Single-crossing Orthogonal Axial Lines in Orthogonal Rectangles

Berhane Semere Mengisteab

A dissertation submitted to the Faculty of Science, University of the Witwatersrand, Johannesburg, in fulfilment of the requirements for the degree of Master of Science

Johannesburg, 2007

Declaration

I declare that this dissertation is my own, unaided work. It is being submitted for the degree of Master of Science in the University of the Witwatersrand, Johannesburg. It has not been submitted before for any degree or examination in any other university.

Berhane Semere Mengisteab
17^{th} of October 2007

Abstract

The axial map of a town is one of the key components of the space syntax method – a tool for analysing urban layout. It is derived by placing the longest and fewest lines, called axial lines, to cross the adjacencies between convex polygons in a convex map of a town. Previous research has shown that placing axial lines to cross the adjacencies between a collection of convex polygons is NP-complete, even when the convex polygons are restricted to rectangles and the axial lines to have orthogonal orientation.

In this document, we show that placing orthogonal axial lines in orthogonal rectangles where the adjacencies between the rectangles are restricted to be crossed only once (ALP-SC-OLOR) is NP-complete. As a result, we infer the single adjacency crossing version of the general axial line placement problem is NP-complete. The transformation of NP-completeness of ALP-SC-OLOR is from vertex cover for biconnected planar graphs. A heuristic is then presented that gives a reasonable approximate solution to ALP-SC-OLOR based on a greedy method.

Acknowledgements

I would like to thank my supervisor, Prof. Ian Sanders, for his constructive advice and guidance. I am also grateful to members of my supervisory committee, Prof. Scott Hazelhurst and Mr. Brynn Andrew, for their valuable comments and suggestions. I would also like to take this opportunity to thank my family for their support and encouragement throughout my learning journey. Finally, my thanks are due to Seare Araya, my friends, and the academic and administration staff of the School of Computer Science for all their support.

Contents

List of Figures

List of Algorithms

List of Tables

Chapter 1

Introduction

1.1 Introduction to the Problem

Computational geometry is a discipline that deals with the algorithmic aspects of geometrical problems. Most geometric problems are defined for *two*-dimensional space as it is a simplified form of the *n*-dimensional space. Art gallery and visibility problems are among the well known geometric problems that are defined in a *two*-dimensional space and have been the focus of research in the field of computational geometry. Another problem in the field which was introduced recently is the axial line placement (ALP) problem. Sanders [2002] introduced the axial line placement problem in attempting to automate the space syntax method [Hillier *et al.* 1983] – a technique of determining how complex an urban layout is by analysing its spatial structure.

Space syntax is composed of four essential phases, one of which is deriving the axial map of the area (region) of interest. According to Hillier *et al.* [1983], the axial map is constituted by placing the fewest and longest lines (called axial lines) to cross the adjacencies between the convex polygons that form the convex map of the region of interest. Space syntax is described in more depth in Chapter 2. Sanders [2002] showed that placing axial lines in a collection of convex polygons is NP-complete. Depending on the configuration of the convex polygons and the orientation of the axial lines, ALP can be divided into three subproblems [Sanders 2002]: the convex polygons restricted to orthogonal rectangles while the axial lines have orthogonal orientation (ALP-OLOR), the convex polygons restricted to orthogonal rectangles while the axial lines have arbitrary orientation (ALP-ALOR), and arbitrary orientation of axial lines to cross the adjacencies between the convex polygons (ALP-ALCP). Each of these subproblems have been shown to be NP-complete [Sanders 2002] which implies that they may not be solved in reasonable time for large data sets.

In this dissertation, we considered the problem of placing a minimum number of orthogonal axial lines to cross the adjacencies between a collection of orthogonal rectangles where the adjacencies are restricted to be crossed only once, and investigated if the problem is polynomially solvable or if it is NP-complete. The problem is referred to as ALP-SC-OLOR for the rest of the document. Note that for the rest of the document ALP-OLOR, ALP-ALOR and ALP-ALCP refer to the multiple adjacency crossing version of the subproblems.

In this chapter, a brief description of the problem area is given accompanied by an example that demonstrates the relationship and difference between ALP-OLOR and ALP-SC-OLOR. Section 1.3 presents the significance of the problem in context of the areas where the research is applied to, followed by an overview of the research approach. A brief overview of the results of the research is also included in this chapter. The final section presents the road-map of the whole document.

1.2 Statement of the Problem

The axial line placement problem takes a collection of convex polygons as input and produces a minimum number of lines (called axial lines) that can cross the adjacencies between the convex polygons such that each adjacency is crossed at least once while the lines should cross as many adjacencies as possible. Previous research on ALP and its subproblems were considering the axial lines to cross as many adjacencies as possible. In this dissertation, we studied the single adjacency crossing of ALP whereby we considered the problem of placing single-crossing orthogonal lines in orthogonal rectangles and investigated whether the problem is polynomially solvable or if it is NP-complete. The aim of studying the most restrictive subproblem of ALP was that the result obtained during the course of the research could give insight of the single adjacency crossing of the general ALP problem.

In proving the NP-completeness of ALP-OLOR, Sanders [2002] categorized the axial lines into two: essential and choice lines. The essential lines are the lines that cross the adjacencies that have never been crossed while the choice lines are the lines some of which are required while others are redundant. Selecting the minimum number of choice lines may require exponential time and hence led to the NP-completeness of the problem. This is discussed in more detail in Subsection 2.7.4.1. In ALP-SC-OLOR, due to the restriction imposed on the adjacencies we may not have choice lines. In other words, the NP-completeness of ALP-OLOR may not have any effect in ALP-SC-OLOR which implies ALP-SC-OLOR could be solved in polynomial time. Figure 1.1 shows an example of such scenario. Lines 'a' and 'd' are essential while 'b' and 'c' are choice lines. On the other hand, selecting the minimum

combination of single-crossing lines may require exponential time due to the many possible combinations of single-crossing lines. Hence there was also possibility that ALP-SC-OLOR could be NP-complete. The ALP-SC-OLOR problem may have more than one solution due to the restriction imposed on the adjacencies. Figure 2.3 illustrates two possible solutions of ALP-SC-OLOR. The research aimed at investigating if any one solution can be obtained in polynomial time or if it is NP-complete.

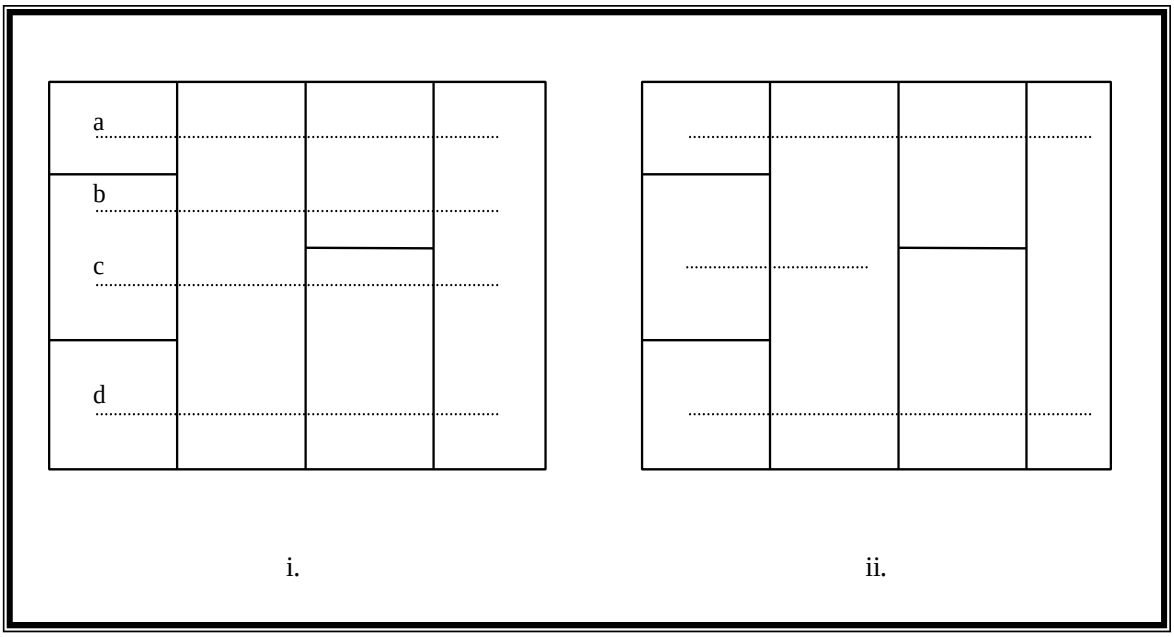

Figure 1.1: An example of ALP-OLOR [Sanders 2002] (i) and ALP-SC-OLOR (ii)

1.3 Significance of the Problem

Space syntax is a method of analysing an urban layout by studying its spatial structure [Hillier *et al.* 1983]. As it requires large data input, automating the technique would highly benefit the end-users. The space syntax method has four essential phases [Mills 1992]: separating "space", where one can move, from the "non-space", where one cannot move in an aerial map of a town; partitioning the "space" region which is the focus of interest into a minimum number of convex polygons which produces the convex map of the town; placing the fewest and longest lines (called axial lines) to cross the shared adjacencies between the convex polygons; and integrating the local and global relationships of a "space" with its

neighbours and the entire urban grid. The phases of the space syntax method are illustrated in Figure 2.1. Space syntax is discussed in Chapter 2 in more depth.

Sanders [2002] has shown that the problem of placing axial lines to cross the adjacencies between a collection of convex polygons, which is the third step in space syntax, is NP-complete. As computing for the exact solution may grow exponentially with respect to the input size, heuristic algorithms have been provided which give reasonable approximate solutions to ALP-OLOR [Sanders 2002], ALP-ALOR [Sanders and Kenny 2001] and ALP-ALCP [Hagger 2005] that can be applied in automating the space syntax method. Furthermore, Sanders *et al.* [2000] developed a polynomial time algorithm that solves the problem of placing orthogonal axial lines to cross special cases of configuration of orthogonal rectangles (chains and trees of rectangles).

The dissertation conceived another approach of approximating the general axial line placement problem by considering adjacencies to be crossed only once. More specifically, the research considered ALP-OLOR where the adjacencies between the orthogonal rectangles can only be crossed once, and explored if it can be solved in polynomial time or if it is NP-complete. As ALP-OLOR is a more simplified form of ALP-ALOR and ALP-ALCP, the result of the research could be extended to the single-crossing version of the subproblems. If the problem had been found to be polynomially solvable, the next step would be to adopt the approach to convex polygons and axial lines with arbitrary orientation. This would contribute in developing a tight heuristic for the general axial line placement problem. On the other hand, a proof of NP-completeness implies that the single-crossing of ALP may not be solved efficiently for large data input, and hence suggests such approach may not be helpful for automating the space syntax method.

In addition to its application in space syntax, ALP can also be considered as an art gallery problem in which the gallery is protected by ray guards – guards that can see along a single ray [Sanders 1999]. For the case of ALP-ALOR, the orthogonal rectangles represent an art gallery of rectangular rooms while the adjacencies between the orthogonal rectangles represent walls that separate the rooms of the gallery. The problem is to place the doorways and the guards of the gallery in such a way that the gallery will be protected by a minimum number of guards. On the ALP-SC-OLOR case, the walls between the adjacent rooms will have exactly one doorway. The problem would then be placing the doorways in such a way that the gallery is protected by a minimum number of horizontal (or vertical) ray guards. In other words, each ray guard is responsible to protect some doorways while a doorway is patrolled by exactly one guard.

1.4 Overview of the Research Approach

ALP-SC-OLOR is a simplified form of ALP in which a minimum number of single-crossing orthogonal axial lines are required to intersect the adjacencies between a collection of orthogonal rectangles. The aim of the proposed research was to investigate if ALP-SC-OLOR can be computed in polynomial time or to show that it is an NP-complete problem if it seems to have exponential complexity.

In order to attain the aim of the research, there were two general approaches: either to transform ALP-SC-OLOR to a known problem in the class of problems of P in polynomial time in which case ALP-SC-OLOR is polynomially solvable or to transform a known NP-complete problem to ALP-SC-OLOR in polynomial time which implies ALP-SC-OLOR is NP-complete.

The research proceeded with the second approach – transforming a known NP-complete problem to ALP-SC-OLOR. As a starting point, we introduced a graph theory problem which is defined only on bipartite graphs, namely bipartite independent dominating set (BIDS). BIDS is shown to be NP-complete by a reduction from the vertex cover problem for planar graphs. Then, using the fact that any bipartite planar graph can be represented in a grid using vertical and horizontal segments, we transformed BIDS into its equivalent segment representation called orthogonal bipartite independent dominating set (OBIDS). The final transformation was from OBIDS to a special case of ALP-SC-OLOR in which the lines are required to be maximal when they cross choice adjacencies. Choice adjacencies are adjacencies that can be crossed by more than one maximal lines. The research proceeded to develop a heuristic algorithm that gives an approximate solution to ALP-SC-OLOR based on a greedy method. The next section presents highlights of the results of the research.

1.5 Overview of the Research Results

In the previous section, the methodology adopted in attaining the aim of the research was highlighted. The findings of the research shows that ALP-SC-OLOR is NP-complete. The transformation is three way: from vertex cover for biconnected planar graphs to bipartite independent dominating set (BIDS) for planar graphs, then to a grid representation of BIDS (OBIDS) and finally to a normalized single-crossing orthogonal axial lines in orthogonal rectangles (ALP-NSC-OLOR). As ALP-SC-OLOR is a special case of the single-crossing versions of ALP-ALOR and ALP-ALCP, both problems remain NP-complete despite the restriction imposed on the adjacencies.

We also presented a heuristic algorithm that gives a reasonable approximate solution to ALP-SC-OLOR based on a greedy method. The heuristic has three phases: determining vertical adjacencies between a collection of rectangles, generating all possible single-crossing lines to cross the vertical adjacencies and selecting the minimum number of lines to cover all the adjacencies. The algorithm is discussed in depth in Chapter 5.

1.6 Structure of the Document

The rest of the dissertation is organized as follows. Chapter 2 begins by presenting a brief description of the space syntax method and its components. This gives a basis for describing the problem area in more depth and in relation to the general axial line placement problem. Terms and definitions used in the field of computational geometry in general and on problems related to ALP in particular are also given as they will appear frequently in the rest of the dissertation. As ALP-SC-OLOR was found to be NP-complete, the concept of NP-completeness is presented along with the techniques of proving NP-completeness of a new problem. Previous research on ALP is also included in the chapter focusing on polynomial time algorithms developed to solve special cases of the ALP-OLOR problem and heuristic algorithms for ALP-OLOR and ALP-ALCP. Finally, a brief overview of domination and dominating set is presented as the bipartite independent dominating set problem introduced in Chapter 4 belongs to the dominating set domain.

Chapter 3 outlines the research methodology. First, the research question is given – determining whether ALP-SC-OLOR is polynomially solvable or whether it is NP-complete, followed by the two general approaches to follow in answering the research question. The two approaches are transforming ALP-SC-OLOR to a known problem in P and transforming a known NP-complete problem to ALP-SC-OLOR. Section 3.3 presents a brief discussion on the focus of the research based on whether ALP-SC-OLOR is polynomially solvable or it is NP-complete.

In Chapter 4, a proof of NP-completeness of ALP-SC-OLOR is presented. A formal definition of the bipartite independent dominating set problem is given first followed by its proof of NP-completeness. Section 4.4 discusses different approaches to represent a graph using segments in a grid. An outline of representing a graph using vertical and horizontal segments is also presented in the section. Section 4.5 focuses on the final transformation from orthogonal bipartite independent dominating set to the problem of placing normalized single-crossing axial lines in orthogonal rectangles (ALP-NSC-OLOR). ALP-NSC-OLOR is a special case of ALP-SC-OLOR in which the axial lines are required to be maximal when

they cross choice adjacencies. A choice adjacency is a shared edge between two rectangles where it can be crossed by two or more maximal lines.

Chapter 5 presents a heuristic algorithm for ALP-SC-OLOR based on a greedy method. First, the three phases of the heuristic are discussed along with their correctness and complexity analysis. Section 5.4.2 details the empirical analysis of the heuristic. The final section presents the experimental results of the algorithm which reflects the solution produced by the heuristic is close to the optimal.

In Chapter 6 future work identified during the course of the research is presented. It is mainly developing heuristics for ALP-SC-OLOR using various approaches and exploring special configurations where ALP-SC-OLOR is polynomially solvable.

The last chapter highlights the main findings and conclusions of the document. It also points out the main contributions of the dissertation. Section 7.4 states the limitations of the research which are mainly lack of large data sets to analyse the performance and running time of the heuristic. The chapter concludes by presenting a summary of the whole document.

Chapter 2

Background and Related Work

2.1 Introduction

Visibility and guarding are well known geometric problems that have been studied extensively. ALP, which was briefly described in Chapter 1, is a recently introduced problem in the area of computational geometry that attempts to automate the space syntax method – a technique of urban analysis. The axial line placement (ALP) problem is to find the fewest and longest lines that cross each adjacency between a collection of convex polygons. Based on the configuration of the convex polygons and the orientation of the axial lines, ALP is classified into ALP-OLOR, ALP-ALOR and ALP-ALCP. Each of these subproblems has been found to be NP-complete. In this dissertation, we considered a variation of ALP-OLOR where the adjacencies between the orthogonal rectangles are restricted to be crossed only once and determined if the problem still remains NP-complete or if it can be solved in polynomial time.

This chapter introduces the space syntax method in more depth which will be the basis for describing the axial line placement problem and the problem area considered for the dissertation. Terms and definitions in the field of computational geometry that are used in the rest of the document are given next. Section 2.6 discusses the theory of NP-completeness and the techniques of transformation in proving the NP-completeness of a new problem. The visibility and guarding problems have some commonalities with ALP and hence they are discussed in depth in Section 2.7. A discussion on previous research on ALP is also included in this chapter, focusing on the algorithms and methods of transformation used in proving NP-completeness of the problems. Lastly, domination and dominating set, a well known graph theory problem, is described along with its variations as the bipartite independent dominating set introduced in Chapter 4 fits into the domination domain.

2.2 Space Syntax

Space syntax is a technique introduced by Hillier *et al.* [1983] for analysing and describing architectural patterns at the building and urban level. By studying the space representation of an urban layout, space syntax helps in determining how the structure of the urban layout is related to the social and economic factors. Space syntax is based on the idea that an architectural pattern of a town is an interconnected space where any "space" is accessible from every other "space" [Jiang 1998]. The method considers the global and local relations of a "space" in determining the integration factor [Hillier 1996] – the global measure is determined by integrating the relation of the "space" with the overall layout whereas the local measure is determined by the relation of the "space" and its neighbours.

Space syntax has been serving as a tool to assist architects to simulate the effects of their design by modeling and analysing urban patterns with respect to human activities such as pedestrian movement and traffic flow [Thomson 2003]. There are four essential steps required when the space syntax method is applied to an urban layout [Mills 1992; Jiang 1998; Sanders 2002].

- The first step is to take an aerial map of a town and then separate the "space" where one can move (roads, parking areas etc.) from the "non-space" where one cannot move (buildings, street blocks, etc). Figure 2.1(i) is an example of this phase – the shaded region represents the "non-space" while the rest of the region represents the "space". The result of this process is the layout of the town, and as described above, the "space" is the object of interest. Image processing techniques can be applied to the aerial map in separating the "space" from the "non-space". Furthermore, the "space" is approximated to a polygon with holes (the "non-space"). Figure 2.1(ii) shows the approximated polygon with holes of the original layout.

- The second step is to partition the polygon (with holes) to produce the convex map of the area, i.e. the polygon is partitioned into a minimum number of convex polygons. A polygon is convex if a line segment joining any two points of the polygon lies entirely within the polygon including its boundaries. New vertices (called Steiner points) could be introduced during the partitioning process. Unlike a polygon without holes which could be partitioned into a minimum number of convex polygons in polynomial time irrespective of Steiner points [Chazelle and Dobkin 1979; Keil and Snoeyink 2002], partitioning a polygon with holes is known to be NP-hard whether Steiner points are allowed [Lingas 1982] or not [Keil 1985]. Consequently, in automating this phase

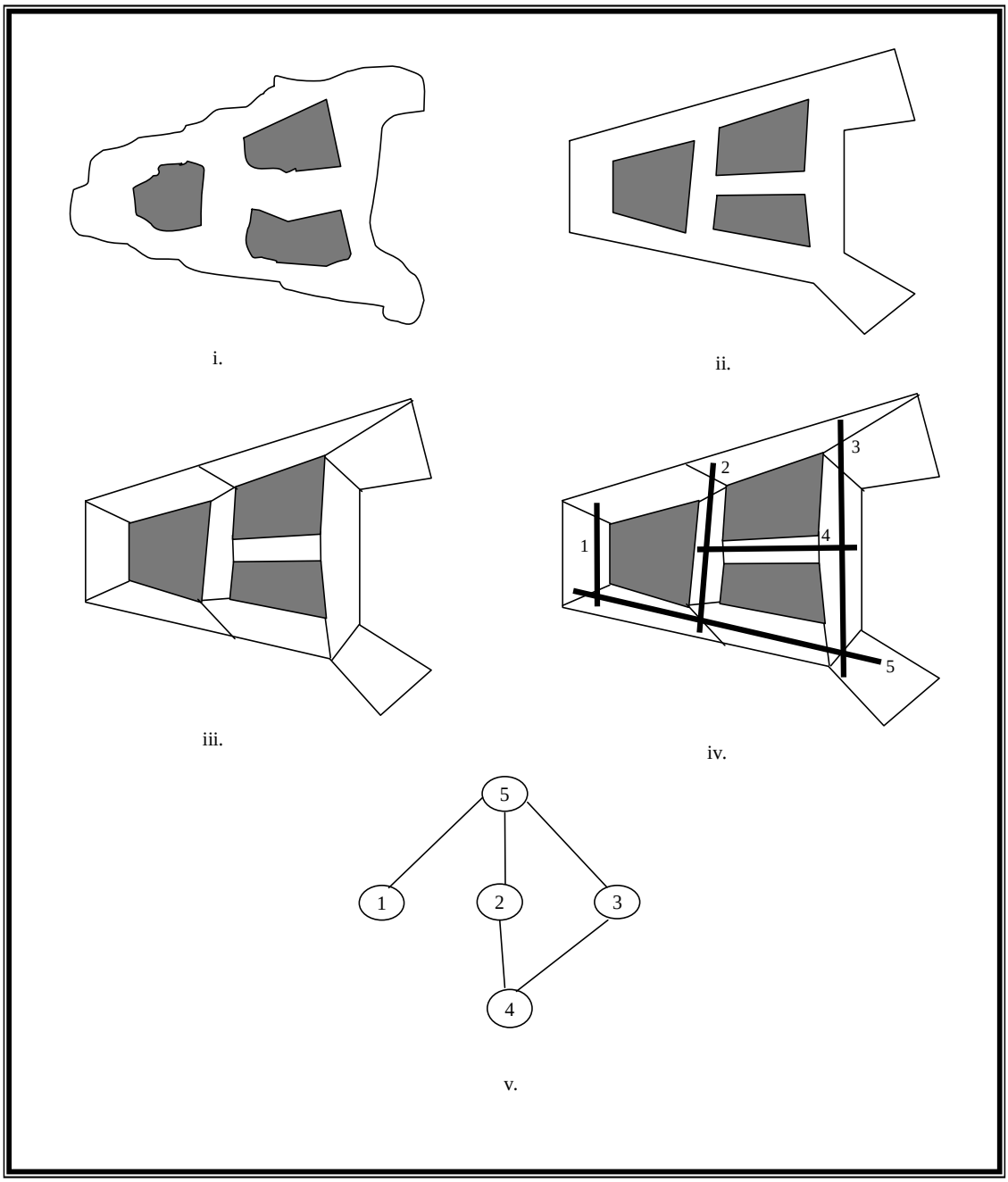

Figure 2.1: The steps of space syntax

heuristic algorithms that approximate the exact solution could be applied since the exact solution may not be computed efficiently for large data inputs. Lien and Amato [2004] developed an algorithm that partitions a simple polygon with or without holes into "approximated" convex polygons in $O(nr)$ time, where n is the number of vertices of the polygon and r is the number of reflex angles of the polygon. The algorithm recursively partitions the polygon until a user specified bound is attained. The lower the specified bound, the higher will be the number of convex components. If the specified bound is zero, the polygon will be partitioned into components which are all convex. Figure 2.1(iii) shows the convex map of the area (region) of interest.

- During the third step, an axial map of the town is derived from the convex map. An axial map is composed of the longest and fewest number of lines (called axial lines) that cover the convex map. The axial lines are placed to cross each adjacency (shared edge) of the convex polygons in the convex map, and must be maximal – they must cross as many adjacencies as possible while each adjacency must be crossed by at least one axial line. The placement of the axial lines is associated with the lines of sight. The problem of placing the minimum number of axial lines was found to be NP-complete [Sanders 2002] which implies it may only be solved efficiently for small data sets in reasonable time. However, there are heuristic algorithms that give approximate solutions for the general problem and specific configurations. More about the problem of placing axial lines and its variations is given in the next section as it is the focus of the dissertation. Figure 2.1(iv) is an example of placing axial lines to cross the adjacencies between the convex polygons produced during the previous step.

- The final step is to compute the local and global relationships of a "space" with its neighbours and the entire grid. These are computed by assigning values to the axial lines based on their depth (the degree to which they are integrated), and then applying graph algorithms to the axial map of the town – the axial lines are represented as nodes (vertices) and an edge is placed between the vertices if the axial lines intersect. If two axial lines intersect, they are considered as adjacent and an edge is placed to join their associated vertices. Usually, the graph representation of the axial map is undirected and unweighted [Penn 2003]. This approach avoids the metric distance between the lines that form the axial map since both long and short lines are represented by dimensionless nodes. As longer lines are more likely to intersect other lines, they are more integrated (depth) than the shorter ones. Based on the depth of the axial line (and hence the degree of the vertex) the integration values are computed using

standard formulae [Jiang 1998]. Figure 2.1(v) is a graph representation of the axial map.

The rest of the document will focus on the third step – placing axial lines in a collection of convex polygons and its variations.

2.3 The Axial Line Placement Problem

In automating the space syntax method, each of its phases need to have either an efficient algorithm which gives the exact solution or an algorithm which gives an approximated solution in reasonable time. As we have seen in the previous section, each phase of the space syntax method has such algorithm and hence can contribute in automating the method. Since the focus of the dissertation is on the third phase – placing axial lines to cross the adjacencies between convex polygons (ALP) – we assume that we are given a collection of non-overlapping convex polygons. More formally ALP can be stated as: given a collection of non-overlapping convex polygons, find the longest and fewest number of lines (axial lines) that cross each adjacency between the polygons. Each adjacency must be crossed at least once while an axial line should cross as many adjacencies as possible. Moreover, the lines must lie entirely within the area defined by the polygons. ALP has been known to be NP-complete [Sanders 2002]. Based on the configuration of the convex polygons and the orientation of the axial lines, ALP can be divided into three subproblems.

i. Orthogonal axial lines to cross the adjacencies between a collection of orthogonal rectangles (ALP-OLOR). Here, the convex polygons are restricted to be orthogonally aligned rectangles while the axial lines have orthogonal orientation. By orthogonal rectangles, we mean that the boundaries of the rectangles are parallel to the Cartesian x and y axes while the orthogonal axial lines refer to the horizontal lines (to cross vertical adjacencies) and vertical lines (to cross horizontal adjacencies). As one can be computed from the other by rotating 90° (or 270°), usually only one of these variations is considered. ALP-OLOR is the most restrictive subproblem of ALP and is known to be NP-complete [Sanders 2002].

ii. Axial lines of arbitrary orientation to cross the adjacencies between a collection of orthogonal rectangles (ALP-ALOR). This problem considers the convex polygons to be orthogonally aligned rectangles while the axial lines may have arbitrary orientation. In contrast to ALP-OLOR, an axial line in ALP-ALOR could cross both vertical and

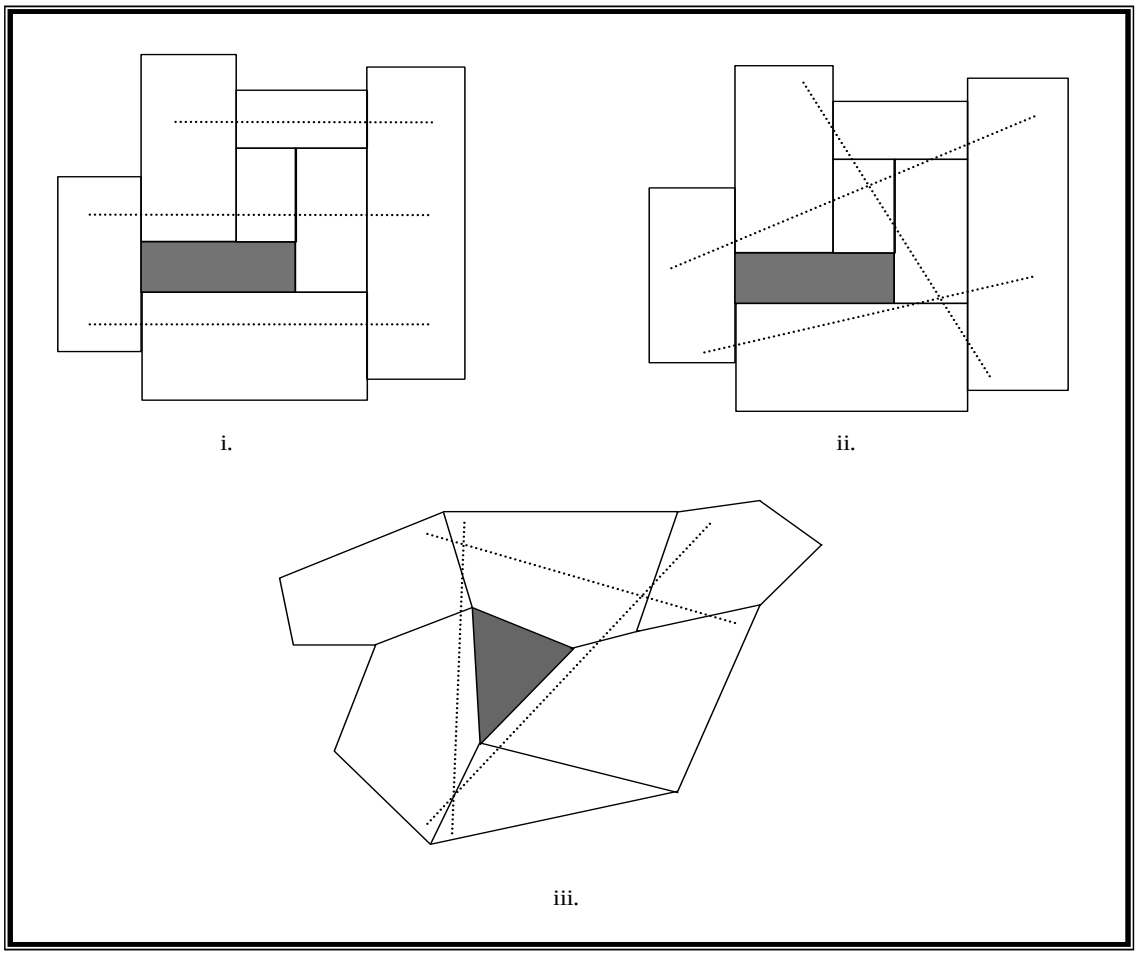

Figure 2.2: ALP-OLOR, ALP-ALOR and ALP-ALCP each having three axial lines

horizontal adjacencies between the orthogonal rectangles. ALP-ALOR was found to be a member of NP-complete class of problems [Sanders 2002].

iii. Axial lines of arbitrary orientation to cross the adjacencies between a collection of convex polygons (ALP-ALCP). This is the most general subproblem. A collection of non-overlapping convex polygons are crossed by axial lines of arbitrary orientation. As ALP-OLOR and ALP-ALOR are NP-complete, and they are special cases of the ALP-ALCP, it follows that ALP-ALCP is also NP-complete [Sanders 2002].

An example of each of the subproblems of ALP is given in Figure 2.2. The subproblems are discussed in more depth in Section 2.7 focusing on the method of transformation used in proving the NP-completeness of the problems, heuristics developed and special configurations of rectangles (or convex polygons) in which the axial lines can be placed in polynomial time.

2.4 The Single-crossing Version of ALP

As described above, ALP and its subproblems are NP-complete which implies that the problems may not be solved efficiently (unless $P = NP$). Consequently, focusing on special cases where the problem can be solved in polynomial time and heuristic algorithms that give reasonably approximated solutions are essential. One way of considering a variation of ALP is to restrict the adjacencies between the collection of convex polygons to be crossed only once while the lines can cross one or more adjacencies. In this dissertation, we considered the problem of placing orthogonal axial lines to cross the adjacencies between orthogonally aligned rectangles only once (ALP-SC-OLOR) and investigated if the problem can be solved efficiently or if it remains NP-complete.

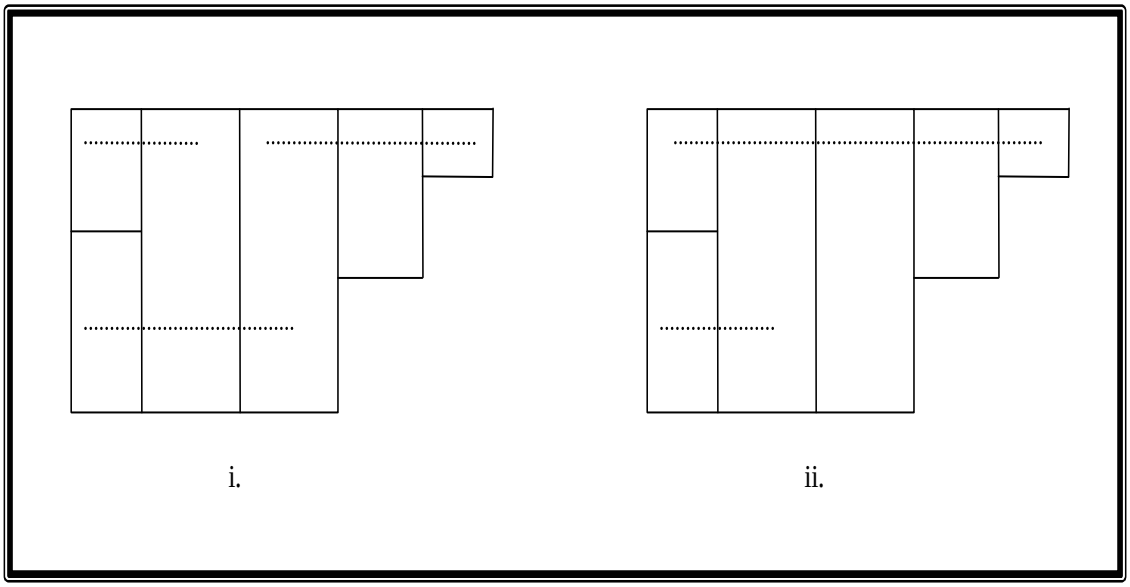

Figure 2.3: The choice of constructing the axial lines is essential in ALP-SC-OLOR to get minimum number of lines

ALP-SC-OLOR can be formally stated as: given a collection of non-overlapping orthogonally aligned rectangles, find the fewest number of orthogonal axial lines that can cross the adjacencies between the rectangles only once while the axial lines can cross one or more adjacencies. By orthogonally aligned rectangles, we mean that the sides of the rectangles are parallel either to the horizontal or to the vertical axes. Likewise, by orthogonal axial lines we mean that the lines are parallel to either the horizontal axis (to cross vertical adjacen-

cies) or the vertical axis (to cross horizontal adjacencies). In this document, we considered horizontal axial lines and vertical adjacencies as the vertical axial lines to cross horizontal adjacencies version can be computed from the other.

For the same configuration of orthogonal rectangles, ALP-SC-OLOR and ALP-OLOR may not have the same solution due to the restriction imposed on the adjacencies in ALP-SC-OLOR. Figure 1.1 has shown the difference between these two versions. Consequently, it might be possible to find an efficient algorithm to compute the ALP-SC-OLOR problem despite the NP-completeness of ALP-OLOR. On the other hand, the choice of constructing the axial lines in ALP-SC-OLOR is essential to get minimum axial lines (Figure 2.3), which is also the case for ALP-OLOR. The aim of the dissertation was to explore if a polynomial time algorithm can be developed for ALP-SC-OLOR or a proof of NP-completeness if the problem seems to require exponential time to compute the minimum number of combination of single-crossing lines.

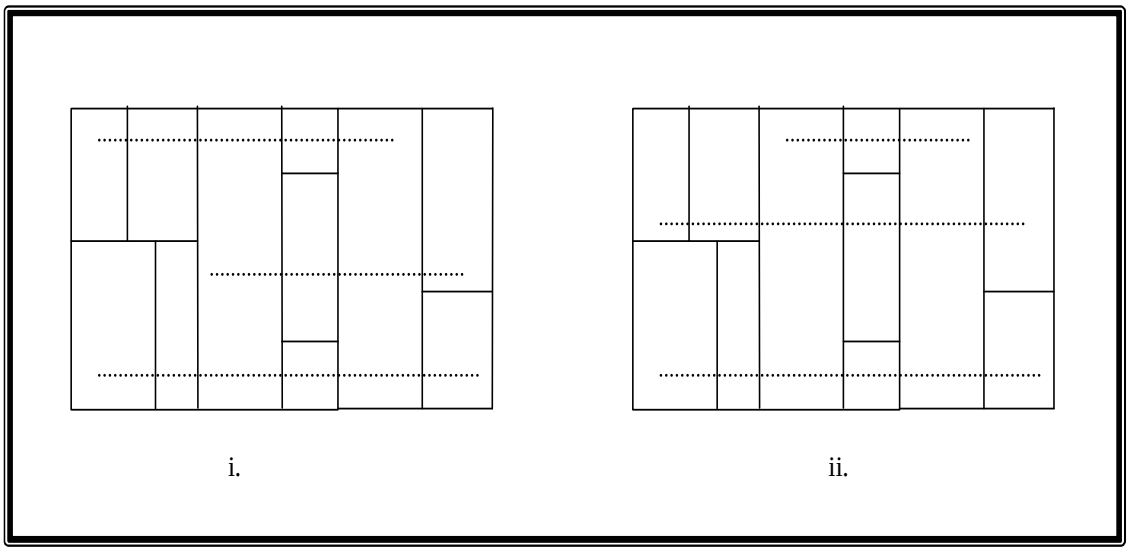

Figure 2.4: ALP-SC-OLOR with two possible solutions

By definition, the axial lines are the fewest and longest lines that cross every adjacency of a collection of convex polygons. For a given collection of convex polygons, it is possible to have more than one solution that differ in the way the axial lines are placed and the adjacencies they cross. Moreover, due to the restriction on the adjacencies between the

orthogonal rectangles to be crossed exactly once in ALP-SC-OLOR, we may have more than one solution depending on the lines that cross the adjacencies. Figure 2.4 shows two possible solutions of ALP-SC-OLOR to the same configuration of orthogonal rectangles. The focus of the dissertation was on one possible way of placing a minimum number of single-crossing lines to cross the adjacencies between the rectangles.

2.5 Basic Terminologies and Definitions

There are various terms that are common and widely used in the field of computational geometry. Some of these terms are defined below as they will be used frequently throughout the document. Most of the definitions are taken from Manber [1989], Shermer [1992] and Asano *et al.* [2000].

- A *point* p is defined by a pair of Cartesian coordinates (x, y).
- A *line* is defined by a pair of distinct points p and q and is denoted by $-pq-$.
- A *line segment* is represented by a pair of distinct points p and q in which the points are the end points of the line segment. It is denoted by pq.
- An *orthogonal line (or orthogonally aligned line)* is a line that is parallel to either the x-axis or the y-axis.
- A *path* P is a sequence of points $p_1, p_2, ..., p_n$ and the line segments joining the points.
- A *closed path* is a path whose last point is the same as the first point. It is also called a *polygon*.
- A *polygon* can also be defined as a sequence of three or more points $p_1, p_2, ..., p_n$ and the line segments joining the points $p_1p_2, p_2p_3, ..., p_np_1$. The points that define the polygon are called *vertices* and the line segments joining the points (vertices) are called *edges*. The edges can also be represented as $(p_1, p_2), (p_2, p_3), ..., (p_n, p_1)$.
- A *simple polygon* is a polygon such that if any two of its edges are non-consecutive, then they do not intersect.
- An *interior angle* of a polygon is the angle between two consecutive edges of the polygon on the inside of the polygon.
- A *reflex angle (or reflex vertex)* of a polygon is an interior angle of a polygon which is greater than 180^0.

- A *convex polygon* is a polygon such that the line segment joining any two points of the polygon lies entirely within the polygon (and its boundaries). In other words, a *convex polygon* is a polygon with no reflex angle.
- A *sub-polygon* is a simple polygon that lies entirely within another polygon.
- A *polygon with holes* is a polygon with one or more sub-polygons removed from it. The sub-polygons must lie entirely within the polygon. Figure 2.1(ii) shows a polygon with three holes. Note that a convex polygon cannot have holes.
- A *rectangle* is a polygon with four vertices and four edges such that a pair of its consecutive edges form an interior angle of 90°.
- An *orthogonal rectangle* is a rectangle with all of its edges orthogonally aligned, that is, parallel to the x-axis or the y-axis.
- A *comb polygon* is a polygon with k-prongs, each prong having two edges and adjacent prongs are separated by an edge [O'Rourke 1995]. An example of a comb polygon with 6-prongs is given in Figure 2.5.

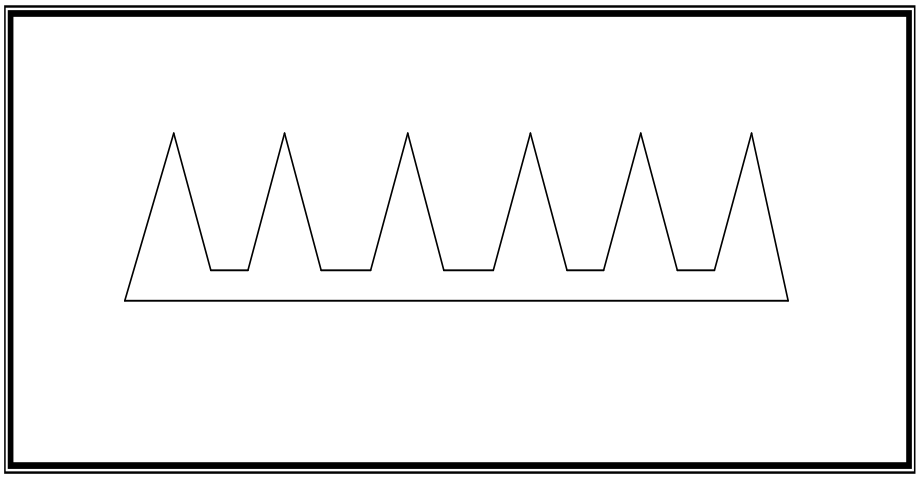

Figure 2.5: A comb polygon with 6-prongs

- An *orthogonal comb polygon* is a comb polygon in which the prongs are orthogonally aligned. An example of such a polygon is given in Figure 2.6.
- A *decomposition* of a polygon P is called a *partition of* P if the sub-polygon components do not overlap except at their boundaries [Keil 2000]. If the sub-polygons are

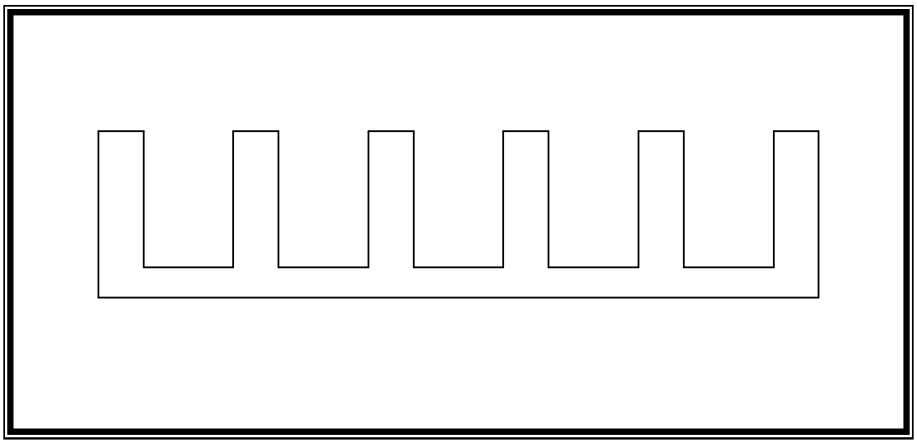

Figure 2.6: An orthogonal comb polygon with 6-prongs

allowed to overlap, then the decomposition is called a *cover of P*. In either case of the decomposition, each sub-polygon must lie entirely within the original polygon while the union of the sub-polygons constitutes the original polygon.

- A *Steiner point* is a point that is not part of the original vertices of a polygon but introduced during a decomposition of the polygon [Keil 2000]. Although introducing a Steiner point makes the partitioning process complex, it is helpful in partitioning the polygon into the fewest sub-components. An example of partitioning a polygon with and without a Steiner point is given in Figure 2.7.

- A *triangulation* is a decomposition of a polygon into triangles by a maximal set of non-intersecting diagonals (line segments joining non-adjacent vertices). Figure 2.8 shows a simple polygon and a possible way of its triangulation.

- In a polygon two points, q and r, are *visible to one another* if the line segment qr joining the two points lies within the polygon and its boundaries or $q = r$ (a point is visible to itself). Figure 2.9(i) illustrates point a can see points b and c, but not d.

- Two points q and r in a polygon are *link-j visible (or L_j visible)* if they can be connected by j or fewer line segments that lie within the polygon and its boundaries. A point is visible to itself, and hence it is L_0 visible. Moreover, any two distinct points in a convex polygon are L_1 visible, since the line segment joining them lies within the polygon (and its boundaries). In Figure 2.9(ii), points a and b are L_3 visible.

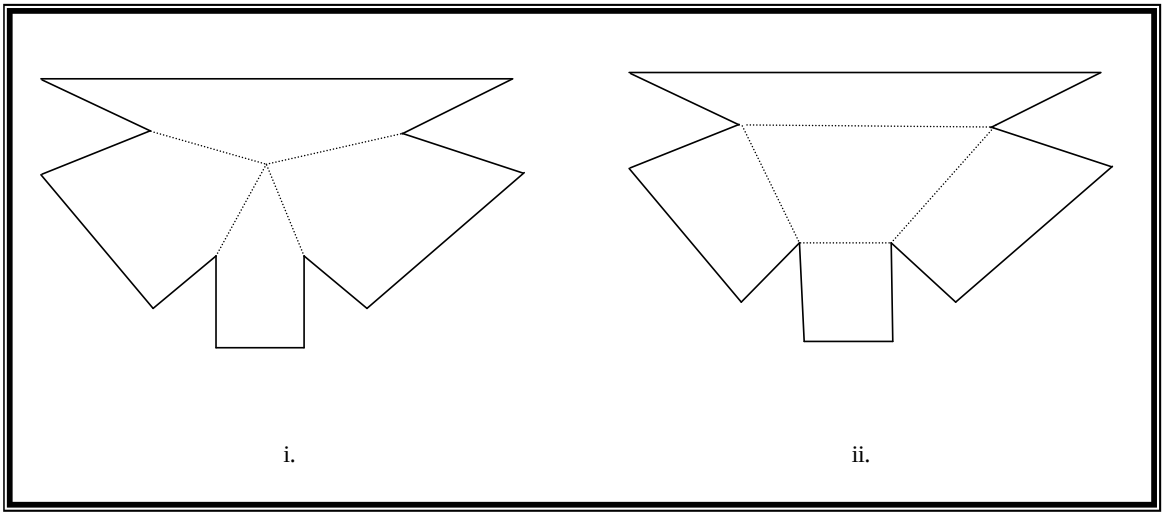

Figure 2.7: Partitioning a polygon into convex polygons with and without Steiner point

- A *visibility polygon V(p)* of a point p in a polygon P is the set of all points of P that are visible from p [Asano *et al.* 2000]. p can be any point inside, outside or on the boundary of the polygon.
- In a polygon a *point q is visible from an edge uv* if there is a point w in uv such that

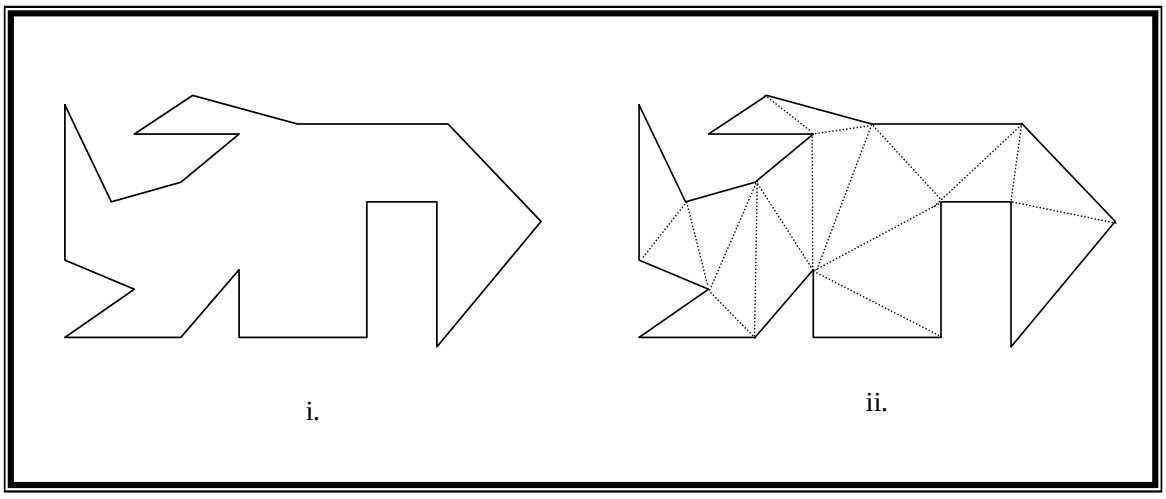

Figure 2.8: A simple polygon and a possible way of its triangulation [de Berg *et al.* 2000]

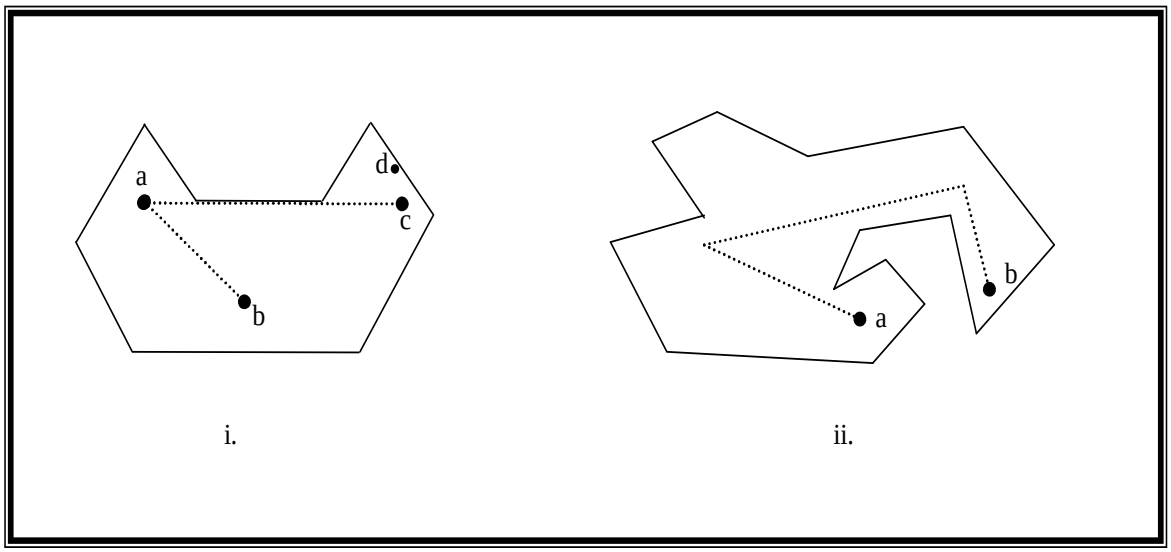

Figure 2.9: (i) Point a is visible from b and c but not from d, and (ii) Points a and b are L_3 visible

qw lies entirely within the polygon. The point w is considered as mobile point within the edge.

- A polygon P is *completely visible* from an edge uv if every point in P is visible from every point in uv.
- A polygon P is *strongly visible* from an edge uv if every point in P is visible from some point in uv.
- A polygon P is *weakly visible* from an edge uv if there exists at least one point in P which is visible from some point in uv. Figure 2.10(i-iii) shows a polygon visibility from an edge uv.
- A *visibility polygon V(e)* of an edge e in a polygon P is the set of all points of P that are weakly visible from e.
- A *graph G(V,E)* is a set of vertices V and a set of edges E that joins the vertices. A graph can be directed or undirected.
- A graph $G(V, E)$ is *planar* if it can be drawn in a plane by representing the vertices as points and the edges as simple curves joining the points such that none of its edges cross each other.

- A graph $G(V, E)$ is *bipartite* if V can be partitioned into two disjoint sets of vertices V_1 and V_2, ($V = V_1 \cup V_2$ and $V_1 \cap V_2 = \emptyset$), such that if $(v_1, v_2) \in E$ then $v_1 \in V_1$ and $v_2 \in V_2$.

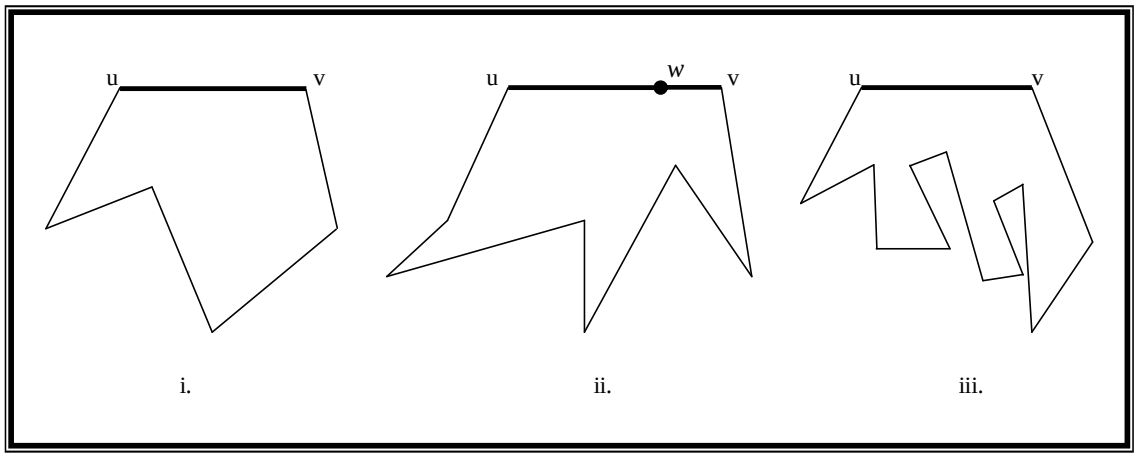

Figure 2.10: A polygon (i) completely visible, (ii) strongly visible and (iii) weakly visible from the edge uv [Avis and Toussaint 1981]

2.6 NP-complete Problems

2.6.1 Introduction

Computers are helpful in solving real-life problems. However, not all problems are solvable by computers. This could be due to the fact that either the problems are too expensive (in terms of resources – memory, processor, etc.) to be solved efficiently or they cannot be solved by computers whatever resource we may have [Garey and Johnson 1979]. Some geometric problems fall into the first category – they may not be solved efficiently. In the previous sections, we used the term NP-complete (or NP-hard) to describe such problems. Moreover, it was stated that if a problem is NP-complete it is worthwhile to search for algorithms that can give approximate solutions or special cases of the problem where it can be solved in polynomial time.

Garey and Johnson [1979] classified algorithms based on the relationship between the input size and their corresponding running time into polynomial time and exponential time algorithms. Polynomial time algorithms are algorithms whose time complexity is bounded by a polynomial function P, that is, $O(P(n))$ for some polynomial function P, where n is

the size of the input. Such algorithms are considered to be efficient and their corresponding problems are tractable since the running time of these algorithms is bounded by a polynomial function with respect to their input size. If an algorithm's time complexity is not bounded by a polynomial function, then it is categorized as an exponential time algorithm[1]. Since the running time of the latter algorithms grow exponentially with respect to the input size, solutions for these problems are feasible only for small size of input.

The section begins by introducing decision problems which will be the basis for defining the P and NP classes of problems. The notion of efficient algorithms is also given followed by common properties of NP-complete problems which focuses on the way of proving NP-completeness of a new problem. The last subsection presents the concept of NP-hard problems.

2.6.2 Decision Problems

In relation to complexity theory, there are problems, called decision problems that can only have *yes* or *no* answers. Manber [1989] described the concept of decision problems using language-recognition problems: if U is considered to be the set of all inputs to the decision problem and $L \in U$ be the set of all inputs where their answer is *yes*, the decision problem will be to determine whether an input instance belongs to L. The input to a decision problem has two parts [Baase and Van Gelder 2000]: the *instance description* part which contains information about the expected input (could be sets, graphs, etc.), and the *question* part that contains the $yes - no$ question. In other words, a decision problem matches all the input values of a problem into *yes* or *no* output. For example,

Instance: Given two positive integers m and n.

Question: Is there a positive integer k such that $m = k * n$?

Decision problems are important since most problems can be converted to decision problems and they can also determine whether a certain property is satisfied for their inputs [Harel 1987]. In converting optimization problems to decision problems, we need to introduce an additional argument with the input which bounds the value to be optimized. For example, the optimization problem *"Given G(V,E), find the path with a minimum number of edges to traverse from u to v where u,v* $\in$ *V"* can be converted into a decision problem as

[1]This definition also includes functions such as $n^{\log n}$ which may not be considered as exponential functions in other instances.

"Given G(V,E), is there a path to traverse from u to v, u,v ∈ V with the number of edges fewer than or equal to k, for some positive integer k?" More formally,

Instance: Given $G(V, E)$ and two vertices u, $v \in V$ and a positive integer k.

Question: Is there a path in G from u to v with fewer than or equal to k number of edges?

If an efficient algorithm is derived to solve a decision problem, it can be modified to solve the corresponding optimization problem. On the other hand, if a decision problem is shown to be hard, then its related optimization problem is also hard [Cormen *et al.* 2001].

Associated with the decision problems, there are two classes: P and NP [Baase and Van Gelder 2000]. The class P (polynomial time) contains decision problems that can be solved in deterministic polynomial time – problems that can be solved by algorithms in which their worst case time complexity is bounded by a polynomial function. Although the polynomial function can be very large, it can be considered that if a problem is not in P then it would be expensive or impossible to get a solution. In practice, most problems with polynomial time algorithms have low-degree polynomial running time [Manber 1989]. Moreover, it is possible to solve complex problems by combining several algorithms of simpler problems in P as combination of polynomials is still a polynomial (the closure property of polynomials for addition, multiplication and composition operations). On the other hand, the NP (nondeterministic polynomial time) class contains decision problems in which a given proposed solution can be verified in polynomial time whether it is a solution or not.

The NP class can also be defined more formally using nondeterministic algorithms [Garey and Johnson 1979]. Nondeterministic algorithms comprise of two phases: the *guessing* phase and the *checking* phase. Given a problem instance I of a decision problem, the first phase guesses some structure S, and then both I and S are provided as inputs to the checking phase. If the output of the checking phase is *true* then the algorithm outputs *yes* whereas if the checking phase returned *false*, there is no output. The class NP contains decision problems that can be solved in nondeterministic polynomial time.

Clearly $P \subseteq NP$ since we can consider any polynomial time deterministic algorithm as a special case of polynomial time nondeterministic algorithm using the deterministic algorithm as a checking phase and ignoring the guessing phase [Garey and Johnson 1979]. However, it is not yet known whether $P = NP$ or not. It is widely believed that $P \neq NP$.

2.6.3 NP-completeness

A problem D is NP-complete if the problem is in NP and every problem in NP is polynomially reducible to D [Manber 1989]. By polynomial reduction, we mean that for two problems D_1 and D_2 and input spaces U_1 and U_2, there exists a polynomial function f such that $u \in U_1$ if and only if $f(u) \in U_2$. NP-complete problems have a common property that if an efficient algorithm exists to any of them then there exist efficient algorithms for all NP-complete problems.

When presenting a proof of NP-completeness of a new problem, we need to show the following conditions hold [Baase and Van Gelder 2000]:

- the problem is in NP.
- it is possible to transform a known NP-complete problem to the new problem.
- the transformation function is polynomial.

Two problems, P_1 and P_2 are equivalent if each problem is polynomially reducible to the other [Manber 1989]. The one sided polynomial reducibility (transformation) in condition two above reflects the fact that when proving the NP-completeness of a problem, we are intending to show how hard the problem is. In other words, when problem P_1 is reduced to another problem P_2, P_2 is considered to be harder than P_1 [Manber 1989]. Polynomial reducibility is transitive since composition of polynomial functions is polynomial. Consequently, if a known NP-complete problem is polynomially reducible to a new problem, then every NP-complete problem is polynomially reducible to the new problem.

In order to make use of the second condition of proving NP-completeness of a new problem, we need to have a known NP-complete problem that could be polynomially reducible to a new problem. The first NP-complete problem which was proved by Cook [1971] is the Satisfiability (SAT) problem. Let S be a boolean expression in conjunctive normal form (CNF), that is, a sequence of AND expressions each of which contains OR expressions. For example,

$$S = (x \vee \neg y \vee z) \wedge (\neg x \vee y \vee z) \wedge (\neg x \vee y \vee \neg z)$$

where $\wedge$ corresponds to the boolean operator AND, $\vee$ corresponds to OR and $\neg$ corresponds to NOT, is a boolean expression. The satisfiability problem is then to determine whether a given boolean expression is satisfiable. A boolean expression is satisfiable if there exists an assignment of *true* and *false* to the variables such that the value of the expression is *true*. For the above example, $x = FALSE$, $y = TRUE$ and $z = TRUE$ satisfies the

expression S. SAT is in NP since given a boolean expression and a proposed solution, we can verify if the proposed solution satisfies the expression in polynomial time. The complete proof of NP-completeness of the SAT problem can be found in Garey and Johnson [1979]; Cook [1971].

When proving the NP-completeness of a problem, most of the time, it is easier to show that the problem is in NP than to find a polynomial transformation function that transforms a known NP-complete problem to the new problem. Garey and Johnson [1979] present general transformation approaches that can be used when dealing with proving the NP-completeness of a new problem. These are restriction, local replacement and component design [Garey and Johnson 1979, p. 63–74].

a. **Restriction:** This reduction approach is based on the idea that if a special case of a problem is NP-complete then the problem is also NP-complete (provided that it is in NP). There should be one to one correspondence between the instances of the target problem and its special case. In other words, the answer to the special case is *'yes'* if and only if the answer to the target problem is *'yes'*. As discussed in Chapter 4, this method will be used in proving the NP-completeness of ALP-SC-OLOR by showing that its special case is NP-complete.

b. **Local Replacement:** In this reduction, the basic unit of an instance of a known NP-complete problem is replaced in the same way as an instance of the target problem. That is we need to pick some aspects of a known NP-complete problem instance to make a collection of basic units, and then correspond with the instances of the target problem by replacing each basic unit, in a uniform way but with a different structure. The local replacement method will be used later in the transformation presented in Chapter 4.

c. **Component Design:** This is the hardest of the three reduction methods. In component design reduction we need to design instances of components of the target problem and then "realize" instances of a known NP-complete problem by combining the components. Generally, there are two basic components in this approach: making choices (like selecting vertices) and testing properties (like checking if all the edges are covered). The two components are joined together in a target instance so that the choices are communicated to the property testers, and the property testers then check whether the choices made satisfy the required constraints.

Note that for a problem to be NP-complete, we need to explicitly show that it is in

NP in addition to the above reduction methods. Otherwise, the problem remains NP-hard (which is discussed in the next subsection).

2.6.4 NP-hardness

A problem D is said to be NP-hard if any NP-complete problem is polynomially reducible to D while D may not be necessarily in NP [Cormen *et al.* 2001]. The notion of NP-complete problems is that if any NP-complete problem can be solved in polynomial time, then every problem in NP can be solved in polynomial time algorithm. On the other hand, NP-hard problems constitute a lower bound to the NP-complete problems – they are at least as hard as any NP-complete problem. When we want to prove that a problem is NP-hard, we only need to show that the second and third conditions given in Subsection 2.6.3 hold. That is,

- reducing (transforming) a known NP-complete problem to the new problem.
- proving the transformation function is polynomial.

There are problems that are NP-hard but not NP-complete. The problem of determining chromatic number of a graph – the fewest number of colours needed to colour a given graph – is an example of such a problem [Brassard and Bratley 1996]. The problem can be stated as:

Question: *Given a graph G, find the chromatic number of G.*

By reducing the 3-colourable problem, which is a known NP-complete problem, it follows that determining the chromatic number of a given graph G is NP-hard. However, as a given proposed solution may not be verified in polynomial time, the problem of determining the chromatic number of a given graph is not in NP and hence it is not NP-complete. NP-hard problems may not also be decision problems [Brassard and Bratley 1996].

There are also problems that cannot be solved by any algorithm. Turing [1936, as cited in Garey and Johnson [1979]] has shown that given a program and its input, there is no algorithm that could determine whether the program will terminate or not. The problem is known as halting problem and can be formally stated as:

Instance: Given a program and its input.

Question: Will the program terminate?

The halting problem is a decision problem but it is not decidable. Hopcroft and Ullman [1979] defined a problem to be undecidable if there is no algorithm that could determine

whether the answer to a given instance of the problem is *yes* or *no*. In other words, the undecidability of a problem is associated with the existence or non-existence of algorithms that could solve a problem with infinite instances.

2.7 Related Work

2.7.1 Introduction

This section focuses on previous work on visibility and guarding problems with a brief historical background and how they relate to the axial line placement problem followed by previous research on ALP. Visibility and guarding problems have been the focus of research areas as they are helpful in solving real-life problems. Visibility problems are mainly of the form: *"Given two objects, which could be points or edges in a polygon with or without holes, are they visible to one another?"* while guarding problems are of the form: *"Given a point or an edge and a polygon with or without holes, what proportion of the polygon can be seen by the object?"*

The section is organized as follows. First an overview of previous work on visibility theory is given in context to the general axial line placement problem. The art gallery problem, a guarding problem that requires the minimum number of guards to protect an art gallery, is presented next highlighting how ALP is considered as a variation of the art gallery problem, followed by research done on ALP focusing on the algorithmic aspects and the results obtained. Lastly, the dominating set problem is briefly described as the bipartite independent dominating set problem introduced in Chapter 4 fits into the domination domain.

2.7.2 Visibility Problems

Visibility problems have been the focus of research for quite a long time – as long as Brunn's theorem of kernel set [Asano *et al.* 2000]. Since then several variations of the problem have been studied. The basis for most visibility problems is the visibility graph of a polygon – graphs that have nodes for representing the vertices of the polygon and lines between the nodes if the vertices see each other [O'Rourke 1997], that is, the graph will have a node for every vertex of the polygon and an edge joining the nodes if their associated vertices of the polygon are visible.

Another variation of the visibility problem is to find the visibility polygon of a point – given a polygon and a point (inside or outside the polygon) what proportion of the polygon is visible from the point. Davis and Benedikt [1979] developed an algorithm to find the

visibility polygon of a point, which first computes the vertices of the polygon that are visible to the point of interest, and then computes the closest point of intersection between the line that contains the point of interest and lies within the polygon, and the edges of the polygon that contains the invisible vertices. The algorithm runs in $O(n^2)$ time. Later a linear algorithm was developed by Joe [1990] that computes the visibility polygon of a point. On the other hand, Asano *et al.* [1986, as cited in Asano *et al.* [2000]] considered a variation of the problem in which the polygon contains holes. Using an "angular plane sweep" method, they developed an algorithm that runs in $O(n \log h)$, where n is the number of edges while h is the number of holes within the polygon, to solve the visibility polygon of a point when the polygon contains holes.

Bose *et al.* [1992] also studied the visibility polygon of a point which could be inside or outside of a simple polygon. The method they followed was to decompose the polygon into visibility regions as a preprocessing phase so that queries of visibility regions from a point can be recovered efficiently. The preprocessing time, which is the dominant cost, is $O(n^3 \log n)$ with space complexity of $O(n^3)$. Their algorithm retrieves the visibility polygon of a point in $O(\log n + k)$ time, where k is the size of the visibility polygon. Furthermore, the number of vertices that are visible from a given point can be retrieved in $O(\log n)$ time.

By introducing the weak and strong visibility notions, Avis and Toussaint [1981] considered the visibility of a polygon from an edge. The complete, strong and weak visibility terms were defined in Section 2.5. The problem was motivated using the question of how a guard can be placed in a polygon [Avis and Toussaint 1981]. If a polygon is completely visible from an edge, a guard can be placed anywhere on the edge, and if it is strongly visible from an edge, there is at least one place a guard can be placed to protect the polygon. Finally, the weak visibility of a polygon from an edge implies that the polygon cannot be guarded from a single position of the edge. They developed an algorithm to determine the visibility polygon from a given edge in linear-time. Their work was extended to the visibility between two edges of a polygon [Avis *et al.* 1986]. As the visibility between edges is more applicable to this dissertation, the extended definitions [Avis *et al.* 1986] are given as follows.

- Edge uv is completely visible from edge xy if every point on edge uv is visible to every point on edge xy.
- Edge uv is strongly visible from edge xy if there exists a point on edge xy that is visible to every point on edge uv.
- Edge uv is weakly visible from edge xy if every point on edge uv is visible to some point on edge xy.

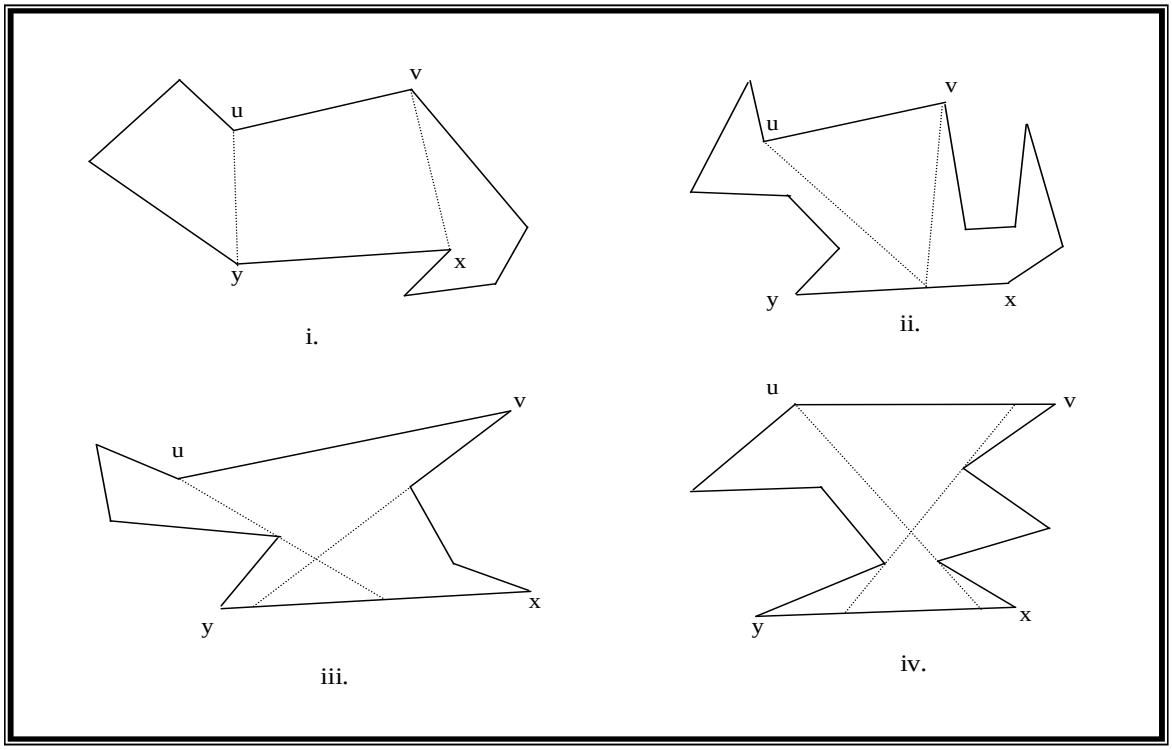

Figure 2.11: Visibility between two edges, uv and xy [Avis *et al.* 1986]

- Edge uv is partially visible from edge xy if there exists a point on edge uv which is visible to some point on edge xy.

Figure 2.11(i-iv) shows complete visibility, strong visibility, weak visibility and partial visibility between edges uv and xy respectively.

The partial edge visibility is directly applicable in placing axial lines of arbitrary orientation to cross shared edges between a collection of convex polygons. In the case of ALP-ALOR and ALP-ALCP, if a line is placed to cross the adjacencies between the convex polygons (orthogonal rectangles in the case of ALP-ALOR), the adjacencies must be partially visible to one another. du Plessis and Sanders [2000] conducted research on partial edge visibility in chains of rectangles and developed a linear time algorithm to compute the problem. In the case of ALP-OLOR and ALP-SC-OLOR, the partial visibility between the adjacencies is restricted to be in horizontal orientation (vertical orientation for horizontal adjacencies), that is based on common y-range value (and based on common x-range value for horizontal adjacencies).

2.7.3 The Art Gallery Problem

The art gallery problem is a well-known guarding problem introduced by Victor Klee in 1973 [O'Rourke 1987]. The problem deals with determining the minimum number of guards required to protect an art gallery with n walls. In terms of a graph representation, the problem would be to determine the minimum covering points (guards or cameras) that can survey 360^0 with respect to their fixed position in a simple polygon of n vertices (art gallery) [de Berg *et al.* 2000]. There are many variations of the art gallery problem – restricting the polygons to be orthogonal polygons, the polygons to have holes or be without holes, considering mobile guards (edge or diagonal guards) and stationary guards (vertex or point guards), and variations on the power of the guards, etc. [O'Rourke 1987; Urrutia 2000; Michael and Pinciu 2003]. In this subsection previous work on some of these variations is discussed.

The problem of determining the minimum number of guards required to cover a given polygon is NP-hard [O'Rourke 1987]. The transformation was from an instance of 3SAT, a well known NP-complete problem [Garey and Johnson 1979]. Another approach to look at the guarding problem is to determine the number of guards that are always sufficient and sometimes required to cover a given polygon. A convex polygon can be guarded by a single guard, and the guard can be placed anywhere within the polygon and its boundaries. Finding the lower bound on the number of guards for any simple polygon is more challenging as it depends on the configuration (shape) of the polygon and the placement of the guards. In solving the art gallery problem, Chvátal [1975] was the first to prove that $\lfloor n/3 \rfloor$ guards are always sufficient and sometimes necessary to patrol an art gallery, where n is the number of vertices of the gallery. The proof is based on the claim that every polygon can be partitioned into at most $\lfloor n/3 \rfloor$ triangles in such a way that each partition is patrolled by a single guard, which was proved by induction. A simplified and easier proof to the problem was presented by Fisk [1978] by triangulating the polygon and then 3-colouring the vertices of each triangle. By placing guards in the least frequently used colour in colouring the resulting graph, $\lfloor n/3 \rfloor$ guards are always sufficient to patrol each triangle and hence the polygon. A comb polygon (see Figure 2.5) with k-prongs, $n \geq 3k$, requires $\lfloor n/3 \rfloor$ guards to cover it since each prong needs to be covered by one guard. For a complete proof, the interested reader can refer to O'Rourke [1987, p.4–13]; de Berg *et al.* [2000, p.47–58]. Moreover, de Berg *et al.* [2000] presented an algorithm to compute the number of guards that are always sufficient and sometimes required to cover an art gallery with n walls in $O(n \log n)$ time.

Kahn *et al.* [1983] considered a simplified version of the art gallery problem where the

gallery is restricted to be an orthogonal polygon. An orthogonal polygon is one whose edges are parallel to one of the coordinate axes. Orthogonal polygons are useful since they can be used for approximating arbitrary simple polygons [O'Rourke 1987]. The solution to the problem was given by Kahn *et al.* [1983] mainly based on the work of Fisk [1978], and that $\lfloor n/4 \rfloor$ guards are sufficient to cover an orthogonal art gallery. The $\lfloor n/4 \rfloor$ guards are required to cover an orthogonal comb polygon (see Figure 2.6). The method they employed was first partitioning the polygon into convex quadrilaterals (quadrilateralization) and then placing diagonals on the non-intersecting vertices of the convex quadrilateral [Kahn *et al.* 1983]. The resulting graph can be coloured using 4-colours (the intersection points of the diagonals are not considered as vertices of the graph). By placing guards in the position of the least frequently used colour, the entire orthogonal polygon can be guarded by $\lfloor n/4 \rfloor$ guards.

Another variation of the art gallery problem studied by Michael and Pinciu [2003] is to find the minimum number of guards to protect the gallery not only from external entities but also from untrustworthy guards – guards must patrol each other besides protecting the gallery. To state the problem formally, let P be a simple polygon (art gallery) to be guarded. A set S of points in P is a "guarded guard set for P" if the following conditions hold.

i. For every point $x \in P$, there exists a point $w \in S$ such that x is visible from w.

ii. For every point $w \in S$, there exists another point $v \in S$ such that w is visible from v.

Michael and Pinciu [2003] have shown that $\lfloor (3n-1)/7 \rfloor$ guards are sufficient to cover an art gallery of size n, where $n \geq 5$ and the "guards are guarded" each other. Furthermore, for orthogonal art gallery problem of size $n \geq 6$ whose "guards are guarded", $\lfloor n/3 \rfloor$ guards are sufficient to protect the gallery and the guards from each other. The method they applied in computing the "guarded guard set" is based on Fisk [1978]'s work of the art gallery problem and then proving their results hold by induction on the number of vertices of the polygon.

Sanders [1999] considered the application of ALP in guarding problems by posing a new variation in the area where the guards can only see along a single ray. The problem was called ray guarding. By considering the collection of orthogonal rectangles as rooms of an art gallery and the adjacencies between the orthogonal rectangles as walls that separate the gallery, the problem is to place the doorways of the gallery in such a way that it can be protected by a minimum number of guards. The doorways of the gallery would be placed in the orientation of the axial lines that cross the adjacencies between the orthogonal rectangles. In the case of ALP-SC-OLOR, the problem would be to place the doorways in such a way that they can be protected by the minimum number of horizontal ray guards – guards that can only see along a horizontal ray. Moreover, each doorway is allowed to be guarded by

only one guard while a guard can patrol one or more doorways. This makes the guards responsible for the doorways they are assigned to.

2.7.4 The Axial Line Placement Problem

The axial line placement problem deals with finding the minimum number of straight line segments that can cross the adjacencies between a collection of convex polygons. Variations of this problem are the convex polygons to be orthogonal rectangles while the axial lines have orthogonal orientation (orthogonal axial lines in orthogonal rectangles), the convex polygons to be orthogonal rectangles the axial lines have arbitrary orientation (arbitrary axial lines in orthogonal rectangles). Sanders [2002] has shown that each of these problems is NP-complete. Previous research on the subproblems of ALP are discussed next.

2.7.4.1 ALP-OLOR

As stated above, Sanders [2002] has shown that the problem of placing orthogonal axial lines to cross adjacencies in a configuration of adjacent orthogonal rectangles is NP-complete. The proof of NP-completeness was presented in two phases: first transforming the vertex cover problem for planar graphs [Garey and Johnson 1979] to a Stick diagram mapping the vertices of the graph to horizontal lines and the edges of a graph to vertical lines; and the second transformation is from Stick diagram to an instance of ALP-OLOR mapping the horizontal lines to a choice of axial lines and the vertical lines to the adjacencies between orthogonal rectangles that are crossed by the axial lines. The vertex cover problem can be stated as [Garey and Johnson 1979]:

> *Given a graph G(V,E) find the fewest number of vertices $V_c \subseteq V$ such that each edge in the graph is incident to at least one of the vertices in V_c, i.e. if $pq \in E$, then at least one of p or q is in V_c.*

Sanders [2002] also developed a heuristic algorithm which gives an approximated solution to the ALP-OLOR problem. The heuristic identifies the adjacencies using a line sweeping method and then applies a greedy algorithm to determine the axial lines. The heuristic algorithm runs in $O(n^3)$ although the worst case time-complexity is $O(n^4)$ (this configuration of the orthogonal rectangles is a special case where the choice of extending the axial lines would be $O(n)$). The heuristic algorithm has five essential steps [Sanders 2002].

1. Determining adjacencies between the orthogonal rectangles using a line sweeping method. The algorithm sorts the rectangles in ascending order of their x-coordinate (only vertical adjacencies are considered as the same can be applied to the horizontal adjacencies)

while breaking ties based on y-coordinates. Each adjacency is determined by comparing the coordinates of the right edge of a rectangle with the coordinates of the left edge of the next rectangle. If the rectangles have the same x-coordinate, the rectangles could share an adjacency and this would be verified using the y-coordinate. Otherwise, the rectangles do not share an adjacency, and consequently the algorithm considers the next rectangle.

2. Generating all orthogonal axial lines that cross the adjacencies between the rectangles. This step is based on the common range of y-coordinate of the adjacencies to determine how far an axial line can be extended.

3. Determining the essential lines. These lines are the ones that cross at least one adjacency that has never been crossed by another axial line.

4. Removing redundant lines. These are the lines that cross only the adjacencies that are already crossed by the essential ones.

5. Once the redundant lines are removed, there could also exist "choice lines" with the essential lines. These are lines only some of which are necessary while the others can be removed. The algorithm resolves the choice conflict by repeatedly choosing the "choice line" which crosses the highest number of previously uncrossed adjacencies.

Furthermore, Sanders *et al.* [2000] studied special cases of the ALP-OLOR problem that can be solved in polynomial time. The result of their study reveals that orthogonal axial lines can be placed in chains and trees of rectangles in polynomial time.

A chain of rectangles is a collection of orthogonally aligned rectangles such that each rectangle is horizontally adjacent to at most one other rectangle on both sides [Sanders *et al.* 2000]. They applied a greedy technique to develop an algorithm that solves the problem in $O(n \lg n)$ time complexity for chains of rectangles. The algorithm comprises five steps [Sanders *et al.* 2000].

1. Sorting the rectangles in ascending order of their x-coordinate while breaking ties based on minimum y-coordinate. This can be done using the merge-sort algorithm in $O(n \lg n)$ time complexity, where n is the number of orthogonal rectangles.

2. Defining the chain of the rectangles. By traversing through the sorted list of orthogonal rectangles from left to right, this step can be done in $O(n)$.

3. Determining forward lines. The forward lines are produced by traversing through the chain of rectangles having common range of y-coordinate which implies a line can be

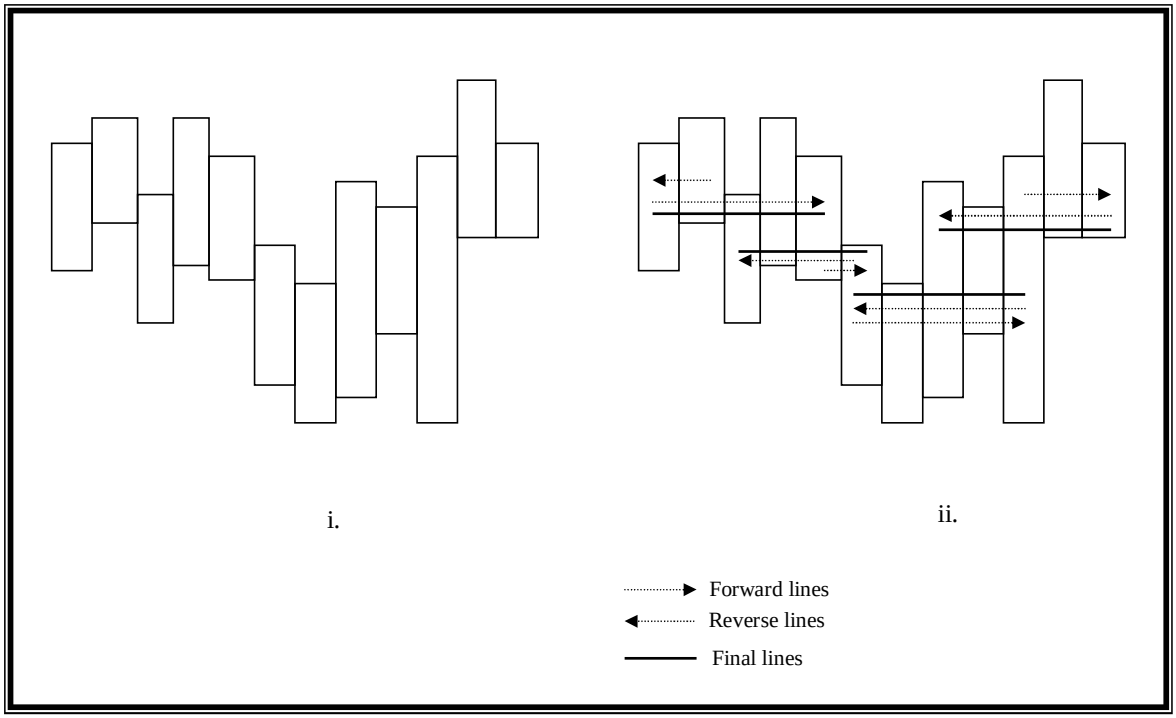

Figure 2.12: An example of a chain of rectangles and the forward, reverse and final lines produced by the $O(n \lg n)$ algorithm

placed to cross the adjacencies between the chain of rectangles. If the last adjacency fails to have common range of y-coordinate with the previous chain, a new line is started from the last adjacency. Clearly, determining the forward lines can be done in $O(n)$.

4. Determining reverse lines. The same as step 3 but here the algorithm starts from the rightmost rectangle and traverse towards the left. This process can also be done in linear time.

5. Merging the forward and reverse lines. The forward lines are extended as far as possible to the right while the reverse lines are extended as far as possible to the left. Moreover, each adjacency is crossed by both forward and reverse lines. By merging these lines, the algorithm produces maximal lines, and hence the final lines. The merging step can be done in linear time since every rectangle is visited once – from left to right.

Figure 2.12 shows an example of a chain of rectangles and the forward, reverse and final lines produced by the algorithm.

A tree of rectangles is a collection of orthogonally aligned rectangles where each rectangle is horizontally adjacent to zero or one rectangles on the left (right) and to zero or more rectangles on the right (left) [Sanders *et al.* 2000]. Sanders *et al.* [2000] developed an algorithm that runs in $O(n^2)$ time complexity to compute the minimum axial lines in trees of rectangles. An example of a tree of rectangles and the worst performance of the algorithm is given in Figure 2.13.

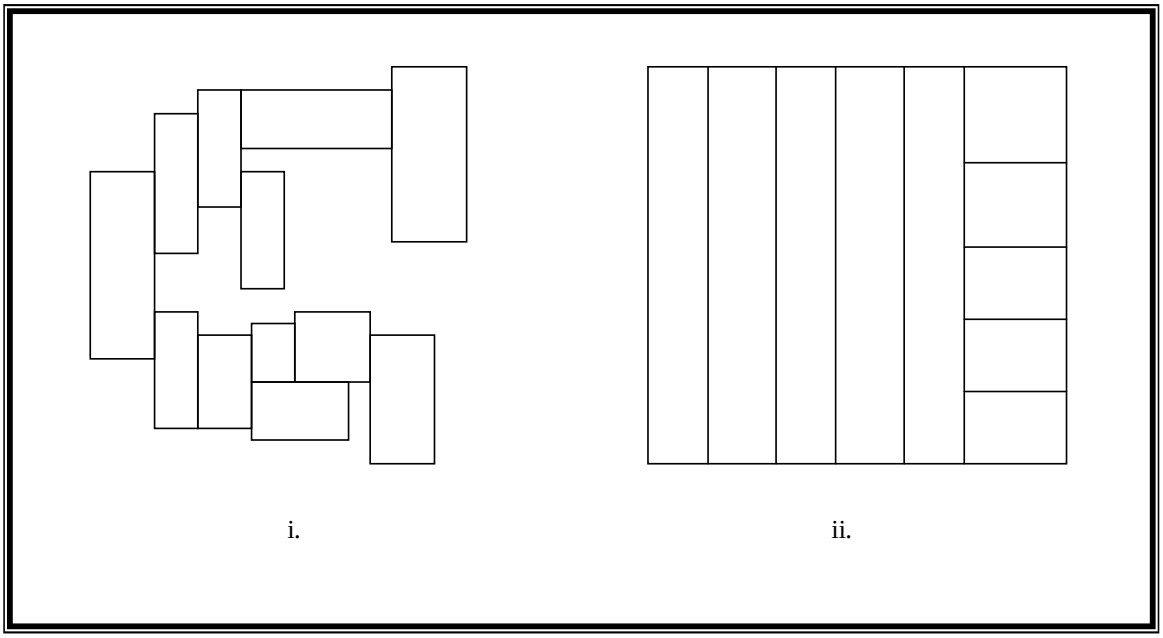

Figure 2.13: A tree of orthogonal rectangles and the worst case of the algorithm developed by Sanders *et al.* [2000]

The results obtained for trees and chains of rectangles are generalized to any hole free collection of orthogonal rectangles. Kruger and Sanders [2005] have shown that ALP-OLOR is polynomially solvable for all hole free collections of rectangles. A hole in a collection of rectangles is a region which is entirely bounded by the collection of rectangles but is not part of the rectangles. And thus a hole free collection of rectangles is a configuration of rectangles with no holes.

The method used by Kruger and Sanders [2005] was first to construct a graph from the collection of rectangles in such a way that the adjacencies between the rectangles are represented by vertices and an edge is placed between two vertices if their corresponding adjacencies can be crossed by the same line. Then a minimum clique cover of the resulting

graph produces the minimum number of lines required to cross each adjacency between the collection of rectangles. Kruger and Sanders [2005] have shown that if the collection is hole free, then the resulting graph can be triangulated and hence the minimum clique cover can be computed in polynomial time [Golumbic 1980]. Consequently, ALP-OLOR can be solved in polynomial time for hole free collections of rectangles.

The approach can be extended to a collection of rectangles with holes where the resulting graph does not contain chord-less cycles of more than three vertices [Kruger and Sanders 2005].

2.7.4.2 ALP-ALOR

Sanders [2002] has shown that the problem of placing axial lines of arbitrary orientation to cross adjacencies between orthogonal rectangles is NP-complete by transforming the vertex cover problem for planar graphs into an instance of ALP-ALOR. The transformation was done in two steps: first from vertex cover for planar graphs to a stick diagram and the second step from stick diagram to an instance of ALP-ALOR mapping the horizontal lines to the axial lines while the vertical lines to a canonical choice unit.

Sanders and Kenny [2001] considered a number of other heuristics that could give approximate solutions to ALP-ALOR. Some of these are presented below.

- *Calculating a solution in two passes.* The heuristic is a modification of the ALP-OLOR heuristic with two passes. The first pass is to extend lines from the leftmost rectangle to the rightmost rectangle, while the second pass is extending lines from top to lower neighbours. The two passes generate all possible lines. Then the final three steps are the same as ALP-OLOR: determining the essential lines, removing redundant ones and resolving choice conflicts which were discussed in the previous subsection.

- *Finding the longest axial line.* By locating the extreme rectangles, the heuristic generates the longest chain until all of the adjacencies are crossed. The idea behind the heuristic is that the line that crosses these extreme rectangles could also cross other intermediate adjacencies. The algorithm continues until all the adjacencies are crossed, and the result would be an approximate solution to ALP-ALOR.

- *Crossing one adjacency at a time.* Once all adjacencies have been determined, the heuristic stores the information in an adjacency matrix. The adjacencies are then sorted in order of their x-coordinate while breaking ties with minimum y-coordinate first. By taking one adjacency at a time, the heuristic algorithm crosses the adjacencies

with a line as long as the line can be extended.

- *Improved crossing one adjacency at a time.* Ferris [2003] has developed a heuristic algorithm to the ALP-ALOR problem by improving the "Crossing one adjacency at a time" heuristic. The improvement of the new heuristic lies in the formation of a new chain of rectangles when a line fails to cross the chain of rectangles. That is, the "Crossing one adjacency at a time" heuristic starts with the next adjacency once a chain is formed whereas the improved heuristic creates a new chain starting from the last rectangle of the current chain when a line fails to cross all the adjacencies within the chain of rectangles. Another improvement is on extending the axial lines to be maximal and removing redundant lines. The worst case running time of the heuristic is $O(n^3)$ while its average case running time is $O(n^2)$.

Sanders [2003] also developed an algorithm that determines the exact solution to ALP-ALOR. As the problem is NP-complete, the exact solution algorithm could only compute the solution in reasonable time for small input size. The development of this algorithm is essential in evaluating the heuristics developed for ALP-ALOR and ALP-OLOR with respect to the exact solution in terms of their running time and performance. The method employed in determining the exact solution is by repeatedly extending each line while considering every adjacency provided that the line can cross the adjacency.

2.7.4.3 ALP-ALCP

As discussed in the previous subsections, the problem of placing arbitrary axial lines to cross the adjacencies between a collection of convex polygons is NP-complete [Sanders 2002]. The proof of NP-completeness was derived from the NP-completeness of ALP-ALOR since it is a special case of ALP-ALCP. Hagger [2005] developed a heuristic algorithm for ALP-ALCP based on a greedy method. The algorithm selects a starting polygon and then a set of lines are generated using a depth first search. Then the algorithm recursively chooses exactly one line that extends the furthest from the current set of lines. It terminates when all adjacencies are crossed. The complexity of the algorithm is $O(ln^3)$ where n is the number of polygons and l is a parameter passed to the algorithm that limits the number of lines found by the search.

Hagger [2005] outlined three possible ways to select a starting polygon: choosing a polygon randomly and then generating lines from it; choosing a polygon based on the number of adjacencies and position in the configuration, and generating lines from it; and detecting special configurations in the collection of convex polygons and then placing lines using spe-

cific algorithms applied to the special configurations. Associated with the third option of selecting a starting polygon, Hagger [2005] identified two possible configurations of collections of convex polygons that can be solved in polynomial time. These are chains of convex polygons and stars of convex polygons.

A chain of convex polygons is a configuration of convex polygons in which each polygon is adjacent to exactly to two polygons except the first and the last polygons in the chain that are adjacent to only one polygon. ALP in chains of convex polygons can be transformed to interval point cover by a quadratic time algorithm [Hagger 2005]. A star of convex polygons is a collection of convex polygons all of which are adjacent to a central polygon. The algorithm of chains of convex polygons can be extended to transform ALP in stars of convex polygons to maximum cardinality matching. A network of stars consists of a collection of stars of convex polygons. Figure 2.14 shows examples of configurations of a chain of convex polygons and a star of convex polygons.

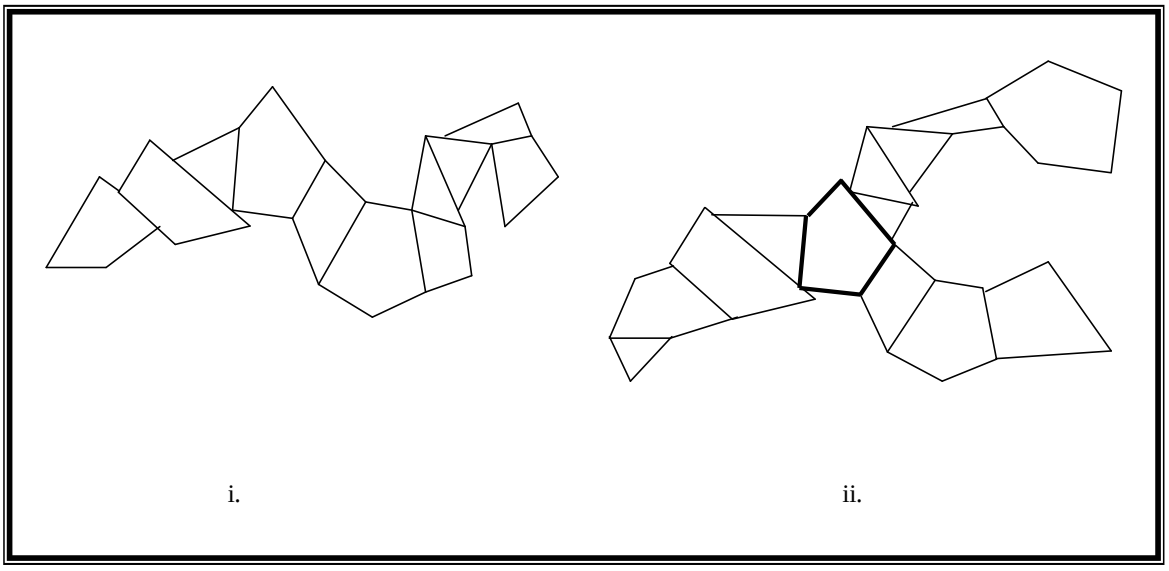

Figure 2.14: (i) A chain of convex polygons and, (ii) a star of convex polygons

2.7.4.4 Summary

ALP-OLOR is the simplified version of the ALP-ALOR and ALP-ALCP problems. In general, various research on ALP-OLOR could be extended to ALP-ALOR and ALP-ALCP by adopting the variations that could arise due to the arbitrary orientation of the axial

lines and the configuration of the convex polygons. For instance, the NP-completeness of ALP-OLOR can also be extended to the other two. Moreover, the heuristic algorithm for ALP-OLOR was adopted to approximate the exact solution to ALP-ALOR with some modifications due to the orientation of the axial lines, i.e. in ALP-OLOR only horizontal adjacencies are considered while in ALP-ALOR vertical adjacencies need also be considered. In this dissertation, we considered ALP-SC-OLOR, which is a single-crossing version of the ALP-OLOR problem, and investigated if a polynomial time algorithm could be developed to solve the problem. The research is vital in that the outcome could be extended to the single-crossing versions of ALP-ALOR and ALP-ALCP as research has not been done on the single-crossing version of these problems.

2.7.5 Domination and Dominating Sets

Domination and dominating sets have been active research areas in graph theory. As the transformation presented in Chapter 4 is based on a variation of a dominating set problem, a description of the general dominating set and its variations are presented in this subsection.

Let $G = (V, E)$ be a graph where V is the set of vertices and E the set of edges. A vertex v dominates a vertex u (or vice versa) if either $(u, v) \in E$ or $u = v$. A k-dominating set of G is a set $D \subseteq V$ with k vertices in which every vertex of V is dominated by at least one vertex of D. When there is no edge in E connecting any two vertices of D, then D is called the independent dominating set of G. The k-dominating problem was found to be NP-complete [Corneil and Perl 1984]. Furthermore, they showed that the independent dominating set problem for arbitrary graphs is NP-complete. However, Corneil and Perl [1984] proved that the k-domination set problem is polynomially solvable for trees and permutation graphs. Based on the number of vertices that can dominate a vertex and those being dominated, there are several variations of the dominating set problem.

Perfect domination set (or perfect codes) D of a graph $G = (V, E)$ is a variation of the dominating set in which for every vertex $v \in V$ either v is in D or has exactly one neighbour in D. This problem was studied by, among others, Livingston and Stout [1990]; Klostermeyer and Eschen [2000]; and Fellows and Hoover [1991]. Moreover, Fellows and Hoover [1991] considered semi-perfect and weakly perfect domination problems. A semi-perfect domination set D of a graph G is a subset of the set of vertices V where D is a dominating set and each vertex not in D is adjacent to exactly one vertex in D. In semi-perfect domination, there is no restriction on the number of adjacencies between the vertices in D. On the other hand, a weakly perfect domination set D of a graph G is a subset of the vertices V in which

each vertex not in D has one neighbour in D, and the vertices in D are adjacent to at most one vertex in D. Fellows and Hoover [1991] showed that the perfect domination, semi-perfect domination and weakly perfect domination problems are members of the class of NP-complete problems. The reduction was from the three dimensional matching problem. But the perfect domination set problem is polynomially solvable for a family of trees of graphs [Fellows and Hoover 1991].

Klostermeyer and Eschen [2000] considered a generalization of the independent and distance domination problems. $D \subseteq V$ is a (k, q) independent dominating set of G if every vertex of V is within a distance of k of at least one vertex of D and any pair of vertices of D are at a distance of at least q+1. They proved $(k, k + 1)$ is NP-complete by a reduction from one in three 3SAT with no negated trials. They also presented algorithms that determine if a graph contains a perfect code in polynomial time for graphs of families of trees and interval graphs.

2.8 Conclusion

The chapter introduced the space syntax method, a tool for analysing urban layout and its components. ALP, one of the components of space syntax, has been found to be NP-complete which implies that it can only be computed for small input size in reasonable time. Consequently, developing heuristics that give approximated solutions and investigating special cases of the problem that can be solved in polynomial time is essential. Some of the heuristics and special cases solvable in polynomial time are discussed in the chapter. The dissertation considered another approach of approximating the axial line placement problem whereby the adjacencies between the orthogonal rectangles are restricted to be crossed only once, and explored if the problem can be solved efficiently or if it is NP-complete. Depending on the outcome of the research, the result could be extended to the single-crossing of ALP-ALOR and ALP-ALCP by adopting the variations on the orientation of the axial lines and the configuration of the convex polygons.

The chapter also discussed the theory of NP-completeness focusing on the common properties of the problems in the P and NP class of problems and techniques for proving NP-completeness of a new problem. It was also highlighted why we need to concentrate on heuristics and polynomial time solvable special cases of an NP-complete problem rather than attempting to compute for the exact solution. Visibility and guarding problems were discussed in depth as they have some commonalities with ALP. The partial edge visibility is essential in placing the axial lines to cross adjacencies between a collection of convex

polygons. In the ALP-SC-OLOR case, the partial edge visibility is restricted to common y-coordinate range as the focus is on vertical adjacencies and horizontal lines. ALP can also be considered as a variation of the art gallery problem where the gallery is protected by ray guards – guards that can only see along a ray. The bipartite independent dominating set problem, which is used during the transformation of NP-completeness of ALP-SC-OLOR, is a variation of the dominating set problem. As a result, a brief discussion on domination was included in the chapter. A formal definition of the bipartite independent dominating set problem is given in Chapter 4.

Chapter 3

Research Method

3.1 Introduction

Chapter 2 stated the aim of the research along with the general ALP problem and the space syntax method. It also presented a brief description of the theory of NP-complete problems focusing on the method of proving NP-completeness of a new problem and the common properties of NP-complete problems. Previous work on visibility and guarding problems were also discussed in more depth as the problems have some commonalities with ALP and ALP-SC-OLOR.

The chapter begins by addressing the research question. Next, an outline of the approach followed in attempting to answer the research question is given. An overview on the direction of the research once a decision was made whether ALP-SC-OLOR is polynomially solvable or NP-complete is also included in the chapter. Lastly, the conclusion of the chapter is presented.

3.2 Research Question

As discussed in the previous chapters, the axial line placement problem was introduced by Sanders [2002] in attempting to automate the space syntax method. Sanders [2002] identified two major research questions when he was dealing with ALP: multiple adjacency crossings and single adjacency crossings. In multiple adjacency crossings, the adjacencies between the collection of rectangles are crossed by at least one axial line whereas in single adjacency crossings, the adjacencies are crossed by exactly one axial line. The multiple adjacency crossing of ALP has been studied extensively, some of which were discussed in Section 2.7.4. On the contrary, research had not been done on the single-crossing version of ALP. In this dissertation, we dealt with the single-crossing of ALP where the axial lines

are restricted to have orthogonal orientation while the collection of convex polygons are restricted to orthogonal rectangles (ALP-SC-OLOR). Considering the most simplified form of ALP gives insight into the single-crossing versions of ALP-ALOR and ALP-ALCP.

In the previous chapters, it was discussed that ALP-OLOR which is the multiple adjacency crossing version of ALP-SC-OLOR is NP-complete. The NP-completeness of the problem is due to the 'choice' axial lines that cross two or more adjacencies but some of which are redundant, i.e. the 'choice' of axial lines may not be resolved in polynomial time. In this dissertation, we considered the single adjacency crossing version of placing orthogonal axial lines in orthogonal rectangles (ALP-SC-OLOR) and examined whether the restriction imposed on the adjacencies could lead the problem to be solved in polynomial time or if it is NP-complete. To formally state the research question:

> *Given a collection of non-overlapping adjacent orthogonal rectangles, can a minimum number of single-crossing orthogonal axial lines be placed in polynomial time to cross the adjacencies between the rectangles?*

The problem of placing single adjacency crossing of orthogonal axial lines in orthogonal rectangles has some commonalities with the general partitioning problem which is a known NP-complete problem [Garey and Johnson 1979]. In order to elaborate the commonality, first let us transform ALP-SC-OLOR into a decision problem. Since ALP-SC-OLOR is an optimization problem, an additional variable is introduced that bounds the number of single-crossing axial lines. That is,

> *Instance*: Given a collection of non-overlapping orthogonal rectangles $R_1, R_2, ..., R_n$ where each R_i, $1 \leq i \leq n$, is adjacent to one or more other rectangles and a positive integer $k \leq 2n - 4$.
>
> *Question*: Is there a set of orthogonal axial lines L such that each adjacency between the orthogonal rectangles is crossed only once and $|L| \leq k$?

The number of vertical adjacencies formed by a collection of orthogonal rectangles is bounded by $2n - 4$, where n is the number of rectangles. This can be verified by Euler's formula for triangle-free graphs which is discussed in more depth in Section 5.2.2. If the single-crossing lines are placed in such a way that a line crosses only one vertical adjacency, the total number of single-crossing lines required will maximally be $2n - 4$. But the challenge is to compute the minimum single-crossing lines to cover all vertical adjacencies in the configuration.

As it is outlined above, ALP-SC-OLOR has some commonalities with the general partitioning problem. The partitioning problem can be written as a decision problem in the following form:

Instance: Given a finite set A and a size $s(a) \in Z^+$ for each $a \in A$.

Question: Is there a subset $A' \subseteq A$ such that

$$\sum_{a \in A'} s(a) = \sum_{a \in A - A'} s(a)?$$

The adjacencies between the orthogonal rectangles can be computed in polynomial time, i.e. $O(n \log n)$ using a line sweeping method [Sanders 2002]. Let C be a collection of the adjacencies. If we can partition C into k disjoint sets, for some positive integer k, such that if $u, v \in C_i$, $1 \leq i \leq k$, are partially visible based on y-coordinate value, then each element belonging to the same subset would be crossed by the same axial line.

On the contrary, the geometrical configuration of the rectangles could possibly force ALP-SC-OLOR to be solved in polynomial time as the partitioning problem may not be represented in a geometrical grid.

3.3 Approach

There were two general approaches followed to answer the research question:

- to transform ALP-SC-OLOR to a problem in P in polynomial time which implies ALP-SC-OLOR is polynomially solvable.
- to transform a known NP-complete problem into ALP-SC-OLOR in polynomial time which implies ALP-SC-OLOR is NP-complete.

If ALP-SC-OLOR had been found to be polynomially solvable, the focus of the research would have been on:

- developing an algorithm that solves ALP-SC-OLOR.
- analysing the correctness and running time of the algorithm.
- the analysis on the lower bound of the problem.

On the other hand, if the problem appeared to be NP-complete, the focus of the research would be on:

- developing a heuristic that gives a reasonably approximated solution of ALP-SC-OLOR.

- theoretical and empirical analysis of the performance and running time of the heuristic.

- exploring special configurations of rectangles in which ALP-SC-OLOR is polynomially solvable.

In answering the research question, the second approach was followed which is attempting to transform a known NP-complete problem to ALP-SC-OLOR. As a starting point, we defined a graph theory problem which is applied only to bipartite graphs, named bipartite independent dominating set (BIDS), and showed that it is NP-complete by a reduction from the vertex cover problem. Previous research has shown that any bipartite planar graph can be represented in a grid using vertical and horizontal segments. Using the outline of transforming a bipartite planar graph into a grid of vertical and horizontal segments presented in Chapter 4, we defined an orthogonal bipartite independent dominating set (OBIDS) problem which is a segment representation of BIDS. Then using a suitable choice unit we transformed OBIDS to a special case of ALP-SC-OLOR – crossing the adjacencies between the rectangles using normalized lines (ALP-NSC-OLOR). The normalized lines are single-crossing with the additional property that the lines are required to be maximal when they cross choice adjacencies. Choice adjacencies are adjacencies that can be crossed by more than one maximal line.

A heuristic for ALP-SC-OLOR was also developed based on a greedy method. The heuristic consists of three parts: determining adjacencies between the collection of rectangles, generating all possible single-crossing lines, and selecting a minimum number of single-crossing lines that cross all the vertical adjacencies only once. Theoretical and empirical analysis on the performance and running time of the heuristic was also carried out which is presented in Chapter 5 in detail. The last task in the category outlined above, which is exploring for special cases of ALP-SC-OLOR that are polynomially solvable, is left as future work.

3.4 Conclusion

The chapter has presented the research question which was to determine whether a polynomial time algorithm could be developed for ALP-SC-OLOR or whether it is an NP-complete problem. The two general approaches that could have been taken to answer the research

question were also discussed followed by an outline of the direction of the research depending whether ALP-SC-OLOR is polynomially solvable or whether it is NP-complete.

As ALP-SC-OLOR is found to be NP-complete, the research carried on with developing a heuristic for ALP-SC-OLOR but exploring special configurations of rectangles in which ALP-SC-OLOR is polynomially solvable is left as future work due to time constraints. The next chapter presents a transformation of NP-completeness of ALP-SC-OLOR.

Chapter 4

NP-completeness of ALP-SC-OLOR

4.1 Introduction

In the previous chapters, it was stated that ALP was found to be NP-complete, even when it is restricted to placing orthogonal axial lines in orthogonal rectangles [Sanders 2002]. This research considered the problem of placing orthogonal axial lines in orthogonal rectangles where the adjacencies between the orthogonal rectangles are allowed to be crossed only once. At first glance, the restriction imposed on the adjacencies to be crossed only once seems to simplify the problem, and hence it might be solvable in polynomial time. But it turned out that selecting the minimum combination of single-crossing lines may require exponential time and hence the single-crossing version of orthogonal axial lines in orthogonal rectangles has been found to be NP-complete. This chapter deals with the proof of NP-completeness of ALP-SC-OLOR.

In the following section, a formal definition of the bipartite independent dominating set (BIDS) problem is given along with its proof of NP-completeness. Then a discussion on representing a planar graph in a grid is presented accompanied by an outline of transformation of a bipartite planar graph into a grid of segments. The last section deals with the transformation of NP-completeness of ALP-SC-OLOR.

4.2 Bipartite Independent Domination Set

As introduced in the previous chapters, the axial line placement problem is to find a minimum number of maximal lines that cross all the adjacencies between a collection of convex

polygons. One approach of representing ALP as a bipartite graph problem is to represent all the axial lines as a set of nodes of one category and all the adjacencies between the convex polygons as another category of set of nodes such that an edge is placed between two nodes of the resulting bipartite graph if the corresponding axial line intersects the associated adjacency. The solution to the problem would be to select the minimum number of nodes associated with the lines that dominate all the nodes associated with the adjacencies. Likewise, the single-crossing version of the orthogonal lines in orthogonal rectangles (ALP-SC-OLOR) can be represented as a bipartite graph problem from the all possible single-crossing lines to the adjacencies they cross, i.e. the nodes in the bipartite graph which correspond to the single-crossing lines are connected to the nodes associated with the adjacencies that can be intercepted by the lines. The solution to ALP-SC-OLOR would be the minimum number nodes associated with the single-crossing lines that dominate all the nodes associated to the adjacencies provided that each node that corresponds to the adjacencies is dominated by exactly one node in the solution set. An example of a bipartite graph representation of an instance of ALP-SC-OLOR is shown in Figure 4.1. As a result of the bipartite graph representation of ALP, we introduced the following graph theory problem which is defined only for bipartite graphs.

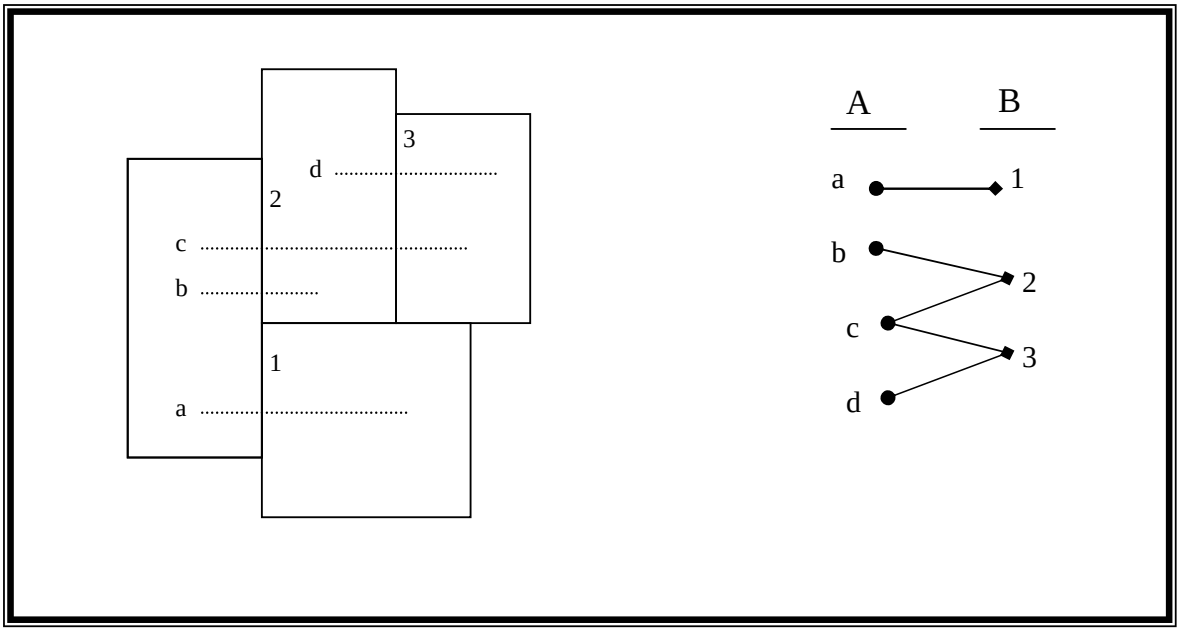

Figure 4.1: Bipartite graph representation of ALP-SC-OLOR

Definition 4.1. Given a bipartite graph $G^{'} = (A, B, E^{'})$, the bipartite independent dominating set (BIDS) of $G^{'}$ is a minimum set $A^{'} \subseteq A$ such that for every $b \in B$, there exists $a \in A^{'}$ in which $(a, b) \in E^{'}$, and for all $a_1, a_2 \in A^{'}$, if $(a_1, b) \in E^{'}$ and $(a_2, b) \in E^{'}$, where $b \in B$, then $a_1 = a_2$.

If $A^{'}$ is a bipartite independent dominating set of $G^{'}$, then each vertex of the second category should be dominated by exactly one element from the set $A^{'}$.

The bipartite independent dominating set problem is an optimization problem. By introducing a new variable M which bounds the cardinality of the solution set, the BIDS problem can be written as a decision problem in the following form:

Instance: Given a bipartite graph $G^{'} = (A, B, E^{'})$ and a positive integer $M \leq |A|$.

Question: Is there a bipartite independent dominating set $A^{'} \subseteq A$ and $|A^{'}| \leq M$?

Theorem 4.1 shows that the BIDS problem is NP-complete for planar graphs. The NP-completeness transformation is from vertex cover for planar graphs. The vertex cover problem can be defined as:

Definition 4.2. Given a graph $G = (V, E)$, the vertex cover of G is a minimum set $V^{'} \subseteq V$ such that for every edge $(u, v) \in E$, at least one of u or v is in $V^{'}$. We say that an edge (u, v) is covered by vertex u (or v) if $u \in V^{'}$ (or $v \in V^{'}$).

The vertex cover problem is NP-complete even when it is restricted to planar graphs [Garey *et al.* 1976]. It is one of the six basic NP-complete problems [Garey and Johnson 1979] which have been used in proving NP-completeness of various optimization problems. When written as a decision problem, vertex cover has the following form:

Instance: Given a graph $G = (V, E)$ and a positive integer $K \leq |V|$.

Question: Is there a vertex cover $V^{'} \subseteq V$ and $|V^{'}| \leq K$?

In vertex cover, at least one of the two vertices that form an edge should be in the solution set.

The following theorem states the NP-completeness of the BIDS problem. The proof method used is local replacement. This type of transformation has been discussed in Garey and Johnson [1979] as one of the three techniques of proving the NP-completeness of a

problem. The transformation by replacing edges with a suitable choice unit also appeared in Kariv and Hakimi [1979]; Fellows and Hoover [1991]; and Corneil and Perl [1984].

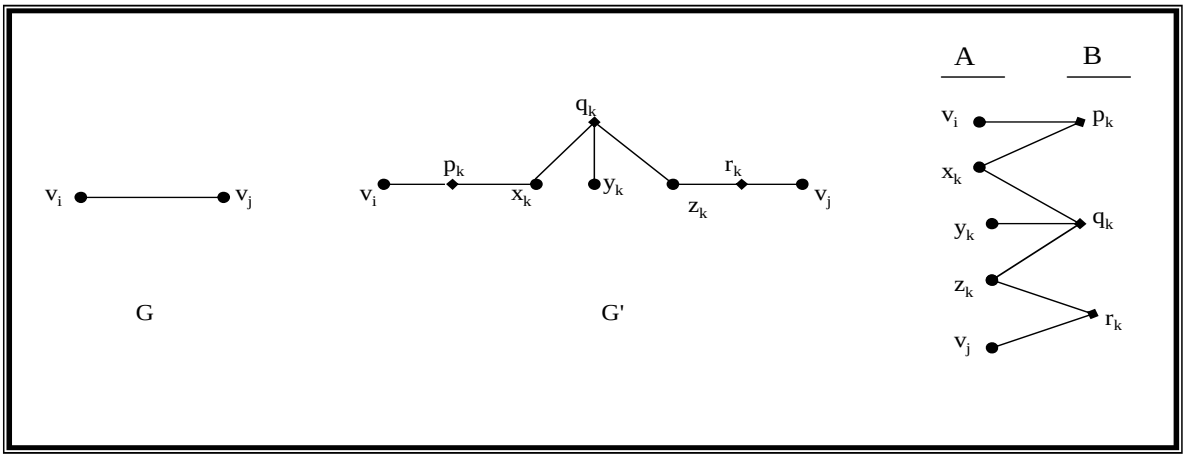

Figure 4.2: Unit of transformation from vertex cover to BIDS

Theorem 4.1. *The BIDS problem is NP-complete for planar graphs.*

Proof. Consider the bipartite graph $G^{'} = (A, B, E^{'})$ and suppose that $A^{'} \subseteq A$ is given, where $|A^{'}| = M$. To check this given $A^{'}$ is a BIDS of $G^{'}$ the following has to be done.

- it has to be established that every vertex of B is dominated by exactly one vertex from $A^{'}$.
- it has to be established that $A^{'}$ is a minimum set with this property.

The above two tasks can clearly be done in polynomial time, and hence the BIDS problem is in NP.

We show that the problem is NP-hard by transforming an instance of vertex cover for planar graphs. Let $G = (V, E)$ be a planar graph. The basic units of the instance of vertex cover are the edges of G. The local replacement substitutes for each edge in E by the choice unit shown in Figure 4.2 in a uniform way. That is, each edge $(v_i, v_j) \in E$ is replaced by a collection of edges (v_i, p_k), (x_k, p_k), (x_k, q_k), (y_k, q_k), (z_k, q_k), (z_k, r_k) and (v_j, r_k), where $(v_i, v_j) = e_k$, to form an instance of BIDS of $G^{'}$. By construction, $G^{'}$ is bipartite as its vertices can be categorized into two sets, and there is no edge in $E^{'}$ connecting the vertices of the same category. The instance of $G^{'} = (A, B, E^{'})$ can be stated as:

$A = V \cup (X \cup Y \cup Z)$, where $X = \cup_k x_k$, $Y = \cup_k y_k$ and $Z = \cup_k z_k$

$B = P \cup Q \cup R$, where $P = \cup_k p_k$, $Q = \cup_k q_k$ and $R = \cup_k r_k$

$E^{'} = \{(v_i, p_k), (x_k, p_k), (x_k, q_k), (y_k, q_k), (z_k, q_k), (z_k, r_k), (v_j, r_k) | (v_i, v_j) \in E$ where $(v_i, v_j) = e_k\}$

Clearly $|A| = |V| + 3|E|$, $|B| = 3|E|$ and $|E^{'}| = 7|E|$, and the instance can be constructed in polynomial time. Moreover, $G^{'}$ is planar since G is planar and replacing each individual segment by the unit choice does not change the planarity of the resulting graph.

We want to show that G has a vertex cover of size K if and only if $G^{'}$ has a BIDS of size $K + |E|$. Suppose $V^{'}$ is a vertex cover of G of size K or less. Since adding additional vertices to $V^{'}$ does not change the property of its vertex cover, we can assume that $|V^{'}| = K$. Since for every edge (v_i, v_j) of G we have at least $v_i \in V^{'}$ or $v_j \in V^{'}$, we can develop a set $A^{'}$ of size $K + |E|$ using the following rules which constitute a bipartite independent dominating set of G'.

1. If $v_i \in V^{'}$ but $v_j \notin V^{'}$, then add v_i and z_k to $A^{'}$.
2. If $v_i \notin V^{'}$ but $v_j \in V^{'}$, then add v_j and x_k to $A^{'}$.
3. If $v_i \in V^{'}$ and $v_j \in V^{'}$, then add v_i, v_j, and y_k to $A^{'}$.

Clearly $V^{'} \subseteq A^{'}$. Moreover at each choice unit of $G^{'}$, and hence every edge consideration of G, exactly one element of X or Y or Z is included in $A^{'}$, and hence $|A^{'}| = K + |E|$. Furthermore, the above rules and the choice of transformation guarantee that the elements of B are dominated by exactly one element of A'. Hence, $A^{'}$ satisfies all the properties of BIDS.

Conversely, suppose A' is a BIDS of G' of size M or less. Without loss of generality, we can assume that $|A^{'}| = M^{'}$. As $A^{'} \subseteq A$, there exist two disjoint sets $V^{'} \subseteq V$ and $D \subseteq (X \cup Y \cup Z)$ such that $A^{'} = V^{'} \cup D$.

From the transformation unit, it is easy to see that each q_i can only be dominated by d_i, $d_i \in D$. Moreover, we cannot have more than one of d_i in $A^{'}$ which is the BIDS of $G^{'}$, otherwise q_i will be dominated by more than one element of $A^{'}$. Therefore, for each edge replacement (choice unit) we have exactly one element of D in $A^{'}$. On the other hand, none of the d_i 's, $d_i \in A^{'}$, can dominate all vertices in B. x_i can dominate p_i and q_i but not r_i; z_i can dominate r_i and q_i but not p_i; and y_i can only dominate q_i but not p_i and r_i. Therefore for every choice unit there must exist at least one $v_i \in V$ such that $v_i \in A^{'}$ and hence $v_i \in V^{'}$. The final assertion implies that every edge in G is covered by at least one

element of $V^{'}$, which fulfils the required property of vertex cover.

From the above discussion, it can easily be deduced that $|V^{'}| = M - |E|$. This is due to the fact that $A^{'} = V^{'} \cup D$, and $V^{'}$ and D are disjoint sets. And for each choice unit of $G^{'}$, we need exactly one element of D in $A^{'}$ and $|D| = |E|$.

Therefore, every BIDS of $G^{'}$ should include the vertex cover of G. That completes the transformation.

□

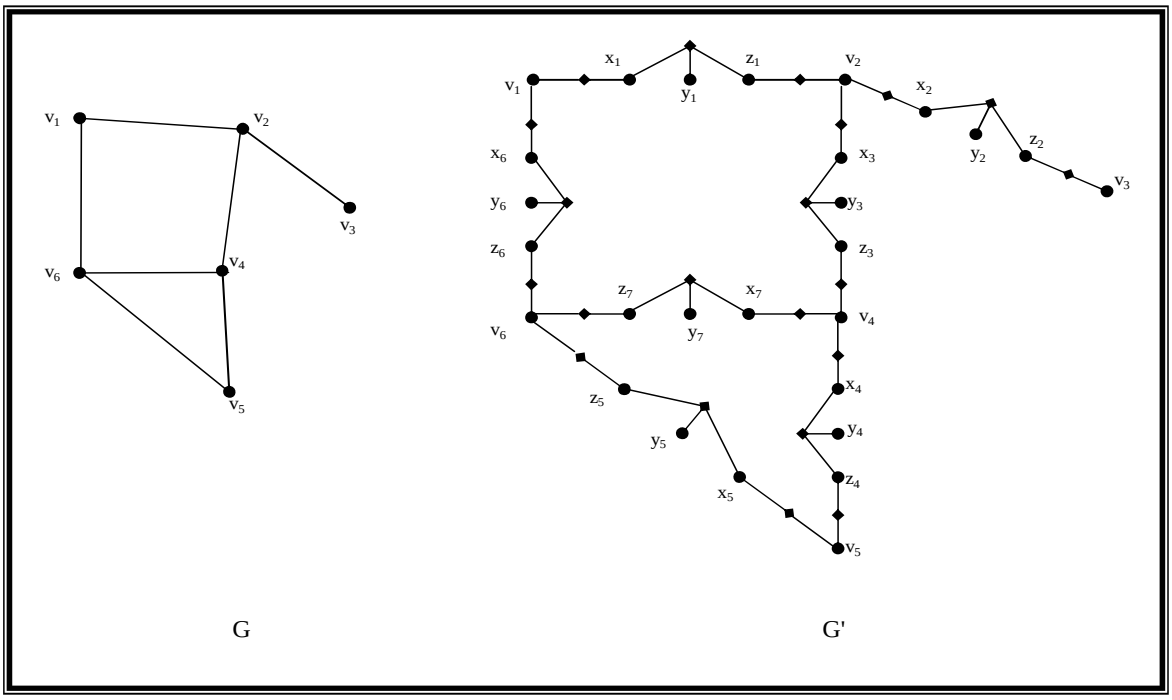

Figure 4.3: A complete transformation from an instance of a vertex cover problem to an instance of BIDS

In Figure 4.3 an illustration is given on how the transformation from G to G' proceeds. For clarity reasons, only the label of the vertices in the first category of $G^{'}$ are given. A brief description of how $A^{'}$ (BIDS of $G^{'}$) is constructed from $V^{'}$ (vertex cover of G) by applying the three rules stated above is presented next.

Let $V^{'} = \{v_2, v_5, v_6\}$. Furthermore, let the first edge of G to consider be (v_1, v_2). Then by rule 2, v_2 and x_1 are added to $A^{'}$, i.e. $A^{'} = \{v_2, x_1\}$.
Then, consider the edge (v_1, v_6). By rule 2, v_6 and x_6 are added to $A^{'}$.

$\Longrightarrow A^{'} = \{v_2, v_6, x_1, x_6\}$.

Then, the next edge to consider would be (v_2, v_3). By rule 1, z_2 is added to $A^{'}$ (v_2 is already in A').

$\Longrightarrow A^{'} = \{v_2, v_6, x_1, x_6, z_2\}$.

Then, consider the edge (v_2, v_4). By rule 1, z_3 is added to $A^{'}$ (v_2 is already in A').

$\Longrightarrow A^{'} = \{v_2, v_6, x_1, x_6, z_2, z_3\}$.

The next edge to consider would be (v_4, v_5). By rule 2, v_5 and x_4 are added to $A^{'}$.

$\Longrightarrow A^{'} = \{v_2, v_5, v_6, x_1, x_4, x_6, z_2, z_3\}$.

The next edge to consider would be (v_4, v_6). By rule 2, x_7 is added to $A^{'}$ (v_6 is already in $A^{'}$).

$\Longrightarrow A^{'} = \{v_2, v_5, v_6, x_1, x_4, x_6, x_7, z_2, z_3\}$.

The last edge to consider would be (v_5, v_6). By rule 3, y_5 is added to $A^{'}$ (v_2 and v_6 are already in $A^{'}$).

$\Longrightarrow A^{'} = \{v_2, v_5, v_6, x_1, x_4, x_6, x_7, y_5, z_2, z_3\}$.

It can be easily verified that $V^{'}$ is vertex cover of G, and $A^{'}$ is BIDS of $G^{'}$.

4.3 Bipartite Independent Dominating Set for Biconnected Planar Graphs

As we have seen in the previous section, the BIDS problem was proved to be NP-complete. The transformation was from the vertex cover problem by replacing the edges with a suitable transformation unit. As our objective is to prove ALP-NSC-OLOR, which will be explained in detail in Section 4.5, is NP-complete we need to represent BIDS in a grid using vertical and horizontal segments. We also need to maintain the property that each of the horizontal segments intercept at least two vertical segments. As a result, the BIDS problem for biconnected planar graphs is considered.

A graph $G = (V, E)$ is biconnected if there is no cut vertex in V, i.e. the removal of any vertex from V does not make the graph disconnected. The vertex cover problem for biconnected planar graphs was shown to be NP-complete [Sanders 2002]. The transformation of NP-completeness was from vertex cover for planar graphs.

Now, suppose the graph G in Theorem 4.1 is a biconnected plane graph. Clearly, the $q_i^{'}s$ in the resulting graph $G^{'}$ are cut vertices. By introducing a new vertex t to the choice unit in Figure 4.2 and new edges that connect t to vertices x, y and z, q is no longer a cut vertex. The new transformation unit is shown in Figure 4.4. For any biconnected planar graph $G = (V, E)$, $|V| \geq 3$, and applying the new transformation in Figure 4.4, $G^{'}$ is a

biconnected bipartite planar graph. The instance of $G^{'} = (A, B, E^{'})$ can be stated as:

$A = V \cup (X \cup Y \cup Z)$, where $X = \cup_k x_k$, $Y = \cup_k y_k$ and $Z = \cup_k z_k$

$B = P \cup Q \cup R \cup T$, where $P = \cup_k p_k$, $Q = \cup_k q_k$, $R = \cup_k r_k$ and $T = \cup_k t_k$

$E^{'} = \{(v_i, p_k), (x_{i,j}, p_k), (x_k, q_k), (x_k, t_k), (y_k, q_k), (y_k, t_k), (z_k, q_k), (z_k, t_k),$
$(z_k, r_k), (v_j, r_k) | (v_i, v_j) \in E$, where $(v_i, v_j) = e_k\}$

Clearly $|A| = |V| + 3|E|$, $|B| = 4|E|$ and $|E^{'}| = 10|E|$, and the instance can be constructed in polynomial time, since every edge of G is replaced by the suitable choice. By applying the rules for the general BIDS, we state the following corollary which is a result of Theorem 4.1.

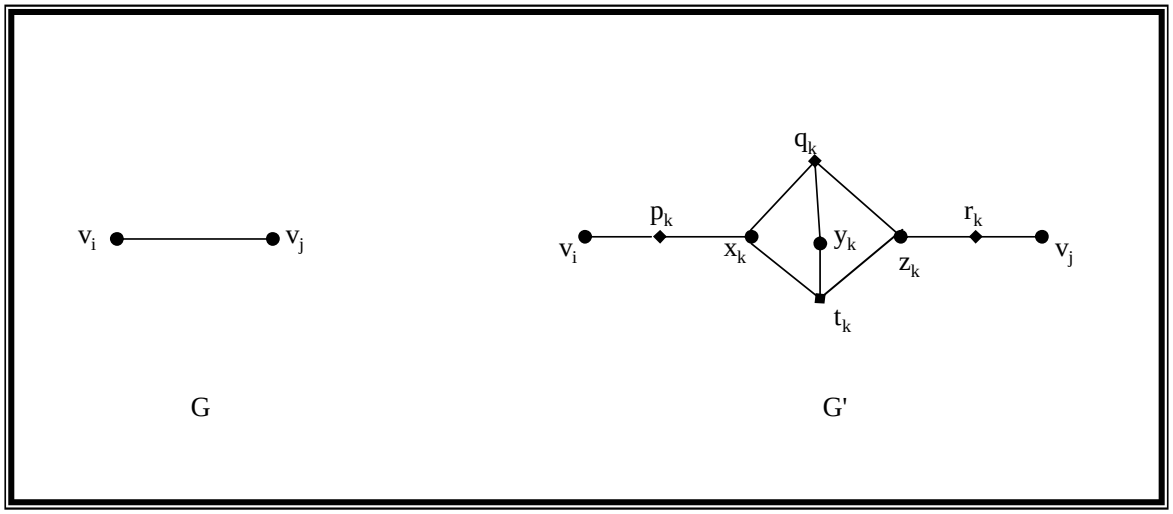

Figure 4.4: A transformation unit for biconnected planar graphs

Corollary 4.1. *The BIDS problem for biconnected planar graphs is NP-complete.*

The instance of a BIDS problem for biconnected planar graphs guarantees that every vertex in the second category is connected to at least two vertices of the first category of the bipartite graph. The biconnected property of the bipartite graph will be used later in our transformation.

In the next section, a discussion on representing a planar graph using segments is presented followed by an outline of representing bipartite planar graphs in a grid of vertical and horizontal segments.

4.4 Grid Representation of Bipartite Planar Graphs

As we have seen in the previous sections, the BIDS problem is NP-complete. In order to transform an instance of the BIDS problem into an instance of the ALP-SC-OLOR problem, the bipartite planar graph should be represented as a grid of vertical and horizontal segments in a plane.

Research has been done in transforming planar graphs into a grid representation of vertical and horizontal segments in a plane. Tamassia and Tollis [1986] and Sanders [2002] studied independently the representation of a planar graph in a grid of vertical and horizontal segments. Tamassia and Tollis [1986] called the approach visibility representation of planar graphs. Two segments are visible if they can be joined by a segment that does not intersect other segments. They studied three variations of the representation – weak visibility representation, ϵ-visibility representation and strong visibility representation. In weak visibility or w-visibility, the vertices are represented by horizontal segments and a vertical segment joins two horizontal segments when their corresponding vertices are adjacent. Here, it is possible that the segments are visible even if their corresponding vertices are not adjacent. For example, segments v_1 and v_3 in Figure 4.5(ii) are visible even though their associated vertices are not adjacent. The second representation, ϵ-visibility, is representing the vertices by horizontal interval lines in such a way that two vertices are adjacent if and only if their corresponding interval lines are visible. An interval line may contain none, one or both of its end points. ϵ-visibility differs from w-representation in that the interval lines may not be segments. The third variation, strong visibility or s-visibility, represents the vertices by horizontal segments such that two vertices are adjacent if and only if their associated segments are visible. s-visibility differs from ϵ-visibility in that the vertices are always represented by segments, and not interval lines. An example of visibility representation of a graph is given in Figure 4.5.

Castro *et al.* [2002] presented a different approach of representing any triangle-free planar graph as an intersection graph of segments in which the segments have only horizontal, vertical or oblique orientation and the segments do not cross each other. A graph is triangle-free if it does not contain a cycle of length three. In the intersection graph representation, every vertex has a corresponding segment such that two vertices are adjacent if their corresponding segments intersect. Their approach is based on Grøotzsch's Theorem which states that every planar triangle-free graph is 3-colourable [Thomassen 1994]. The vertices with the same colour are then represented by segments of the same orientation in such a way that if two vertices are adjacent, their associated segments intersect. Figure 4.6 illustrates

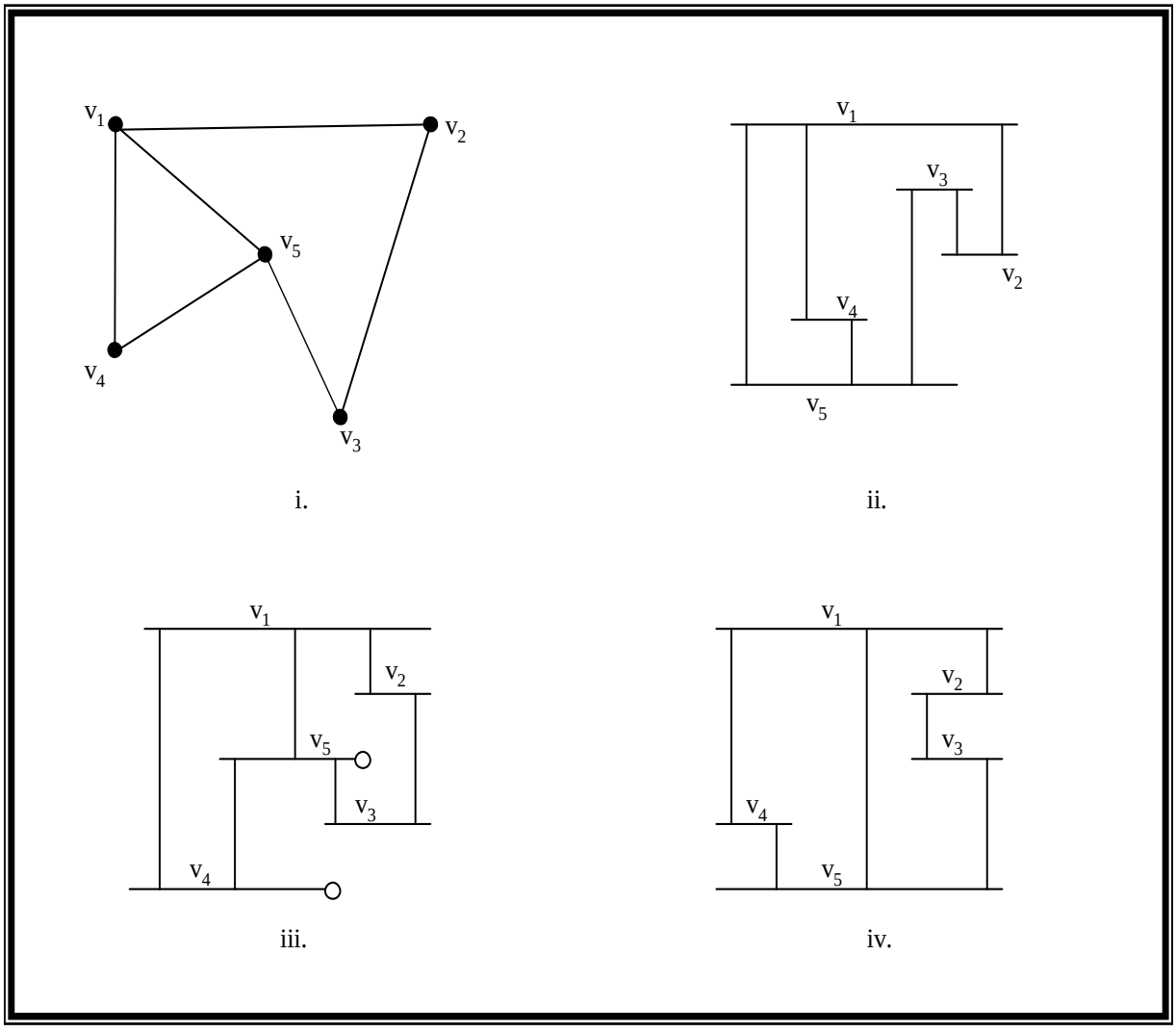

Figure 4.5: (i) A graph with five vertices, (ii) its w-visibility representation, (iii) its ϵ-visibility representation, and (iv) its s-visibility representation

an intersection graph representation of a planar graph.

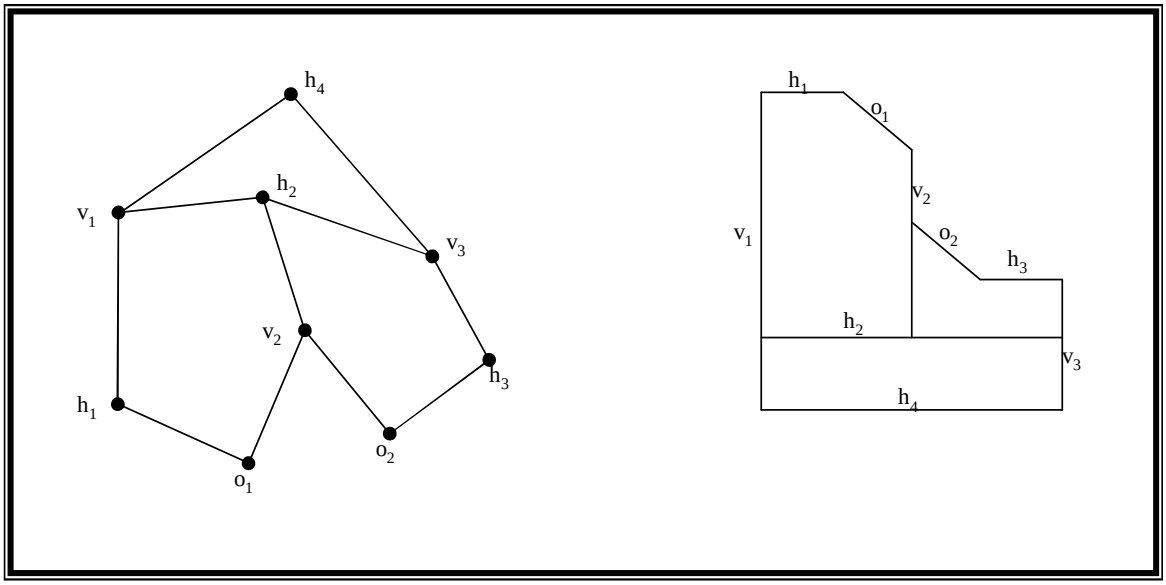

Figure 4.6: A graph and its vertical, horizontal and oblique orientation segment representation

The representation of a planar graph using a grid of segments has a special effect when the graph is bipartite. As any bipartite graph is 2-colourable, studies have been done whether it is always possible to represent the vertices of the same colour as vertical (or horizontal) segments while the vertices of the other colour as horizontal (or vertical) segments such that two segments intersect if their corresponding vertices are adjacent. de Fraysseix *et al.* [1991] and Hartman *et al.* [1991] have independently shown that any bipartite planar graph can be represented as vertical and horizontal segments in such a way that two segments have a point in common if and only if their associated vertices are adjacent. More specifically, the vertices of the first category are represented as vertical (or horizontal) segments while the vertices of the second category are represented as horizontal (or vertical) segments in which two segments intersect when their corresponding vertices are adjacent. Furthermore, de Fraysseix *et al.* [1991] provided a linear time algorithm to transform a bipartite planar graph into a grid representation of orthogonal vertical and horizontal segments in a plane. Their algorithm is based on dividing all the faces of the graph into quadrilaterals (dummy edges and vertices are introduced so that each face is bounded by a cycle, if necessary) and then joining the vertices of the same colour that form a quadrilateral face with a directed diagonal based on the orientation of the vertices in that face. Then they show that both

diagonals have bipolar orientation and hence each diagonal will have exactly one sink and one target vertex. The final phase of the algorithm is to represent the diagonals that connect all black-coloured vertices as horizontal (or vertical) segments and the diagonals that connect all the white-coloured vertices as vertical (or horizontal) segments where two segments intersect whenever their associated vertices in the planar graph are adjacent.

Czyzowicz *et al.* [1998] also developed a linear time algorithm to represent a bipartite planar graph in a ladder form of vertical and horizontal segments in which for every vertex in the bipartite graph, there is a corresponding segment (vertical or horizontal) such that two segments intersect if and only if their corresponding vertices are adjacent. The algorithm given by Czyzowicz *et al.* [1998] first divides all the faces of the bipartite planar graph into quadrilaterals. Dummy edges and vertices could be introduced during quadrilateralization. Then the vertices that make up the outer face of the graph are represented as alternating vertical and horizontal segments while intersecting whenever they are adjacent. Then the remaining vertices are represented by segments in the ladder while traversing from left to the right. When a vertex that was not represented by a segment is detected, a new segment is introduced in the ladder while extending the ladder towards the left or downward to represent the vertex, and extending the segment if its corresponding vertex is re-used. Finally, the segments introduced as a result of the dummy vertices are removed and the remaining segments are resized as a result of the dummy segments. The proof of their work is based on the mathematical induction on the number of vertices.

As any bipartite planar graph can be transformed to a biconnected bipartite planar graph by adding dummy edges, and our interest is on biconnected planar graphs, the proof of the following theorem assumes the graph is biconnected.

Theorem 4.2. *Any bipartite planar graph $G = (X, Y, E)$ can be represented by a grid of orthogonal vertical and horizontal segments $I = (H \cup V)$ such that two segments intersect if and only if their corresponding vertices are adjacent.*

Proof. The proof is by induction on the number of internal faces of G. The basis case is the graph forming a simple cycle, in which it can be represented in a grid by alternating vertical and horizontal segments in a ladder form as shown in Figure 4.7.

The induction hypothesis is a biconnected bipartite planar graph with $n - 1$ internal faces, $n \geq 2$ can be represented as a grid of vertical and horizontal segments. We want to show that a biconnected bipartite planar graph with n internal faces can be represented in a grid of segments.

Let G be a biconnected bipartite planar graph with n internal faces, $n \geq 2$. Moreover, let P be an internal face of G which shares a boundary with the external face. By deleting all the vertices and edges that lie between P and the external face only, we have a planar graph G' with $n-1$ internal faces. Clearly, G' is biconnected and bipartite as deletion of the vertices and edges does not change the bipartite or biconnected property of the graph. As G' is 2-colourable, say black and white, by the induction hypothesis, G' has a grid representation of segments in such a way that the vertices with black colour are associated with vertical (or horizontal) segments while the vertices with white colour are associated with the horizontal (or vertical) segments such that two segments intersect if their corresponding vertices are adjacent.

Now, consider the bipartite graph G. There are two cases that may arise depending on the number of edges between P and the external face in G.

i. There is only one edge, say (u_1, u_2), between P and the external face. Then each vertex of G has a corresponding segment (vertical or horizontal – depending on the colour of the vertex) in the grid. As u_1 and u_2 are of different colour, one of them has a corresponding vertical segment while the other has a corresponding horizontal segment in the grid. By connecting the segments associated with u_1 and u_2, G is represented in the grid. An example is shown in Figure 4.8.

ii. There are more than one edges, say (u_1, u_2), ..., (u_{k-1}, u_k), $k > 2$ between P and the external face. Then extend the segments associated with the vertices u_1 and u_k, and introduce a sequence of alternating vertical and horizontal segments to correspond the new vertices (the vertices that fall in P and the external face) in a ladder form as shown in Figure 4.8. Consequently, G is represented in the grid of vertical and horizontal segments.

Therefore any bipartite planar graph can be represented by a grid of vertical and horizontal segments. The generalization is due to the fact that any bipartite planar graph can be transformed to a biconnected bipartite planar graph by introducing additional edges and vertices. Sanders [2002] has presented a linear time algorithm to transform a planar graph into a biconnected planar graph. It is not hard to adopt the algorithm to bipartite graphs.

□

The following steps describe a grid representation of biconnected planar graphs based on Theorem 4.2. The approach is a variation of the work by Czyzowicz *et al.* [1998] in

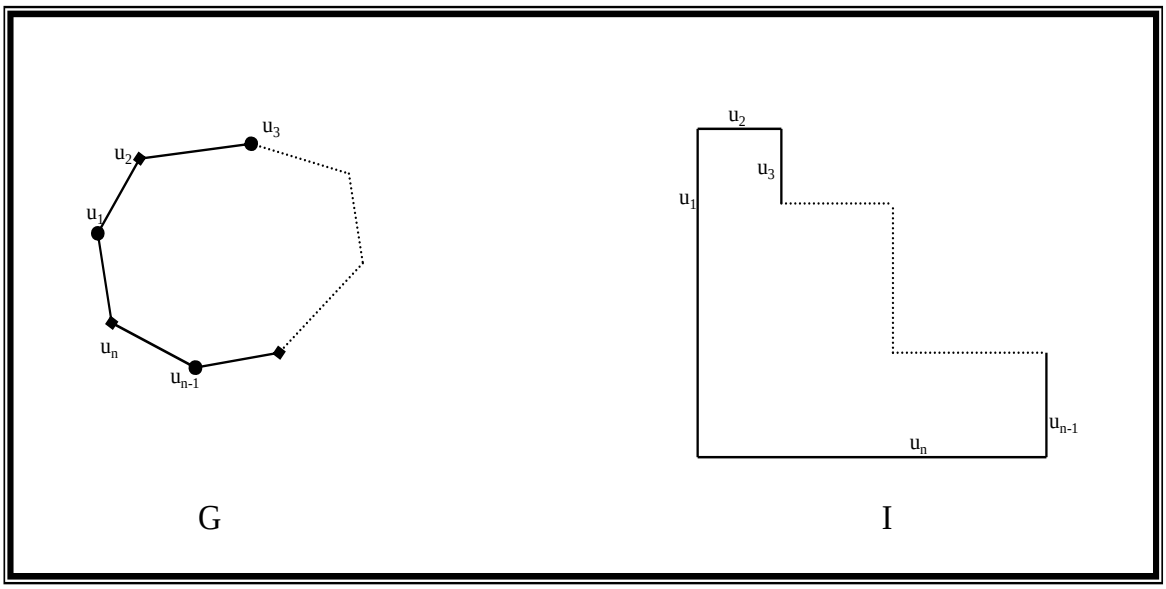

Figure 4.7: Ladder representation of a bipartite planar graph (basis case)

which they repeatedly represent the outer face of the graph in the grid and then deleting the vertices that are already represented from the bipartite graph.

- Introduce dummy edges if there are any edges which fall entirely in the external face, so that all edges lie either between the internal and external faces or between two internal faces. The dummy edges join two vertices of different colour. As the graph is bipartite it is two colourable. Introduce dummy vertices also if it is necessary (to maintain the planarity property of the graph).

- Pick the left most internal face and represent it in a ladder form.

- Consider the subsequent internal faces in breadth search first (BFS) order while updating the ladder representation, as shown in Figure 4.8. The ladder representation is updated by extending the segments associated with the vertices in the new cycle if they are already represented by segments, and including new segments for those vertices that do not have corresponding segments in the grid.

- Resize the segments which were connected as a result of the introduction of dummy edges. Similarly, delete the segments in the resulting grid that are associated with the dummy vertices.

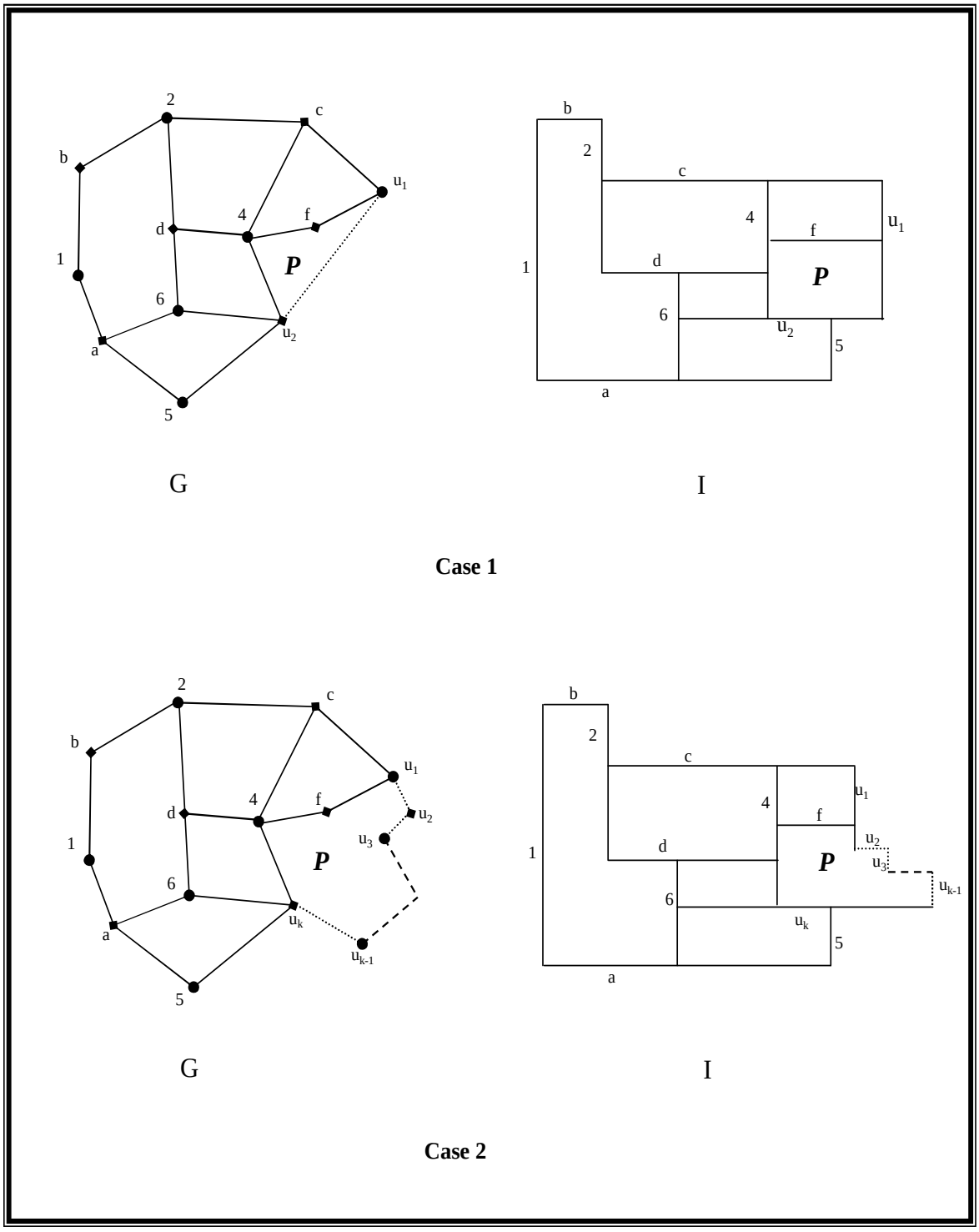

Figure 4.8: Ladder representation of a bipartite planar graph (induction step)

The above approach differs from the work by Czyzowicz *et al.* [1998] in that the above steps form the grid representation of a graph by extending and merging the smaller ladders while the latter approach partitions the ladder that corresponds to the outer face into a collection of smaller ladders.

The transformation using the above approach (and the ladder representation in general) is linear on the number of vertices since every vertex is not visited more that the number of its incident edges times. Since G is bipartite it is triangle free, i.e. G contains no cycles of length three. And according to Euler's formula for triangle free graphs the number of edges is bounded by $2n - 4$. Hence the sum of the degree of the vertices is G is bounded by $4n - 8$ and consequently the transformation is linear.

Once a bipartite planar graph is transformed into a grid of vertical and horizontal segments, we define a set similar to BIDS for planar graphs but applied in a grid of orthogonal segments.

Definition 4.3. Given a grid of orthogonal configuration $I = H \cup V$, of horizontal segments H and vertical segments V, an orthogonal bipartite independent dominating set (OBIDS) of I is a minimum set $H^{'} \subseteq H$ of horizontal segments that intersect all the vertical segments in such a way that no two segments of $H^{'}$ cross the same vertical segment.

As the orthogonal bipartite independent dominating set problem is an optimization problem, it has the following form when written as a decision problem.

Instance: Given a grid $I = H \cup V$ of horizontal and vertical segments and a positive integer $P \leq |H|$.

Question: Is there an orthogonal bipartite independent dominating set $H^{'} \subseteq H$ and $|H^{'}| \leq P$?

The orthogonal bipartite independent dominating set problem is a special case of the packing problem applied to a grid of segments. Although the optimal packing problem is NP-complete [Fowler *et al.* 1981], the geometrical configuration of the grid of segments seems to simplify the problem of orthogonal bipartite independent dominating set. The following theorem states that the OBIDS problem is NP-complete.

Theorem 4.3. *The OBIDS problem is NP-complete.*

Proof. Clearly OBIDS is in NP – given a set of horizontal segments $H^{'}$ where each horizontal segment intercepts at least two vertical segments, and a positive integer P, $|H^{'}| \leq P$, we can verify whether the segments in $H^{'}$ cross all the vertical segments such that no vertical segment is crossed by two or more horizontal segments of $H^{'}$ in polynomial time.

We show the problem is NP-hard by reducing an instance of BIDS for biconnected planar graphs. Let $G = (A, B, E)$ be a bipartite biconnected planar graph. By Theorem 4.2, G has an equivalent orthogonal grid representation $I = H \cup V$ in such a way that the vertices of A are represented by the horizontal segments H while the vertices of B are represented as vertical segments. Since each vertex in G has degree greater than one (G is biconnected), each horizontal segment intercepts at least two vertical segments. As detailed above, the transformation can be done in polynomial time since a vertex in G is not visited more than its incident edges times.

Suppose $A^{'}$ is BIDS of G of size M or less. Clearly, $A^{'} \subseteq A$, and hence has an equivalent set of horizontal segments $H^{'}$, $H^{'} \subseteq H$. By Theorem 4.2, each vertex in $A^{'}$ has a corresponding horizontal segment in $H^{'}$, which implies that if we have a BIDS of G of size M, then a set of horizontal segments of I of size M can be found that intercept each vertical segment exactly once.

Conversely, if a set of horizontal segments $H^{'}$ in I of size P that intersect each vertical line exactly once can be found, then by Theorem 4.2, there can be found a corresponding BIDS of G of size P. Therefore, OBIDS is NP-complete.

□

The following definition is entailed to maintain the connectivity property of graphs in their corresponding grid representation.

Definition 4.4. A grid representation is biconnected if it requires removal of at least two segments in order for the grid to be disconnected.

Theorem 4.2 states any bipartite planar graph has a corresponding grid representation using vertical and horizontal segments. Suppose $G(A, B, E)$ is a biconnected bipartite planar graph. Then G can be represented in a grid of vertical and horizontal segments, say I. Clearly I is a biconnected grid. From Corollary 4.1, in general the BIDS of G is NP-complete. Consequently, OBIDS of I is NP-complete as it is the corresponding grid representation of G. The following corollary states the OBIDS problem for biconnected grids is NP-complete.

Corollary 4.2. *The OBIDS problem for biconnected grids is NP-complete.*

In the next section, a transformation from an instance of OBIDS for biconnected grids to a normalized orthogonal axial lines in orthogonal rectangles, which is a special case of ALP-SC-OLOR, is presented.

4.5 Normalized Orthogonal Axial Lines in Orthogonal Rectangles

The previous section detailed the NP-completeness of the OBIDS problem. In this section, a proof of NP-completeness of ALP-SC-OLOR is given which is based on placing normalized orthogonal axial lines in orthogonal rectangles.

In ALP-SC-OLOR, a minimum number of orthogonal lines are required to cross the adjacencies between the orthogonal rectangles in such a way that the adjacencies are allowed to be crossed only once. There are many ways to cross the adjacencies between the collection of rectangles using combinations of single-crossing lines some of which are redundant. In this section, we introduce a new problem by imposing an additional restriction to ALP-SC-OLOR using the notion of maximality. A line is maximal if it crosses as many adjacencies as possible. Based on the maximality of the orthogonal lines, the adjacencies between the orthogonal rectangles fall into two categories: essential and choice adjacencies. Essential adjacencies are those adjacencies that can be crossed by only one particular maximal line, while choice adjacencies are those that can be crossed by two or more maximal lines. Figure 4.9 shows the essential and choice adjacencies.

In order to obtain a minimum number of single-crossing lines, the choice adjacencies should be crossed by a minimum number of lines since a line is always required to cross the essential adjacencies. Based on the property of the single-crossing lines and the notion of maximality, we considered a special case of the ALP-SC-OLOR problem as follows:

Definition 4.5. A normalized orthogonal line is a single-crossing line such that when the line crosses a choice adjacency then it is required to be maximal, i.e. the line should cross as many choice adjacencies as possible. The problem of normalized orthogonal lines in orthogonal rectangles (ALP-NSC-OLOR) is to place a minimum number of normalized orthogonal axial lines to cross the adjacencies between a collection of orthogonal rectangles.

An example of normalized orthogonal lines placed to cross the adjacencies between orthogonal rectangles is shown in Figure 4.9. In the first diagram, the line which crosses choice adjacencies 'a' and 'b' is not maximal, although it crosses choice adjacencies and hence it is

not normalized line. The same reasoning applies to the lines that cross choice adjacencies 'c', 'd' and 'e'. As a result, the problem is not a valid ALP-NSC-OLOR. In the second diagram The choice adjacencies 'a', 'b', 'c', 'd' and 'e' are crossed by a maximal line, and hence the problem is a valid ALP-NSC-OLOR with 4 normalized lines. The third diagram is also a valid ALP-NSC-OLOR, but the number of normalized lines is 6 which is not minimum.

The proof of NP-completeness of ALP-SC-OLOR presented in this chapter is based on the restriction method described in Section 2.6.3. First we consider the problem of placing normalized orthogonal lines to cross the adjacencies between a collection of orthogonal rectangles (ALP-NSC-OLOR) and show that it is NP-complete. Then we show that solving ALP-SC-OLOR cannot be easier than solving ALP-NSC-OLOR.

As ALP-NSC-OLOR is an optimization problem, it has the following form when written as a decision problem.

> *Instance*: Given a collection of orthogonal rectangles R_1, R_2, ..., R_n such that each rectangle is adjacent to at least one another rectangle, and a positive integer $T \leq 2n - 4$.
>
> *Question*: Is there a set Q of normalized orthogonal lines in which each vertical adjacency between the rectangles is crossed by the lines in Q exactly once, and $|Q| \leq T$?

The following theorem states that ALP-NSC-OLOR is NP-complete.

Theorem 4.4. *The problem of placing a minimum number of normalized orthogonal lines in orthogonal rectangles (ALP-NSC-OLOR) is NP-complete.*

Proof. Clearly ALP-NSC-OLOR is in NP since given a set of horizontal normalized orthogonal lines Q and a positive integer T, $|Q| \leq T$, it can be checked if every adjacency in the configuration is crossed by only one line of Q in polynomial time.

In order to show that ALP-NSC-OLOR is NP-hard, we transform the problem of OBIDS for biconnected grids to ALP-NSC-OLOR. Suppose I is a biconnected grid of vertical and horizontal segments. The basic unit of instances of OBIDS are the vertical segments of I. The local replacement substitutes for each vertical segment in I, a choice unit of two orthogonal rectangles that share an adjacency as shown in Figure 4.10 and whenever a horizontal segment intercepts two vertical segments, their associated orthogonal rectangles are joined by an orthogonal rectangle (connector) to produce I'. That is, if v_i and $v_j \in I$

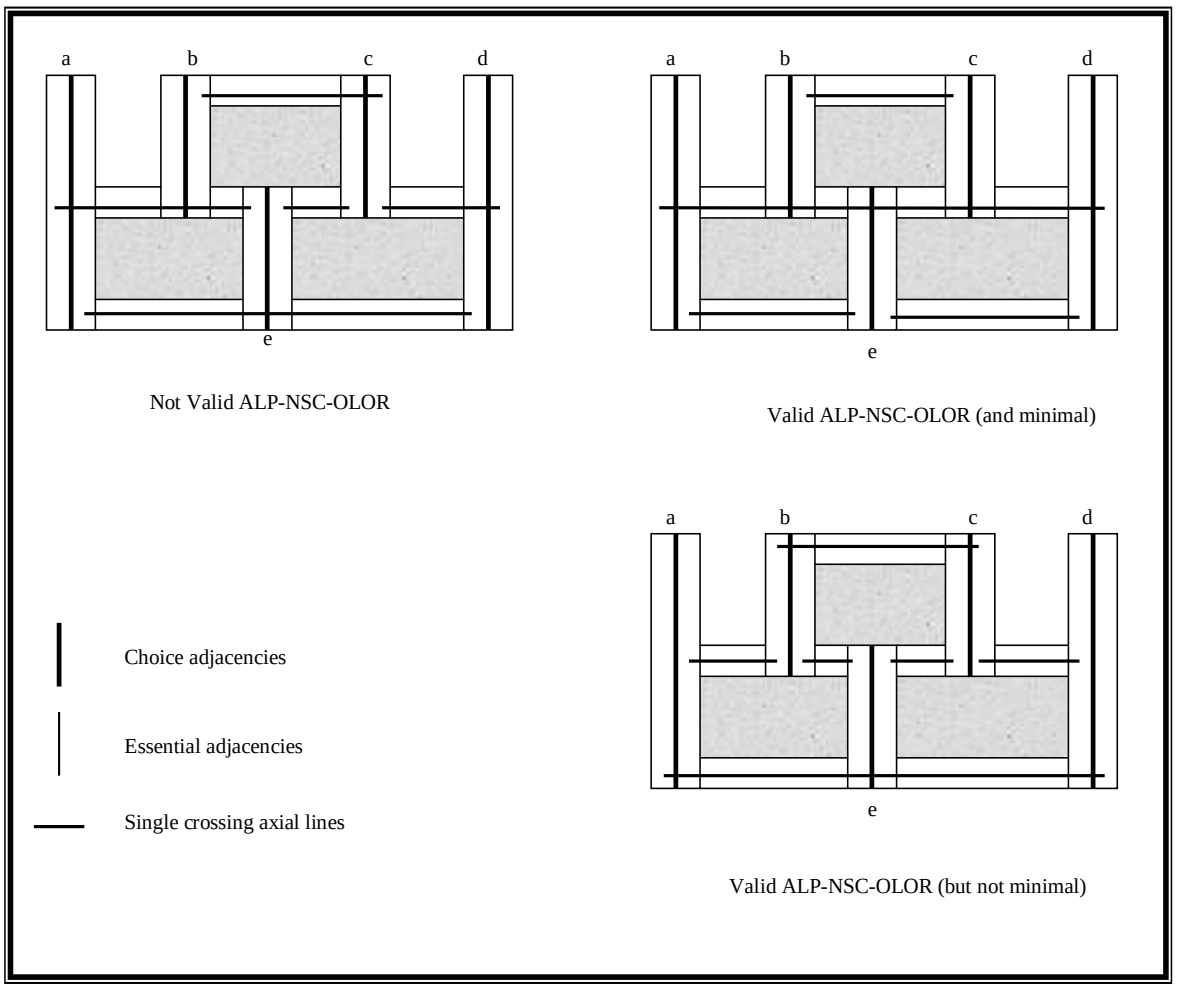

Figure 4.9: An example of ALP-NSC-OLOR

are vertical segments that are intercepted by a horizontal segment h_k, $h_k \in I$, then v_i would be replaced by two adjacent orthogonal rectangles, R_{il} and R_{ir}, while v_j would be replaced by two other adjacent orthogonal rectangles, R_{jl} and R_{jr}. Then R_{il}, R_{ir}, R_{jl}, $R_{jr} \in I^{'}$. Moreover, an orthogonal rectangle, R_{kh}, is included that connects R_{ir} and R_{jl} assuming that R_{ir} is to the left of R_{jr} since their associated vertical segments are intercepted by a horizontal segment.

The transformation can be done in polynomial time since each vertical line is visited not more than $(1+k)$ times when the segment is replaced by the choice unit, where k is the number of horizontal segments that intercept the vertical segment.

Now, suppose $H^{'}$ is an OBIDS of I. Then we have a set of horizontal lines of size $P = |H^{'}|$

in which each vertical line in I is intercepted by exactly one horizontal line of $H^{'}$. Since I is biconnected and every vertical line of I is replaced by two orthogonal rectangles that form an adjacency in $I^{'}$, essentially each vertical line in I has a corresponding choice adjacency in I'. Every connector in I' introduces two essential adjacencies that can be crossed by the same normalized line which should be included in the solution of ALP-NSC-OLOR. Moreover, some of the adjacencies formed by the connectors and the choice units could be crossed by the maximal lines. Therefore, the total number of normalized axial lines required in I' is:

$$|Q| = P + \textit{Total Number of Connectors} - \textit{Shared Connectors}$$

where shared connectors are the number of connectors in which the adjacencies they formed are crossed by the maximal lines and P is the size of OBIDS of I.

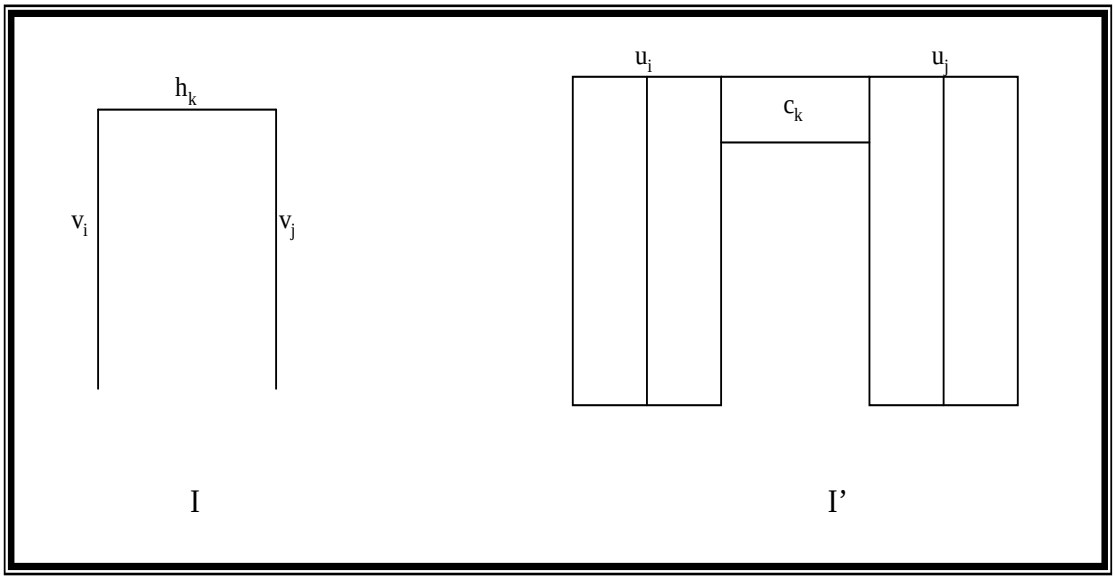

Figure 4.10: Transformation unit of an OBIDS problem to an instance of ALP-NSC-OLOR

Conversely, suppose Q, is a set of normalized axial lines that cross each adjacency between the rectangles in I', $|Q| \leq T$. Now, consider a set of lines

$$H^{'} = \{l | l \in Q \text{ and } l \text{ crosses choice adjacencies.}\}$$

Since the lines in Q are normalized, $H^{'}$ contains the maximal lines that crosses the choice adjacencies. By construction, each vertical line of I has a corresponding choice adjacency in $I^{'}$, and if two vertical lines are crossed by the same horizontal line in I, the rectangles that form their corresponding adjacencies are connected by another rectangle. Therefore, we can get a set of horizontal lines of size $|H^{'}|$ which is OBIDS of I. Figure 4.11 shows a

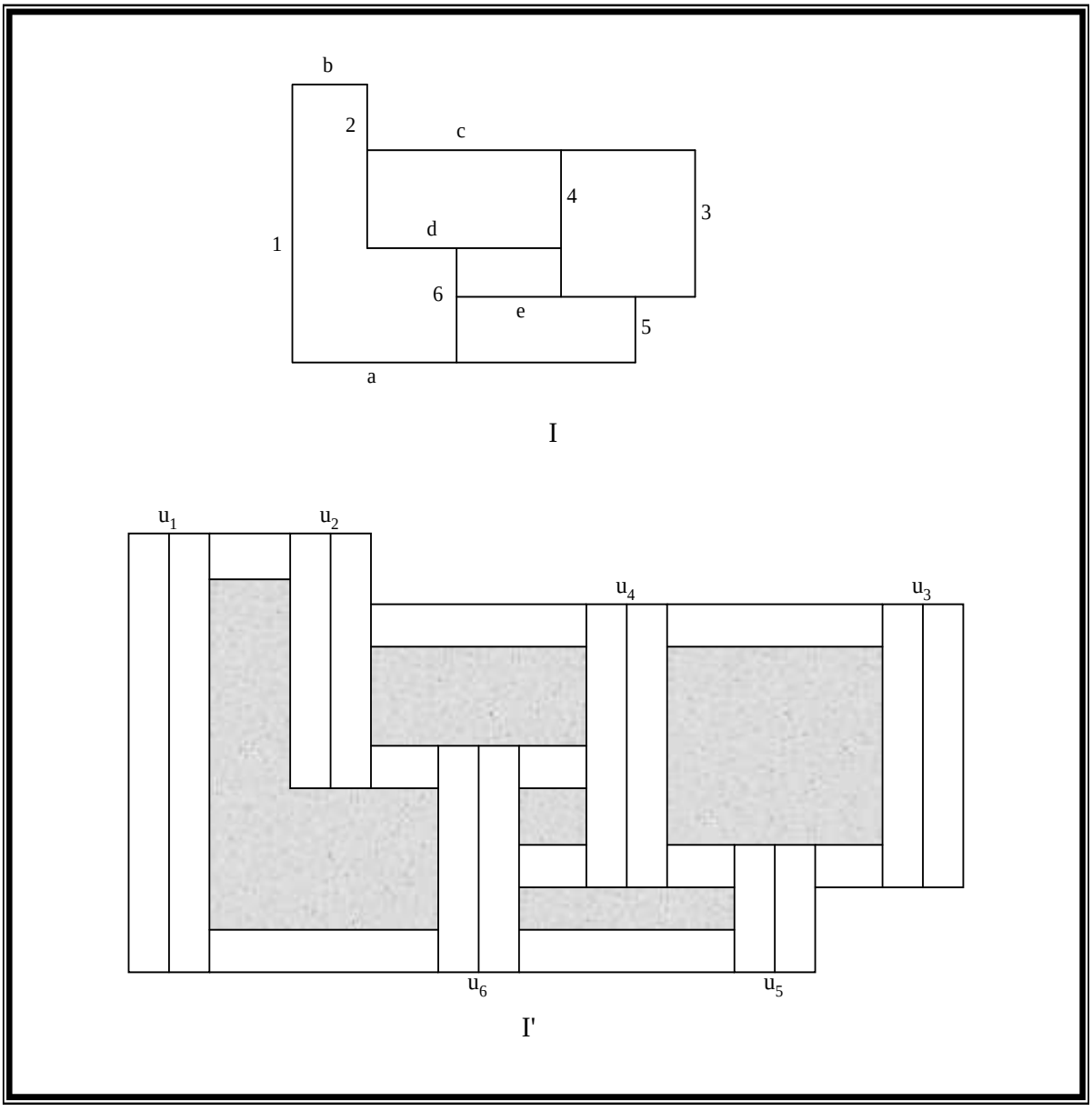

Figure 4.11: An illustration on how the transformation proceeds from an instance of OBIDS to ALP-NSC-OLOR

complete transformation of an instance of OBIDS for biconnected grids to an instance of ALP-NSC-OLOR.

□

Before proceeding to the proof of NP-completeness of ALP-SC-OLOR, we demonstrate how a solution to the OBIDS problem gives a solution to ALP-NSC-OLOR. Let us consider the diagrams in Figure 4.12. For the sake of clarity, only the horizontal lines are labeled in the first diagram, while the maximal lines and the choice adjacencies are labeled in the second diagram. Our interest is to find a solution of the OBIDS problem in the first case, and a solution to the ALP-NSC-OLOR problem in the second one.

In the first diagram, there are three possible combination of horizontal lines to intercept the vertical lines exactly once: lines 1, 3, 5; lines 2, 4, 6; and lines 6, 7. As the problem of OBIDS is an optimization problem, we are interested in the minimum solution, and hence the solution is combination of lines 6 and 7. In the case of ALP-NSC-OLOR, there are also three possible solutions – the maximal lines 1, 3, 5 along six other single crossing lines which cross only essential lines (9 normalized single crossing lines in total); the maximal lines 2, 4, 6 along six other single crossing lines that only cross the essential adjacencies (9 normalized single crossing lines in total); and the maximal lines 3 and 7 along with five other single crossing lines which cross only essential adjacencies (7 normalized single crossing lines in total). And hence the solution to the OBIDS problem corresponds to the solution to the ALP-NSC-OLOR problem.

Next, we show that ALP-SC-OLOR is NP-complete using the restriction technique. In the restriction method, the focus is on the target problem whereby we add restrictions while retaining the essential properties of the problem so that the resulting restricted problem is identical to a known NP-complete problem [Garey and Johnson 1979]. In our case, the essential aspects of ALP-SC-OLOR is that all adjacencies between the collection of rectangles should be crossed by exactly one line in the solution set. And this property is maintained in ALP-NSC-OLOR, i.e crossing the adjacencies in the collection of rectangles by normalized lines.

The restriction method is based on the concept that an algorithm that solves a given problem in polynomial time should also solve a special case of the problem in polynomial time. ALP-NSC-OLOR is a special case of ALP-SC-OLOR whereby the single-crossing lines are restricted to be normalized orthogonal lines. The following theorem is based on Theorem 4.4 and the above discussion.

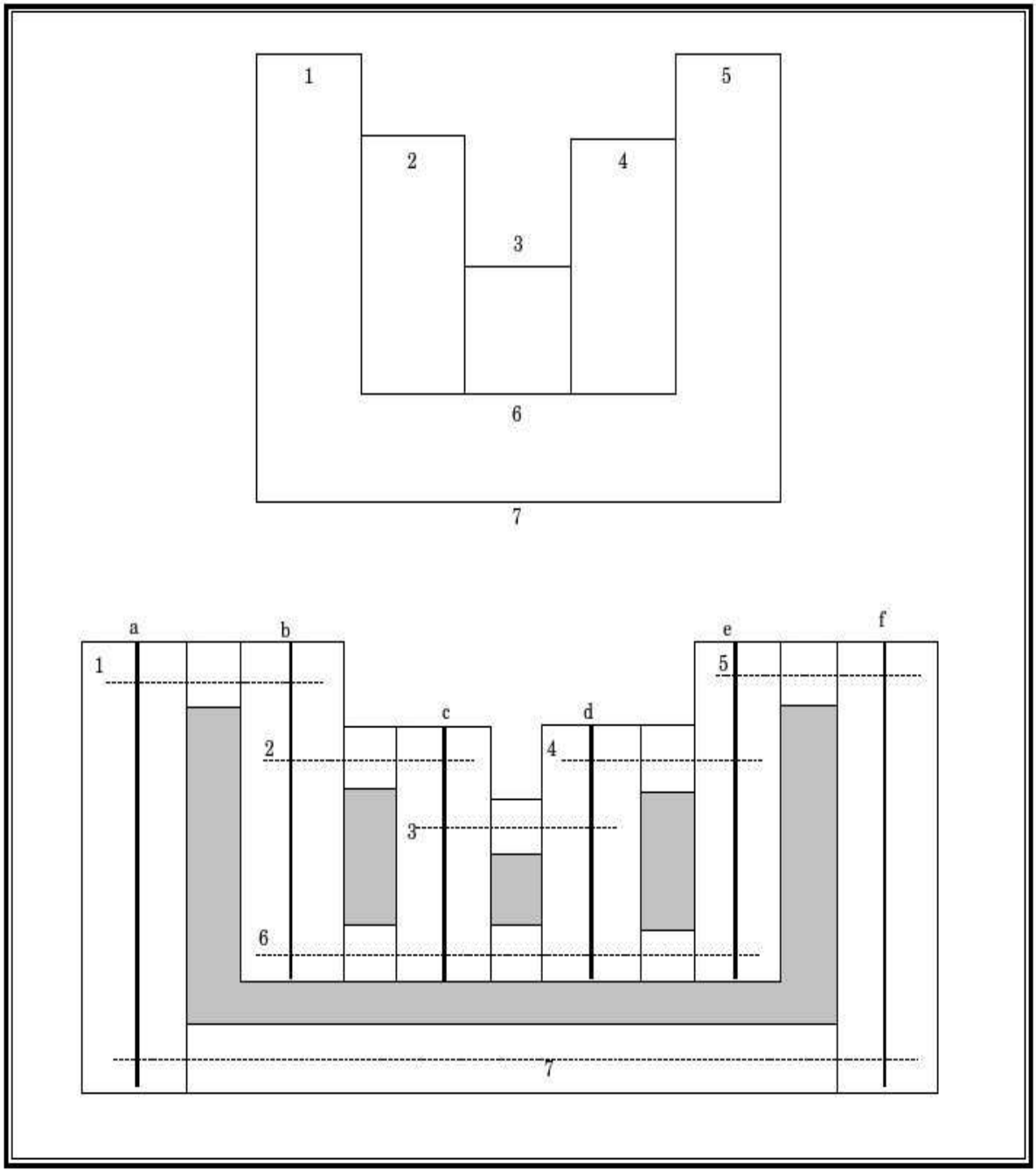

Figure 4.12: How the solution of OBIDS gives a solution to ALP-NSC-OLOR

Theorem 4.5. *ALP-SC-OLOR is NP-complete.*

Proof. Clearly, ALP-SC-OLOR is in NP since given a set of horizontal lines L and a positive integer K, it is possible to check if every adjacency between the orthogonal rectangles is crossed only once in polynomial time.

Theorem 4.4 stated that placing normalized axial lines in orthogonal rectangles is NP-complete. The ALP-NSC-OLOR problem is a special case of ALP-SC-OLOR in which the orthogonal axial lines are restricted to be normalized orthogonal lines. Therefore, ALP-SC-OLOR is NP-complete.

□

The NP-completeness of ALP-NSC-OLOR arises due to the fact that selecting the minimum combination of normalized orthogonal axial lines to cross all the adjacencies between the collection of rectangles exactly once takes exponential time. ALP-NSC-OLOR is a simplification of ALP-SC-OLOR in that it greatly reduces the number of combinations of single-crossing lines to cross the adjacencies in the configuration. That is, in ALP-NSC-OLOR the choice adjacencies are required to be crossed by maximal lines whereas in ALP-SC-OLOR the lines that cross the choice adjacencies may not be maximal which could form a number of combinations of single-crossing lines to cover the adjacencies in the configuration.

Now, consider the configuration of rectangles in Figure 4.13. Adjacencies 3 and 4 are choice adjacencies while adjacencies 1, 2 and 5 are essential. Adjacency 3 can be crossed by lines a or b. As adjacencies are restricted to be crossed only once, which line crosses adjacency 3 affects which line (or lines) is going to cross its neighbouring adjacencies 4 and 5. That is, the choice (decision) we make at a choice adjacency clearly determines which lines will cross the neighbouring adjacencies. Figure 4.13 is a simple configuration of rectangles with two choice adjacencies. With complex configuration of rectangles and many choice adjacencies, computing for the exact solution requires exhaustive search.

4.6 Single-crossing Versions of ALP-ALOR and ALP-ALCP

The previous section stated that ALP-SC-OLOR is NP-complete. The single-crossing version of ALP-ALOR, which will be referred to as ALP-SC-ALOR, is a more generalized problem of ALP-SC-OLOR whereby the single-crossing axial lines have freedom to have arbitrary orientation. This freedom of orientation produces more choice adjacencies which makes it even harder to make decisions locally at every choice adjacency.

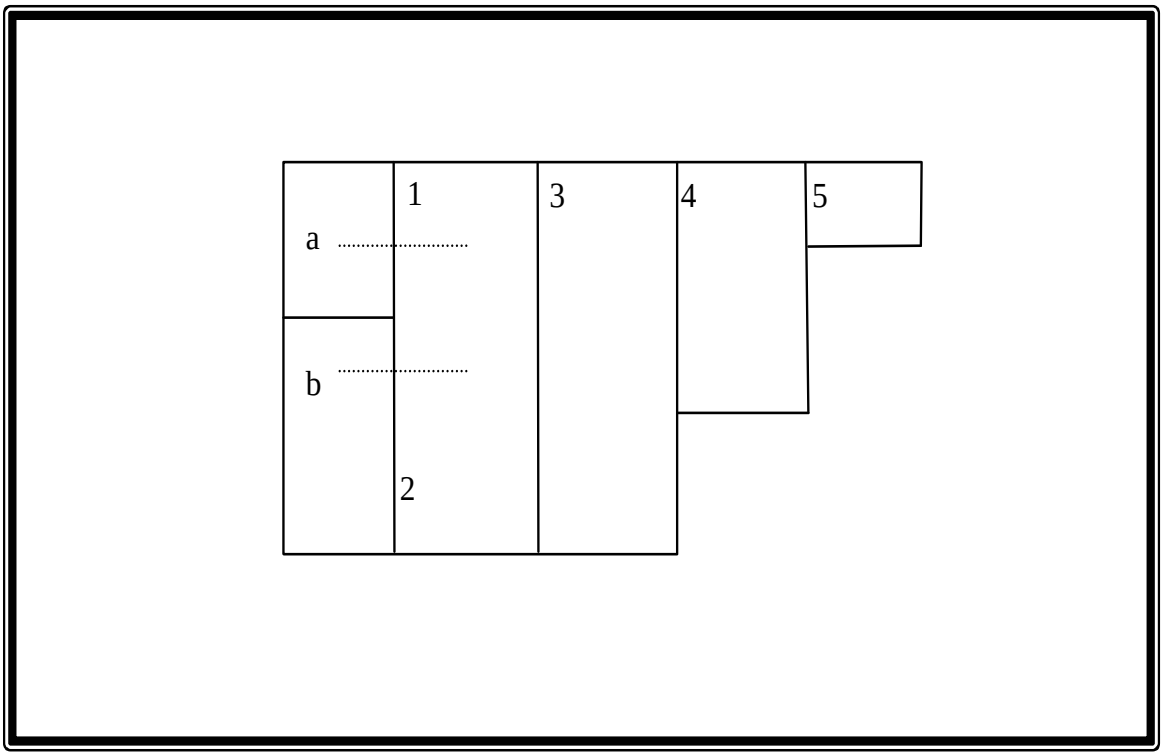

Figure 4.13: Placing single-crossing lines to cross choice adjacencies

In the case of the single-crossing of ALP-ALCP, which will be referred to as ALP-SC-ALCP, the freedom is not only on the orientation of the single-crossing axial lines but also on the shape of the collection of polygons. If ALP-SC-OLOR, which is the more restricted problem of ALP-SC-ALOR and ALP-SC-ALCP, cannot be solved in polynomial time then neither of the generalized problems can be solved in polynomial time. As a result, the NP-completeness of ALP-SC-OLOR can be extended to ALP-SC-ALOR and ALP-SC-ALCP.

4.7 Conclusion

The research question the dissertation aimed to answer was to determine whether ALP-SC-OLOR could be solved in polynomial time or it is NP-complete. As discussed in the previous sections, ALP-SC-OLOR has been found to be NP-complete. The proof of NP-completeness presented in the chapter was based on the reduction from vertex cover to ALP-NSC-OLOR which is a special case of ALP-SC-OLOR where the axial lines are required to be maximal

when they cross choice adjacencies.

As detailed above, the proof of NP-completeness of ALP-SC-OLOR is given by restricting the problem into a special case of the general problem where the single-crossing orthogonal lines in ALP-SC-OLOR are restricted to be maximal when they cross choice adjacencies. Garey and Johnson [1979] described that an algorithm that solves a given problem in a general case should also solve a special case of the general problem. As a result, if a special case of a given problem is found to be NP-complete, the general problem should also be NP-complete. And hence ALP-SC-OLOR is NP-complete.

In general, if a problem is found to be NP-complete, computing the exact solution takes exponential time in the worst case, which implies the problem can only be solved efficiently for data of small size input. As a result, if a problem is found to be NP-complete, it is worthwhile to focus on heuristic algorithms that give reasonably approximated solutions and special configurations that are solvable in polynomial time than attempting to develop an algorithm that gives an optimal (exact) solution. In the next chapter, a heuristic algorithm for ALP-SC-OLOR based on a greedy method is given.

Chapter 5

Heuristic for ALP-SC-OLOR

5.1 Introduction

Chapter 4 presented a proof of NP-completeness of placing single-crossing orthogonal axial lines in orthogonal rectangles. The proof was a three way reduction: from vertex cover for biconnected planar graphs to a bipartite independent dominating set (BIDS) for biconnected planar graphs, and then to orthogonal biconnected bipartite independent dominating set (OBIDS) and finally, to a special case of ALP-SC-OLOR where the adjacencies between the orthogonal rectangles are crossed by normalized orthogonal axial lines. As it is discussed in more detail in Garey and Johnson [1979], if a special case of a given problem is NP-complete, then the general problem is also NP-complete since any algorithm that could solve the general problem in polynomial time should also solve the special case in polynomial time. And hence, we concluded that ALP-SC-OLOR is NP-complete.

In general, if a problem is found to be NP-complete, it cannot be solved in polynomial time unless $P = NP$. As this is still an open question, if a problem is found to be NP-complete it is worthwhile to concentrate on heuristic algorithms that give reasonable approximate solutions and special cases of the problem that could be solved in polynomial time rather than attempting to solve for the exact solution. This is due to the fact that exhaustive search for the exact solution grows exponentially with the size of the input of the problem. As a result a heuristic algorithm for ALP-SC-OLOR based on a greedy method is presented in this chapter.

In the next section, the general description of the heuristic algorithm is discussed followed by a high level outline of the heuristic. The theoretical and empirical complexity analysis of the heuristic is also included in this chapter. The final section presents a discussion on the performance of the heuristic.

5.2 Heuristic Algorithm

As detailed in the previous chapters, the ALP-SC-OLOR problem aims at crossing the adjacencies between a collection of orthogonal rectangles by a minimum number of orthogonal axial lines in such a way that each adjacency is crossed by only one line. In this dissertation, we considered a greedy method with local improvement for developing a heuristic algorithm for ALP-SC-OLOR that gives a reasonable approximate solution.

Before presenting the heuristic, let us define the notion of axial lines. By definition, axial lines are lines that may cross an adjacency or adjacencies between a collection of convex polygons – in our case orthogonal rectangles. There could be a number of axial lines that may cross an adjacency. If two or more axial lines cross the same adjacencies, they are considered to be equivalent. In other words, if we consider the axial lines L_1 and L_2 as sets, and the elements each set being the adjacencies the lines crosses, then we can say that the axial line L_1 is a subset of L_2 if all the adjacencies crossed by L_1 are also crossed by L_2. And consequently, two axial lines are considered to be equivalent if both lines are subsets of one another.

The approach of the heuristic is first to generate all possible single-crossing lines and then repeatedly add the lines that cross the highest number of adjacencies not crossed by the lines in the solution set to the solution set provided that no two lines in the solution set cross the same adjacency. As adjacencies are restricted to be crossed only once, the order of placing the lines in the solution set may affect the final number of lines produced to cover the adjacencies. As a result, a local improvement has been considered in Section 5.2.4.

Once all single-crossing lines are generated, the ALP-SC-OLOR problem could be considered as an instance of a set packing exact-cover problem – given a set and subsets to the set, selecting a minimum number of subsets that cover all the elements and none of the elements is in more than one subset of the solution set. And a greedy method is well known to give a reasonably approximated solution to the set packing problem [Skiena 1998].

The heuristic has three phases – determining the adjacencies between the orthogonal rectangles, placing all possible single-crossing orthogonal lines to cross each adjacency and the final phase is to select the fewest number of single-crossing orthogonal lines to cross all the adjacencies between the rectangles. Algorithm 1 presents a high-level outline of the heuristic.

Before discussing the three phases of the heuristic algorithm, the data structures used in the algorithm are described below.

Algorithm 1 A high-level algorithm to compute an approximate minimum number of single-crossing lines to cross the adjacencies between a collection of rectangles

INPUT: An array of rectangle record.
OUTPUT: An approximate minimum number of single-crossing lines

{*NextApproximateSolution is the set of minimum single-crossing lines produced during one iteration of the outer WHILE loop. It is initialized with empty list.*}

{*BestApproximateSolution is the set of minimum single-crossing lines so far.*}

{*an untraversed adjacency is an adjacency where no line has been considered which originates from the adjacency and is being extended to the right*}

{*an uncrossed adjacency is an adjacency where it is not crossed by any of the lines in the set NextApproximateSolution*}

Determine the adjacencies between the collection of rectangles
for each adjacency **do**
Form a list of the lines that originate from the adjacency and are extended to the right
Sort the list of lines in descending order of the number of adjacencies the lines cross
end for

while there is an untraversed adjacency **do**
Add the first line from the list of lines originating from the current adjacency to the set of *NextApproximateSolution*

while there is an uncrossed adjacency **do**
Goto the next uncrossed adjacency OR to the first adjacency if we reach the last adjacency
Add the line that crosses the only adjacencies which are not crossed by the lines in the set of *NextApproximateSolution* to that set (*NextApproximateSolution*)
end while

if the number of lines in the local solution set is less than *BestApproximateSolution* **then**
Update the set of *BestApproximateSolution* to be *NextApproximateSolution*
end if

Set the *NextApproximateSolution* to be empty
Goto the next adjacency
end while
Output *BestApproximateSolution*

5.2.1 Data Structures

There are three records defined in the heuristic: records of rectangles, adjacencies and lines. The rectangles are defined by their left-bottom coordinates and right-top coordinates. Associated with the rectangle record are *adjListLeft* and *adjListRight* that keep track of the left and right adjacencies of a rectangle, respectively. The record also has a field *name* that uniquely identifies a particular rectangle. The rectangles are considered to be non-overlapping and orthogonal.

The adjacencies are defined by the left (*leftRectangle*) and right (*rightRectangle*) rectangles that form the adjacency, and their bottom and top (*bottomMargin* and *topMargin* respectively) $y-$value range of their common edge. A field, *name*, is also included in the adjacency record in order to identify an adjacency uniquely. The field is computed based on Algorithm 2 starting from 1 and incrementing when a new adjacency is detected. Although an adjacency can uniquely be identified by the two rectangles it is formed by, the *name* field is used for checking whether the adjacency is crossed or not by a line in the solution set in constant time. The values of all the fields associated with the adjacency record are computed from the collection of orthogonal rectangles.

The line record is defined by the list of adjacencies the line crosses (*adjList*), the minimum $y-$value of the upper margin of the adjacencies (*upperMargin*) and the maximum $y-$value of the lower margin of the adjacencies (*lowerMargin*). The line record also contains the field *degree* to keep track of the number of adjacencies the line crosses. Likewise, all the values of the fields associated with the line record are computed from the configuration of the orthogonal rectangles and the adjacencies formed by the rectangles.

Initially, the rectangles are stored in an array. An array of adjacency record is defined to store the adjacencies between the rectangles. By storing the adjacency record in an array, it can be accessed using indices in constant time. Moreover, it is possible to check whether an adjacency is already crossed or not by a line in the solution set in constant time as the problem restricts adjacencies to be crossed only once.

5.2.2 Determining Adjacencies

The total number of vertical and horizontal adjacencies formed by a collection of non-overlapping orthogonal rectangles is linear with respect to the number of rectangles. This can be verified using Euler's formula where given a planar graph $G = (V, E)$, the number of non-crossing edges $|E|$ that joins the vertices is bounded by $3n - 6$, where n is the number of vertices of the graph. As described in Sanders [2002] in more detail, by representing the

rectangles as a vertices of a graph and placing edges between the vertices if their corresponding rectangles are adjacent, the number of vertical and horizontal adjacencies formed by the rectangles is bounded by $3n - 6$. In ALP-SC-OLOR, we are interested in vertical adjacencies and horizontal lines, and hence we give a tighter bound on the number of vertical adjacencies.

The following lemma serves as the basis for determining the upper bound on the total number of vertical adjacencies formed by a collection of non-overlapping orthogonal rectangles. The proof is omitted as it is trivial.

Lemma 5.1. *Two rectangles that are vertically adjacent to the same rectangle cannot be vertically adjacent to one another.*

Suppose R_1, R_2 and R_3 be three adjacent rectangles. Moreover, suppose rectangles R_1 and R_2 are adjacent to rectangle R_3. Then there are four possible configurations of the rectangles: R_1 and R_2 are to the left of R_3, R_1 and R_2 are to the right of R_3, R_3 is to the right of R_1 and to the left of R_2, and R_3 is to the left of R_1 and to the right of R_2.

Figure 5.1 shows the possible configurations of three non-overlapping adjacent orthogonal rectangles. The fourth configuration is omitted as it is symmetric to Figure 5.1(iii). The following theorem states the upper bound on the total number of vertical adjacencies in a collection of non-overlapping orthogonal rectangles.

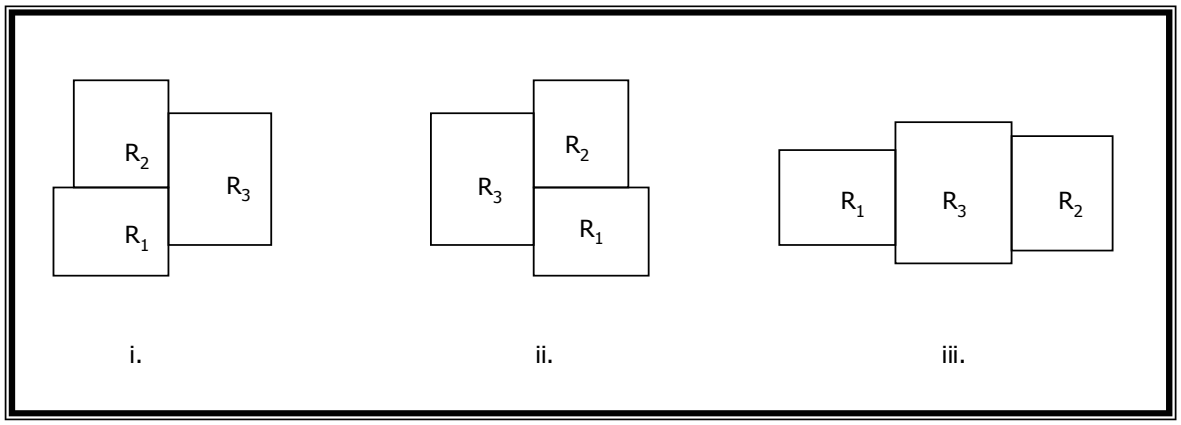

Figure 5.1: Possible configurations of three non-overlapping adjacent orthogonal rectangles

Theorem 5.1. *The number of vertical adjacencies in a collection of non-overlapping orthogonal rectangles is at most $2n - 4$, $n \geq 3$, where n is the number of rectangles.*

Proof. From Euler's formula, the total number of edges in a simple planar graph is at most $3n - 6$, where n is the number of vertices. If the graph does not contain a cycle of three (it is triangle-free) then the total number of edges is at most $2n - 4$ [Wilson and Watkins 1990].

Let R be a collection of n non-overlapping rectangles. Now, construct a graph G from the collection of rectangles in R in such a way that each rectangle in R has a corresponding vertex in G and, if two rectangles in R are vertically adjacent then connect their associated vertices in G by an edge. Clearly the resulting graph is planar with n vertices.

To show that the total number of vertical adjacencies in R is at most $2n - 4$, it suffices to show that G does not contain cycles of three, i.e. G is triangle-free.

Suppose G contains a cycle of three, and let V_i, V_j and V_k be the vertices that form a cycle of three. Then by construction of G, there are three rectangles in R associated with V_i, V_j and V_k. Let R_i, R_j and R_k be the three rectangles associated with V_i, V_j and V_k respectively. As the vertices are connected by an edge in G, R_i, R_j and R_k are vertically adjacent to each other. But by Lemma 5.1, the three rectangles cannot be vertically adjacent to each other. Therefore, G must be a triangle-free graph and from Euler's formula for triangle-free graphs, the total number of edges in G is at most $2n - 4$.

As each edge in G corresponds to a vertical adjacency in R, the total number of vertical adjacencies in a collection of non-overlapping orthogonal rectangles is at most $2n - 4$.

□

Algorithm 2 presents a method of determining adjacencies between a collection of rectangles. As the aim of the problem was to determine the minimum number of lines to cross the adjacencies, it was assumed that initially the rectangles are in an array. The algorithm first defines an array of adjacency record (***line 2***) which stores the adjacencies between the rectangles. The algorithm proceeds by picking one rectangle and then checks repeatedly if it forms an adjacency with any of the rectangles, while updating the array of adjacencies when an adjacency is detected. As the checking of rectangles is from left to right, the array of rectangle record is copied to *LSorted*, and sorted based on their left edge (*LSorted*$[i]$.*left*), while breaking ties in minimum *LSorted*$[i]$.*bottom* first (***lines 3-5***). There are three indices defined in the algorithm: i is the index of the current rectangle in *LSorted* which is under consideration, j is the index of the adjacency record, and k is the index of the subsequent rectangles in *LSorted* to be checked for potential adjacency with *LSorted*$[i]$. Initially the

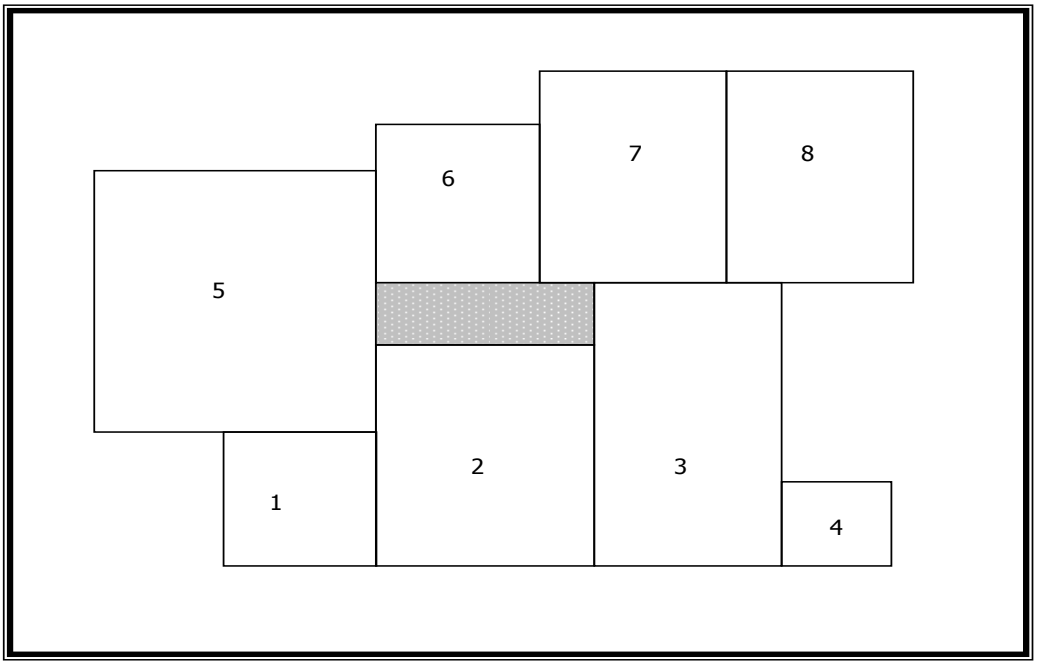

Figure 5.2: An example of a collection of non-overlapping orthogonal rectangles

index of the current rectangle (i) and the index of the adjacency record (j) have a value of one (***line 6***).

When we compare two rectangles for adjacency, one of the following three conditions may occur:

- The $x-$value of the right edge of the current rectangle ($LSorted[i]$) is less than the $x-$value of the left edge of the rectangle to be checked for potential adjacency ($LSorted[k]$). As there is a horizontal gap between the rectangles, they cannot be adjacent. And hence none of the rectangles in *LSorted* can be adjacent with the current rectangle under consideration (***line 10***). In that case increment i, that is, advance to the next rectangle for consideration (***line 35***).

- The $x-$value of the right edge of the current rectangle ($LSorted[i]$) is greater than the $x-$value of the left edge of the rectangle to be checked for potential adjacency ($LSorted[k]$) – (***line 10***). The two rectangles cannot be adjacent, but the subsequent rectangles in *LSorted* can be adjacent with the current rectangle which is $LSorted[i]$. In that case increment k, that is, consider the next rectangle after $LSorted[k]$ to check for adjacency with $LSorted[i]$ (***line 32***).

Algorithm 2 An algorithm that determines the adjacencies between a collection of non-overlapping orthogonal rectangles

INPUT: An array of rectangle record *Rect*.
OUTPUT: An array of adjacency record *Adj*.

```
{Let the array of rectangles be Rect with size n}
Create an array of adjacency record Adj (with size = 2n − 4)
Create an array of rectangle record LSorted
Copy Rect to LSorted
Sort LSorted in ascending order of LSorted.left while breaking ties in LSorted.bottom
i ← j ← 1
{i is the index of the current rectangle under consideration in LSorted, j is the index of
Adj and k is the index of the rectangle from LSorted to be checked for adjacency with
LSorted[i]}
while (i ≤ n − 1) do
  k ← i + 1
  while (k ≤ n and LSorted[i].right ≥ LSorted[k].left) do
    {there is no horizontal gap between the two rectangles}
    if (LSorted[i].right = LSorted[k].left) then
      {the rectangles have the same x−value. Therefore, there is a possibility to have
      a common adjacency}
      if (LSorted[i].bottom < LSorted[k].top AND
          LSorted[i].top > LSorted[k].bottom) then
        {the rectangles are adjacent}
        Create adjacency j and store it in Adj[j]
        Adj[j].leftRectangle ← LSorted[i]
        Adj[j].rightRectangle ← LSorted[k]
        Adj[j].upperMargin ← Minimum(LSorted[i].top, LSorted[k].top)
        {the Minimum function returns the argument with the smallest value}
        Adj[j].lowerMargin ← Maximum(LSorted[i].bottom, LSorted[k].bottom)
        {the Maximum function returns the argument with the largest value}
        {append the adjacency into the list of adjacencies of its neighbourhood rectan-
        gles that form the adjacency}
        Append Adj[j] into the list of adjacencies of Rect[LSorted[i].name].adjListRight
        Append Adj[j] into the list of adjacencies of Rect[LSorted[k].name].adjListLeft
        j ← j + 1
      else if (LSorted[i].top ≤ LSorted[k].bottom) then
        {LSorted[i] is not adjacent to LSorted[k]. It cannot be adjacent to the rest of
        rectangles in LSorted either}
        Exit inner while loop
      end if
    end if
    k ← k + 1
    {go to the next rectangle for potential adjacency checkup}
  end while
  i ← i + 1
  {consider next rectangle to check for adjacency}
end while
```

- The $x-$value of the right edge of the current rectangle ($LSorted[i]$) is equal to the $x-$value of the left edge of the rectangle to be checked for potential adjacency ($LSorted[k]$) – (***line 12***). Here, depending on the value of the top and bottom edges of the rectangles three cases may arise:
  - The $y-$value range of the current rectangle ($LSorted[i]$) overlaps with the $y-$value range of the rectangle to be checked for adjacency ($LSorted[k]$) – ***line 14***. Then the two rectangles are adjacent. The information concerning the adjacency (the left and right rectangles, and the lower and upper margins of the adjacency) are stored in the Adj array. Furthermore, the adjacency is appended to the left and right list of adjacencies of the rectangles respectively (***lines 16-25***). And finally, k is incremented (***line 32***).
  - The $y-$value of the top edge of the current rectangle ($LSorted[i]$) is less than the $y-$value of the bottom edge of the rectangle being checked for adjacency ($LSorted[k]$) – ***line 27***. The two rectangles cannot be adjacent and the current rectangle cannot be adjacent to any of the rest of $LSorted$. In that case i is incremented – advance to the next rectangle and start checking for potential adjacency (***line 35***).
  - The $y-$value of the bottom edge of the current rectangle ($LSorted[i]$) is greater than the $y-$value of the top edge of the rectangle being considered for adjacency ($LSorted[k]$). The two rectangles cannot be adjacent but the rest of the rectangles in $LSorted$ could be adjacent to the current rectangle under consideration ($LSorted[i]$). Hence k is incremented (***line 32***).

In order to demonstrate how the algorithm determines the adjacencies between a collection of non-overlapping orthogonal rectangles, consider the configuration of rectangles in Figure 5.2. In the algorithm, i is the index of the current rectangle under consideration, k is the index of the rectangle to be checked for potential adjacency against the current rectangle ($LSorted[i]$) and j is the index of the adjacency record. ***Lines 2 and 3*** of the algorithm copies the array $Rect$ to $LSorted$. Then $LSorted$ is sorted by $x-$value of the left edge of the rectangles in ascending order while breaking ties in increasing $y-$value of the bottom edge of the rectangles (***line 5***). So, once the rectangles are sorted, they will be in the order of 5, 1, 2, 6, 7, 3, 8, and 4. i is initialized to 1 while k is initialized to 2 (***line 9***). As $LSorted[1]$ is rectangle 5 and $LSorted[2]$ is rectangle 1, the algorithm first checks if rectangle 5 shares a common edge (adjacency) with rectangle 1. Since $LSorted[1].right$ is greater than $LSorted[2].left$, k is incremented (***line 32***) which will be 3. The algorithm

A List of Functions:

- **head(ALine):** returns the first line from the list of lines *ALine*.
- **head(Adj):** returns the first adjacency from the list of adjacencies *Adj*.
- **tail(ALine):** deletes the first line from the list of lines *ALine*, and returns the rest of the list.
- **tail(Adj):** deletes the first adjacency from the list of adjacencies *Adj*, and returns the rest of the list.
- **computeDegree(ALine):** returns the number of adjacencies crossed by line *ALine*.
- **partiallyVisible(ALine, Adj):** returns true if there is a horizontal partial visibility between the adjacencies crossed by line *ALine* and adjacency *Adj*.
- **isMarked(ALine, Crossed):** returns ***true*** if any of the adjacencies crossed by line *ALine* is already crossed, and ***false*** otherwise. If an adjacency k is already crossed, then $Crossed[k]$ is ***true***, and ***false*** if it is not yet crossed.
- **markAdj(ALine, Crossed):** marks the indices of the adjacencies crossed by line *ALine* as ***true*** in the boolean array *Crossed*, and returns the index of the next uncrossed adjacency higher than $head(ALine.adjList)$; or the smallest index of uncrossed adjacency if we are at the last adjacency; or -1 if all the adjacencies are crossed.
- **unMarkAdj(ALine, Crossed):** marks the indices of the adjacencies crossed by line *ALine* as ***false*** in the boolean array *Crossed*, i.e. if an adjacency with index k is crossed by *ALine*, then $Crossed[k]$ will be marked as ***false***.

Figure 5.3: A list of functions used in the heuristic algorithm

next considers rectangle 5 ($LSorted[1]$) and rectangle 2 ($LSorted[3]$). As $LSorted[5].right$ is equal to $LSorted[2].left$, ***line 15*** checks if there is a vertical overlap between the two rectangles. As there is an overlap, the rectangles are adjacent and the information of adjacency is kept in the array Adj (***lines 16-21***). Moreover, the adjacency is appended to the right adjacency list ($adjListRight$) of rectangle 5 and to the left adjacency list ($adjListLeft$) of rectangle 2 (***lines 24 and 25***). Then k is incremented, and the algorithm proceeds until all the adjacencies between the rectangles are determined.

The complexity of the algorithm is $O(n^2)$ in the worst case although its performance is much better in the average case as it skips unnecessary computations. The algorithm checks each rectangle against the others if they have common adjacency until either there is horizontal gap between the rectangles (e.g. Rectangles 5 and 7 in Figure 5.2 where the rest of the rectangles in $LSorted$ cannot be adjacent with Rectangle 5) or the right edge of the current rectangle is the same as the left edge of the other rectangle but there is a vertical gap between the two rectangles (e.g. Rectangles 1 and 6 in Figure 5.2 in which the rest of the Rectangles in $LSorted$ cannot be adjacent to rectangle 1). A discussion on complexity analysis of the algorithm is given in Section 5.4.

5.2.3 Generating All Possible Single-crossing Orthogonal Lines

Once the adjacencies between the orthogonal rectangles are determined, the next phase is to place all possible single-crossing orthogonal lines to cross the adjacencies. A method of generating all possible single-crossing lines is given in Algorithm 3.

The algorithm starts by defining two arrays of line record (***lines 2 and 4***) with the same size as the number of adjacencies: $AllLines[i]$ which stores the list of lines that originate from adjacency i (list of lines whose left most adjacency is i) and that could be extended towards the right, and $Extend[i]$ which stores the list of lines that are extended up to adjacency i. In other words, all lines that may originate from a particular adjacency i and be extended to the right are stored in $AllLines[i]$ and the lines extended as far as adjacency i are stored in $Extend[i]$. First a line $AllLines[i]$ is created to cross the current adjacency i and another one $Extend[i]$ that is extended up to adjacency i (***line 8***). Then the algorithm checks if any previously generated lines can possibly be extended towards the current adjacency under consideration. In order to do that, the algorithm picks the rectangle to the left of the current adjacency. Then if any previous line can be extended towards the current adjacency, it should be already extended (or included) up to the left adjacency (left edge) of the current rectangle which is to the left of the current adjacency. So, ***line 14*** of the algorithm stores

Algorithm 3 An algorithm that generates all possible single-crossing orthogonal lines to cross the adjacencies between a collection of non-overlapping rectangles

INPUT: An array of adjacency record *Adj*[] *(produced by Algorithm 1).*
OUTPUT: An array of line record *AllLines*[].

```
    m ← number of vertical adjacencies
    Create an array of line record AllLines[] with size m
    {to keep track the list of lines that originate from a particular adjacency and are extended
    towards the right}
    Create an array of line record Extend[] with size m
 5: {to keep track the list of lines that are extended up to a particular adjacency}

    for i = 1 to m do
      {loop through the array of adjacency record}
      Create lines AllLines[i] and Extend[i]
      Add Adj[i] to the list of adjacencies crossed by AllLines[i]
10:   Add Adj[i] to the list of adjacencies crossed by Extend[i]
      {currect is the rectangle to the left of current adjacency}
      currect ← Adj[i].leftRectangle
      {leftadj is the list of left adjacencies of currect}
      leftadj ← currect.leftAdj

15:   while (leftadj is not empty) do
        extline ← Extend[head(leftadj).name]
        {extline is the list of lines that cross leftadj and hence candidates for possible ex-
        tension towards the current adjacency}

        while (extline is not empty) do
          if (partiallyVisible(head(extline), Adj[i]) = true) then
20:         {the head of extline can be extended towards the current adjacency}
            {copy the head of extline to newline}
            newline ← head(extline)
            {add the current adjacency to the list of adjacencies crossed by newline}
            Append Adj[i] to newline.adjList
25:         first ← the left most (first) adjacency crossed by newline

            {add the extended line (newline) to the list of lines originated from adjacency
            first; and to the list of lines that are extended up to the current adjacency and
            are candidates for possible extension towards the right}
            Append newline to the list of AllLines[first]
            Append newline to the list of Extend[i]
          end if

30:       extline ← tail(extline)
        end while

        leftadj ← tail(leftadj)
      end while

    end for
```

the list of these left adjacencies in *leftadj*. Then the algorithm considers each line that crosses the left adjacency of the rectangle towards the current adjacency (***lines 16-18***) and checks if any of them is horizontally partially visible with the current adjacency under consideration (***line 19***). For those lines that are horizontally partially visible, a copy of the line is formed and the current adjacency is included to the list of adjacencies crossed by the new line (copied line) (***line 21***). Then the new line is appended to the list of lines that are extended up to the current adjacency (*Extended*[*i*]). Moreover, the line is appended to the list of lines that are originated from the same adjacency (***lines 25-28***). This process continues until all the adjacencies in the configuration of non-overlapping rectangles are considered in turn.

To demonstrate how the algorithm generates all possible single-crossing lines to cross the adjacencies, consider the configuration of rectangles in Figure 5.4. For the sake of convenience, we represent the lines by the adjacencies they cross, i.e, $-2-$ is used to represent the line that crosses adjacency 2 while $-2-4-6-$ represents the line that crosses adjacencies 2, 4 and 6. The first adjacency to consider is adjacency 1. Then *AllLines*[*1*] = *Extend*[*1*] = $-1-$. Likewise, for adjacencies 2 and 3, we have *AllLines*[*2*] = *Extend*[*2*] = $-2-$, and *AllLines*[*3*] = *Extend*[*3*] = $-3-$ respectively. The next adjacency to consider would be adjacency 4, in which case *AllLines*[*4*] = *Extend*[*4*] = $-4-$. Moreover, ***lines 22 and 24*** of the algorithm extends line $-2-$ to include adjacency 4, and append the line $-2-4-$ in the list of lines of *AllLines*[*2*] (***line 27***). Furthermore, ***line 28*** appends line $-2-4-$ to the list of lines extended up to adjacency 4, which is *Extend*[*4*]. The algorithm continues until all adjacencies are considered. The lines generated and their representation is shown in Figure 5.4.

5.2.4 Selecting an Approximate Minimum Number Single-crossing Orthogonal Axial Lines

Algorithm 4 presents a greedy method to determine the approximate minimum number of lines from all possible single-crossing lines. The idea behind the algorithm is that by repeatedly selecting the lines that cross the highest number of adjacencies from the list of previously uncrossed adjacencies, we could get a reasonably approximated solution which is close to the optimal lines that cross all adjacencies exactly once. As the problem restricts adjacencies to be crossed only once, a boolean array *Crossed* is defined to keep track of which adjacencies are already crossed by the lines in the solution set. Initially, the array is set to *false* (***lines 2 and 4***). The *computeDegree* function (***line 5***) computes the number of adjacencies a line crosses and the *degree* field of the line record is updated accordingly.

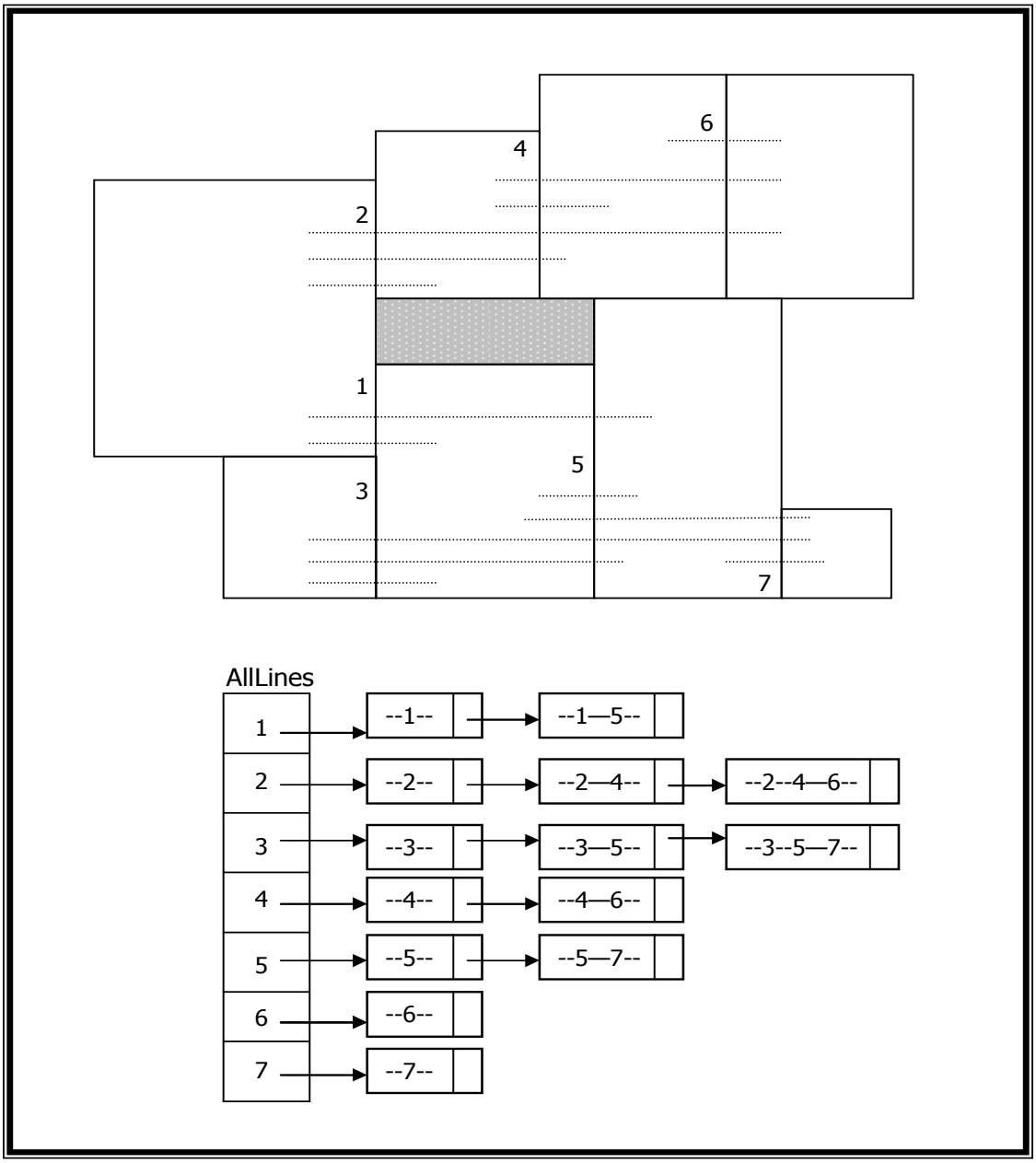

Figure 5.4: Placing all possible single-crossing orthogonal lines in orthogonal rectangles

For each index of the adjacency record, Algorithm 4 sorts the list of lines that originate from the adjacency based on the number of adjacencies they cross (*degree*) in descending order. The algorithm traverses through the list of lines that originate from the adjacency under consideration, add the line that crosses the highest number of previously uncrossed adjacencies (not crossed by the lines in the local solution set) to the set of local solution and marks the adjacencies as *crossed*. There is at least one line that crosses previously uncrossed adjacencies – the line that crosses the current adjacency only.

The iteration starts from the first adjacency and selects the first line, i.e. the highest degree, from the list of lines and marks the adjacencies crossed by the line as *true* in the boolean array *Crossed* (***lines 9-15***). Moreover, the *markAdj* function returns the smallest adjacency higher than the current adjacency which is not yet crossed (the smallest index of *Crossed* higher than the current adjacency whose value is *false*); or the smallest uncrossed adjacency if we reach the last adjacency; or *-1* if all the adjacencies are crossed by the lines in the current solution. If there are uncrossed adjacencies, we pick our next line in the solution by traversing through the list of lines originating from the uncrossed adjacency and are extended towards the right (***lines 20-23***). At each step an uncrossed adjacency is detected, a line that originates from the uncrossed adjacency and crosses previously uncrossed adjacencies is included to the local solution and the number of lines in the local solution is incremented subsequently (***lines 25-26***). Once all the adjacencies are crossed, the number of lines in the local solution is compared to the global number of lines (the smallest so far) and the global solution is updated if the local solution is minimum (***lines 29-32***). Before starting the next iteration, the boolean array *Crossed* is reset to *false* and a new iteration begins from the next adjacency. This process continues until all the adjacencies are considered in turn.

To demonstrate how the heuristic algorithm proceeds, consider the configuration of orthogonal rectangles in Figure 5.2. Algorithm 3 generates all possible single-crossing orthogonal lines to cross the adjacencies, as shown in Figure 5.4. The heuristic algorithm then begins by selecting the line that crosses the highest number of adjacencies from the list of lines of $AllLines[1]$, which is $-1-5-$. The next minimum uncrossed adjacency higher than one is 2, in which case, we select the line $-2-4-6-$ from the list of lines originating from adjacency 2. The next adjacency to consider would be adjacency 3. The line $-3-5-7-$ cannot be included to the local solution because adjacency 5 is already crossed by a line in the local solution. By the same reasoning, the line $-3-5-$ cannot be part of the local solution. So, line $-3-$ is included to the local solution set. The final uncrossed adjacency

Algorithm 4 An algorithm to compute the approximate minimum number of single-crossing orthogonal lines to cross the adjacencies between a collection of orthogonal rectangles

INPUT: An array of all possible lines *AllLines*[] *(produced by Algorithm 2).*
OUTPUT: A list of single-crossing lines *globalsoln.*

```
    m ← number of vertical adjacencies
    Create a boolean array Crossed with size m
    {to mark the adjacencies as crossed, if a line that crosses the adjacency is included in
    the local solution}
    Initialize Crossed as false

 5: computeDegree(AllLines)
    Sort the list of lines in AllLines in descending order of the number of adjacencies they
    cross (AllLines[i].degree)
    globalnum ← m
    {globalnum stores the minimum number of single-crossing lines detected so far. It is
    initialized with the trivial solution – each adjacency is crossed by a line that crosses the
    adjacency only}

    for i = 1 to m do
10:   {loop through the array of list of lines AllLines}
      currline ← AllLines[i]
      {the list of lines that originate from adjacency i and are extended towards the right}
      localsoln ← head(currline)
      localnum ← 1
15:   k ← markAdj(head(currline), Crossed)

      while (k ≠ −1) do
        {there are uncrossed adjacencies}
        currline ← AllLines[k]
        {the list of lines that originate from adjacency k and are extended towards the
        right}

20:     while (isMarked(head(currline), Crossed)) do
          {check if any of the adjacencies crossed by currline are already crossed. If so, go
          to the next line in the list, otherwise include the line in the list of localsoln}
          currline ← tail(currline)
        end while

        {update the local solution}
25:     Append head(currline) to localsoln
        localnum ← localnum + 1
        k ← markAdj(head(currline), Crossed)
      end while

      if (localnum < globalnum) then
30:     {a smaller number of single-crossing lines detected, and hence the global solution is
        updated}
        globalnum ← localnum
        globalsoln ← localsoln
      end if

      unMarkAdj(localsoln), Crossed)
35:   {reset array Crossed to false}

    end for
```

is adjacency 7 and hence the line $-7-$ is included in the local solution. During the first iteration we need four lines to cross the adjacencies between the collection of orthogonal rectangles, and hence the global solution is updated with the current local solution.

The second iteration starts by selecting the lines that originate from adjacency 2, in which the line $-2-4-6-$ is included in the local solution. The next uncrossed adjacency higher than two is adjacency 3. The line with highest adjacency crossing is $-3-5-7-$ in which case it is included in the local solution as none of the adjacencies are previously crossed by the lines in the local solution. The final uncrossed adjacency is adjacency 1, and hence the line $-1-$ is selected. As we found fewer lines than the first iteration, the global solution is updated by the new local solution. The algorithm continues by selecting the lines that originate from each adjacency in turn until the lines originating from all adjacencies are considered in turn, and the heuristic returns the minimum of all as the approximated solution to the ALP-SC-OLOR problem.

In the example above, the heuristic algorithm gives the optimal solution, but this is not always true. An example of a configuration of orthogonal rectangles that the heuristic algorithm fails to give an optimal solution is given in Figure 5.5. The heuristic gives six single-crossing orthogonal lines, while the adjacencies between the collection of rectangles can be crossed by five single-crossing orthogonal lines.

Once a line is included in the set of local solution, another approach is to start from the smallest uncrossed adjacency rather than going to the next uncrossed adjacency. This was due to the fact that as adjacencies are restricted to be crossed only once, the order of the lines added to the local solution affects the final solution. But in most cases, the result obtained by this approach coincides with the approach discussed above. This is because selecting a line from the list originating the left most adjacency is extended further towards the right, and hence may cover the adjacencies with higher indices. But the converse is not true – a line originating from a higher adjacency cannot cross adjacencies with the smaller indices as the lines are extended from the left most towards the right. As a result it was considered as redundant, and left out from the algorithm.

5.3 Correctness of the Heuristic

The correctness of the heuristic algorithm depends on two factors: each adjacency must be crossed by a line from the set of lines in the final solution and no adjacency is crossed more than once. The lines generated by Algorithm 3 are organised in such a way that the lines that have the same leftmost adjacency are placed in a list and the list is stored in an array

(*AllLines*) of the same index as the leftmost adjacency. Algorithm 4 starts by selecting the first line from the list of lines that cross the leftmost adjacency, while traversing to the right by considering the leftmost uncrossed adjacency at every step. And when a line is included in the local solution, all the adjacencies the line crosses are marked as *crossed* so that another line that may cross any previously crossed adjacency will not be included in the final solution. The *isMarked* function checks if any of the adjacencies crossed by a line is already crossed by a line in the final solution. This guarantees that none of the adjacencies formed by the collection of rectangles is crossed more than once or is left uncrossed.

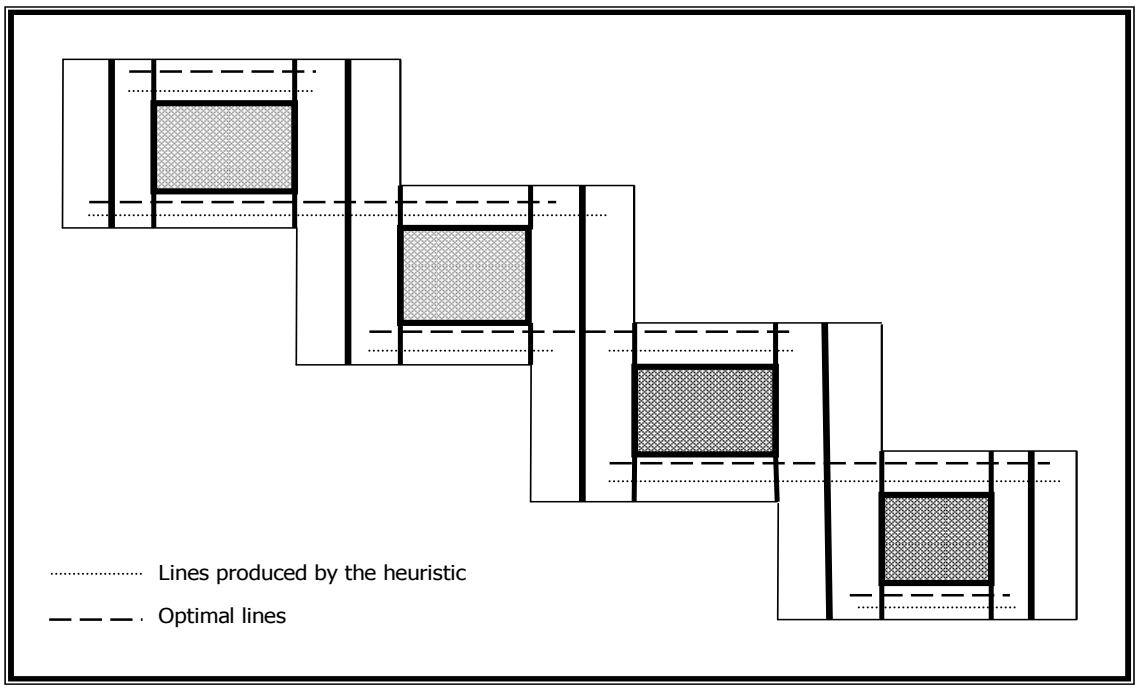

Figure 5.5: An example of a configuration of orthogonal rectangles where the heuristic fails to give the optimal solution (1)

At every new addition of a line to the local solution, the adjacencies the line crosses are marked as *crossed*, and the index of the smallest uncrossed adjacency k higher than the current adjacency is computed (*markAdj* function). ***Line 20*** of Algorithm 4 selects a line that crosses previously uncrossed adjacency. There is at least one line that originates from the current adjacency under consideration and crosses previously uncrossed adjacencies by a line in the local solution – the line that crosses the current adjacency only. The *while* loop (***lines 16-28***) guarantees us that all adjacencies are crossed. At every iteration, the global solution (so far minimum number of single-crossing lines identified by Algorithm 4)

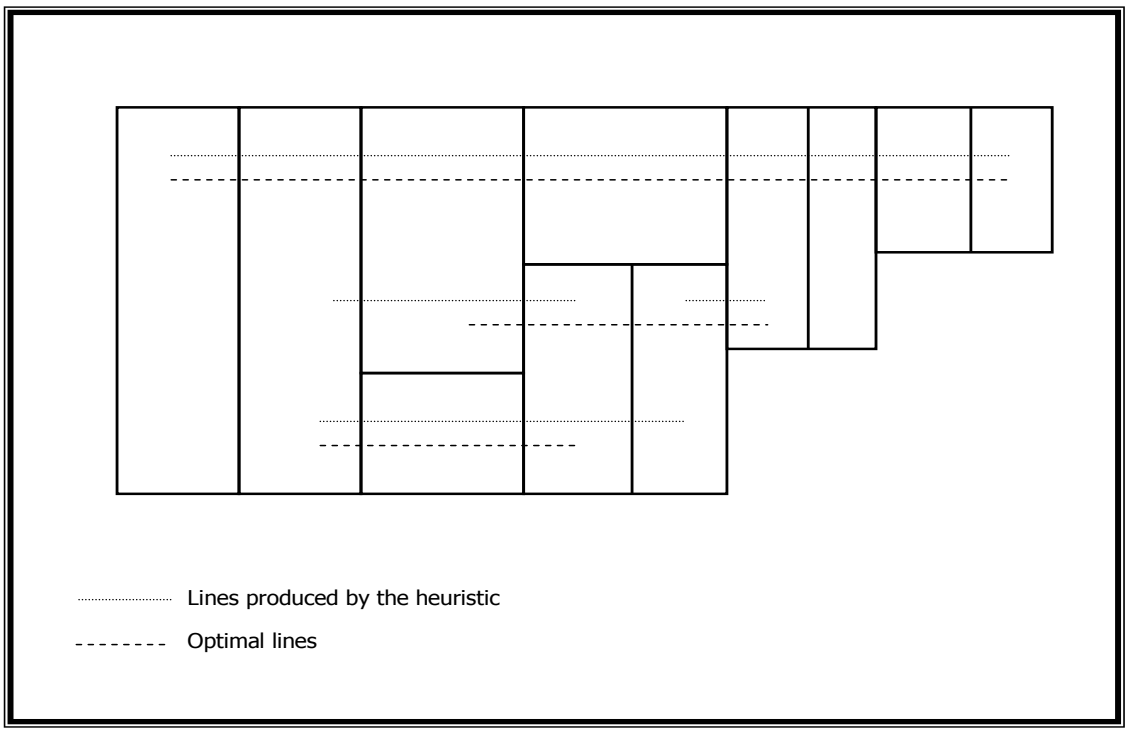

Figure 5.6: An example of a configuration of orthogonal rectangles where the heuristic fails to give the optimal solution (2)

is updated by the local solution if the number of lines in the local solution is less than the number of lines in the global solution (***lines 29-32***).

There is a trivial solution to the problem where each single-crossing line crosses a single adjacency, in which case the number of single-crossing lines would be the same as the number of vertical adjacencies. As the objective of the problem is to compute for the minimum single-crossing lines, the lines with higher degree are selected to the local solution provided that the adjacencies were not previously crossed.

The heuristic does not always produce the optimal solution. This is because the heuristic always includes at least one maximal line whereas the optimal solution may not contain a maximal line. An example of such scenario is given in Figure 5.5. The heuristic may also fail to give the optimal solution due to the order of the lines the heuristic chooses to include in the solution even if the optimal solution contains one or more maximal lines.

A loose guaranteed solution analysis of the heuristic could be performed based on the set packing exact-cover problem – given a collection of sets, computing the union of a minimum

number of disjoint sets such that every set in the collection is a subset of the union set (solution set). A greedy algorithm produces a solution within a factor of k of the optimal, where k is the highest cardinality of the collection of sets. This is because choosing the set with maximal cardinality may prevent the greedy algorithm from selecting other k sets in the collection that may have a common element with the maximal set but disjoint to one another and hence may constitute the solution. In the case of ALP-SC-OLOR, Algorithm 4 considers the maximal lines that may originate from each adjacency in turn. As a result, the solution obtained by the heuristic is much tighter than the k factor bound, where k is the maximum number of adjacencies that can be crossed by the same line.

5.4 Complexity Analysis

Theoretical and empirical analysis of the heuristic algorithm has been performed. This is based on how close the solution of the heuristic algorithm is as compared to the optimal solution, and determining the running time of the heuristic. Since running time to compute the exact solution of an NP-complete problem grows exponentially with respect to the input size, a trade-off is necessary on a heuristic which runs faster but the result may not be optimal. In the following subsections, the theoretical and empirical complexity analysis of the heuristic are discussed.

5.4.1 Theoretical Analysis

As described in the previous section, the heuristic algorithm has three phases: determining the adjacencies between a collection of orthogonal rectangles, generating all possible single-crossing lines to cross the adjacencies between the rectangles, and selecting the minimum number of single-crossing lines to cover all the adjacencies between the collection of orthogonal rectangles. As these phases are independent to each other, they are analysed individually below.

5.4.1.1 Determining Adjacencies

The first phase of the heuristic is to determine the adjacencies between the orthogonal rectangles. Since in ALP-SC-OLOR we are interested in vertical adjacencies only Theorem 5.1 shows the number of vertical adjacencies is bounded by $2n - 4, n \geq 3$, where n is the number of rectangles.

When analysing the complexity of Algorithm 2, copying the array of rectangles is $O(n)$. Merge sort can be used in sorting the array of rectangles, which will be $O(n \log n)$. The

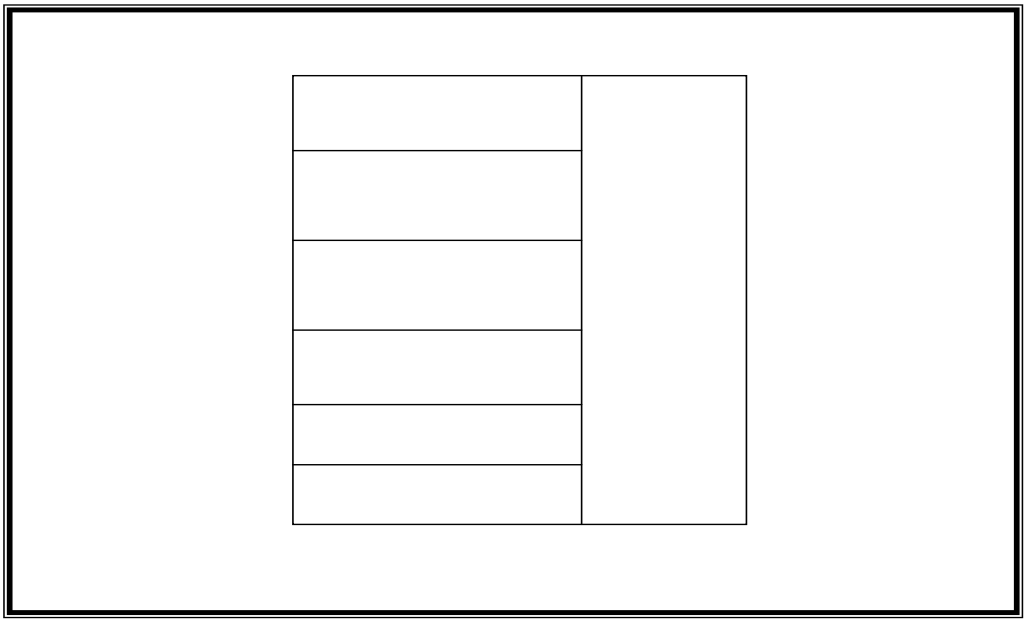

Figure 5.7: A configuration of orthogonal rectangles that exhibit the worst performance of Algorithm 2

outer *while* loop (***line 8***) runs n times and the inner *while* loop (***line 10***) runs n times in the worst case although there is an improvement in the average case – when a gap is detected between two rectangles, the two rectangles cannot be adjacent. In other words, if there are horizontal and vertical gaps between the current rectangle under consideration and another rectangle then there is no adjacency between the two rectangles and the rest of the rectangles in *LSorted*, and hence the rest of the sorted array of rectangles are skipped from checking for adjacency with the current rectangle. Figure 5.7 shows the worst case scenario of the algorithm of determining adjacencies. In that case, all the rectangles are checked against one another whether they are adjacent or not, which would be $O(n^2)$.

The best performance of Algorithm 2 is when the configuration of rectangles form a chain of rectangles. In that case, the outer *while* loop runs n times whereas the inner *while* loop runs only once. As the sorting operation would be dominant, the complexity of the algorithm in the best case is $O(n \log n)$. An example of a chain of rectangles is given in Figure 5.8.

Sanders [2002] presented an $O(n \log n)$ algorithm for determining adjacencies between a collection of rectangles using a line sweeping method. During implementation of the heuristic, Algorithm 2 was used as the overall complexity of the heuristic cannot be improved

even if we used the line sweeping method to determine the adjacencies between the collection of rectangles.

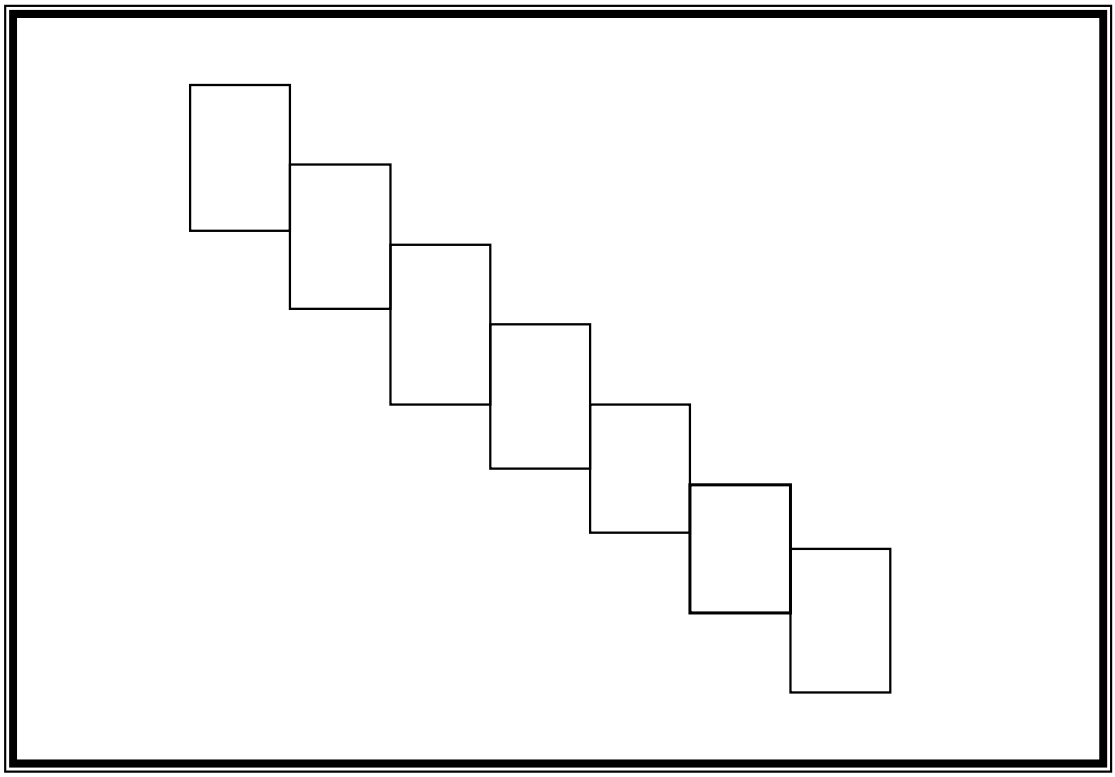

Figure 5.8: An example of a configuration of orthogonal rectangles the heuristic exhibits its best running time

5.4.1.2 Generating All Possible Single-crossing Orthogonal Lines

An algorithm for generating all possible non-redundant single-crossing lines that might be part of the solution set of single-crossing lines is given in Algorithm 3. The lines that originate from the same adjacency (left-most adjacency) form a list and then the list is stored in an array with the same index as the left-most adjacency. In the worst case, the number of lines originating from a particular adjacency is linear with respect to the number of adjacencies and hence the number of rectangles. If we have adjacencies a_1, a_2, ..., a_k such that every adjacency is horizontally partially visible to the other adjacencies then there are k lines that originate from adjacency 1 (a_1) and are extended towards the right adjacencies, $(k-1)$ lines that originate from adjacency 2 (a_2) and extended towards the right adjacencies, and so on,

and there will be one line that originates from adjacency k (a_k). Clearly, the total number of lines produced is quadratic with respect to the number of adjacencies.

When analysing the complexity of Algorithm 3, the *for* loop (***line 6***) iterates m times where m is the number of adjacencies. *leftadj* stores the adjacencies a rectangle may form with its left neighbours, and hence the outer *while* statement (***lines 15***) could loop $O(n)$ times. The inner *while* loop (***line 18***) iterates through the list of lines that could be extended further to the right which is $O(m)$. Copying a line (***lines 23***) could take $O(m)$ work. Adding an adjacency to a list of adjacencies crossed by the same line or appending a line to a list of lines (***lines 24–28***) can be done in constant time. Therefore, the overall complexity of the algorithm for generating all possible single-crossing lines is $O(m^3)$. But then since the number of adjacencies is linear with respect to the number of rectangles the overall complexity of Algorithm 3 in the worst case is $O(n^3)$.

The worst running time of the algorithm is when the collection of rectangles form a chain where all the adjacencies formed are horizontally partially visible to one another. In that case, every line originating from any adjacency will be extended to all the adjacencies to its right. The *for* statement will iterate m times and the inner *while* statement loops $O(m)$ times for each iteration of the *for* loop. Making a new copy of a line could be done in $O(m)$ time. The outer *while* loop executes only once for every loop of the *for* statement. Therefore, the overall complexity of Algorithm 3 is $O(n^3)$ as the number of adjacencies is linear with respect to the number of rectangles.

The best case is when there is only one line generated from each adjacency in which the rectangles form a chain as shown in Figure 5.8. The total number of lines produced will be m which is the same as the number of adjacencies. The *for* statement loops m times whereas the outer *while* iterates only once since each rectangle in the configuration has one left adjacency and the inner *while* loop is executed only once. Therefore, the overall best case complexity is $O(n)$ since m is linear with respect to the number of rectangles.

5.4.1.3 Selecting an Approximate Minimum Number of Single-Crossing Orthogonal Lines

As discussed previously, the algorithm of selecting an approximate minimum number of single-crossing lines is presented in Algorithm 4. It was also discussed in the previous section that the maximum number of possible single-crossing lines is quadratic with respect to the number of adjacencies. The *computeDegree* subroutine (***line 5***) counts the number of adjacencies each line can cross. Clearly, the running time of the subroutine is bounded by

$O(n^3)$, where n is the number of adjacencies and hence the number of rectangles. This is because the maximum number of lines is quadratic (with respect to the number of rectangles) and, in the worst case, we need to traverse through the list of adjacencies crossed by each line.

Sorting the lines that originate from the same adjacency by their degree (***line 6***) was implemented based on insertion sort, which would be $O(n^2)$ in the worst case. This is because the single-crossing lines are kept in a list and the overall complexity cannot be improved even if the running time of sorting is improved to $O(n \log n)$. As there are m adjacencies which is linear with respect to the number of rectangles, the total complexity of sorting is $O(n^3)$. The *for* loop on ***line 9*** iterates through the number of adjacencies which is m times. The *markAdj* subroutine (***line 15***) marks the indices of the adjacencies crossed by the line passed as parameter as *true* in the boolean array and then computes for the next adjacency k which was previously uncrossed by the lines in the local solution. Clearly the subroutine takes linear time with respect to the number of adjacencies.

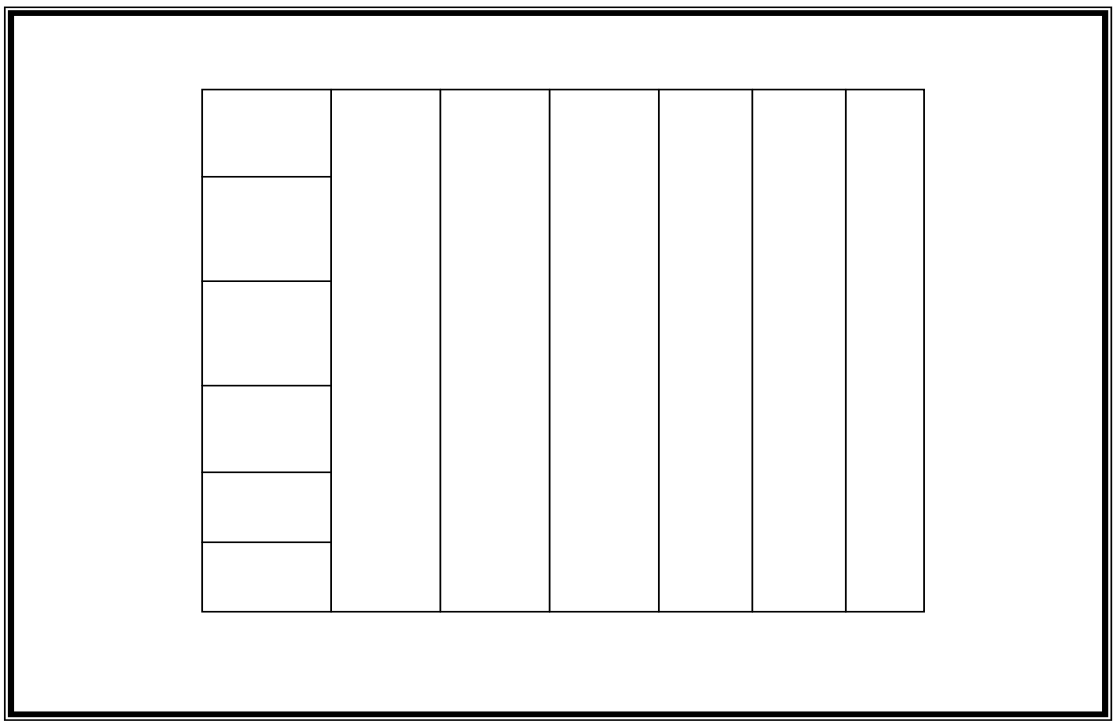

Figure 5.9: A configuration of rectangles in which lines originated from $O(n)$ adjacencies are extended to other $O(n)$ adjacencies

The outer *while* loop on ***line 16*** traverses through the adjacencies. Marking the adjacencies as crossed and computing for k can be done in $O(n)$. The inner while loop is bounded by $O(n^2)$ – n for the list of lines and n for *isMark* function – as the number of adjacencies is linear with respect to the number of rectangles. Therefore, the overall complexity of the outer *while* loop is $O(n^3)$. Resetting adjacencies as unmarked is linear, and hence the complexity of the *for* loop is $O(n^4)$. As it is the dominant component of the algorithm, the complexity of selecting a minimum number of single-crossing lines to cross adjacencies between a collection of rectangles is $O(n^4)$.

The complexity analysis of Algorithm 4, as presented above, is based on the potential number of iterations. As a result, the worst case of the algorithm could not be constructed. Figure 5.9 shows a configuration of rectangles where $n/2$ rectangles are adjacent to a particular rectangle forming $O(n)$ adjacencies, and the lines originated from these adjacencies are extended further to $O(n)$ other adjacencies. The time complexity of Algorithm 4 for the configuration of rectangles of the form of Figure 5.9 is $O(n^3)$ – $O(n)$ work for traversing through the lines in a list of lines originated from $O(n)$ adjacencies, and this work is repeated for each adjacency in turn.

The best performance of Algorithm 4 is when the orthogonal rectangles form a chain of rectangles where the adjacencies are not horizontally partially visible to one another. This configuration forces the generation of all possible lines to produce only one line from each adjacency. The *for* loop executes m times whereas the outer *while* loop runs m times, and hence the overall complexity in the best case is $O(n^2)$, as the inner *while* loop is executed only once. Figure 5.8 shows a chain of rectangles in which Algorithm 4 performs best.

5.4.1.4 Summary

The theoretical analysis of the heuristic algorithm was categorized into three phases. Since each phase is independent of one another, a discussion on the best and worst cases of each phase was presented individually. In the worst case, selecting the approximate minimum single-crossing lines is the dominant method and hence the complexity is $O(n^4)$. In the best case, selecting the approximate minimum single-crossing lines is the dominant method. Hence, the running time of the heuristic is $O(n^2)$ in the best case. The configuration of rectangles that exhibit best performance of the heuristic is given in Figure 5.8.

5.4.2 Empirical Analysis

As discussed above, the theoretical analysis of the heuristic shows that its complexity is $O(n^4)$ in the worst case. In order to verify the theoretical analysis, the running time of the heuristic was tested on randomly generated rectangles of size of 100 to 1550 at an interval of 50. For each data size, five instances were generated and each instance was repeated five times. This is in order to minimize the external factors (processes) that may affect the timing measurement. The equipment in which the empirical analysis was performed was a Pentium III 870MHz processor with 128MB RAM. The operating system was Linux and the language of implementation was Java.

The empirical result of the approximation algorithm is given in Table 5.1. The data has been plotted and a curve fitted in order to determine the complexity of the heuristic empirically for the tested data. As shown in Figure 5.10, the curve that best fits was $Y = 0.026X^2 - 4.2X + 456$. A cube function ($Y = 0.0000048X^3 + 0.017X^2 - 1.45X + 100$) curve was also fitted to the graph of the running time of the heuristic, but the quadratic function curve was found to be better fitted (line fit coefficient of the cube function was 0.968 while that of the quadratic function is 0.974). Moreover, the small coefficient of the highest degree in the cube curve reflects the fact that the running time of the heuristic for the tested data was quadratic. A possible reason for the quadratic complexity of the heuristic could be none or few of the rectangles generated during experimentation exhibit the worst performance of the algorithm.

Another possible reason for the gap between the theoretical and empirical complexity analysis is that in the theoretical analysis, the loops and functions are analysed based on the potential number of iterations they can loop, i.e. bound on the number of iterations.

5.5 Experimental Results

Algorithm 2, Algorithm 3 and Algorithm 4 were coded and tested on a number of non-overlapping orthogonal rectangles. In order to verify the performance of the heuristic, an exact solution algorithm was also implemented based on exhaustive search. In order to make the exact solution more efficient, only lines that originate from adjacencies which were not previously crossed were considered in searching for minimum combination. That is, if a line is selected from a list of lines originating from a particular adjacency, the other lines in the list are not considered during that particular combination. Since the number of possible

Number of Rectangles	Average Number of Lines Produced (to the nearest integer)	Standard Deviation	Average Running Time (in ms)	Standard Deviation
100	47	2.30	314.32	2.83
150	73	2.59	470.99	28.67
200	95	5.86	1198.32	15.38
250	124	8.17	1552.35	78.36
300	151	12.21	2389.20	41.70
350	164	6.83	3167.00	63.93
400	200	10.86	5745.56	38.91
450	224	18.30	6568.43	122.96
500	237	9.62	8026.24	104.91
550	267	13.91	11470.72	583.29
600	291	13.77	12467.08	303.74
650	323	15.27	14565.44	151.38
700	340	20.28	16905.44	300.39
750	366	18.84	20713.69	232.84
800	388	19.10	22575.68	1095.39
850	413	28.40	29307.21	927.51
900	446	12.46	32766.24	553.42
950	464	18.46	35392.65	965.07
1000	498	6.02	37399.00	673.12
1050	510	31.60	42053.56	1348.30
1100	548	9.42	44542.80	1100.17
1150	574	3.40	47733.12	1904.86
1200	579	13.65	55771.32	806.48
1250	625	14.86	57717.52	1515.04
1300	638	10.21	64076.65	1301.13
1350	665	8.62	71283.02	1440.24
1400	697	12.50	77304.75	1511.26
1450	718	7.41	86063.43	2371.60
1500	737	20.06	88944.80	2331.47
1550	777	2.65	94633.80	1772.91

Table 5.1: Empirical analysis of the heuristic algorithm

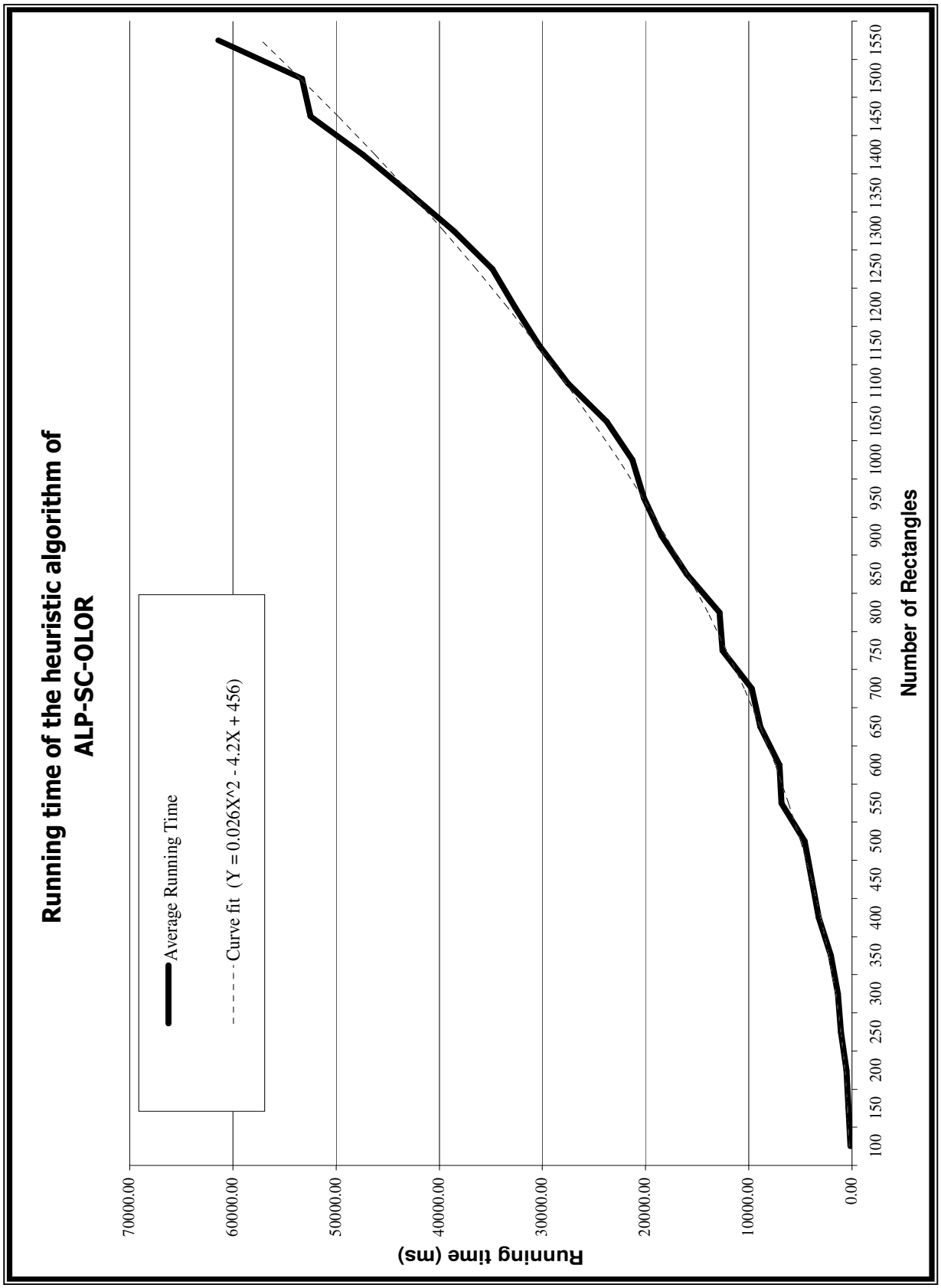

Figure 5.10: Running time of the heuristic algorithm

Number of rectangles	30	40	50
Number of tests	20	15	10
Max. number of vertical adjacencies	46	59	75
Min. number of vertical adjacencies	28	41	44
Average number of vertical adjacencies	30	51	61
Standard deviation	4.26	4.50	10.87
Max. number of single-crossing lines	316	371	457
Min. number of single-crossing lines	70	116	109
Average number of single-crossing lines	165	227	273
Standard deviation	61.86	61.32	125.47
Max. heuristic deviation from optimal	2	3	2
Min. heuristic deviation from optimal	0	0	0
Average heuristic deviation from optimal	0.45	0.60	0.60
Standard deviation	0.69	0.91	0.84
Average heuristic solution	13.55	19.27	23.3
Average optimal solution	13.10	18.67	22.7
Max. running time (optimal solution)	11654.27 sec	5374.93 min	489.27 hrs
Min. running time (optimal solution)	10.56 sec	10.73 min	9.84 hrs
Average running time (optimal solution)	875.44 sec	1321.72 min	175.74 hrs
Relative standard deviation (%)	299.47	145.12	108.93

Table 5.2: Experimental result

single-crossing lines could be quadratic with respect to the number of adjacencies and hence the number of rectangles, computing the exact solution was only possible for small number of rectangles. As a result, twenty cases were tested for rectangles of size 30, fifteen cases for rectangles of size 40 and ten cases for rectangles of size 50.

The rectangles used for testing were randomly generated. The width and height of the rectangles were randomly generated between units of 1 and 10. The rectangles were generated on top of one another until the height reached $(Integer(\sqrt{n})) * 10 * p$ where p is a random number between 0.1 and 0.75. This is in order to favour more rectangles being extended to the right than to the top, and hence more vertical adjacencies would be produced between the rectangles. That is, as p gets higher the rectangles would be extended towards the top than to the right. Then the next rectangles were placed (generated) adjacent to the previously created ones towards their right hand side. When the rectangles were generated on top of one another, a hole was included between the rectangles based on a probability of 0.05% to 5% in order to generate different configurations. A hole is a region which is entirely bounded by the collection of rectangles but is not part of the rectangles (the shaded region in Figure 5.2).

The heuristic performed well for the data used during experimentation. In most cases, it was able to determine the minimum single-crossing lines required to cross all adjacencies, although there were differences from the optimal solution in some cases. The difference arises due to the fact that the heuristic includes at least one maximal line that may originate from a particular adjacency in the final solution, in which case the optimal solution may not contain any of these lines. As it is illustrated in Figure 5.5, the optimal solution may not contain any of the maximal lines whereas the heuristic produces one maximal line (second from top) in its final solution. The order in which the heuristic chooses the single-crossing lines may also prevent the heuristic from producing the optimal solution even if the optimal solution contains maximal lines. An example of such scenario is shown in Figure 5.6.

The results of the experiment are given in Table 5.2. The results reflect the heuristic gives a reasonable approximate solution. But the observed result could be due to the small number of rectangles used for experimentation, and hence the heuristic may not perform well with the increase of the number of rectangles.

5.6 Conclusion

This chapter presented the experimental results obtained by the heuristic algorithm proposed for ALP-SC-OLOR. The heuristic is based on the greedy method. It was implemented and

the accuracy of the solution obtained was evaluated with respect to the exact solution. But as the implementation of the exact solution was a brute force algorithm, it was only possible to run for configuration of small number of rectangles. As a result, the conclusion is limited for small data sets and may not be generalized for large data sets.

In order to determine the running time of the heuristic empirically, the heuristic algorithm was coded and tested on several randomly generated rectangles. The result shows that the average case complexity for the tested data was $O(n^2)$, although in the worst case the running time complexity could be $O(n^4)$. This is based on best curve fit with least square analysis. As the theoretical analysis was performed based on the potential number of iterations, the worst case may not be constructed.

Chapter 6

Future Work

6.1 Introduction

The problem of placing a minimum number of single-crossing orthogonal lines to cross the adjacencies between a collection of orthogonal rectangles has been found to be NP-complete. As it is open question whether any NP-complete problem could be solved efficiently for a given input or not, research associated with NP-complete problems are mainly developing heuristic algorithms which give approximate solutions and exploring special cases of the NP-complete problem that is polynomially solvable. Accordingly the future research presented in this chapter is mainly developing heuristics for the ALP-SC-OLOR and BIDS problems based on different approaches and special configurations in which ALP-SC-OLOR and BIDS are polynomially solvable.

In the next section, future work associated with ALP-SC-OLOR is presented followed by possible research on single adjacency crossing axial lines in deformed urban grids. Lastly, an outline of future research on BIDS is given.

6.2 ALP-SC-OLOR

6.2.1 Special Cases of ALP-SC-OLOR

As it is presented in Section 2.7.4.1 although ALP-OLOR is in general NP-complete, a number of configurations of orthogonal rectangles have been found in which ALP-OLOR is polynomially solvable. As a generalization to the special cases, ALP-OLOR has been found to be polynomially solvable for any hole free collection of rectangles. A collection of rectangles is hole free if it does not contain holes. A possible research would be to explore special configurations of collections of rectangles in which ALP-SC-OLOR is solvable in

polynomial time.

6.2.2 Heuristic for ALP-SC-OLOR

The heuristic for ALP-SC-OLOR discussed in Chapter 5 is based on a greedy method where it first produces all possible single-crossing lines and then recursively checks if there are a minimum number of single-crossing lines less than the current solution by considering the maximal line that can originate from a particular adjacency and extended towards the left at every iteration. Another approach is to categorize the collection of rectangles into groups of rectangles where ALP-SC-OLOR is polynomially solvable and compute the single-crossing lines independently. Possible research is to develop a heuristic for ALP-SC-OLOR using the latter approach.

6.2.3 Modern Heuristic Techniques for ALP-SC-OLOR

Another possible research area is to look at heuristics for ALP-SC-OLOR based on modern heuristic techniques such as simulated annealing and tabu search. These heuristic techniques are well known to give good approximate solutions to combinatorial and permutation problems [Dowsland 1995; Glover and Laguna 1995]. Simulated annealing investigates new and unknown areas in the domain of the solution of a problem while making use of already determined solution knowledge in computing the optimal solution. On the other hand, tabu search computes a number of adjacent solutions and then selects the solution that best improves the value of the objective function.

6.3 Single-crossing Axial Lines in Deformed Urban Grids

Hillier *et al.* [1983] defined axial lines as the longest and fewest lines to cross the adjacencies formed in a collection of convex polygons. A deformed urban grid is an urban grid in which the corridors are deformed rectangles and the intersection corridors are deformed squares [Sanders 2000]. Placing axial lines in deformed urban grids was found to be NP-complete [Wilkins and Sanders 2004]. As an NP-complete problem may not be solved efficiently for a given data input, one approach to benefit the end users of the system is to look at approaches where the problem could be solved efficiently. Penn *et al.* [1998, as cited in Turner *et al.* [2005]] redefined the notion of axial lines to be associated with the decomposition of the convex polygons and then analysed the placement of the axial lines to the pedestrian movement in which case they were found to be correlated.

A possible research question is to redefine the axial lines based on the property of single adjacency crossing and then determine if the axial lines can be placed to cross the adjacencies in the grid in polynomial time or if the problem remains NP-complete. If it is polynomially solvable, the next step will be to analyse whether the placement of the lines correlates with pedestrian movement which could be endorsed in automating the space syntax method if the result is positive.

6.4 Bipartite Independent Dominating Set

In Chapter 4 we have defined a graph theory problem, bipartite independent dominating set, which is applied to bipartite graphs only. It has been shown that BIDS is NP-complete by a transformation from the vertex cover problem. The bipartite independent dominating set has applications in packing and job allocation problems. For example, consider the following problem:

> *Given a collection of items some of which may not be placed together, how should the items be packed using minimum packing materials?*

The problem can be reformulated as a bipartite graph problem where the items are represented as vertices in one category and packing materials as vertices in another category such that each vertex which corresponds to the packing material is connected by an edge to the vertices which correspond to the items that could be packed together. The solution to the BIDS problem of the resulting graph gives a possible way of packing the items using minimum packing materials.

A possible research area will be to develop heuristics for BIDS and identifying configurations of bipartite graphs in which the BIDS problem is polynomially solvable.

6.5 Conclusion

The chapter presented some possible research areas identified during the course of the research. The future research detailed above is based on the concept that if a problem is found to be NP-complete, it is worthwhile to develop heuristic methods that give reasonable approximate solutions and special cases in which the problem is polynomially solvable rather than attempting to solve the exact solution of the problem. This is because in general solving the problem may require exponential time. As a result some of the research presented above deal with developing heuristics for ALP-SC-OLOR using different approaches.

The bipartite independent dominating set which was identified during the transformation from vertex cover to ALP-NSC-OLOR has applications in packing and job allocation problems. Consequently, it would be worthwhile to develop heuristic algorithms that give approximate solutions.

Chapter 7

Conclusion

7.1 Introduction

The aim of the dissertation, as stated in Chapter 1, was to explore whether single-crossing orthogonal axial lines can be placed to cross each adjacency between a collection of orthogonal rectangles (ALP-SC-OLOR) in polynomial time or if it is NP-complete. As research had not been done before on the single-crossing version of the axial line placement problem, the outcome of the research could give insight on the single-crossing of the general ALP problem and its subproblems.

In this chapter, a brief description of the summary of findings and conclusions of the dissertation will be presented along with the contribution of the research. Some limitations of the research are stated next. Lastly, the overall conclusion of the dissertation is provided.

7.2 Summary of Findings and Conclusions

The problem area considered in the dissertation fits into axial line placement which was discussed in Chapter 2 in more depth. Placing the fewest and longest lines to cross each adjacency of a convex map forms the axial map of the region of interest. The axial map is one of the four components that constitute the space syntax method. Space syntax, which comprises four essential phases, is a method of analysing and determining how complex a town plan is by studying its spatial structure. The four phases of space syntax are separating the "space" from "non-space" in an aerial map of a town and then approximating the grid to a polygon with holes; deriving the convex map of the town by partitioning the polygon into convex polygons; deriving the axial map of the town by placing the longest and fewest lines (called axial lines) to cross all adjacencies between the convex polygons; and integrating the local and global relationships of the "space". As the input to the space syntax method is

usually a large data set, its automation highly benefits the end users. Consequently, each of the stated phases need to have an efficient algorithm that gives the exact solution or heuristic algorithms that give approximate solutions.

The problem of placing the axial lines to cross the adjacencies between a collection of convex polygons (ALP-ALCP) was found to be NP-complete which implies that it can only be solved in reasonable time for small data sets or special configurations. Furthermore, the simplifications of the problem where the convex polygons are restricted to be orthogonal rectangles and the axial lines with orthogonal orientation (ALP-OLOR) and arbitrary orientation (ALP-ALOR) were found to be NP-complete. The dissertation considered a variation of the ALP-OLOR problem where the adjacencies between the collection of orthogonal rectangles are restricted to be crossed exactly once (ALP-SC-OLOR).

Section 2.6 dealt with decision problems and the theory of NP-completeness. NP-complete problems have a common property that if any of them can be solved in polynomial time, then all NP-complete problems can be solved in polynomial time which in turn implies $P = NP$. On the other hand, if any of the NP-complete problems is shown that it cannot be solved in polynomial time then all NP-complete problems cannot be solved in polynomial time which in turn implies $P \neq NP$. NP-complete problems are considered to be the 'hardest' in the NP class. Computing the exact solution of NP-complete problems requires exponential time, and hence in general these problems can only be solved in reasonable time for small data sets. Consequently, if a problem is NP-complete it is worthwhile to find special cases of the problem that can be solved in polynomial time or develop heuristic algorithms that approximate the exact solution.

In Section 2.7 review of geometric problems that have commonalities with ALP and previous work on ALP were presented. The section discussed the visibility and guarding problems along with their relationship to ALP. In placing the axial lines to cross the adjacencies between a collection of convex polygons, the adjacencies need to be partially visible to one another. In the case of ALP-SC-OLOR, the adjacencies between the collection of rectangles should be horizontally partially visible in order to place a line which crosses them. On the other hand, ALP can be considered as a variation of an art gallery problem where the gallery is protected by 'ray guards' – guards that can only see along a ray.

Chapter 3 presented the research question and the methodological strategy followed in answering the research question. The research question was to determine if ALP-SC-OLOR can be solved in polynomial time or if it is NP-complete. There were two general approaches to answer the research question: transforming ALP-SC-OLOR to a known problem in P in

polynomial time which would imply ALP-SC-OLOR is polynomially solvable or transforming a known NP-complete problem to ALP-SC-OLOR in polynomial time which would imply that ALP-SC-OLOR is NP-complete. Although the problem has some similarities with the general partitioning problem, which is a known NP-complete problem, the geometrical configuration of the rectangles could simplify ALP-SC-OLOR. The direction of the research once a decision was made on the problem were also outlined in the chapter.

In Chapter 4, a proof of NP-completeness of ALP-SC-OLOR was presented. The reduction was three way: from vertex cover for biconnected planar graphs to bipartite independent dominating set (BIDS), and then to orthogonal bipartite independent dominating set (OBIDS), and lastly to a normalized single-crossing orthogonal lines in orthogonal rectangles (ALP-NSC-OLOR) which is a special case of the ALP-SC-OLOR problem having the additional property that the lines are required to be maximal when they cross choice adjacencies. Choice adjacencies are those that could be crossed by two or more maximal lines. As is discussed briefly in Section 2.6, if a special case of a general problem is NP-complete, the general problem is also NP-complete. And hence, we concluded that placing single-crossing orthogonal axial lines in orthogonal rectangles (ALP-SC-OLOR) is NP-complete.

The running time of an NP-complete problem grows exponentially with the increase in the input size. As a result, a trade-off is necessary on a heuristic which runs faster but the result may not be optimal. Chapter 5 presented a heuristic algorithm for ALP-SC-OLOR based on a greedy method. The heuristic has three phases: determining vertical adjacencies between a collection of orthogonal rectangles, generating all possible single-crossing lines that cross the vertical adjacencies, and selecting the approximate minimum number of single-crossing lines to cross all the vertical adjacencies between the rectangles. The heuristic was coded and run on randomly generated orthogonal rectangles in order to compare the solution obtained against the exact (optimal) solution. The experimental results show that in most cases the solution obtained by the heuristic matches or only slightly differs from the optimal solution. However, this may not be the case as the number of rectangles increases. The heuristic is guaranteed to produced a solution within a k factor of the optimal solution, where k is the maximum number of adjacencies a line can cross in the collection of rectangles. The performance of the heuristic is expected to be close to the optimal as it assumes local improvement. Although the theoretical worst case time complexity of the algorithm is $O(n^4)$, the running time complexity of the heuristic for the tested data was $O(n^2)$. This is based on the line fit least square analysis. The gap between the theoretical and empirical analysis is due the theoretical analysis of the heuristic which was done based

on the potential number of iterations. As a result, the configuration of rectangles that exhibit the worst performance may not be constructured.

In general, if a problem is found to be NP-complete, it cannot be solved in polynomial time unless $P = NP$. As a result, research on NP-complete problems mainly focuses on good heuristic algorithm that give approximate solution close to the exact solution or special configurations of the problem that are polynomially solvable. The future work identified during the course of the research, presented in Chapter 6, is mainly applying different approaches of devising heuristics and considering polynomially solvable configurations of ALP-SC-OLOR and BIDS.

7.3 Contribution of the Research

ALP-OLOR is a special case of ALP-ALOR and ALP-ALCP. Consequently, as a result of NP-completeness of ALP-OLOR, one can conclude that ALP-ALOR and ALP-ALCP are also NP-complete.

Similarly, as discussed in Chapter 4, ALP-SC-OLOR has been found to be NP-complete. And hence the NP-completeness of ALP-SC-OLOR can be extended to the single-crossing versions of ALP-ALOR and ALP-ALCP. This is due to the fact that any polynomial time algorithm that solves a general problem should also solve the special case. The NP-completeness of the special case implies such algorithm may not exist unless $P = NP$. Based on the result of the research we can conclude that the single-crossing version of the general ALP problem is also NP-complete.

Another contribution of the dissertation is that some new NP-complete problems have been identified during the course of the research. This contributes to the open question of whether $P = NP$ or not. Besides, the newly identified NP-complete problems can also be used to prove the NP-completeness of other new problems.

7.4 Limitations of the Research

The limitations associated with the research are due to the small size of data used for the empirical analysis of the heuristic described in Chapter 5. This is due to the time constraint as more time was spent on the transformation of NP-completeness (that is, attempting to answer the research question). The limitations can be summarized as follows:

- The implementation of the exact solution algorithm of ALP-SC-OLOR used in Section 5.5 was based on exhaustive search. As a result, it was only possible to compute the

exact solution for a small number of rectangles. Based on the result obtained from the exact solution algorithm, it was concluded that the heuristic produced fairly close to the optimal solution. However, the heuristic may not produce solutions that are close to the optimal for a large number of rectangles.

- The data used during the empirical analysis of the heuristic of ALP-SC-OLOR were not enough to deduce the average case analysis. With larger data sets, the empirical complexity of the algorithm may not remain the same. And hence the conclusion of the empirical analysis is only for the tested data, and may not reflect the average case time complexity of the heuristic.

7.5 Overall Conclusion

The axial line placement problem, which was introduced in attempting to automate the space syntax method, is to place the fewest and longest lines to cross all adjacencies between a collection of convex polygons. As the problem was found to be NP-complete, another approach to approximate the problem is to consider adjacencies to be crossed only once. The aim of the research was to determine if axial lines can be placed to cross the adjacencies between a collection of rectangles when adjacencies are restricted to be crossed only once (ALP-SC-OLOR) in polynomial time or if it is NP-complete. The result obtained reveals that ALP-SC-OLOR is NP-complete despite the restriction imposed on the adjacencies. Moreover, the result implies that the single-crossing version of the general axial line placement problem is NP-complete. And hence, the single-crossing version may not be useful to approximate the general ALP problem. A heuristic that gives approximate solution to ALP-SC-OLOR based on a greedy method is also presented in this dissertation.

Appendix

List of Abbreviations

The following list contains abbreviations used in the document.

ALP	Axial line placement
ALP-ALCP	Axial line placement: Axial lines of arbitrary orientation in convex polygons
ALP-ALOR	Axial line placement: Axial lines of arbitrary orientation in orthogonal rectangles
ALP-OLOR	Axial line placement: Orthogonal axial lines in orthogonal rectangles
ALP-NSC-OLOR	Axial line placement: Normalized single-crossing orthogonal axial lines in orthogonal rectangles
ALP-SC-ALCP	Axial line placement: Single-crossing axial lines of arbitrary orientation in convex polygons
ALP-SC-ALOR	Axial line placement: Single-crossing axial lines of arbitrary orientation in orthogonal rectangles
ALP-SC-OLOR	Axial line placement: Single-crossing orthogonal axial lines in orthogonal rectangles
BIDS	Bipartite independent dominating set
OBIDS	Orthogonal bipartite independent dominating set

References

[Asano *et al.* 1986] Ta. Asano, Te. Asano, L. Guibas, L. Hershberger, and H. Imai. Visibility of disjoint polygons. *Algorithmica*, 1:49–63, 1986.

[Asano *et al.* 2000] T. Asano, S. Ghosh, and T. Shermer. Visibility in the plane. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 829–876. Elsevier Science B.V., 2000.

[Avis and Toussaint 1981] D. Avis and G. Toussaint. An optimal algorithm for determining the visibility of a polygon from an edge. *IEEE transactions on Computers*, C-30(12):910–914, 1981.

[Avis *et al.* 1986] D. Avis, T. Gum, and G. Toussaint. Visibility between two edges of a simple polygon. *The Visual Computer*, 2:342–357, 1986.

[Baase and Van Gelder 2000] S. Baase and A. Van Gelder. *Computer Algorithms: Introduction to Design and Analysis*. Addison-Wesley, third edition, 2000.

[Bose *et al.* 1992] P. Bose, A. Lubiw, and J. Munro. Efficient visibility queries in simple polygons. In *Proceedings of the 4th Canadian Conference on Computational Geometry*, pages 23–28, St John's, Newfound, Canada, 1992.

[Brassard and Bratley 1996] G. Brassard and P. Bratley. *Fundamentals of Algorithms*. Prentice Hall Inc., Englewood Cliffs, New Jersey, 1996.

[Castro *et al.* 2002] N. Castro, F. Cobos, J. Dana, Márquez, and M. Noy. Triangle-free planar graphs as segment intersection graphs. *Journal of Graph Algorithms and Applications*, 6(1):7–26, 2002.

[Chazelle and Dobkin 1979] B. Chazelle and D. Dobkin. Decomposing a polygon into its convex parts. In *Proceedings of the 11th Annual ACM Symposium on Theory of Computing*, pages 38–48, Atlanta, Georgia, 1979.

[Chvátal 1975] V. Chvátal. A combinatorial theorem in plane geometry. *Journal of Combinatorial Theory*, Series B(18):39–41, 1975.

[Cook 1971] S. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing*, pages 151–158, New York, 1971.

[Cormen *et al.* 2001] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, Cambridge, Massachusetts, second edition, 2001.

[Corneil and Perl 1984] D. Corneil and Y. Perl. Clustering and domination in perfect graphs. *Discrete Applied Mathematics*, 9:27–39, 1984.

[Czyzowicz *et al.* 1998] J. Czyzowicz, E. Kranakis, and J. Urrutia. A simple proof of the representation of bipartite planar graphs as the contact graphs or orthogonal straight line segments. *Information Processing Letters*, 66:125–126, 1998.

[Davis and Benedikt 1979] L. Davis and M. Benedikt. Computational models of space: Isovists and isovist fields. *Computer Graphics and Image Processing*, 11:49–72, 1979.

[de Berg *et al.* 2000] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin, 2000.

[de Fraysseix *et al.* 1991] H. de Fraysseix, P. de Mendez, and J. Pach. Representation of planar graphs by segments. *Intuitive Geometry*, 63:110–117, 1991.

[Dowsland 1995] K. Dowsland. Simulated annealing. In C. Reeves, editor, *Modern Heuristic Techniques for Combinatorial Problems*, pages 20–69. McGraw-Hill Book Company, 1995.

[du Plessis and Sanders 2000] N. du Plessis and I. Sanders. Partial edge visibility in chains of orthogonal rectangles. Technical Report TR-Wits-CS-2000-15, School of Computer Science, University of the Witwatersrand, Johannesburg, 2000. `http://www.wits.ac.za/research/pubs.html`.

[Fellows and Hoover 1991] M. Fellows and M. Hoover. Perfect domination. *Australian Journal of Combinatorics*, 3:141–150, 1991.

[Ferris 2003] K. Ferris. Heuristics for arbitrary axial line placement in orthogonal rectangles. Honours Research Report, School of Computer Science, University of the Witwatersrand, Johannesburg, 2003.

[Fisk 1978] S. Fisk. A short proof of Chvátal's watchman theorem. *Journal of Combinatorial Theory*, Series B(24):374, 1978.

[Fowler *et al.* 1981] R. Fowler, M. Paterson, and S. Tanimoto. Optimal packing and covering in the plane are NP-complete. *Information Processing Letters*, 12(3):133–137, 1981.

[Garey and Johnson 1979] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. W.H Freeman and Company, 41 Madison Ave., New York, 1979.

[Garey *et al.* 1976] M. Garey, D. Johnson, and L. Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1(3):237–267, 1976.

[Glover and Laguna 1995] F. Glover and M. Laguna. Tabu search. In C. Reeves, editor, *Modern Heuristic Techniques for Combinatorial Problems*, pages 70–150. McGraw-Hill Book Company, 1995.

[Golumbic 1980] M. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Academic, New York, 1980.

[Hagger 2005] L. Hagger. A greedy heuristic for axial line placement in collections of convex polygons. Master's dissertation, Faculty of Science, University of the Witwatersrand, Johannesburg, 2005.

[Harel 1987] D. Harel. *Algorithmics: The Spirit of Computing*. Addison-Wesley, first edition, 1987.

[Hartman *et al.* 1991] B. Hartman, I. Newman, and R. Ziv. On grid intersection graphs. *Discrete Mathematics*, 87:41–52, 1991.

[Hillier *et al.* 1983] B. Hillier, J. Hanson, J. Peponis, J. Hudson, and R. Burdett. Space syntax, a different urban perspective. *Architects' Journal*, 178:47–63, 1983.

[Hillier 1996] B. Hillier. *Space is the Machine: A Configurational Theory of Architecture*. Cambridge University Press, Cambridge, UK, 1996.

[Hopcroft and Ullman 1979] J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.

[Jiang 1998] B. Jiang. A space syntax approach to spatial cognition in urban environment. In *Workshop on Cognitive Models of Dynamic Phenomena and their Representations*, 1998. `http://www.sis.pitt.edu/~cogmap/ncgia/jiang.html`.

[Joe 1990] B. Joe. On the correctness of a linear-time visibility polygon algorithm. *International Journal of Computer Mathematics*, 32:155–172, 1990.

[Kahn *et al.* 1983] J. Kahn, M. Klawe, and D. Kleitman. Traditional galleries require fewer watchmen. *Algebraic and Discrete Methods*, 4(2):194–206, 1983.

[Kariv and Hakimi 1979] O. Kariv and S. Hakimi. An algorithmic approach to network location problems. *SIAM Journal of Applied Mathematics*, 37:513–538, 1979.

[Keil and Snoeyink 2002] J. Keil and J. Snoeyink. On the time bound for convex decomposition of simple polygons. *International Journal of Computational Geometry and Applications*, 12(3):181–192, 2002.

[Keil 1985] J. Keil. Decomposing a polygon into simpler components. *SIAM Journal of Computing*, 14(4):799–817, 1985.

[Keil 2000] J. Keil. Polygon decomposition. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 491–518. Elsevier Science B.V., 2000.

[Klostermeyer and Eschen 2000] W. Klostermeyer and E. Eschen. Perfect codes and independent dominating sets. *In Congressus Numerantium*, 142:7–28, 2000.

[Kruger and Sanders 2005] H. Kruger and I. Sanders. Orthogonal axial line placement in hole free collections of rectangles. In *Research for a changing world, Proceedings of SAICSIT2005*, pages 48–55, White River, South Africa, September 2005.

[Lien and Amato 2004] J. Lien and N. Amato. Approximate convex decomposition of polygons. In *Proceedings of the 20th Annual ACM Symposium on Computational Geometry*, Brooklyn, New York, June 2004.

[Lingas 1982] A. Lingas. The power of non-rectilinear holes. In *The 9th International Colloquium on Automata, Languages and Programming, Lecture Notes in Computer Science*, volume 140, pages 369–383. Springer-Verlag, 1982.

[Livingston and Stout 1990] M. Livingston and Q. Stout. Perfect dominating sets. *In Congressus Numerantium*, 79:187–203, 1990.

[Manber 1989] U. Manber. *Introduction to Algorithms: A Creative Approach.* Addison-Wesley, 1989.

[Michael and Pinciu 2003] T. Michael and V. Pinciu. Art gallery theorems for guarded guards. *Computational Geometry*, 26(3):247–258, 2003.

[Mills 1992] G. Mills. The spatial structure of ideology in informal settlements: A case study in southern Africa. *Building and Environment*, 27(1):13–21, 1992.

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

[O'Rourke 1995] J. O'Rourke. *Computational Geometry in C.* Cambridge University Press, Cambridge, 1995.

[O'Rourke 1997] J. O'Rourke. Visibility. In J. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, pages 467–479. CRC Press, Boca Raton, 1997.

[Penn *et al.* 1998] A. Penn, B. Hillier, D. Banister, and J. Xu. Configurational modelling of urban movement networks. In J. Ortuzar, D. Henshar, and S. Jara-Diaz, editors, *Travel Behavior Research: Updating the State of Play*, pages 362–399. Pergamon, Amsterdam, 1998.

[Penn 2003] A. Penn. Space syntax and spatial cognition or why the axial line? *Environment and Behavior*, 35(1):30–65, 2003.

[Sanders and Kenny 2001] I. Sanders and L-A. Kenny. Heuristics for placing non-orthogonal axial lines to cross the adjacencies between orthogonal rectangles. Technical Report TR-Wits-CS-2001-6, School of Computer Science, University of the Witwatersrand, Johannesburg, 2001. `http://www.wits.ac.za/research/pubs.html`.

[Sanders *et al.* 2000] I. Sanders, C. Watts, and A. Hall. Orthogonal axial line placement in chains and trees of rectangles. *South African Computer Journal*, 25:56–67, 2000.

[Sanders 1999] I. Sanders. Non-orthogonal ray guarding. In J. Snoeyink, editor, *Abstracts for the 11th Canadian Conference on Computational Geometry*, pages 80–83, University of British Columbia, Vancouver, 1999. `http://www.cs.ubc.ca/conferences/CCCG/elec_proc/elecproc.html`.

[Sanders 2000] I. Sanders. Placing axial lines in urban grids. *South African Computer Journal*, 26:145–153, 2000.

[Sanders 2002] I. Sanders. *The Axial Line Placement Problem.* PhD thesis, Faculty of Natural and Agricultural Science, University of Pretoria, 2002.

[Sanders 2003] I. Sanders. Determining the exact solution for placing axial lines to cross the adjacencies between orthogonal rectangles. Technical Report TR-Wits-CS-2003-4, School of Computer Science, University of the Witwatersrand, Johannesburg, 2003. `http://www.wits.ac.za/research/pubs.html`.

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

[Skiena 1998] S. Skiena. *The Algorithm Design Manual*. TELOS – the Electronic Library of Science, 1998.

[Tamassia and Tollis 1986] R. Tamassia and I. Tollis. A unified approach to visibility representations of planar graphs. *Discrete and Computational Geometry*, 1(4):321–341, 1986.

[Thomassen 1994] C. Thomassen. Grötzsch's 3-colour theorem and its counterparts for the torus and the projective plane. *Journal of Combinatorial Theory B*, 62:268–279, 1994.

[Thomson 2003] R. Thomson. Bending the axial line: smoothly continuous road centre-line segments as a basis for road network analysis. In *Proceedings of the 4th international space syntax symposium*, London, 2003.

[Turing 1936] A. Turing. On computable numbers, with an application to the Entscheidungsproblem. In *Proceedings of the London Mathematical Society*, volume 42 of *2*, pages 230–265, 1936.

[Turner *et al.* 2005] A. Turner, A. Penn, and B. Hillier. An algorithmic definition of the axial map. *Environment and Planning B: Planning and Design*, 32:425–444, 2005.

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

[Wilkins and Sanders 2004] D. Wilkins and I. Sanders. Axial line placement in deformed urban grids. *South African Computer Journal*, 33:10–23, 2004.

[Wilson and Watkins 1990] R. Wilson and J. Watkins. *Graphs: An Introductory Approach*. John Wiley & Sons, Inc., 1990.