Intersections. *IEEE Transactions on computers*, 28(09):643–647, 1979. doi:10.1109/TC.1979.1675432.

[13] Pritam Bhattacharya, Subir Kumar Ghosh, and Sudebkumar Pal. Constant Approximation Algorithms for Guarding Simple Polygons Using Vertex Guards. *arXiv preprint arXiv:1712.05492*, abs/1712.05492, 2017.

[14] Therese Biedl and Saeed Mehrabi. On r-Guarding Thin Orthogonal Polygons. *arXiv preprint arXiv:1604.07100*, 2016. (extended version of ISAAC'2016).

[15] Therese Biedl and Saeed Mehrabi. On Orthogonally Guarding Orthogonal Polygons with Bounded Treewidth. *Algorithmica*, 83(2):641–666, 2021. doi:10.1007/s00453-020-00769-5.

[16] Therese Biedl, Mohammad T. Irfan, Justin Iwerks, Joondong Kim, and Joseph S. B. Mitchell. The Art Gallery Theorem for Polyominoes. *Discrete & Computational Geometry*, 48(3):711–720, 2012. doi:10.1007/s00454-012-9429-1.

[17] Therese C. Biedl and Saeed Mehrabi. On r-Guarding Thin Orthogonal Polygons. In Seok-Hee Hong, editor, *27th International Symposium on Algorithms and Computation, ISAAC 2016*, volume 64 of *LIPIcs*, pages 17:1–17:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.ISAAC.2016.17.

[18] Therese C. Biedl, Mohammad Tanvir Irfan, Justin Iwerks, Joondong Kim, and Joseph S. B. Mitchell. Guarding Polyominoes. In *Proceedings of the 27th ACM Symposium on Computational Geometry, SoCG 2011*, pages 387–396. ACM, 2011. doi:10.1145/1998196.1998261.

[19] Burton H. Bloom. Space/Time Trade-offs in Hash Coding with Allowable Errors. *Communications of the ACM*, 13(7):422–426, 1970. doi:10.1145/362686.362692.

[20] Édouard Bonnet and Tillmann Miltzow. Parameterized Hardness of Art Gallery Problems. In *24th Annual European Symposium on Algorithms, ESA 2016*, volume 57 of *LIPIcs*, pages 19:1–19:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.ESA.2016.19.

[21] Édouard Bonnet and Tillmann Miltzow. An Approximation Algorithm for the Art Gallery Problem. In *33rd International Symposium on Computational Geometry, SoCG 2017*, volume 77 of *LIPIcs*, pages 20:1–20:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.SoCG.2017.20.

[22] Robert King Brayton. On the Asymptotic Behavior of the Number of Trials Necessary to Complete a Set with Random Selection. *Journal of Mathematical Analysis and Applications*, 7(1):31–61, 1963. doi:10.1016/0022-247X(63)90076-3.

[23] Björn Brodén, Mikael Hammar, and Bengt J. Nilsson. Guarding Lines and 2-Link Polygons is APX-hard. In *Proceedings of the 13th Canadian Conference on Computational Geometry (CCCG'01)*, pages 45–48, 2001.

[24] Hervé Brönnimann and Michael T. Goodrich. Almost Optimal Set Covers in Finite VC-dimension. *Discrete & Computational Geometry*, 14(4):463–479, 1995. doi:10.1007/BF02570718.

[25] Francisc Bungiu, Michael Hemmer, John Hershberger, Kan Huang, and Alexander Kröller. Efficient Computation of Visibility Polygons. In *Proceedings of the 30th European Workshop on Computational Geometry (EuroCG 2014)*, 2014.

[26] Constantin Carathéodory. Über den Variabilitätsbereich der Fourier'schen Konstanten von positiven harmonischen Funktionen. *Rendiconti Del Circolo Matematico di Palermo (1884-1940)*, 32(1):193–217, 1911. doi:10.1007/BF03014795.

[27] Bernard Chazelle. Triangulating a Simple Polygon in Linear Time. *Discrete & Computational Geometry*, 6(3):485–524, 1991. doi:10.1007/BF02574703.

[28] Li Chen, Rasmus Kyng, Yang P. Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. Maximum Flow and Minimum-Cost Flow in Almost-Linear Time. *arXiv preprint arXiv:2203.00671*, 2022. doi:10.48550/arXiv.2203.00671.

[29] Vasek Chvátal. A Combinatorial Theorem in Plane Geometry. *Journal of Combinatorial Theory, Series B*, 18(1):39–41, 1975.

[30] V. Chvátal. A Greedy Heuristic for the Set-Covering Problem. *Mathematics of Operations Research*, 4(3):233–235, 1979. ISSN: 0364765X, 15265471. doi:10.1287/moor.4.3.233.

[31] Kyung-Yong Chwa, Byung-Cheol Jo, Christian Knauer, Esther Moet, René Van Oostrum, and Chan-Su Shin. Guarding Art Galleries by Guarding Witnesses. *International Journal of Computational Geometry & Applications*, 16(02n03):205–226, 2006. doi:10.1142/S0218195906002002.

[32] W. contributors. Find first set | Hardware support — Wikipedia, The Free Encyclopedia, 2022. (Accessed on 16/06/2022).

[33] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT press, 3rd edition, 2009.

[34] Marcelo C. Couto, Cid C. de Souza, and Pedro Jussieu de Rezende. An Exact and Efficient Algorithm for the Orthogonal Art Gallery Problem. In *SIBGRAPI 2007, Proceedings of the XX Brazilian Symposium on Computer Graphics and Image Processing*, pages 87–94. IEEE Computer Society, 2007. doi:10.1109/SIBGRAPI.2007.15.

[35] Marcelo C. Couto, Cid C. de Souza, and Pedro Jussieu de Rezende. Experimental Evaluation of an Exact Algorithm for the Orthogonal Art Gallery Problem. In Catherine C. McGeoch, editor, *Experimental Algorithms, 7th International Workshop, WEA 2008, Proceedings*, volume 5038 of *LNCS*, pages 101–113. Springer, 2008. doi:10.1007/978-3-540-68552-4_8.

[36] Marcelo C. Couto, Pedro J. de Rezende, and Cid C. de Souza. Instances for the Art Gallery Problem, 2009. https://www.ic.unicamp.br/~cid/Problem-instances/Art-Gallery (Accessed on 07/06/2022).

[37] Pierluigi Crescenzi. A Short Guide to Approximation Preserving Reductions. In *Proceedings of Computational Complexity. Twelfth Annual IEEE Conference*, pages 262–273. IEEE, 1997. doi:10.1109/CCC.1997.612321.

[38] Yefim Dinitz. Dinitz' Algorithm: The Original Version and Even's Version. In Oded Goldreich, Arnold L. Rosenberg, and Alan L. Selman, editors, *Theoretical Computer Science, Essays in Memory of Shimon Even*, volume 3895 of *LNCS*, pages 218–240. Springer, 2006. doi:10.1007/11685654_10.

[39] Irit Dinur and David Steurer. Analytical Approach to Parallel Repetition. In *Symposium on Theory of Computing, STOC 2014*, pages 624–633. ACM, 2014. doi:10.1145/2591796.2591884.

[40] Stephan Eidenbenz, Christoph Stamm, and Peter Widmayer. Inapproximability Results for Guarding Polygons and Terrains. *Algorithmica*, 31(1):79–113, 2001. doi:10.1007/s00453-001-0040-8.

[41] Stephan J. Eidenbenz. Inapproximability Results for Guarding Polygons without Holes. In *Algorithms and Computation, 9th International Symposium, ISAAC '98, Taejon, Korea, December 14-16, 1998, Proceedings*, volume 1533 of *LNCS*, pages 427–436. Springer, 1998. doi:10.1007/3-540-49381-6_45.

[42] Hossam El Gindy. Visibility in Polygons with Applications. Master's thesis, McGill University, 1980.

[43] Hossam El Gindy and David Avis. A Linear Algorithm for Computing the Visibility Polygon from a Point. *Journal of Algorithms*, 2(2):186–197, 1981. doi:10.1016/0196-6774(81)90019-5.

[44] Jeff Erickson. Algorithms – Lecture 25: Extensions of Maximum Flow, 2015. (Lecture notes).

[45] Jeff Erickson. Algorithms – F. Balances and Pseudoflows, 2020. (Lecture notes).

[46] Vladimir Estivill-Castro and Jorge Urrutia. Optimal Floodlight Illumination of Orthogonal Art Galleries. In *Proceedings of the 6th Canadian Conference on Computational Geometry (CCCG'94)*, pages 81–86. University of Saskatchewan, 1994.

[47] Uriel Feige. A Threshold of $\ln n$ for Approximating Set Cover. *Journal of the ACM (JACM)*, 45(4):634–652, 1998. doi:10.1145/285055.285059.

[48] Ulrich Finke and Klaus H. Hinrichs. Overlaying Simply Connected Planar Subdivisions in Linear Time. In *Proceedings of the Eleventh Annual Symposium on Computational Geometry, SoCG 1995*, page 119–126, New York, NY, USA, 1995. ACM. ISBN: 0897917243. doi:10.1145/220279.220292.

[49] Steve Fisk. A Short Proof of Chvátal's Watchman Theorem. *Journal of Combinatorial Theory, Series B*, 24(3):374, 1978.

[50] Efi Fogel and Monique Teillaud. The Computational Geometry Algorithms Library CGAL. *ACM Communications in Computer Algebra*, 49(1):10–12, 2015. doi:10.1145/2768577.2768579.

[51] Edward Fredkin. Trie Memory. *Communications of the ACM*, 3(9):490–499, 1960. doi:10.1145/367390.367400.

[52] Mohsen Ghaffari. *Advanced Algorithms*. Computer Science, ETH Zürich, 2020.

[53] Subir Kumar Ghosh. Approximation Algorithms for Art Gallery Problems in Polygons. *Discrete Applied Mathematics*, 158(6):718–722, 2010. doi:10.1016/j.dam.2009.12.004.

[54] Leonidas J. Guibas and Robert Sedgewick. A Dichromatic Framework for Balanced Trees. In *19th Annual Symposium on Foundations of Computer Science (sfcs 1978)*, pages 8–21. IEEE Computer Society, 1978. doi:10.1109/SFCS.1978.3.

[55] Leonidas J. Guibas and Jorge Stolfi. Primitives for the Manipulation of General Subdivisions and the Computation of Voronoi Diagrams. *ACM transactions on graphics (TOG)*, 4(2): 74–123, 1985. doi:10.1145/282918.282923.

[56] Philip Hall. On Representatives of Subsets. *Classic Papers in Combinatorics*, pages 58–62, 1987. doi:10.1007/978-0-8176-4842-8_4.

[57] Dorit S. Hochbaum. Approximation Algorithms for the Set Covering and Vertex Cover Problems. *SIAM Journal on computing*, 11(3):555–556, 1982. doi:10.1137/0211045.

[58] Hamid Hoorfar and Alireza Bagheri. Guarding Path Polygons with Orthogonal Visibility. *arXiv preprint arXiv:1709.01569*, 2017.

[59] John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ Algorithm for Maximum Matchings in Bipartite Graphs. *SIAM Journal on computing*, 2(4):225–231, 1973. doi:10.1137/0202019.

[60] Russell Impagliazzo and Ramamohan Paturi. On the Complexity of k-SAT. *Journal of Computer and System Sciences*, 62(2):367–375, 2001. doi:10.1006/jcss.2000.1727.

[61] Chuzo Iwamoto and Toshihiko Kume. Computational Complexity of the r-visibility Guard Set Problem for Polyominoes. In *Discrete and Computational Geometry and Graphs - 16th Japanese Conference, JCDCGG 2013, Revised Selected Papers*, volume 8845 of *LNCS*, pages 87–95. Springer, 2013. doi:10.1007/978-3-319-13287-7_8.

[62] Barry Joe and Richard B Simpson. Corrections to Lee's Visibility Polygon Algorithm. *BIT Numerical Mathematics*, 27(4):458–473, 1987. doi:10.1007/BF01937271.

[63] David S. Johnson. Approximation Algorithms for Combinatorial Problems. *Journal of Computer and System Sciences*, 9(3):256–278, 1974. doi:10.1016/S0022-0000(74)80044-9.

[64] Jeff Kahn, Maria Klawe, and Daniel Kleitman. Traditional Galleries Require Fewer Watchmen. *SIAM Journal on Algebraic Discrete Methods*, 4(2):194–206, 1983.

[65] Viggo Kann. *On the Approximability of* NP*-complete Optimization Problems*. PhD thesis, Department of Numerical Analysis and Computing Science, KTH Royal Institute of Technology, 1992.

[66] Richard M. Karp. Reducibility Among Combinatorial Problems. In *Complexity of Computer Computations*, pages 85–103. Springer, 1972. doi:10.1007/978-1-4684-2001-2_9.

[67] Matthew J. Katz. A PTAS for Vertex Guarding Weakly-Visible Polygons – An Extended Abstract. *arXiv preprint arXiv:1803.02160*, 2018.

[68] Matthew J. Katz and Gabriel S. Roisman. On Guarding the Vertices of Rectilinear Domains. *Computational Geometry*, 39(3):219–228, 2008.

[69] James King and David Kirkpatrick. Improved Approximation for Guarding Simple Galleries from the Perimeter. *Discrete & Computational Geometry*, 46(2):252–269, 2011. doi:10.1007/s00454-011-9352-x.

[70] Jon Kleinberg and Éva Tardos. *Algorithm Design*. Pearson Education, Addison-Wesley, 2006.

[71] Ali A. Kooshesh and Bernard M. E. Moret. Three-Coloring the Vertices of a Triangulated Simple Polygon. *Pattern Recognit.*, 25(4):443, 1992. doi:10.1016/0031-3203(92)90093-X.

[72] Erik A. Krohn and Bengt J. Nilsson. Approximate Guarding of Monotone and Rectilinear Polygons. *Algorithmica*, 66(3):564–594, 2013. doi:10.1007/s00453-012-9653-3.

[73] D. T. Lee and Arthur K. Lin. Computational Complexity of Art Gallery Problems. *IEEE Transactions on Information Theory*, 32(2):276–282, 1986. doi:10.1109/TIT.1986.1057165.

[74] Der-Tsai Lee. Visibility of a Simple Polygon. *Computer Vision, Graphics, and Image Processing*, 22(2):207–221, 1983. doi:10.1016/0734-189X(83)90065-8.

[75] Andrzej Lingas, Agnieszka Wasylewicz, and Pawel Zylinski. Linear-Time 3-Approximation Algorithm for the r-Star Covering Problem. In Shin-Ichi Nakano and Md. Saidur Rahman,

editors, *Algorithms and Computation, 2nd International Workshop, WALCOM 2008*, volume 4921 of *LNCS*, pages 157–168. Springer, 2008. doi:10.1007/978-3-540-77891-2_15.

[76] Carsten Lund and Mihalis Yannakakis. On the Hardness of Approximating Minimization Problems. *Journal of the ACM (JACM)*, 41(5):960–981, 1994. doi:10.1145/185675.306789.

[77] Ana Mafalda Martins and António Leslie Bajuelos. Characterizing and Covering Some Subclasses of Orthogonal Polygons. In *International Conference on Computational Science*, pages 255–262. Springer, 2006. doi:10.1007/11758525_34.

[78] Makoto Matsumoto and Takuji Nishimura. Mersenne Twister: a 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 8(1):3–30, 1998. doi:10.1145/272991.272995.

[79] David E. Muller and Franco P. Preparata. Finding the Intersection of Two Convex Polyhedra. *Theoretical Computer Science*, 7(2):217–236, 1978. doi:10.1016/0304-3975(78)90051-8.

[80] Robert Nürnberg. Calculating the Area and Centroid of a Polygon in 2D, 2013. (Lecture notes).

[81] Joseph O'Rourke. *Art Gallery Theorems and Algorithms*, volume 57. Oxford New York, NY, USA, 1987.

[82] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Courier Corporation, 1998.

[83] Dietmar Schuchardt and Hans-Dietrich Hecker. Two NP-Hard Art-Gallery Problems for Ortho-Polygons. *Mathematical Logic Quarterly*, 41(2):261–267, 1995. doi:10.1002/malq.19950410212.

[84] Ilya Sergey. Experience Report: Growing and Shrinking Polygons for Random Testing of Computational Geometry Algorithms. In *Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming, ICFP 2016*, pages 193–199. ACM, 2016. doi:10.1145/2951913.2951927.

[85] Michael Ian Shamos. *Problems in Computational Geometry*. Carnegie-Mellon University, Pittsburgh, 1975.

[86] Moshe Shimrat. Algorithm 112: Position of Point Relative to Polygon. *Communications of the ACM*, 5(8):434, 1962. doi:10.1145/368637.368653.

[87] Petr Slavık. A Tight Analysis of the Greedy Algorithm for Set Cover. *Journal of Algorithms*, 25(2):237–254, 1997. doi:10.1006/jagm.1997.0887.

[88] Emanuel Sperner. Ein Satz über Untermengen einer endlichen Menge. *Mathematische Zeitschrift*, 27(1):544–548, 1928. doi:10.1007/BF01171114.

[89] James Stirling. *Methodus Differentialis: Sive, Tractatus de Summatione et Interpolatione Serierum Infinitarum*. Impensis Ric. Manby, London, 1730.

[90] Ana Paula Tomás. Análise de Alguns Problemas Geométricos em Polıgonos Ortogonais. Technical report, Departamento de Ciência de Computadores, Faculdade de Ciências da Universidade do Porto, 2004. (Accessed on 12/06/2022).

[91] Ana Paula Tomás. Guarding Thin Orthogonal Polygons Is Hard. In *Fundamentals of Computation Theory - 19th International Symposium, FCT 2013, Proceedings*, volume 8070 of *LNCS*, pages 305–316. Springer, 2013. doi:10.1007/978-3-642-40164-0_29.

[92] Ana Paula Tomás and António Leslie Bajuelos. Generating Random Orthogonal Polygons. In *Current Topics in Artificial Intelligence CAEPIA 2003 and 5th Conf. on Technology Transfer, TTIA 2003. Revised Selected Papers*, volume 3040 of *LNCS*, pages 364–373. Springer, 2003. doi:10.1007/978-3-540-25945-9_36.

[93] Ana Paula Tomás, António Leslie Bajuelos, and Fábio Marques. Approximation Algorithms to Minimum Vertex Cover Problems on Polygons and Terrains. In *Computational Science - ICCS 2003, International Conference*, volume 2657 of *LNCS*, pages 869–878. Springer, 2003. doi:10.1007/3-540-44860-8_90.

[94] Ana Paula Tomás, António Leslie Bajuelos, and Fábio Marques. On Visibility Problems in the Plane – Solving Minimum Vertex Guard Problems by Successive Approximations. In *International Symposium on Artificial Intelligence and Mathematics, AI&Math 2006*, 2006.

[95] Csaba D. Toth, Joseph O'Rourke, and Jacob E. Goodman. *Handbook of Discrete and Computational Geometry*. CRC Press, 2017.

[96] Davi C. Tozoni, Pedro J. de Rezende, and Cid C. de Souza. The Quest for Optimal Solutions for the Art Gallery Problem: A Practical Iterative Algorithm. In *Experimental Algorithms, 12th International Symposium, SEA 2013, Proceedings*, volume 7933 of *LNCS*, pages 320–336. Springer, 2013. doi:10.1007/978-3-642-38527-8_29.

[97] Vladimir N. Vapnik and A. Ya. Chervonenkis. On the Uniform Convergence of Relative Frequencies of Events to Their Probabilities. *Theory of Probability and its Applications*, 16 (2):264–280, 1971. doi:10.1137/1116025.

[98] Vijay V. Vazirani. *Approximation Algorithms*, volume 1. Springer, 2001.

[99] Giovanni Viglietta. Lesson 9. The Art Gallery Problem — I628E – Information Processing Theory. Online, January 2020. (Accessed on 20/05/2022).

[100] Ron Wein, Eric Berberich, Efi Fogel, Dan Halperin, Michael Hemmer, Oren Salzman, and Baruch Zukerman. 2D Arrangements. In *CGAL User and Reference Manual*. CGAL Editorial Board, 5.4.1 edition, 2022. (Accessed on 03/02/2022).

[101] David P. Williamson and David B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011.

[102] Chris Worman and J. Mark Keil. Polygon Decomposition and the Orthogonal Art Gallery Problem. *International Journal of Computational Geometry & Applications*, 17(02):105–138, 2007. doi:10.1142/S0218195907002264.