The axial line placement problem

by

Ian Douglas Sanders

Submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy
in the Faculty of Natural and Agricultural Science
University of Pretoria
Pretoria
June 2002

Supervisor: Professor D. G. Kourie, Department of Computer Science

In memory of my father,

Douglas Rutherford Sanders,

26 April 1927 – 18 October 1997

Acknowledgements

A number of people contributed in many different ways to the completion of this work.

- I would like to thank Prof. Glenn Mills (at that time of the Department of Architecture, University of the Witwatersrand) for bringing the problem to my attention and encouraging me to work on it.

- The work on this thesis started under the supervision of Prof. David Lubinsky and Prof. Michael Sears of the University of the Witwatersrand and continued under the supervision of Prof. Derrick Kourie of the University of Pretoria. I would like to thank them for the effort that they put into guiding this work.

- The School of Computer Science at the University of the Witwatersrand assisted me with paying my fees. This year I was also given a lighter workload in order to finish the thesis. I am very grateful for this.

- Some of the work was submitted to conferences and journals and the anonymous referees made comments which assisted me in improving my work. I would thus like to thank these people, particularly the South African Computer Journal referees, for their helpful comments.

- I would also like to thank Joseph O'Rourke for his kind email advice on the naming of my problem and Theresé Biedl for pointing out some related literature to me.

- Steve Hedetniemi and the "algorithms seminar group" at Clemson University gave me lots of encouragement during my sabbatical there in 1996. My thanks to all involved.

- Sigrid Ewert and Derrick Kourie assisted me with the translation of the summary into Afrikaans. I thank them for their efforts and accept full responsibility for any errors.

- I would like to thank my research students who tackled offshoots of the work. Their participation added a lot to my enjoyment of the work and provided stimulation for me at important times.

- My students past, present and future are the reason that I felt that need to start off on this path in the first place. I thus owe them a vote of thanks too.
- My family have always been supportive of my academic endeavours and warrant a vote of thanks for their support in this latest effort as well.
- I owe a very sincere vote of thanks to my friends, and colleagues, Scott Hazelhurst and Conrad Mueller who were always prepared to offer their comments on various pieces of the work and to offer me encouragement and support.
- Finally, my friends offered encouragement and support (typically "Get the bloody thing done!") at various stages of the work. This applies particularly to Sheila Rock, Lex Holt (who also provided LaTeXsupport), Vashti Galpin, Sean Pyott and András Salamon. Thank you all very much for keeping me on track and helping me get past the frequent "thesis avoidance" episodes.

Summary

Visibility, guarding and polygon decomposition are problems in the field of computational geometry which have roots in real world applications. These problems have been the focus of much research over a number of years. This thesis introduces a new problem in the field – The Axial line Placement Problem – which has some commonalities with these other problems. The problem arises from a consideration of the computational issues that result from attempting to automate the space syntax method. Space syntax is used for describing, quantifying and interpreting the spatial patterns in urban designs by analysing the relationship between the space through which one can move (roads, parks, etc.) and the buildings in the urban layout. In particular, this thesis considers the problem of the placing the axial lines, defining paths along which someone can move, to cross the shared boundaries between the convex polygons which represent the space through which someone can move in the town.

A number of simplifications of the original problem are considered in this thesis. The first of these is the problem of placing the smallest number of orthogonal line segments (orthogonal axial lines) to cross the shared boundaries (adjacencies) in a collection of adjacent orthogonal rectangles. This problem is shown to be NP-Complete by a transformation from the vertex cover problem for planar graphs. A heuristic algorithm which produces an approximation to the general solution is then presented. In addition, special cases of collections of orthogonal rectangles which allow polynomial time solutions are described and algorithms to solve some of these special cases are presented.

The problem where the axial lines, that pass through the adjacencies between orthogonal rectangles, can have arbitrary orientation is then considered. This problem is also shown to be NP-Complete and once again heuristic approaches to solving the problem are considered. The problem of placing axial lines to cross the adjacencies between adjacent convex polygons is a more general case of the problem of placing axial lines of arbitrary orientation in orthogonal rectangles. The NP-Completeness proof can be extended to this problem as well.

The final stage of the thesis considers real world urban layouts. Many urban layouts are regular grids of roads. Such layouts can be modelled as general urban grids and this thesis shows that it is possible to find the minimal axial line cover in general urban grids in polynomial time. Some urban layouts are less regular and the idea of a deformed urban grid is introduced to model some of these layouts. A heuristic algorithm that finds a partition of a deformed urban grid in polynomial time is presented and it is conjectured that the axial map of a deformed urban grid can be found in polynomial time. The problem is still open for more general urban layouts which cannot be modelled by deformed urban grids.

The contribution of this thesis is that a number of new NP-Complete problems were identified and some new and interesting problems in the area of computational geometry have been introduced.

Opsomming

Sigbaarheid, waghou en veelhoek-dekomposisie is probleme in berekeningsmeetkunde wat hulle oorsprong in reële toepassings het. Die probleme is sedert jare die onderwerp van vele navorsing. Hierdie tesis voeg 'n nuwe probleem by die navorsingsgebied – die Asselyn Plasingsprobleem – wat sekere gemeenskaplikhede met bogenoemde probleme het. Laasgenoemde probleem vloei voort uit 'n beskouing van die berekeningskwessies wat ontstaan wanneer pogings aangewend word om die ruimte-sintaksis metode te outomatiseer. Ruimte-sintaksis word gebruik vir die beskrywing, kwantifisering en interpretasie van ruimtelike patrone in stedelike ontwerpe en wel deur die verwantskap tussen die ruimte waardeur 'n mens kan beweeg (paaie, parke, ens.) en die geboue in die stedelike uitleg te ontleed. Hierdie tesis beskou, in die besonder, die probleem van die plasing van asselyne op sodanig wyse dat hulle gedeelde grense tussen konvekse veelhoeke kruis, waarby the lyne paaie waarlang mens kan beweeg en die veelhoeke die ruimte waardeur mens deur die stad kan beweeg, verteenwoordig.

'n Aantal vereenvoudigings van die oorspronklike probleem word in hierdie tesis beskou. Die eerste hiervan is die probleem om die kleinste moontlike aantal ortogonale lynsegmente (ortogonale asselyne) op so 'n wyse te plaas dat hulle die gedeelde grense in 'n versameling van aangrensende ortogonale reghoeke kruis. Daar word gewys dat hierdie probleem NP-volledig is, deur 'n transformasie van die nodus-dekkingsprobleem ("vertex cover problem") vir planêre ("planar") grafieke na die problem uit te voer. 'n Heuristiese algoritme wat 'n benaderde oplossing tot die algemene probleem bied, word dan voorgestel. Addisioneel word spesiale gevalle van versamelings van ortogonale reghoeke wat polinomiese tyd oplossings toelaat beskryf. Algoritmes wat sekere van hierdie spesiale gevalle oplos word aangebied.

Daarna word die probleem beskou waarvolgens asselyne wat deur aangrensende ortogonale reghoeke gaan, arbitrêre orientasie mag hê. Hierdie probleem word ook as NP-volledig bewys en weereens word heuristieke benaderings om die probleem op te los, beskou. Die probleem om asselyne te plaas sodanig dat hulle grense tussen aangrensende konvekse veelhoeke te kruis is 'n veralgemening van die probleem om asselyne van arbitrêre orientasie in reghoeke te plaas. Die NP-volledigheidsbewys kan ook na die meer algemene probleem uitgebrei word.

Die finale fase van die tesis beskou die uitleg van reële stede. In die geval van baie stede is die uitleg 'n reëlmatige rooster van paaie. So 'n uitleg kan as 'n algemene stedelike rooster gemodeleer word en hierdie tesis toon aan dat dit moontlik is om die minimum asselyn dekking van sulke roosters in polinomiese tyd te bepaal. Sekere stede se uitleg is minder reëlmatig en die konsep van 'n verwronge stedelike rooster word voorgestel om sommige daarvan te modeleer. 'n Heuristiese algoritme wat in polinomiese tyd 'n partisie van 'n verwronge stedelike rooster vind, word aangebied. Daar word gepostuleer dat die assekaart van 'n verwronge stedelike rooster in polinomiese tyd gevind kan word. Die probleem vir stedelike uitlegte wat nie deur verwronge stedelike roosters gemodeleer kan word nie, bly egter steeds onopgelos.

Die bydrae van hierdie tesis is dat 'n aantal nuwe NP-volledige probleme geïdentifiseer is, en sommige nuwe en interessante probleme tot die gebied van berekeningsmeetkunde toegevoeg is.

Preface

Some of the work in this thesis has been previously published.

- The NP-Completeness proof in Chapter 4 was published in the South African Computer Journal [Sanders *et al.*, 1999].

- The axial line placement problem in chains and trees of orthogonal rectangles presented in Chapter 4 was also published in the South African Computer Journal [Sanders *et al.*, 2000b]. Much of the work for this paper was done under my supervision by two Computer Science Honours students, Claire Watts and Andrew Hall, as the research component of their degrees.

- The axial line placement problem in urban grids and deformed urban grids in Chapter 7 was accepted for the South African Institute of Computer Scientists and Information Technologists 2000 research symposium as a full research paper. It was published in a special issue of the South African Computer Journal [Sanders, 2000].

- The NP-Completeness proof in Chapter 5 was presented at the 11th Canadian Conference on Computational Geometry – an extended abstract was published in a collection of such abstracts and the full paper is available electronically [Sanders, 1999].

- The heuristics proposed in Chapter 5 were presented at the 13th Canadian Conference on Computational Geometry – an extended abstract was published in a collection of such abstracts [Sanders and Kenny, 2001a]. The full paper is available as a technical report in the School of Computer Science at the University of the Witwatersrand [Sanders and Kenny, 2001b]. Some of the work for this paper was done under my supervision by a Computer Science Honours student, Leigh-Ann Kenny, as the research component of her degree.

- Various presentations were made of "work in progress" at the Southern African Computer Lecturers' Association annual conferences and the South African

Institute of Computer Scientists and Information Technologists annual research symposia [Sanders *et al.*, 1995, 1997; Sanders, 1998a,b; Bilbrough and Sanders, 1998].

- Some work has also been published as technical reports in the Department of Computer Science at the University of the Witwatersrand [Watts and Sanders, 1997; Sanders *et al.*, 2000a; du Plessis and Sanders, 2000; Sanders and Kenny, 2001b].

- My Honours students over the years have worked on some small parts of the overall research [Watts, 1997; Zarganakis, 1997; Soares, 1997; Wilson, 1997; Bilbrough, 1998; du Plessis, 1999; Hall, 1999; Ashman, 1999; Bukovska, 2000; Kenny, 2000; Soltész, 2000; Konidaris, 2001; Scott-Dawkins, 2001; Phillips, 2001; Hagger, 2001].

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background to the problem

The advent of computers has changed how people work and how jobs are done in many professions. This is because in many circumstances computers can be used to handle the routine (and often boring) aspects of some jobs, freeing the professional to devote her/his time to the more mentally stimulating/challenging aspects of the work. Baase [1997] discusses many of the benefits and problems of the computer revolution in her book *A Gift of Fire*.

The general area of architecture (designing houses or office buildings, town planning, etc.) is an area where computers could offer benefits to the professional. In some other areas of design this is already happening. For example, shape grammars can be used to understand designs and to create new designs [Koning and Eizenberg, 1981; Mitchell, 1990; Herbert *et al.*, 1994] and computers have been used to automatically generate floor plans for houses [Eastman, 1972; Rinsma *et al.*, 1990]. Town planning or urban design is an area where the professional uses large data sets in order to be able to understand existing town plans and also to be able to improve existing layouts and to design new layouts. Using computers to assist the town planners could free the town planner from some aspects of the routine jobs and thus allow her/him to concentrate on the design phase. This automation raises a number of societal and ethical issues, which will not be discussed here (see Baase [1997] for more detail), but also raises a number of problems that are of interest to computer scientists. This thesis considers some of the interesting computational problems that arise from the possible automation of a design task – that of understanding and designing urban layouts.

Hillier *et al.* [1983] proposed a method, which they called *space syntax*, to describe and analyse patterns of architectural space both at the building and urban level. Hillier [1996] discusses the use of the method in more detail (see particularly pages 153 to 181). The idea is that with an objective and precise method of description it is possible to investigate how well environments work, rigorously relating social variables to architectural forms. They believe that space syntax can help architects to understand the interaction between space and society and thus can be used as a tool to understand why urban areas have developed as they have and also to design new urban layouts.

An architect or town planner would apply the space syntax method ([Mills, 1992]) to a town or city in 4 main steps.

1. Studying the town plans or an aerial photograph of the town and separating out the "space" (roads, parks, etc.) from the "non-space" (buildings, car parks, schools, etc.). The result of this step would be a "deformed grid" which is the town plan reduced to a number of polygons representing the "non-space" separated by "space" – it is the space that is the real object of interest.

2. Creating a convex map of the area. This convex map is made up of the smallest number of "largest" convex spaces that cover all of the space in the area being analysed. It is a partition of the space into the minimum number of convex polygons. A convex space gives some local information about the area in the sense that every point in the convex space is directly visible and directly accessible to every other point in that convex space.

3. Creating an axial map of the area from the convex map. The axial map is made up of the fewest and longest straight line segments (axial lines) that cover the town, crossing through the convex polygons that make up the convex map, and offers a globalising perspective that takes into account how far one can see (or walk) in the town.

4. Combining the information from the convex map and the axial map to produce an integration factor for the town/city. The integration factor, which is the final result of the analysis, gives an idea of how easy it is to move about in the town.

At present most of this work is done manually by the architect/town planner using pencil and tracing paper over the aerial photograph or map. This seems to be an application where computers can be used to assist, or (in some aspects of the work) replace, the architect in performing the routine tasks required to prepare the data for interpretation. Some tasks are boring and can be more efficiently solved by computers. Some tasks are computationally difficult and are unlikely to be performed optimally by humans so computers could be utilised to provide improved solutions. In order to determine whether the processing can be done automatically it is important to consider the tasks that are performed by the architect where there is potential for automation. The next section of this thesis considers the tasks performed in applying the space syntax analysis method, identifies where there is the possibility for automation and also highlights some of the areas of current and possible future research interest for computer scientists. Note that the focus of this thesis is on the computational problems that could arise with automation. The issue of whether space syntax is a good way of understanding and doing design is not within the scope of the research.

1.2 The scope for automation

The first stage in the process of applying space syntax is the separating of space from non-space in the town plan or aerial photograph. This problem has already been the subject of much research in the field of image processing over a number of years. The general approach here would be to take an aerial photograph, landsat image, or any other image (in digital form) over the inhabited areas and to automatically separate space (roads, parks, etc.) from non-space (buildings, etc.) using image segmentation techniques. Gonzalez and Wintz [1987], Gonzalez and Woods [1992], Castleman [1996], Jähne [1997] or Russ [1999] offer good introductions into the field of image processing and more specifically, general image segmentation techniques. Various authors [Huertas and Nevatia, 1988; Liow and Pavlidis, 1990; Ton *et al.*, 1991; Stilla *et al.*, 1996; Geman and Jedynak, 1996; Barzohar and Cooper, 1996; Levitt and Dwolatzky, 1999] have considered the issues of separating roads or buildings from the background in digital images.

An additional problem that could occur after the separation/segmentation phase is that the segmented areas are unlikely to have smooth boundaries. In order to make these areas suitable for further processing it is necessary to be able to "accurately" and "efficiently" represent each area by a bounding polygon. In essence the area should be described by a bounding polygon that matches the boundary as closely as possible while minimising the number of vertices and edges required to define the polygon. Research into this problem has been going on for a number of years ([Ramer, 1972; Pavlidis and Horowitz, 1974; Sarkar, 1993; Perez and Vidal, 1994; Ruskin, 1997; Zhu and Seneviratne, 1997]). The result of applying segmentation and polygon approximation to instances of the problem would be the "deformed grid" – a number of polygons representing the non-space in the area with the space between as the real area of interest. A bounding polygon can then be put around this area of interest. This deformed grid would then be represented in an appropriate fashion for future processing – finding the convex map and the axial map.

It does, therefore, seem that the initial phase of the space syntax method – separating space and non-space – can benefit from automation. Computer programs can be used to generate a deformed grid from an aerial photograph or town plan. If the town planner feels this automatically deformed grid is acceptable then he/she can use it as is in the next phase of the process. Alternatively he/she could use it as a starting point for generating a deformed grid that they feel is acceptable to use in the next phase of the process. Although many of the problems in this area are well understood there are still some open questions and unresolved issues and there is thus still scope for further research in this area.

Once the deformed grid has been found, the next phase of the work is to find the convex map of the area. The convex map is defined as being the minimum number of non-overlapping convex spaces (convex polygons) that cover the space in the deformed grid. Figure 1.1 shows a section extracted from a map of the Johannesburg region. This is essentially the deformed grid of the region – the space (light coloured) and non-space (darker coloured) in the urban layout represented by polygons. Note that the polygons representing non-space (darker coloured) are not of direct interest in applying space syntax. These non-space polygons can be of arbitrary shape. The polygon[s] representing the space in the deformed grid must be partitioned into convex polygons. Figure 1.2 shows a close up of part of the original region that will be used to give an idea of the application of space syntax. Essentially the problem is that of covering or partitioning a general polygon (the boundary of the area under consideration) with holes (the non-space parts of the area under consideration) by the *minimum* number of convex polygons. The general covering problem (where polygons are allowed to overlap) and the general partitioning problem (where polygons may not overlap) have been quite well researched. In addition partitioning and covering of special classes of polygons has also received a lot of interest. Many of the problems in this category have been shown to be computationally intensive and some have been shown to be NP-Hard. Chapter 2 discusses these problems in more detail. The special case of covering or partitioning a polygon with holes, that represents a town plan, and therefore has special constraints, has not been studied but the complexity of some town plans (as shown in Figure 1.1) would seem to indicate that this problem is also inherently hard. However, even if the problem itself is inherently hard – it takes a long time to find the minimum solution – it is likely that approximations to the optimal solution would be sufficient for the town planner to continue meaningfully with the later phases of the analysis. It might also be the case that some town plans can be modeled by simplifications to the general problem and that these could be solved exactly in a reasonable time. It thus seems likely that this phase of the process could also benefit from automation. Interesting areas of research related to this phase of the process are to study approximation algorithms and special cases of the problem that can be solved exactly in a reasonable time.

From the convex map a town planner can generate the axial map over the area. This axial map is defined as the smallest number of axial lines that will cross all of the shared boundaries between the convex spaces in the convex map. Figures 1.3

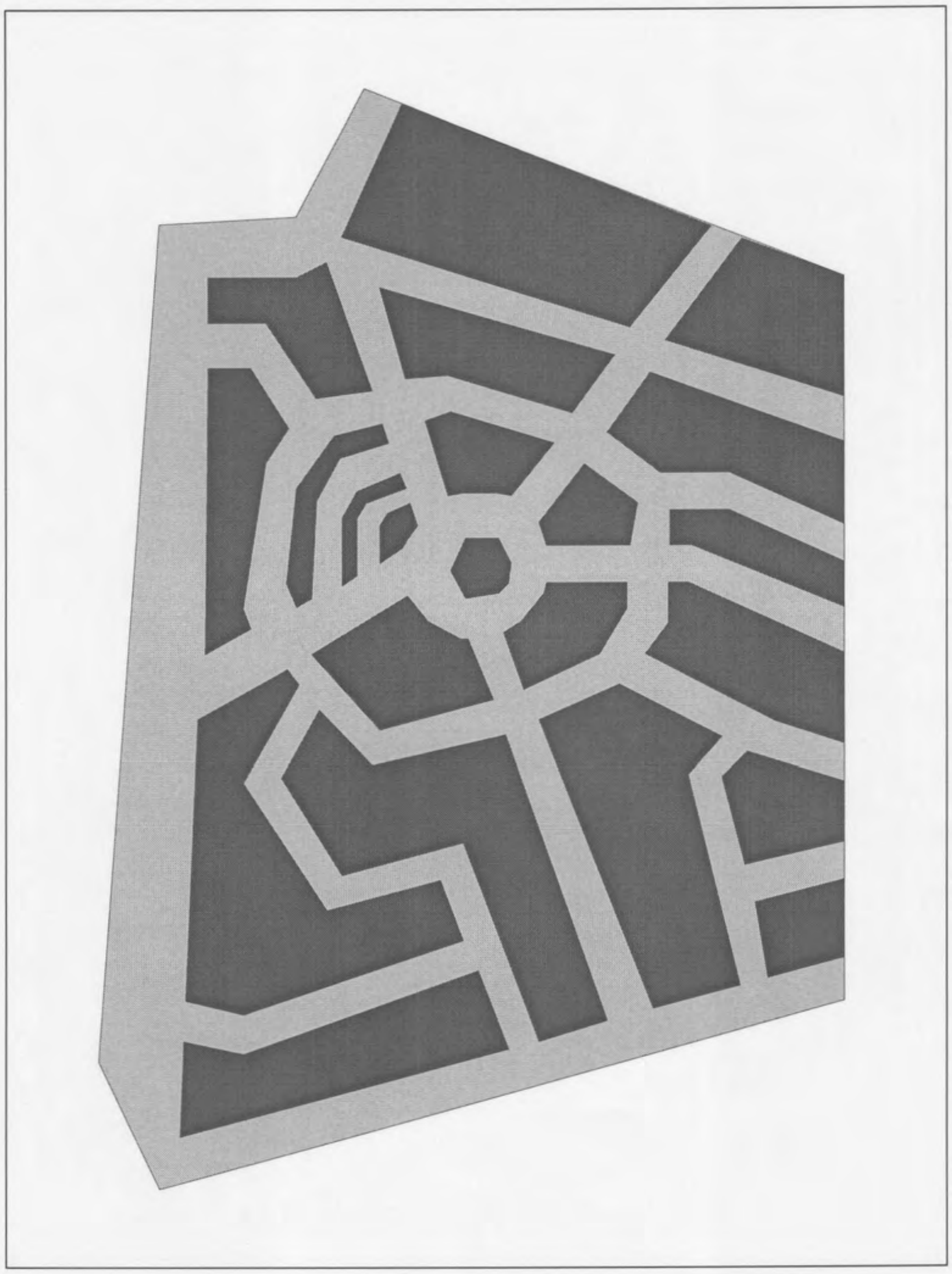
Figure 1.1: An example town plan

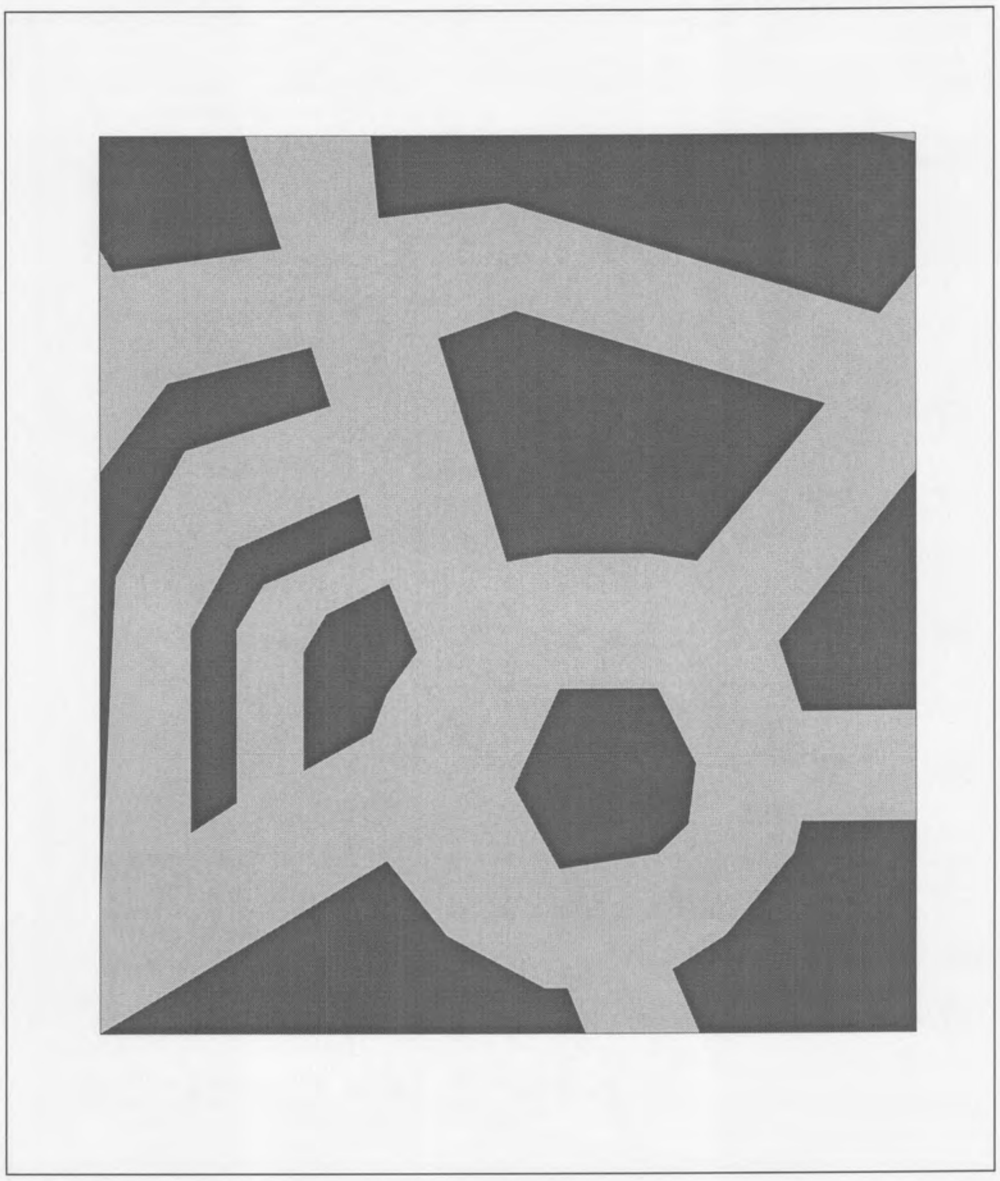

Figure 1.2: A zoomed view of a portion of the example town plan

and 1.4 show how the enlarged portion of the urban layout from Figure 1.1 could be covered by convex polygons and the shared boundaries between the convex polygons crossed by axial lines. It is interesting to note that in the two figures the same number of convex polygons are required to cover the spaces in the layout but that the convex polygons are different. An effect of the different sets of convex polygons in the two partitions is that different numbers of axial lines are required to cross the shared boundaries in the two figures (6 axial lines are required in Figure 1.3 and 7 in Figure 1.4). This example illustrates the inherent difficulty of the problem. It is known to be difficult to find the minimum number of convex polygons to partition a polygon and it seems that is it also difficult to partition the space in a town plan. Then to find the minimum number of axial lines to cross all of the shared boundaries it is necessary to consider all of the combinations of the minimum number of convex polygons and to find the minimum number of axial lines for each of those configurations.

The problem of placing the minimum number of axial lines for any configuration of convex polygons is an area that has not been directly researched in the past. In fact, it was first studied by the author of this thesis. There is, however, an abundance of research in closely related areas – guarding and visibility problems. In guarding problems the aim is to place the minimum number of guards (with different attributes) so that the guards can see the entire area of polygons of different forms. Visibility problems are very similar but focus on determining how much of a polygon can be seen from some point or points inside the polygon. These problems are discussed in detail in Chapter 2. The literature in these areas indicates that finding the minimum number of axial lines could itself be an inherently difficult problem. The problem of finding the axial map from some urban layout (given the convex map) is the major emphasis of this thesis. A number of variations the problem are studied and even more questions are raised. The results presented in this thesis do, however, indicate that this problem can be solved sufficiently well to be of use to the town planner or urban designer.

Having determined the convex map and the axial map the architect would perform the space syntax analysis based on these parameters. This phase of the process is already automated [Hillier, 1996]. This is done essentially using graph algorithms. An area of interest is in determining whether the graph algorithms used here – for example, shortest path algorithms – could be modified for the specific application. In addition, issues related to space usage and representation could be attractive areas for research.

The discussion above makes it clear that the aim of automatically applying space syntax to a town plan or aerial photograph of a town poses many interesting research areas for computer scientists. The range of research areas is too broad to be covered as a single Ph.D. thesis and thus this thesis concentrates on a small part of the overall problem. The next section gives an overview of the research undertaken for

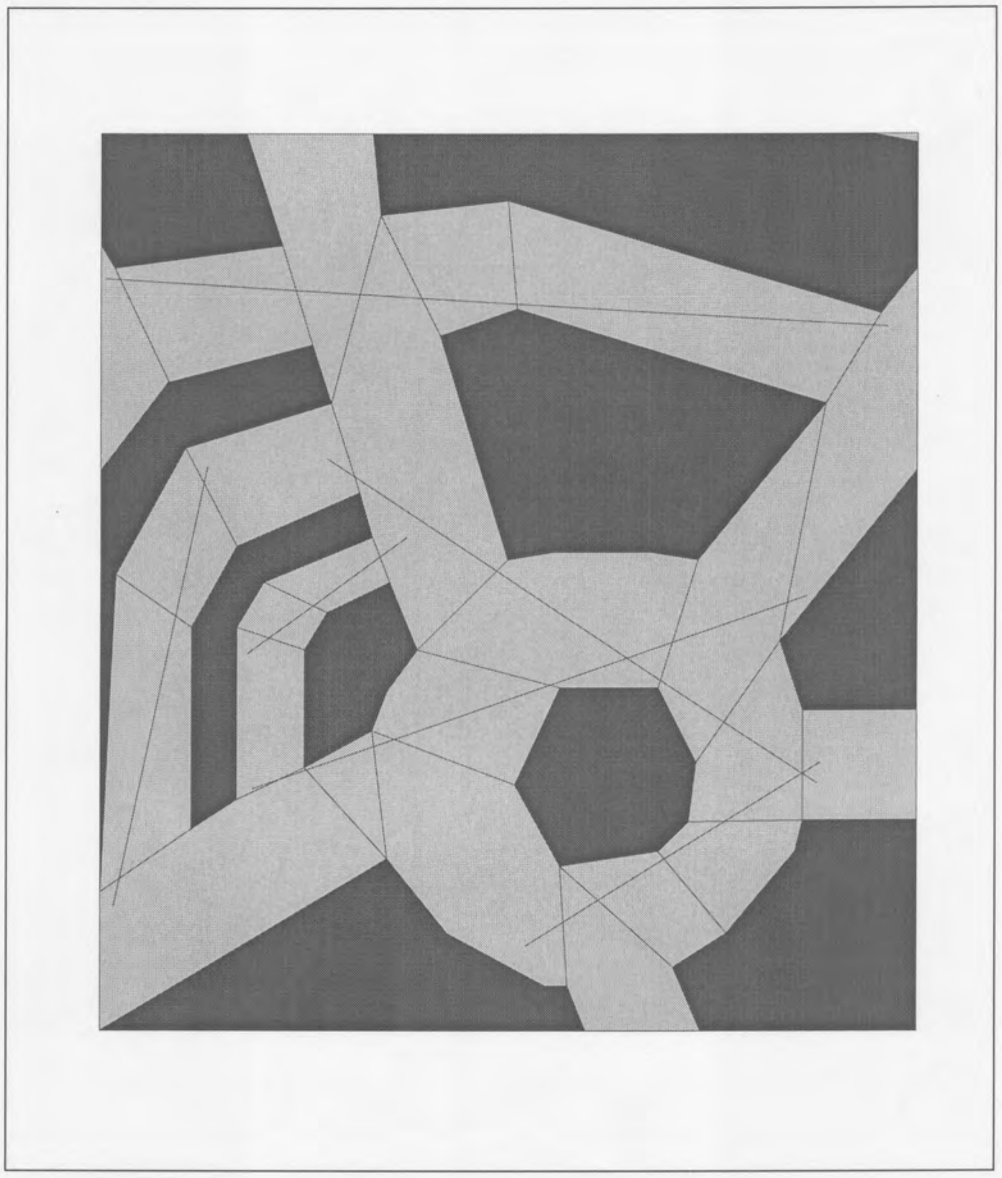

Figure 1.3: A convex map of the enlarged version of the town plan with 24 convex spaces and its associated axial map with 6 axial lines

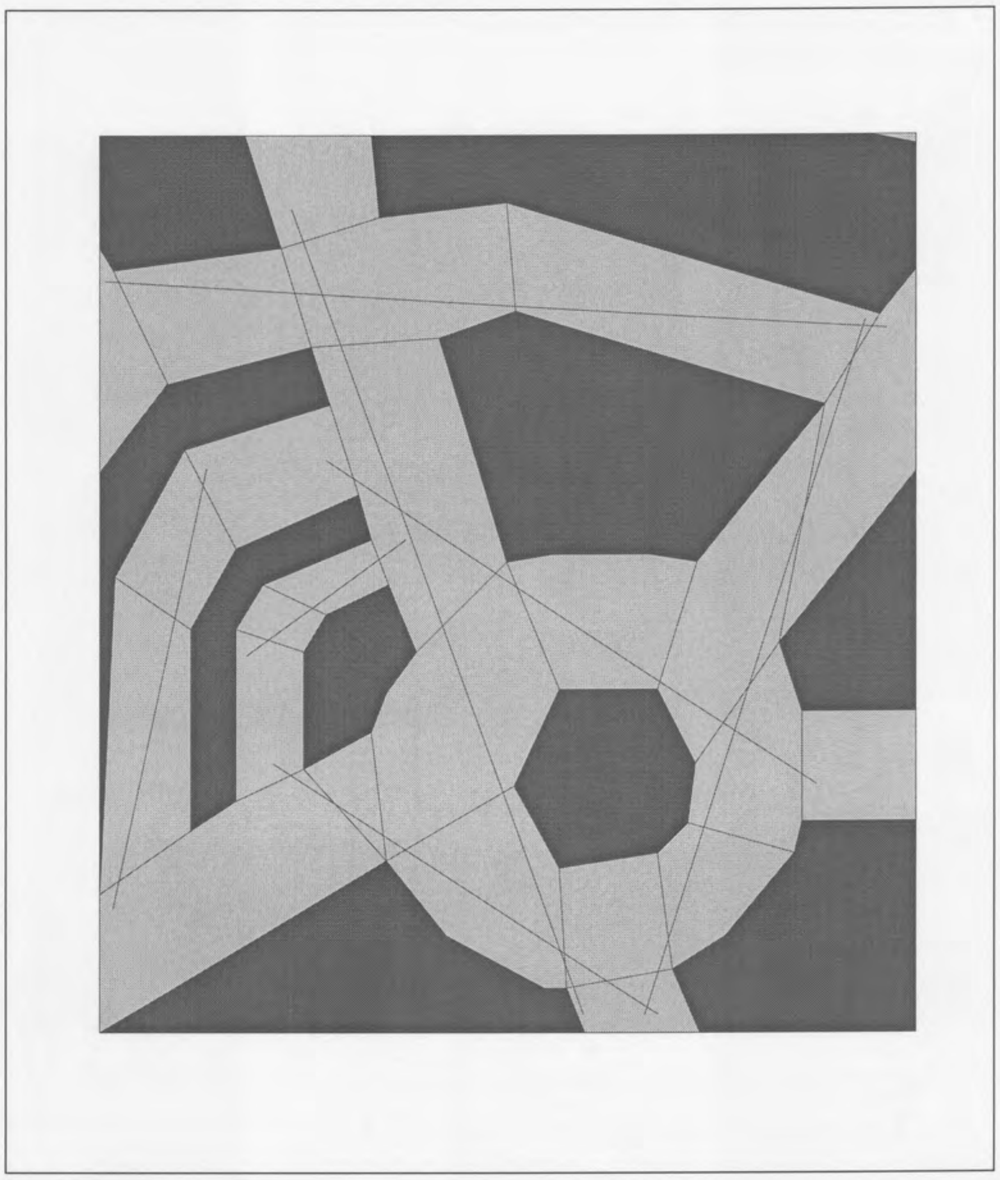

Figure 1.4: A convex map of the enlarged version of the town plan also with 24 convex spaces and its associated axial map with 7 axial lines

the thesis.

1.3 The research focus of the thesis

As discussed in the previous section of this thesis the potential automation of space syntax gives rise to many areas of research. These areas include image processing, computational geometry and algorithms. The range of research questions that could be addressed is very wide and it was thus necessary to concentrate on a subset of the problems. For this reason, the problems of separating space from non-space, determining the convex map and the final analysis stage with its associated algorithms were not considered as part of this research. The decision was made to focus the research for this Ph.D. on the problem of finding the axial lines that cross all of the shared boundaries between the convex polygons in the convex map, that is finding the axial map for a given layout. In the remainder of this thesis the problem will be called the *Axial Line Placement* problem or ***ALP***. This problem on its own is still very big and could not be solved entirely. The research considered a number of simplifications to and variations on ***ALP*** – using simpler types of convex polygons and introducing constraints aimed at making the problems easier to solved. Some new contributions have been made as a result of this research and some progress has been made towards finding solutions to the general problem. There are, however, still a number of unsolved problems and open questions. The next section gives an overview of the thesis, in particular indicating some of the variations of the *Axial Line Placement Problem* that have been considered as part of the research for this thesis.

1.4 Overview of the remainder of the thesis

Many researchers over the years have concentrated on the problems of guarding and visibility in polygons of various shapes. Much attention has also been focussed on the problems of decomposing polygons into smaller more easily handled components. As mentioned in Section 1.3 above these problems have relevance to the current research because they can provide insight into solving ***ALP*** . Chapter 2 of this thesis presents a detailed literature survey of the work in these areas.

Chapter 3 expands somewhat on the range of research possibilities that arise from the problem of finding the convex and axial maps for town plans (urban layouts). This chapter also presents some simplifications or generalisations of the problem that are in themselves interesting problems for further study. The chapter concludes by discussing the specific problems that were tackled as part of this thesis.

The new results, in the form of proofs, algorithms, etc. that arise from tackling these problems constitute the research contribution of the thesis and are discussed in ensuing chapters of the document.

Chapter 4 discusses one of the "simplifications" presented in Chapter 3 in depth – the placement of orthogonal axial lines to cross the shared boundaries between rectangles in a configuration of orthogonal rectangles. The problem is presented and shown to be NP-Complete. A heuristic algorithm that appears to give "acceptable" approximations is tested and compared to a heuristic algorithm that is known to give a solution no worse than twice an optimal solution. Special cases where the problem can be solved exactly in polynomial time are also discussed.

In Chapter 5 the problem discussed in Chapter 4 is changed slightly to allow the axial lines that cross the shared boundaries between rectangles in a configuration of orthogonal rectangles to have arbitrary orientation. This problem is also shown to be NP-Complete. The chapter also introduces some ideas for heuristics for finding acceptable approximations to the exact solution in this case.

The restrictions on problem are relaxed even further in Chapter 6 that considers the axial line placement problem for arbitrary convex polygons (dropping the requirement of orthogonal rectangles). This problem is a generalisation of the problem described in Chapter 5 and thus the NP-Completeness of this problem can be easily proved. The variation of the problem discussed in this chapter is the most general case of ***ALP*** and thus ***ALP*** is NP-Complete.

In Chapter 7 of the thesis, the original problem – placing axial lines to cross the shared boundaries of the convex polygons in the convex map of an urban layout or town plan – is considered. In this thesis the problem of finding the convex map of the area under consideration is not studied in great depth because it has already been shown to be NP-Hard. However in this chapter it is studied as a side issue in the matter of finding the axial map of the area. Here it is shown that although the general problem of partitioning a polygon with holes is NP-hard, there are some instances of the problem that can be solved in polynomial time. In particular it is shown that if the town plan is regular then the convex map can be found in polynomial time and so can the axial map. The chapter also addresses the matter of town plans that are not regular and argues that finding the convex map of such layouts is likely to be NP-Hard but that finding the axial map of such layouts might not be as difficult.

The final chapters of this thesis are devoted to future work and concluding remarks. Chapter 8 discusses some of the problems that have not been tackled in this thesis and Chapter 9 restates the results and conclusions drawn in this thesis. There are still many open questions and unproven conjectures but this thesis has made a significant contribution in tackling some new problems and obtaining some new results.

Chapter 2

Background

2.1 Introduction

As discussed in the introduction to this thesis (Chapter 1) the potential automation of space syntax gives rise to a wide range of research questions and it was thus necessary to concentrate on a subset of the problems. For this reason, the problems of separating space from non-space, determining the convex map and the final analysis stage with its associated algorithms were not considered as part of this research. The decision was made to focus the research for this PhD on ***ALP***, the problem of finding the axial lines that cross all of the shared boundaries between the convex polygons in the convex map. This problem has much in common with other well studied problems in the field of computational geometry.

The most obvious commonality is in the idea of *visibility*. The intent of the axial map of some urban layout is that it offers a globalising perspective that takes into account how far one can *see* (or walk) in the town. In particular placing an axial line to cross the adjacencies between a number of adjacent convex polygons can be thought of as determining a line of sight from some point to another in a given polygon. Visibility in polygons is an area of computational geometry which has received much attention in the last two decades [Asano *et al.*, 1999]. The essential question is this work is: “Can some point in the polygon ‘see’ some other point in the polygon?” However, other visibility questions can also be posed. Included in these are vertex-vertex visibility, vertex-edge visibility, edge-edge visibility, etc.

The problem of *guarding* a polygon is very closely related to visibility in polygons. In fact, all guarding problems are essentially visibility problems. The first “guarding problem” came about as a problem posed by Victor Klee in response to a request by Vasek Chvátal [O’Rourke, 1987]. The original problem was to determine the minimum number of guards necessary to cover the interior of an n-wall art gallery. Many variations on this problem can now be found in the literature. (See the monograph by O’Rourke [1987], the survey papers by Shermer [1992] and

Urrutia [1999], and the summaries of results by O'Rourke [1997] and Suri [1997].) These variations include vertex guards, edge guards, point guards, periscope guards and prison guards.

Another area of computational geometry which in some ways is similar to both visibility and guarding problems is *polygon decomposition* [Keil, 1999]. The focus here is in decomposing given polygons into smaller more easily manageable parts. Polygon decomposition problems occur frequently in such areas as pattern recognition, image processing, computer graphics and VLSI. In polygon decomposition problems the aim is to break the polygon down into constituent parts which can be more easily processed. Polygon decomposition is categorised in two different ways – covering and partitioning. The basic covering problem is to find the minimum number of polygons, with some predefined characteristics, to cover the complete area of an enclosing polygon. This problem has some commonality with the guarding problem – covering a polygon with the minimum number of polygons of some specified type is equivalent to the placement of the minimum number of guards of some specified type so that each point inside the polygon is visible to some guard. The partitioning problem is the same as the covering problem except that the polygons into which the enclosing polygon is decomposed are not permitted to overlap. The phase of the space syntax analysis method which produces a convex map of a given urban layout is clearly a polygon decomposition problem. The town plan is a polygon with holes and the convex map is a minimum partition of that polygon.

The focus of this chapter is to explore the commonalties (and some cases the differences) between ***ALP*** and the work which has been done in the areas of visibility, guarding and polygon decomposition. This is accomplished by looking at the results which have been published in the other areas and relating these to ***ALP***. Before discussing the previous results, however, it is worthwhile introducing some general terms which are used in the literature. Section 2.2 introduces terms which are important for understanding the research in these areas which is discussed. Other more specific terms are introduced only when required.

In addition, a number of problems which are discussed in this chapter have been proved to be NP-Complete or NP-Hard. In addition, the new results which are presented in this thesis rely on a number of NP-Completeness proofs. It thus seems appropriate that the results concerning NP-Complete and NP-Hard problems are summarised here (Section 2.3). Because of the size and the complexity of the topic the presentation here focusses on the more practical aspects of the use and application of the theory.

Once this general background has been covered, the chapter addresses the more directly related background literature and discusses its relevance to ***ALP*** (Section 2.4). The chapter concludes (Section 2.5) by reiterating that ***ALP*** is different from the other problems discussed. ***ALP*** is thus a previously unstudied problem and is worth being investigated. This discussion leads on to the posing of the research questions in Chapter 3.

2.2 Terminology

The general definitions in the fields of computational geometry and graph theory appear below to make the material covered in this chapter more understandable. In addition, many of these definitions are also used in later chapters of the thesis. Most of the definitions come from Manber [1988], O'Rourke [1987], and Shermer [1992] but can be found in other sources as well. A reader who is familiar with the area could skip this section of the thesis.

- A *point* p is represented by a pair of coordinates (x, y) in euclidean space.
- A *line* is represented by two points p and q (which can be any two distinct points on the line) and is denoted $-pq-$.
- A line is *orthogonal*, or *orthogonally aligned*, if it is parallel to one of the Cartesian axes.
- A *line segment* is represented by a pair of points p and q where the points are the endpoints of the line segment and is denoted by pq.
- A line segment is *orthogonal*, or *orthogonally aligned*, if it is parallel to one of the Cartesian axes.
- A *ray* is represented by a pair of points p and q where one point is the endpoint of the ray and the other point is any other distinct point on the ray.
- A ray is *orthogonal*, or *orthogonally aligned*, if it is parallel to one of the Cartesian axes.
- A *path* is a sequence of points $p_1, p_2, \ldots, p_n$ and the line segments joining them.
- The line segments in a path are called *edges*.
- A *closed path* is a path whose last point is the same as its first point.
- A closed path is also called a *polygon*.
- The points defining the polygon are called the *vertices* of the polygon.
- A *polygon P* can also be defined as a collection of n vertices, $v_1, v_2, \ldots, v_n$, and n edges, $v_1v_2, v_2v_3, \ldots, v_{n-1}v_n, v_nv_1$.

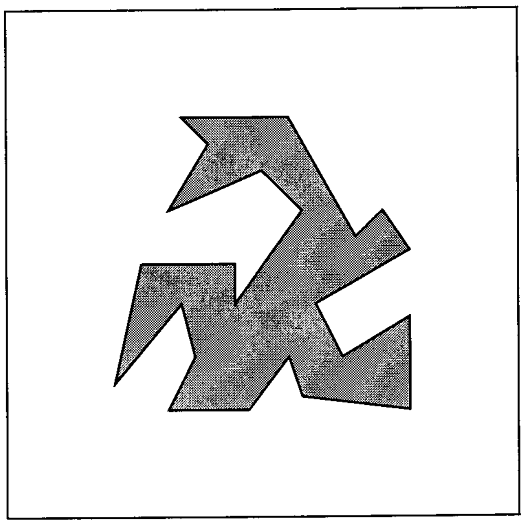

Figure 2.1: A simple polygon (Shermer [1992])

- A *simple polygon* is a polygon where no two non-consecutive edges intersect. Figure 2.1 shows an example of a simple polygon.
- The set of points in the plane enclosed by a simple polygon forms the *interior* of the polygon.
- The set of points on the polygon itself forms the *boundary* of the polygon.
- The set of points surrounding the polygon forms its *exterior*.
- A *hole* in a simple polygon P is another polygon H enclosed by the boundary of P.
- If a simple polygon P contains holes then P is said to be *multiply connected*; if P contains no holes then it is said to be *simply connected*.
- A simple polygon is *convex* if, given any two points on its boundary or in its interior, all points on the line segment joining them are contained in the polygon's boundary or interior.
- A polygon P is a *star* or *star-shaped* if there is some point x in the polygon from which every other point in the polygon can be seen.
- A *kernel* in a star polygon is a point x in the polygon from which every other point in the polygon can be seen.
- A *comb* polygon is as shown in Figure 2.3. Comb polygons exist for any number of vertices which is a multiple of 3.

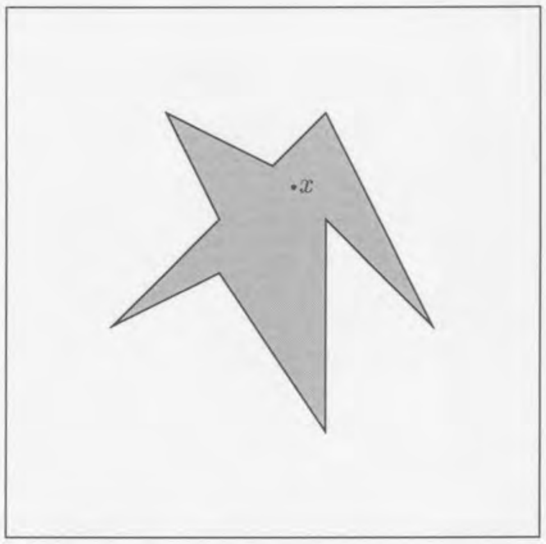

Figure 2.2: A star polygon – x is a kernel of the polygon

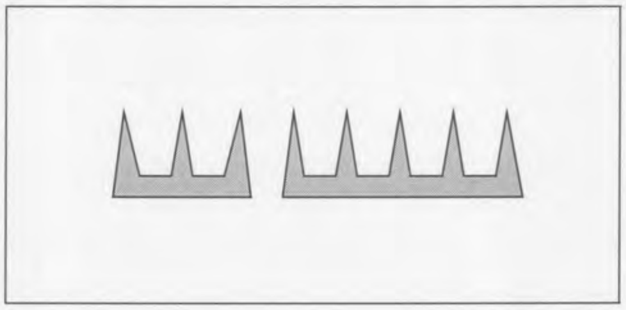

Figure 2.3: Comb polygons (Shermer [1992])

- A *quadrilateral* is a simple polygon with four edges and four vertices.
- An *interior angle* in a polygon is the angle between two consecutive edges of the polygon on the inside of the polygon.
- A *rectangle* is a quadrilateral where all four interior angles are right angles. The pairs of opposite sides of the rectangle are equal in length. If all four sides are of equal length then the rectangle is a square.
- If all of the edges of a rectangle are orthogonally aligned then this rectangle is referred to as an *orthogonal rectangle*.
- A polygon P is an *orthogonal polygon* if all of its edges are parallel to the major axes.
  Note: These polygons have commonly been called "rectilinear" polygons in the literature but O'Rourke [1987] prefers to call them orthogonal because

"rectilinear" (as was pointed out to him by Grunbaum) has a well established meaning: "characterised by straight lines". Orthogonal polygons have also been called *isothetic* polygons [Wood, 1985]. In this thesis O'Rourke's nomenclature is used.

- A *trapezoid* or *trapezium* is a quadrilateral that has one pair of opposite sides parallel, the other pair being nonparallel.
- A vertex v is a *reflex vertex* if it has interior angle ≥ 180 degrees.
- A polygon P is *orthogonally convex* if it is orthogonal and any horizontal or vertical line (that is not co-linear with an edge) intersects the boundary of P in at most two points. Figure 2.4 shows two orthogonally convex polygons. The second polygon is an orthogonally convex star.

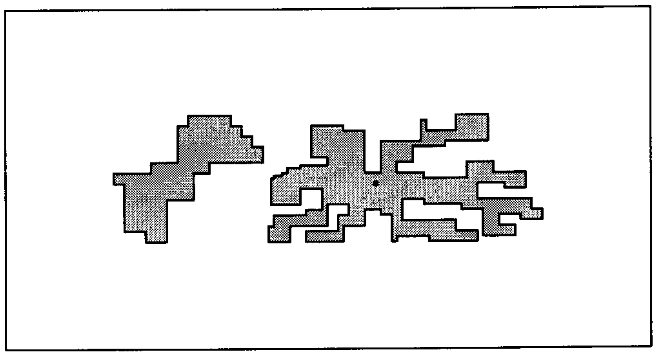

Figure 2.4: An orthogonally convex polygon and orthogonally convex star (Shermer [1992])

- A polygon P is *horizontally (vertically) convex* if any horizontal (vertical) line that is not co-linear with an edge intersects P in at most two points.
- A path inside a polygon P is orthogonally convex if it consists of orthogonal segments and any horizontal or vertical line that is not co-linear with a segment intersects the path in at most one point.
- An *orthogonal comb polygon* is as shown in Figure 2.5.
- A polygon P is said to be *covered* by a collection of subpolygons of P if the union of these subpolygons is exactly P. The collection of subpolygons is called a *cover* of P.
- A cover of a polygon P is said to be a *partition* of P if the intersection of each pair of subpolygons in the cover has zero area. (Note, in some work a partition is also called a *decomposition* but this is misleading terminology and is not used here. In this work *decomposition* includes covering and partitioning.)

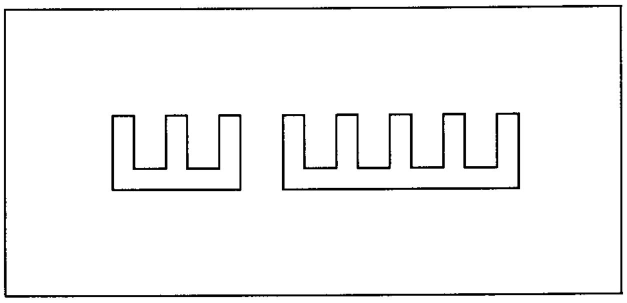

Figure 2.5: Orthogonal comb polygons (Shermer [1992])

- A *triangulation* of a polygon P is a decomposition of the polygon into triangles without adding vertices. This is accomplished by chopping the polygon with diagonals (line segments between nonadjacent vertices). See Figure 2.6 for an example of triangulating a simple polygon.

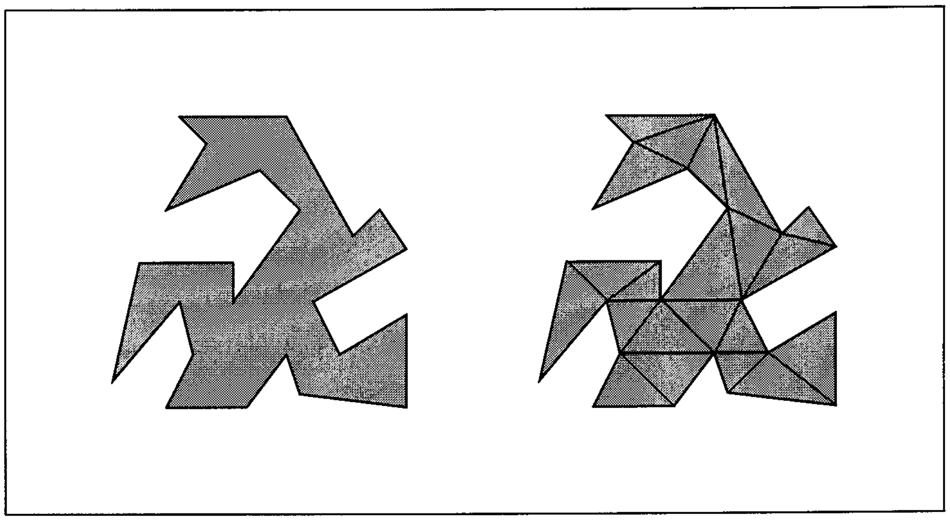

Figure 2.6: A simple polygon and one of its triangulations (Shermer [1992])

- A *Steiner point* is a vertex which is not one of the original points in the polygon. Steiner points are often used in covering and partitioning.
- A *chain* is a sequence $p_1, \ldots, p_k$ of vertices.
- A *reflex chain* of a polygon is a sequence of consecutive reflex vertices.
- A polygon P is *spiral* if it has at most one reflex chain, see Figure 2.7 for an example.

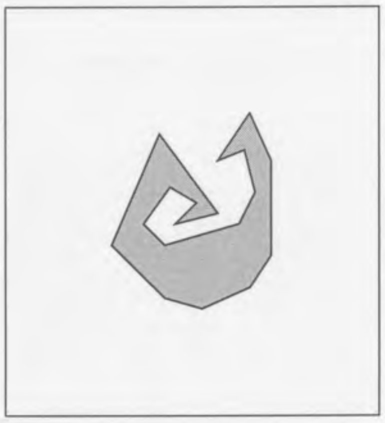

Figure 2.7: A spiral polygon (Shermer [1992])

- Two points in a polygon P are said to be *visible* if the straight line joining them does not intersect the exterior of P [Asano *et al.*, 1999]. Visible points are said to "see" one another. See Figure 2.8 for an example.

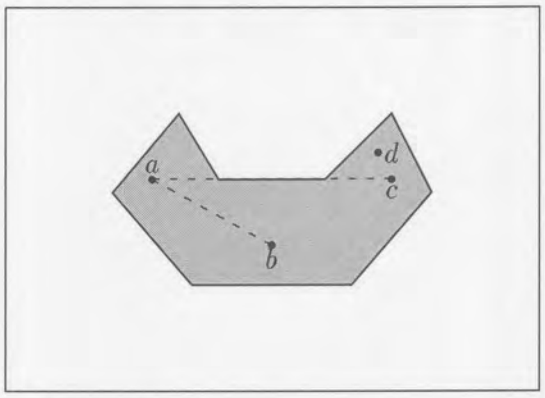

Figure 2.8: Point a can "see" b and c, but not d (Shermer [1992])

- Two points in a polygon are said to be link-j visible (L_j visible) if they can be joined by a path of j or fewer line segments that lie entirely inside the polygon. See Figure 2.9 for an example of L_3 visibility. L_1 visibility is the usual visibility – two points are visible if they can be joined by a path of one segment lying inside the polygon.

- An alternative way to define convexity is to call a set convex if each pair of points in the set is L_1 visible.

- The definition above can be generalised by calling a set link-k convex or L_k convex if every pair of points in the set is L_k visible.

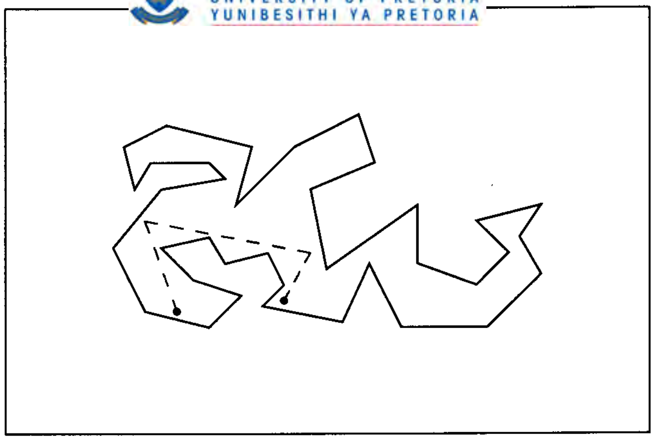

Figure 2.9: A pair of L_3 (or link-3) visible points (Shermer [1992])

- A *visibility polygon* of a point y in some polygon P contains all the points of P visible to y. Figure 2.10 shows the visibility polygon of the point y in the star polygon of Figure 2.2

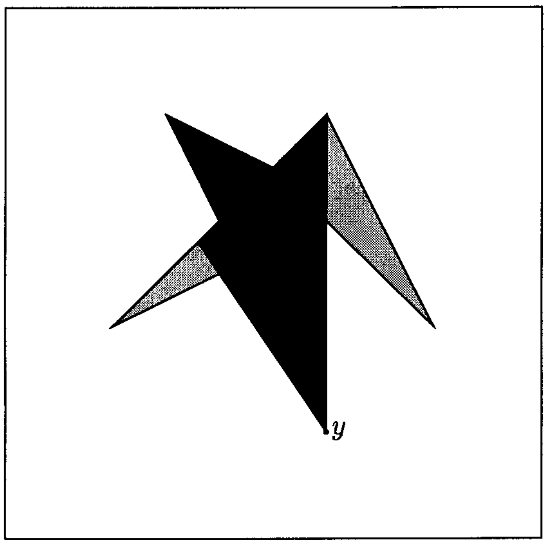

Figure 2.10: The visibility polygon of the point y (shown as the darker shaded subpolygon) – y is the kernel of the visibility polygon but not of the original polygon

- In some contexts, a point in a polygon is identified with a *guard*. A set of such points is called a *guard set*. If all of the elements in a guard set G are vertices of P then G is called a *vertex guard set* and the elements of G are called *vertex guards*. Otherwise G is called a point guard set and its elements are called *point guards*.

- A guard set G is said to cover a polygon P if the union of the visibility polygons of the guards in G is a cover of P. Figure 2.11 shows a covering guard set of a polygon P.

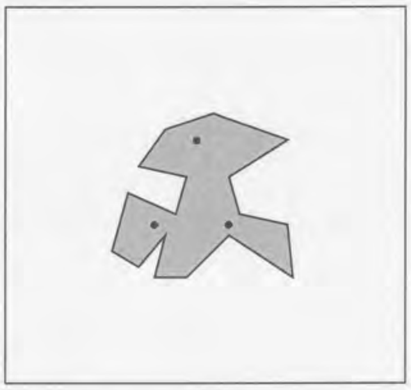

Figure 2.11: A covering guard set (Shermer [1992])

- A *star* polygon can thus be covered by a single guard.
- The *art gallery problem* for a polygon P is to find a minimum-cardinality covering guard set G for P.
- A *mobile guard* is a guard that can patrol an edge or diagonal (or some other line segment) of a polygon.
- An *edge guard* is a mobile guard who is allowed to patrol individual edges of a polygon rather than being restricted to one point.
- The *edge guard problem* then asks for the minimum number of edge guards necessary to cover any polygon of n vertices.
- An edge is called a *guard edge* if a mobile edge guard patrolling along the edge can see every point in the polygon.
- A *diagonal* of a polygon P is a line segment between two non consecutive vertices of P.
- A *chord* of a polygon P is a line segment between two points a and b on two distinct edges of P.
- A *diagonal guard* is a guard capable of moving along an edge or internal diagonal of a polygon [Lu *et al.*, 1998].
- A *chord guard* is a guard capable of moving along a chord wholly contained inside a simple polygon [Lu *et al.*, 1998].

i 16034880

b15432580

- A *line guard* is a guard capable of moving along a line segment wholly contained inside a simple polygon [Lu *et al.*, 1998].

- A polygon which can be guarded by a diagonal guard, chord guard or line guard is called *diagonal-visible*, *chord-visible* or *line-visible* respectively [Lu *et al.*, 1998].

- A *hidden set* is a set of points H in a polygon P such that no pair of points of H is visible. Figure 2.12 shows an example of a hidden set.

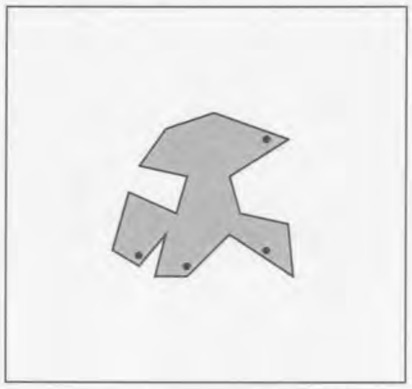

Figure 2.12: A hidden set (Shermer [1992])

- A *hidden guard set* is a hidden set which is also a guard set.

- A *window* of a point x (the *generator*) in a polygon P is an edge of the visibility polygon of x which is not an edge of P and is delimited by two event points b (a reflex vertex called the *base*) and a point q on the boundary of P. Windows constitute borders a guard has to pass when being moved in order to lose (or gain) sight of x since inside P only reflex vertices might block the view.

- A subset of P from which a generator is not visible is called a *pocket*. A pocket is joined to the visibility polygon by a window.

- A *dent edge* is any edge of some polygon where both the vertices are reflex vertices [Wood and Yamamoto, 1993].

- A chain $p_1, \ldots, p_k$ is called *monotone with respect to a line* L if the projections of $p_1, \ldots, p_k$ onto L are ordered the same as in the chain. Two adjacent vertices may project to the same point on L without destroying monotonicity.

- A polygon P is *monotone* if it can be partitioned into two chains that are monotone with respect to the same line.

- A *staircase path* is an orthogonal path such that the path is monotone with respect to the coordinate axes [Wood and Yamamoto, 1993].
- A *staircase polygon* is an orthogonal polygon, a subset of whose vertices constitute a staircase path (see Figure 2.13).

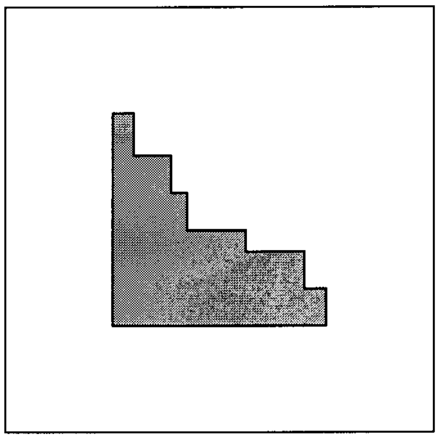

Figure 2.13: A staircase polygon

- The maximum visibility problem asks for locating a point inside the polygon from which the visible area is maximised [Gewali, 1993]. The minimum visibility problem is similarly defined.
- A simple polygon P is said to be an LR-visibility polygon if there exist two points s and t on the boundary of P such that every point of the clockwise boundary of P from s to t (denoted as L) is visible from some point of the counterclockwise boundary of P from s to t (denoted as R) and vice versa [Bhattacharya and Ghosh, 1998].
- A planar graph is a graph which can be drawn or embedded in the plane in such a way that the edges of the embedding intersect only at the vertices of the graph.
- A cut vertex is a vertex whose removal increases the number of components in a graph.
- A biconnected graph is a graph which contains no cut vertices.
- A biconnected planar graph is a planar graph which contains no cut vertices.
- A face in a planar representation of a graph is a planar region bounded by edges and vertices of the representation and containing no edges or vertices in its interior.

This list of terms is not complete. Many of the articles cited in the remainder of the chapter introduce more specific terminology or definitions, in addition to using many of those presented here, to discuss new problems or special cases of existing problems. The terms presented here should, however, be enough to give the reader an understanding of the discussion of the related literature in Section 2.4. Before this discussion of the related literature Section 2.3 presents a brief overview of complexity theory, particularly approaches using in proving problems NP-Complete. This material is presented to help the reader understand related literature presented in Section 2.4. A reader who is familiar with the material is encouraged to skip to Section 2.4

2.3 NP-Complete problems

2.3.1 Introduction

The following sections of the literature review chapter of this thesis discuss a number of problems which have been proved to be NP-Complete or NP-Hard. This section of the chapter is thus aimed at giving the reader an overview of the theory. More detail and more rigorous presentations can be found in Garey and Johnson [1979] and Papadimitriou [1994] and a more accessible discussion appears in Harel [1992]. Section 2.3.2 presents the theory of NP-Completeness and Section 2.3.3 discusses how a new problem can be shown to be NP-Complete. Section 2.3.4 discusses the relationship between NP-Complete and NP-Hard problems.

2.3.2 NP-Complete Problems

The theory of NP-Completeness is designed to be applied to *decision problems* – problems which only have “yes” or “no” answers. Abstractly a decision problem Π consists simply of a set D_Π of *instances* and a subset $Y_\Pi \subseteq D_\Pi$ of *yes-instances.* These decision problems are studied because they have a natural, formal counterpart – “languages” which can be studied in a mathematically precise theory of computation. The correspondence of decision problems and languages is brought about by encoding schemes which can be used to specify problem instances for study. The relationship between recognising languages and solving decision problems is straightforward. A deterministic Turing machine (DTM) program M solves a decision problem Π under an encoding scheme e if M halts for all input strings over its input alphabet, Σ, and $L_M = L[\Pi, e]$. L_M is the language recognised by the program M, that is $L_M = \{x \in \Sigma^* \mid M \text{ accepts } x\}$ and $L[\Pi, e]$ is an encoding of an instance of Π under encoding scheme e. Refer to Garey and Johnson [1979] and Papadimitriou [1994] for more detail on the application of this theory.

The class P is defined as follows:
P $= \{L \mid$ there is a polynomial time DTM program M for which $L = L_M\}$.

Then a decision problem Π belongs to P if there is a polynomial time DTM program that "solves" Π. More informally we could say that $\Pi \in$ P if there is a polynomial time algorithm which solves Π. The class NP can be defined (see Garey and Johnson [1979], page 28 for more detail) in terms of a *nondeterministic algorithm*. Such an algorithm is viewed as being composed of two stages – a *guessing* stage and a *checking* stage. Given a problem instance I, the first stage merely guesses some structure S. The structure is, of course, algorithm-dependent and represents, more or less explicitly, a possible answer to the problem. For example, in the travelling salesperson problem, the structure is a route through the cities to be visited. The checking stage then uses I and S as inputs and proceeds to compute in a normal deterministic manner either eventually halting with an answer "yes", eventually halting with an answer "no" or computing forever without halting. (The last two cases do not always need to be distinguished). A nondeterministic algorithm "solves" a decision problem Π if the following two properties hold for all instances $I \in D_\Pi$:

1. If $I \in Y_\Pi$ then there exists some structure S that when guessed for input I will lead the checking stage to respond "yes" for I and S.

2. If $I \notin Y_\Pi$ then there exists no structure S that when guessed for input I will lead the checking stage to respond "yes" for I and S.

The class NP is then defined informally to be the class of all decision problems Π that, under reasonable encoding schemes, can be solved by polynomial time nondeterministic algorithms.

The relationship between P and NP is fundamental in the theory of NP-Completeness. It is clear that P $\subseteq$ NP – every problem solvable by a polynomial time deterministic algorithm is also solvable by a polynomial time nondeterministic algorithm. Thus if $\Pi \in$ P then $\Pi \in$ NP. The current conjecture is that P $\subset$ NP but this is still an open problem. If P is different from NP then all problems in P can be solved with polynomial time algorithms and the problems in NP$-$P are termed intractable. The theory of NP-Completeness focuses on proving results of a weaker form – if P$\neq$NP then there are problems in NP which are neither solvable in polynomial time nor NP-Complete. The class P can be viewed as consisting of the "easiest" problems in NP and the class of NP-Complete problems contains the hardest problems in NP.

A language L is defined to be NP-Complete if $L \in$ NP and for all other languages $L' \in$ NP, $L' \propto L$ (where $L_1 \propto L_2$ implies a polynomial transformation from a language L_1 to a language L_2). Following from this, a decision problem Π is said to be NP-Complete if $\Pi \in$ NP and for all other decision problems $\Pi' \in$ NP, $\Pi' \propto \Pi$. This implies "the common fate phenomenon" of NP-Complete problems – every NP-Complete problem is polynomially transformable to every other one. If a single NP-Complete problem can be solved in polynomial time, then all problems in NP can be so solved. If any problem in NP is intractable, then so are all NP-Complete problems. At this stage it has not been proven that any NP-Complete problem is inherently intractable. In addition, no one has yet found a polynomial time solution for any NP-Complete problem. Even without a proof that NP-Complete problems are intractable, we know that any new problem which can be proved to be NP-Complete is at least as hard as the other NP-Complete problems and that a major breakthrough will be needed to solve such a problem with a polynomial time algorithm.

2.3.3 Proving the NP-Completeness of a new problem

From the above it seems that in order to prove a new problem Π NP-Complete, one must show that *every* problem in NP transforms to the new problem. *A priori*, it is not even clear that any NP-Complete problem need exist. It turns out that if *at least* one NP-Complete problem is known to exist that it is only necessary to show that

1. $\Pi \in \mathrm{NP}$
2. some known NP-Complete problem Π' transforms polynomially to Π.

This result arises because if L_1 belongs to NP and L_1 is NP-Complete then every other $L' \in \mathrm{NP}$ transforms to L_1. Now if L_2 is also in NP and there exists a polynomial time transformation from L_1 to L_2 then there exists a transformation from every other $L' \in \mathrm{NP}$ to L_2.

Harel [1992] presents the argument in a slightly different form. He states that to prove that Π is NP-Complete it is not necessary to find polynomial transformations from all of the other NP-Complete problems. It is only necessary to transform Π to some problem Π'' which is known to be NP-Complete ($\Pi \propto \Pi''$) and to transform another (or the same) problem Π' already known to be NP-Complete to Π ($\Pi' \propto \Pi$). The first transformation shows that in terms of tractability Π cannot be worse than Π'', that is that Π is in fact in NP (as required above). Then the second transformation shows that in terms of tractability Π cannot be better than Π'. Since Π' and Π'' are both NP-Complete and stand or fall together then Π must be NP-Complete too. Thus if one problem is known to be NP-Complete then other problems can be proved to be NP-Complete too by using the transformation approach twice for each new problem. Harel [1992] notes that in practice only the second of these transformations is carried out in an NP-Completeness proof. To show that Π cannot be any worse than the NP-Complete problems, that is that it is in fact in NP, can often be done more easily directly – by showing that Π can be solved by a polynomial time nondeterministic algorithm.

The "first" NP-Complete problem is the *satisfiability* problem and Cook's Theorem is used to prove that this problem is in fact NP-Complete. Now that a single problem has been shown to be NP-Complete the process of devising an NP-Completeness proof for any decision problem Π consists of the following four steps

1. showing that Π is in NP,
2. selecting a known NP-Complete problem Π'
3. constructing a transformation f from Π' to Π, and
4. proving that f is a polynomial transformation.

This approach is taken in the NP-Completeness proofs in this thesis.

Determining whether the decision problem, Π, is in NP is a matter of showing that given any solution for an instance I it is possible to verify in polynomial time whether or not that solution "proves" that the answer for I is "yes". For example, a nondeterministic algorithm for ***travelling salesperson*** could be constructed by using a guessing stage that simply guesses an arbitrary sequence of cities and a checking stage that checks whether the guessed solution would result in a tour of the desired length. The existence of such a polynomial time nondeterministic algorithm shows that Π is in NP.

Once the new problem has been shown to be in NP, then a known NP-Complete problem must be selected and a transformation must be constructed from the known problem to the new problem. There are three general transformation approaches that are used in NP-Completeness proofs and that can provide some ideas about how to tackle a specific NP-Completeness proof. Note that these approaches cannot be applied to all NP-Completeness proofs. They just give some ideas about how one might approach the task of proving a new problem to be NP-Complete.

These approaches are called [Garey and Johnson, 1979]

1. restriction
2. local replacement
3. component design

Restriction This is the easiest and perhaps most frequently applied of the three types of NP-Completeness proofs. An NP-Completeness proof by restriction for a given problem $\Pi \in$ NP consists simply of showing that Π contains a known NP-Complete problem Π' as a special case. The heart of such a proof lies in the specification of the additional restrictions to be placed on the instances of Π so that the resulting restricted problem will be identical to Π'. There should be an obvious one-to-one correspondence between their instances that preserves "yes" and "no" answers. This one-to-one correspondence, which provides the required transformation from Π' to Π (see point (3)) above is usually so apparent that it need not even be given explicitly. The approach taken in this kind of proof is often to focus on the target problem itself and attempt to restrict away its "inessential" aspects until a known problem appears.

Local replacement In this type of proof the transformations are sufficiently non-trivial to warrant spelling out in the standard proof format but they are still relatively uncomplicated. Proofs of this type rely on picking some aspect of the known NP-Complete problem instance to make up a collection of basic units and then obtaining an instance of the target problem by replacing each basic unit in a uniform way with a different structure.

Component design Proofs of this type tend to be the most complicated. The basic idea is to use the constituents of the target problem instance to design certain "components" (also called gadgets and widgets) that can be combined to realise an instance of the known NP-Complete problem.

Garey and Johnson [1979] discuss these transformations in more detail and give examples of the application of each approach.

2.3.4 NP-Hard problems

The techniques for proving NP-Completeness can also be used for proving that problems outside of NP are hard. Any decision problem Π, whether a member of NP or not, to which we can transform an NP-Complete problem will have the property that it cannot be solved in polynomial time unless P=NP. Such a problem could be said to be "NP-Hard" since it is at least as hard as the NP-Complete problems. The idea of NP-Hardness can be generalised in such a way that not only decision problems can be proved to be "at least as hard" as the NP-Complete problems.

In this thesis only the notion of decision problems will be used. The reader is referred to Garey and Johnson [1979] and Papadimitriou [1994] for more detail on NP-Hardness.

2.3.5 Summary

In summary, the approach taken to prove a new problem, say R, to be NP-Complete is to

1. show R is in NP – this involves showing that R can be solved by a nondeterministic polynomial algorithm,
2. selecting a known NP-Complete problem, say T,
3. constructing a transformation f from T to R, and
4. proving that f is a polynomial transformation.

To show that a problem is NP-Hard only steps 2 to 4 are required.

Many of the problems raised in the areas of guarding, covering, partitioning and visibility have been proved to be "hard" in the sense discussed above so the theory presented in this section is useful in understanding many of the papers presented in the literature and in the section below (Sections 2.4. This theory is also a useful starting point for some of the new results proved in the body of this thesis – Chapters 4 to 7.

2.4 Relation of previous work to ALP

2.4.1 Overview

In Section 2.1 above the commonalities between ***ALP*** and visibility, guarding and polygon decomposition were briefly introduced. These problems, which all have their roots in real world problems, can in some sense be thought of as different castings of the same problem and results presented in one area often apply elsewhere as well. Problems of this type have been studied extensively over the last 30 years and the results of these studies have been collected in a number of very comprehensive surveys. The survey article by Wood [1985] discusses (amongst other problems) research in visibility in orthogonal (he calls them isothetic) polygons. The monograph by O'Rourke [1987] "Art Gallery Theorems and Algorithms" provides an excellent overview of the current state of knowledge of Art Gallery Guarding problems at the time it was written. The survey paper by Shermer [1992] "Recent Results in Art Galleries" extended O'Rourke's work. O'Rourke [1993] also discusses many of these problems in his textbook. The survey papers by Urrutia [1999] "Art Gallery and Illumination Problems", Asano *et al.* [1999] "Visibility in the plane" , and Keil [1999] "Polygon Decomposition" are all published in the Handbook of Computational Geometry and give a very thorough coverage of the material. In addition the summaries of results by Suri [1997] "Polygons" and O'Rourke [1997] "Visibility" published in the "Handbook of Discrete and Computational Geometry" are excellent sources of reference for results in these areas. There is thus little point in trying to repeat these surveys and the reader is encouraged to consult them to get an overall feeling of the current state of research in these areas.

The next section gives a very brief overview of some of the important general results in the areas of guarding, visibility and polygon decomposition. A reader who is very familiar with these areas could skip this section of the thesis. The following section puts ***ALP*** into context with the previous and related research work. The subsequent section focusses in greater depth on results, problems or techniques which give special insight into ways of dealing with ***ALP***.

2.4.2 Short historical perspective

2.4.2.1 Guarding problems

The original art gallery guarding problem, posed by Klee in 1973, asks for the number of guards required to survey an art gallery with n walls and no interior obstructions [O'Rourke, 1987]. The problem was first solved by Chvatal [1975] who showed that there exist polygons where $\lfloor n/3 \rfloor$ vertex guards are both necessary and sufficient to guard the entire area of the polygon. He first showed using comb polygons that any guard set for such a polygon with n vertices, $g(n)$, must have at least $\lfloor n/3 \rfloor$ guards and then by an inductive argument on triangulation graphs of polygons showed that the size of the guard set is less than or equal to $\lfloor n/3 \rfloor$. Thus $g(n) = \lfloor n/3 \rfloor$. A different and very simple sufficiency proof, using three-colouring on triangulation graphs, of the same result was given in 1978 by Fisk [1978]. O'Rourke [1987] presents very lucid discussion on both of these proofs. Avis and Toussaint [1981] used Fisk's proof to implement an $O(n \log n)$ algorithm to assign positions to the guards. Algorithms such as this for finding guard sets are often called "guard placement algorithms". Most guard placement algorithms work by imitating upper-bound art gallery proofs. The algorithm by Avis and Toussaint [1981] is in fact an algorithmic imitation of Fisk's proof.

Later Lee and Lin [1986] tackled the issue of the computational complexity of the art gallery guarding problem. Their work was aimed at the problem of finding the minimum number of guards to cover a given polygon. They proved that the problem for a simply connected simple polygon is NP-Hard for the *minimum vertex guard* problem (where guards must be located at the vertices of the polygon), *minimum point guard* problem (where guards can be placed anywhere in the interior of the polygon or on its boundary) and *minimum edge guard* problem (where guards can only be placed on the edges which make up the polygon boundary and are allowed to move along the edge on which they are placed). Their proof is based on a transformation from Boolean Three Satisfiability (3SAT). (Note: Shermer [1992] credits Aggarwal with first proving the result for the minimum point guard problem but the author of this thesis was unable to obtain a copy of Aggarwal's Ph.D. thesis.)

Lee and Lin's results imply that it is in general impractical to find a minimum set of guards for a given polygon. Some polygons can be guarded by a single guard and there exist polygons where $\lfloor n/3 \rfloor$ are required. Finding the exact number of guards required for a given polygon cannot always be done in reasonable time. Algorithms such as that by Avis and Toussaint [1981] do not guarantee a minimum solution for a given polygon but will place a guard set which is always sufficient (and sometimes necessary) to guard the polygon.

If the art gallery is allowed to have obstructions (pillars etc.) in its interior, the corresponding floor plan is a simple polygon with other simple disjoint polygons, called holes, inside it. In such a polygon, $\lfloor n/3 \rfloor$ guards are no longer sufficient.

O'Rourke showed that $\lfloor (n+2h)/3 \rfloor$ point or vertex guards are always sufficient to guard any polygon with n vertices and h holes. Shermer showed that $\lfloor (n+h)/3 \rfloor$ guards are necessary for some polygons with n vertices and h holes and conjectured that this number of guards was also sufficient for any polygon with h holes. In 1984 Shermer showed that $\lfloor (n+1)/3 \rfloor$ guards are always sufficient and sometimes necessary for any polygon with one hole. All of these results appear in O'Rourke [1987].

In 1991 two independent and dramatically different proofs were generated for the fact that $\lfloor (n+h)/3 \rfloor$ point guards are also sufficient ([Bjorling-Sachs and Souvaine, 1991; Hoffmann *et al.*, 1991]). Neither paper gave details of algorithms for placing the guards but the algorithm derived from the Hoffmann *et al.* [1991] proof has complexity $O(n^3 \log n)$. In 1995 Bjorling-Sachs and Souvaine [1995] presented an $O(n^2)$ algorithm to place the guards. This algorithm is based on a constructive proof. The proof and the algorithm work by first connecting each hole in the polygon to the exterior of the polygon and then triangulating the new hole-free version of the polygon. The channels are constructed in such a way that only one new vertex is added for each channel. In addition, there is always a triangle in the new hole free polygon that "sees" all of the channel – any vertex of this triangle sees the whole channel. These special triangles are included so that a guard placement based on three-colouring in the hole free polygon will automatically cover the channels and so the original polygon would have been guarded.

Other variations of the classic problem arise when specified subsets of the polygon, rather than just points, are allowed as elements of guard sets. The first of these variations is the *edge guard problem*. An edge guard is a guard who is allowed to patrol individual edges of a polygon rather than being restricted to one point. The edge guard problem then asks for the minimum number of edge guards necessary to cover any polygon of n vertices. Toussaint (as cited in Shermer [1992]) conjectured that (if a small number of polygons are excluded) $g^E(n) = \lfloor n/4 \rfloor$ guards are necessary. Figure 2.14 shows the type of polygons where $\lfloor n/4 \rfloor$ edge guards are required. Two types of polygons are known which require $\lfloor (n+1)/4 \rfloor$ guards (see Figure 2.15) but these are thought to be exceptions. O'Rourke [1983] made some progress on Toussaint's conjecture. He was unable to establish an upper bound on the number of edge guards required but was able to place a bound on the number of *mobile guards* necessary. A mobile guard is a guard that can patrol an edge or diagonal of a polygon. Every edge guard is thus a mobile guard and the upper bound on the number of mobile guards thus gives the least upper bound on the number of edge guards $g^M(n) \leq g^E(n)$. O'Rourke [1983] showed that $g^M(n) = \lfloor n/4 \rfloor$ mobile guards are necessary. Shermer [1992] investigated *diagonal guards* and showed that $\lfloor (2n+2)/7 \rfloor \leq g^D(n) \leq \lfloor (n-1)/3 \rfloor$. He also showed that, aside from a small number of exceptions, $\lfloor n/4 \rfloor \leq g^E(n) \leq \lfloor 3n/10 \rfloor$. Sack and Suri [1990] have given an O(n) algorithm to detect if a given polygon can be guarded by one edge guard and Ke [1989] (as cited in Shermer [1992] – the original reference could not be obtained) gave an $O(n \log n)$ algorithm for the related problem of detecting if a given polygon can be guarded by one line-segment guard.

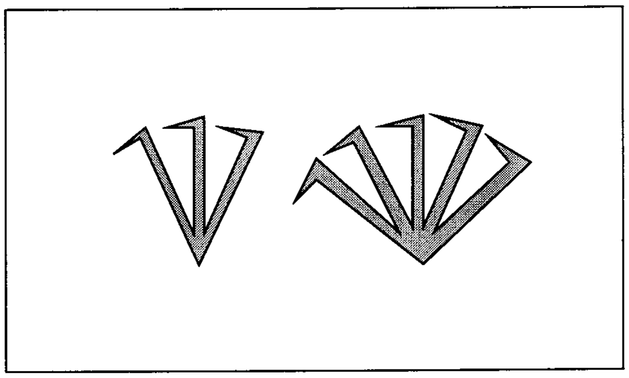

Figure 2.14: Polygons requiring $\lfloor n/4 \rfloor$ edge guards (Shermer [1992])

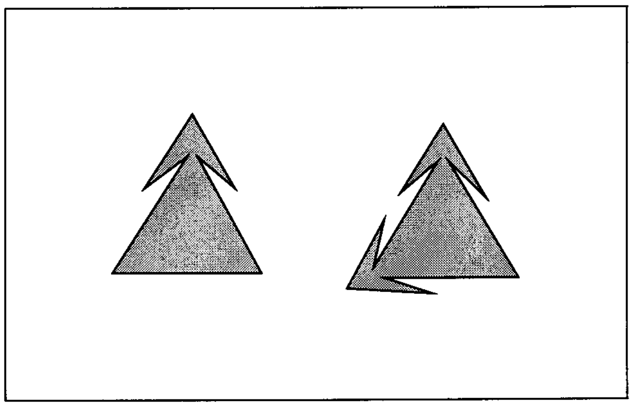

Figure 2.15: The two polygons requiring $\lfloor (n+1)/4 \rfloor$ edge guards (Shermer [1992])

Other variations of the general problem are the orthogonal art gallery theorem [Kahn *et al.*, 1983] (which is discussed in more detail below); guarding rectangular art galleries [Czyzowicz *et al.*, 1994]; generalised guarding of rectilinear polygons [Györi *et al.*, 1996]; the prison guard problem [Kooshesh *et al.*, 1990; Füredi and Kleitman, 1994]; the treasury guard problem [Deene and Joshi, 1992; Carlsson and Jonsson, 1993]; edge guards [Viswanathan, 1993]; edge guards in star polygons [Subramaniyam and Diwan, 1991]; diagonal and chord guards [Lu *et al.*, 1998]; guard edges [Park *et al.*, 1993]; optimally placing k guards in a polygon to maximise the area or portion of boundary visible [Ntafos and Tsoukalas, 1994]; floodlight guards [Bose *et al.*, 1993; Czyzowicz *et al.*, 1993; Contreras *et al.*, 1998b,a]; the searchlight scheduling problem [Sugihara *et al.*, 1990]; hidden guard sets [Shermer, 1989]; watchman paths [Carlsson and Jonsson, 1995]; watchmen in grids [Ntafos, 1986]; periscope guards in grids [Gewali and Ntafos, 1993] and other problems [Liaw *et al.*, 1993; Venkatasubramanian and Cullum, 1993]. Methods for developing approximations [Avis and Toussaint, 1981; Ntafos and Tsoukalas, 1994] or for bounding approximations have also been studied [Eidenbenz *et al.*, 1998].

2.4.2.2 Visibility Problems

Visibility problems have been studied from at least the early part of the twentieth century and a number of different topics have been studied [Asano *et al.*, 1999]. This section gives a brief overview of some of the main topics. More detail can be found in O'Rourke [1987], Asano *et al.* [1999] and the original papers.

The basic question in visibility can be stated as

Given a polygon P and two points $x, y \in P$, are x and y visible?

In this case x and y are visible if the line segment xy contains no points of the exterior of P [Asano *et al.*, 1999]. In a polygon with holes, the holes are taken to be part of the exterior of the polygon and so the same idea of visibility applies.

This idea leads naturally to the problem of determining what part of some polygon P can be seen from a given point a. The *visibility polygon* $V(a)$ of a point a in a polygon P is the set of all points visible to a ($V(a) = \{q \in P | a \text{ sees } q\}$). Any visibility polygon $V(a)$ is star shaped – a is its kernel. The problem of determining the visibility polygon for some point a in a polygon has been well studied (see for example ElGindy and Avis [1981]; Heffernan and Mitchell [1995]).

Avis and Toussaint [1981] extended this work by defining the concepts of *complete visibility*, *strong visibility* and *weak visibility* from some fixed edge uv of P.

1. P is said to be completely visible from an edge uv if for every $z \in P$ and every $w \in uv$, w and z are visible.

2. P is said to be strongly visible from an edge uv if there exists a $w \in uv$, such that for every $z \in P$, w and z are visible.

3. P is said to be weakly visible from an edge uv if for each $z \in P$ there exists a $w \in uv$ such that w and z are visible.

Avis and Toussaint [1981] then presented an $O(n)$ algorithm to determine whether a given polygon P is completely visible, strongly visible or weakly visible from a particular edge in P. Some related work on weak visibility of polygons is due to Sack and Suri [1990], Ghosh *et al.* [1993] and Doh and Chwa [1993].

An area which is related to the above is edge-to-edge visibility in polygons. Here again there are degrees of visibility between the edges concerned [Avis *et al.*, 1986].

1. Edge uv is said to be completely visible from edge xy if for all points on z on edge xy and all points w on edge uv, w and z are visible.

2. Edge uv is said to be strongly visible from edge xy if there exists a point z on edge xy such that for all points w on edge uv, w and z are visible.

3. Edge uv is said to be weakly visible from edge xy if for each point w on edge uv there exists a point z on edge xy such that w and z are visible.

4. Edge uv is said to be partially visible from edge xy if there exists a point w on edge uv and a point z on edge xy such that w and z are visible.

Avis *et al.* [1986] presents a linear algorithm to compute these four edge-to-edge visibilities. This algorithm is discussed is more detail in Section 2.4.4.3.1 because it has some direct relevance to ***ALP***.

The problem of computing the first intersection of the boundary of some polygon P with a light ray from some edge or point in P is also of interest in the field of visibility [Chazelle and Guibas, 1989].

Link visibility is another area that has been the subject of research [Asano *et al.*, 1999]. The *link* distance between two points p and q in a polygon P is the minimum number of line segments (links) in a polygonal path from p to q that stays in P. Two points are called *link-j* visible if the link distance between them is at most j. Letting $j = 1$ gives the standard visibility. Note that this form of visibility is also referred to as L_j visibility.

Another important area of research is in computing visibility graphs of polygons [Asano *et al.*, 1999]. A visibility graph of a polygon is a graph whose vertices are the vertices of the polygon and whose edges are the pairs of visible vertices (see Figure 2.16 for an example). Related to the computing of visibility graphs is the problem of determining for a given graph G if there exist some polygon P that has G as its visibility graph. This is called the visibility graph *recognition* problem. The problem of actually constructing such a P is called the visibility graph *reconstruction* problem. In this area see for example Ghosh [1991], Andreae [1992], Srinivasaraghavan and Mukhopadhyay [1993], Ghosh [1996] and Ghosh [1997].

Other types of visibility have also been studied (see Asano *et al.* [1999] for more information) – minimum and maximum visibility [Gewali, 1993]; clear visibility; dent and staircase visibility [Wood and Yamamoto, 1993]; $\mathcal{O}$-visibility; rectangular visibility; circular visibility; visibility with reflection; X-ray visibility; LR-visibility [Das *et al.*, 1993; Bhattacharya and Ghosh, 1998]; etc.

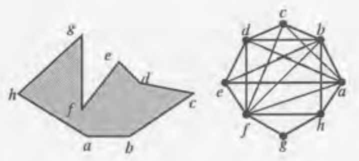

Figure 2.16: A polygon and its visibility graph

2.4.2.3 Polygon Decomposition

2.4.2.3.1 Overview As discussed in Section 2.1 polygon decomposition is categorised in two different ways – covering and partitioning. The basic covering problem is to find the minimum number of polygons, with some predefined characteristics, to cover the complete area of an enclosing polygon. This problem is intimately related to the guarding problem – guarding a polygon with guards chosen from some set S is the same as covering the polygon with visibility polygons of the members of S. The partitioning problem is the same as the covering problem except that the polygons into which the enclosing polygon is decomposed are not permitted to overlap. In addition to the survey articles mentioned earlier Keil and Sack [1985] presents a very readable but now somewhat dated overview of polygon decomposition. The partitioning problem is the version of the decomposition problem which has relevance to this research. Finding the convex map of an urban area is equivalent to partitioning a polygon with holes into the smallest number of convex polygons. This section discusses some of the previous research in both covering and partitioning.

2.4.2.3.2 Covering The problem of covering a polygon with simpler polygons has been the subject of much research and many of the problems in this class have been shown to be NP-Hard. Many NP-Hardness results for covering problems were first established for polygons with holes [O'Rourke and Supowit, 1983]. Using a transformation from 3SAT they proved that minimum convex cover for a multiply connected polygon (a polygon with holes); that minimum star-shaped cover for a multiply connected polygon; and minimum spiral cover for a multiply connected polygon are NP-Hard. Lee and Lin [1986] were the first to establish some NP-Hard results for polygons without holes – in particular they showed that the star cover problem (covering a simple simply connected polygon with the smallest number of star polygons) was NP-Hard. This result follows from the fact that they proved that the minimum point guard problem was NP-Hard – star covering and point guard are different castings of the same problem. They also showed that vertex guarding and edge guarding are NP-Hard and thus that the corresponding cover problems are NP-Hard as well. Later Culberson and Reckhow [1988, 1994] established the NP-Hardness for convex cover and similar problems using a transformation from SAT.

One approach to dealing with the difficulty of covering problems is to consider restrictions of the general problem. For instance, the problem of covering orthogonal polygons with many types of subpolygons has received a lot of attention. Some examples are given below but the reader is referred to the survey articles mentioned above for more complete coverage. Masek has shown that finding the minimum cover of a orthogonal (or rectilinear) polygon with rectangles is NP-Hard (the problem at that stage was termed *rectilinear picture compression*). This result at the time was only known to apply to orthogonal polygons with holes. Franzblau and Kleitman [1984] then showed that the problem can be solved in polynomial time if the orthogonal polygon is vertically convex. Franzblau [1989] presents an approximation algorithm for this problem which gives a solution which is at worst twice the minimum solution if the polygon has no holes. Lubiw [1990] has also worked on this problem – showing it is a special case of the boolean basis problem.

In orthogonal polygons with holes, Conn and O'Rourke [1987] (as cited in Shermer [1992]) have shown that covering either the boundary or the reflex vertices is NP-Complete but they found an $O(n^{2.5})$ time algorithm for covering the convex vertices. Culberson and Reckhow [1988] have shown that covering an arbitrary orthogonal polygon with rectangles is NP-Complete even if only the boundary of the orthogonal polygon is to be covered. They have also done similar work with Dent diagrams in orthogonal polygons [Culberson and Reckhow, 1989b,a]. Motwani *et al.* [1990b] showed that a polynomial time algorithm can be found for orthogonally convex polygon coverings of orthogonal polygons with three dent orientations but not for four dent orientations. Motwani *et al.* [1990a] showed that covering orthogonal polygons with star polygons can be accomplished in polynomial time and Gewali *et al.* [1992] showed that a minimum orthogonal star polygon cover for a horizontally convex orthogonal polygon can also be found in polynomial time.

2.4.2.3.3 Partitioning In the previous section (Section 2.4.2.3.2) many of the polygon covering problems are shown to be NP-Hard. The partitioning problem, however, has been shown to be solvable in polynomial time in many cases.

The early work in partitioning polygons without holes into the minimum number of convex polygons [Feng and Pavlidis, 1975; Schachter, 1978] produced algorithms which could not guarantee a minimum of components. Then Chazelle and Dobkin [1985] were able to show that finding a minimum convex partition of a simple simply connected polygon (a cover by *nonoverlapping* convex pieces) has polynomial-time complexity and presented a polynomial-time algorithm for this problem. Their algorithm is $O(n+c^3)$ where n is the number of vertices in the polygon and c is the number of reflex angles. The algorithm allows for the introduction of Steiner points. It begins by producing a naive decomposition of the polygon by removing each notch (reflex angle) in turn by means of a simple line segment drawn from the notch until it encounters a line already in the decomposition. This naive decomposition is then "improved" until an optimal convex decomposition is achieved. The process for doing the improvement is based on the idea of X-patterns (and Y-patterns) which describe particular interactions of notches. The reader is referred to the original paper for details. Keil [1985] considered partitioning which does not allow Steiner points and presented polynomial-time dynamic programming algorithms for partitioning a simple simply connected polygon into the minimum number of convex polygons, spiral polygons, star polygons and monotone polygons. Recently Keil and Snoeyink [1998] presented an improved algorithm for the same problem.

The convex polygon partitioning problem is NP-Hard in polygons with holes – simple multiply connected polygons [Lingas, 1982]. The proof by Lingas [1982] is based on a transformation from a planar version of 3SAT. Lingas's 1982 paper is a revision of an earlier paper and in the revision he uses some insights from O'Rourke and Supowit [1983] about the truth setting components of the transformation to simplify his paper. Lingas [1982] credits O'Rourke and Supowit [1983] with having proved the same result independently of him.

Partitioning problems where the original polygon is to be partitioned into different types of subpolygons (spiral polygons, star-shaped polygons, etc.) have also been studied (see [Keil, 1999] for an overview of these).

As is the case with guarding, visibility and covering, work has been done on considering approximations to the solution for the general problems and in studying restrictions on the general problems. Some examples of this work are given below. (More detail can be found in the survey articles mentioned earlier).

In studying restrictions of the general problem to partition a orthogonal polygon with the minimum number of rectangles, Imai and Asano [1986] present an $O(n^{1.5}\log n)$ algorithm if the polygon has holes; Liou *et al.* [1989] (as cited in Shermer [1992]) present an $O(n)$ algorithm for this problem if there are no holes; and Soltan and Gorpinevich [1993] showed that this problem can be solved in polynomial time even if the original polygon contains degenerate (point) holes. In addition, Ku and Leong [1995] have studied optimal partitions of rectilinear layouts used in VLSI design.

Other types of restrictions have also been studied. Asano *et al.* [1986] discuss the problem of partitioning a polygon into a minimum number of trapezoids with two horizontal sides. The problem is shown to be NP-Complete for polygons with holes but solvable in polynomial time for polygons without holes. Lingas and Soltan

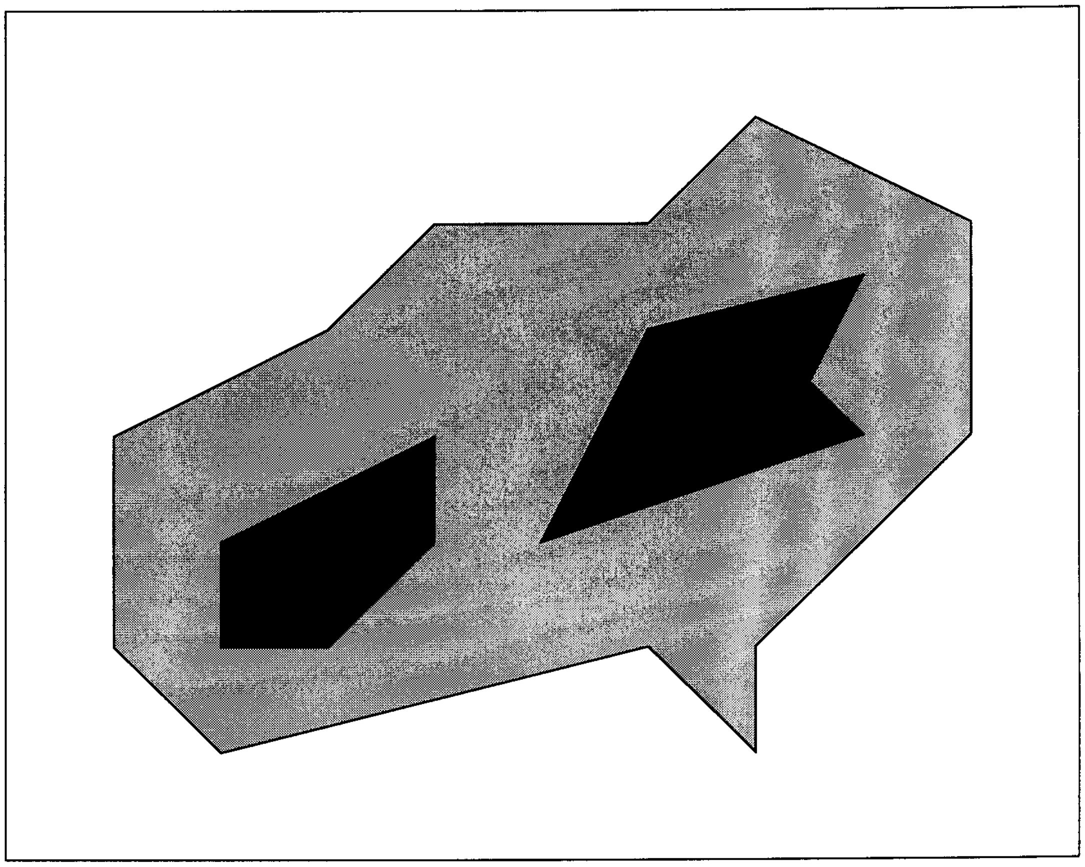

Figure 2.17: A multiply connected simple polygon (a simple polygon with holes)

[1996] prove that the problem of partitioning a polygon with holes into a minimum number of convex polygons by cuts in a family of directions F is NP-Hard if $|F| \geq 3$ and polynomial time for $|F| \leq 2$. Keil [1999] gives additional examples.

2.4.3 Putting *ALP* into context with other research

In Section 2.1 above the fact that there are some commonalities and some differences between ***ALP*** and visibility, guarding and polygon decomposition problems is briefly discussed. The aim of this section is to highlight these commonalities and differences in the context of the previous and related research (see Section 2.4.2).

Consider the polygon with holes shown in Figure 2.17. This polygon is the basis for much of the discussion that follows in this section. This polygon is chosen to be representative of a class of polygons which research in visibility, guarding and decomposition focusses on – a multiply connected simple polygon or a simple polygon with holes. It is a relatively uncomplicated example but is still complicated enough to illustrate the complexity of these problems. It can also be considered as an example of the types of polygons which are used in space syntax and so is of

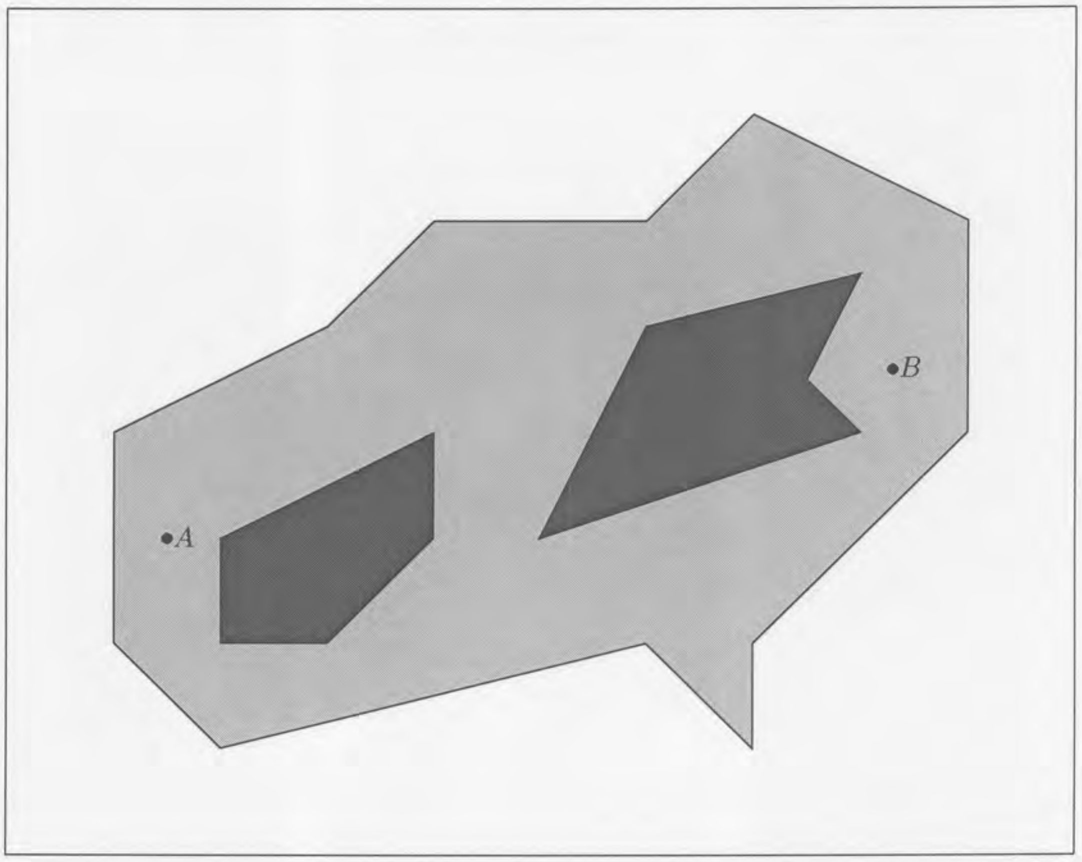

Figure 2.18: Point-point visibility

specific interest in ***ALP***.

As mentioned earlier, the intent of the convex map of some urban layout is to give a localising perspective of the urban layout or town plan. It tells one what can be seen (and directly accessed) from a particular point in the town. The intent of the axial map of some urban layout is that it offers a globalising perspective that takes into account how far one can *see* (or walk) in the town. These two concepts relate to the idea of *visibility* in a polygon. The basic question in visibility research is "can some point in the polygon see some other point in the polygon?". This idea of "point-point" visibility is shown in Figure 2.18. Here the question is "Can point A see point B?" or "Is there a direct line of sight between A and B?". Clearly here the answer is "No". This can be related to space syntax in the sense that A and B are not in the same "locality". Some effort must be put in to get from A to B or if one starts at A to get to a point where one is able to see B. The axial map of the layout will address how much effort is required. This point is addressed later in the discussion. Note however, at this point, that A and B are L_2 visible or *link*-2 visible. They can be joined by a path made up of two line segments. This point is also returned to later.

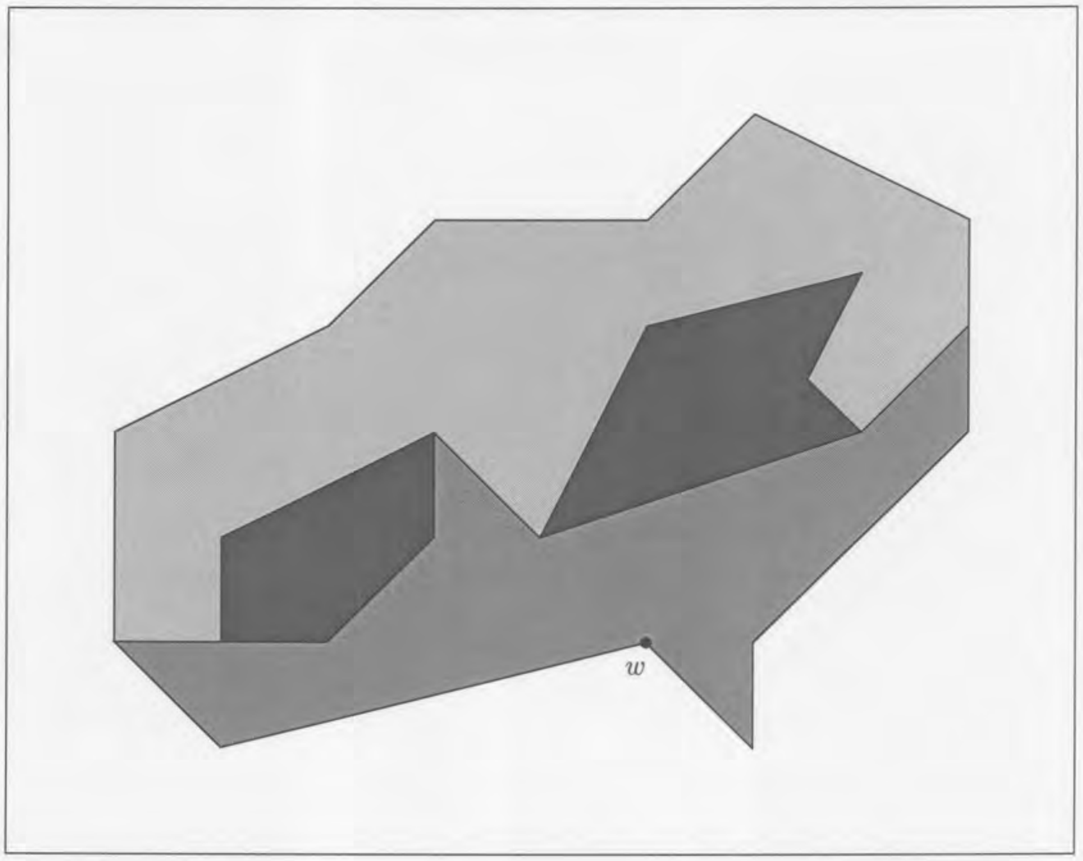

Figure 2.19: Placing a vertex guard

The traditional art gallery guarding problem is to determine the minimum number of guards necessary to cover the interior of an n-wall art gallery [O'Rourke, 1987]. In the example polygon given in Figure 2.17 the aim of guarding would be place the minimum number of guards so that all of the interior of the polygon can be seen by at least one of the guards. From this it is easily seen that the problem of guarding a polygon is very closely related to visibility in polygons. In fact, all guarding problems are essentially visibility problems. In this discussion vertex guards (that is any guard must be placed at a vertex on the exterior of the polygon or at a vertex on the boundary of one of the holes) are used to describe the guarding problem. In Figure 2.19 a single vertex guard, w, has been placed. The darker shaded portion of the interior of the original polygon is the area which can be seen or guarded by this guard. Note that this more darkly shaded area of the interior of the original is also the *visibility polygon* from the point w. Clearly the guard w cannot see all of the interior of the polygon and additional guards are required.

A second guard can be placed at x on the boundary of the original polygon – see Figure 2.20. This guard can see a different subset of the original polygon and thus generates a different visibility polygon. Note that there are some areas of overlap

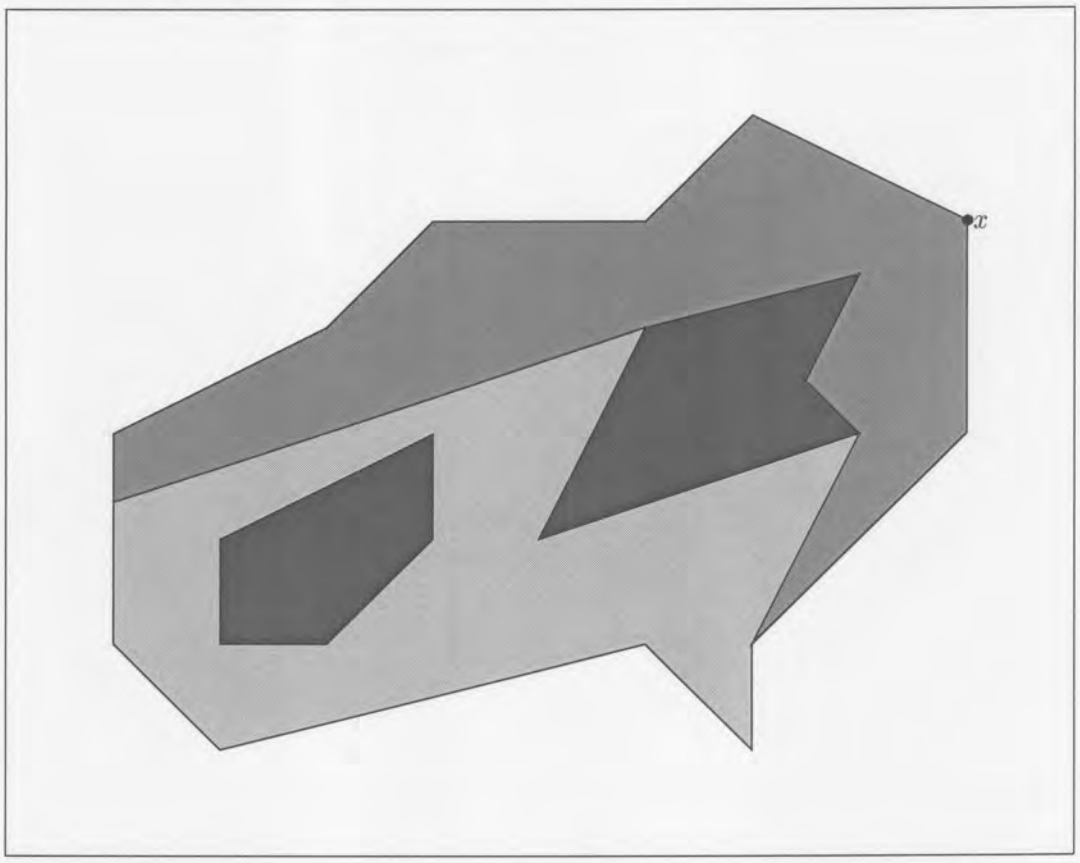

Figure 2.20: Placing the next vertex guard

between the two visibility polygons. These are areas which are guarded by both w and x.

A minimal vertex guard set for the original polygon is shown in Figure 2.21. There could be other guard placements but for this example 4 guards are required. If the problem was that of placing point guards, edge guards, diagonal guards, etc. then similar results would apply.

Note that each guard which is placed generates its own visibility polygon. Each of these visibility polygons is a star polygon – the guard position is the one point in the polygon which is guaranteed to be able to see and be seen by every other point in this polygon. Thus the guard placement gives a star polygon *cover* for this polygon.

As with this specific example, many guarding problems have some commonality with the area of *polygon decomposition* – covering a polygon with the minimum number of polygons of some specified type is equivalent to the placement of the minimum number of guards of some specified type so that each point inside the polygon is visible to some guard. In polygon decomposition the focus is in decomposing given polygons into smaller more easily manageable parts. Polygon de-

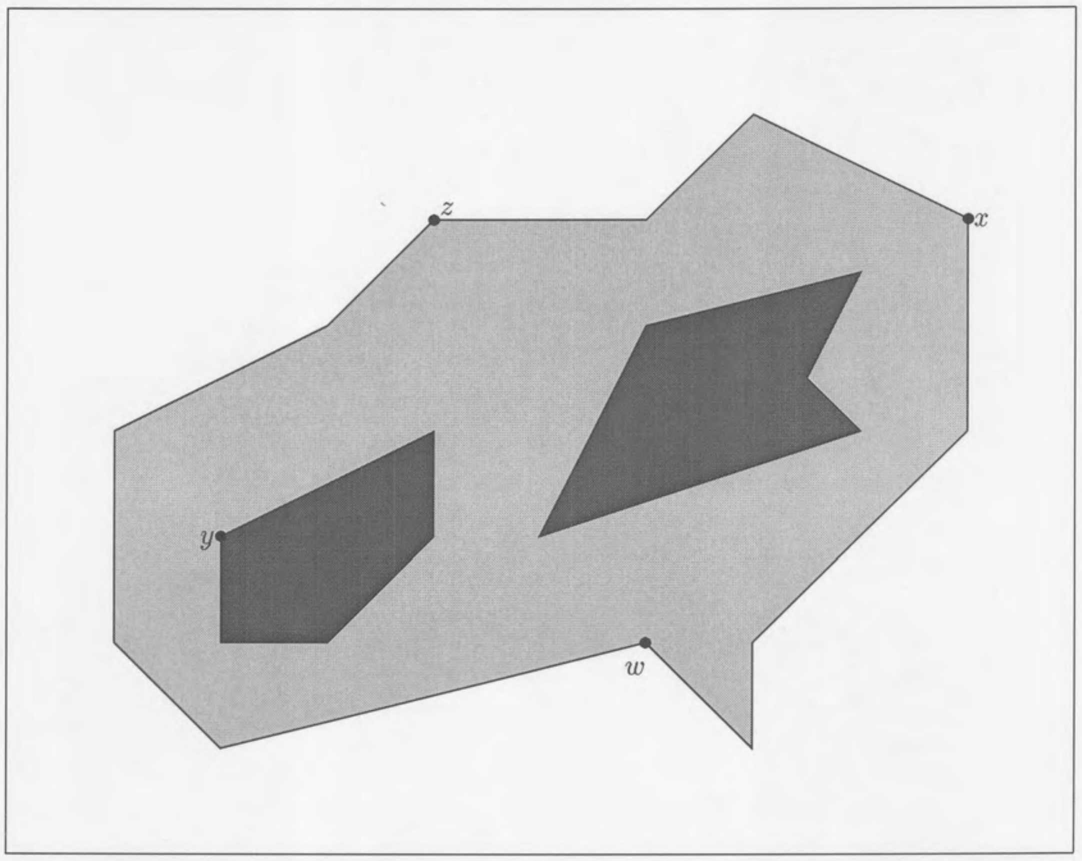

Figure 2.21: A vertex guard set for the example polygon

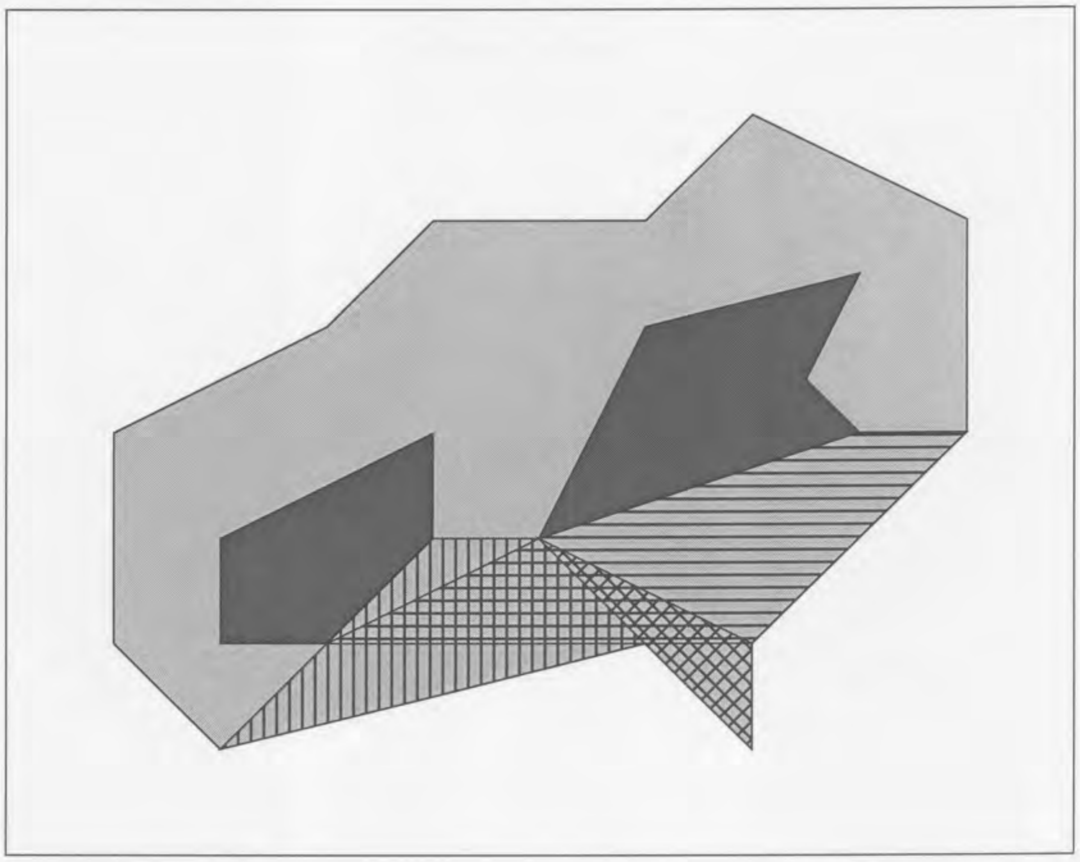

Figure 2.22: Part of a *cover* with convex polygons (note the overlapping of the convex polygons)

composition can be categorised in two different ways – *covering* and *partitioning*. The basic covering problem is to find the minimum number of polygons, with some predefined characteristics, to cover the complete area of an enclosing polygon. The partitioning problem is the same as the covering problem except that the polygons into which the enclosing polygon is decomposed are not permitted to overlap.

The phase of the space syntax analysis method which produces a convex map of a given urban layout is clearly a polygon decomposition problem. The town plan is a polygon with holes and the convex map is a minimum partition of that polygon. In space syntax the convex map of the urban layout is made up of the smallest number of convex spaces that "cover" all of the space in the area being analysed. A convex space gives some local information about the area in the sense that every point in the convex space is directly visible and directly accessible to every other point in that convex space. There are two ways of interpreting this definition. The first, that it is a *cover* in the computational geometry sense of covering. That is that overlapping convex polygons are allowed. Figure 2.22 shows part of a *cover* of the original polygon by convex polygons. The second way to interpret the convex map

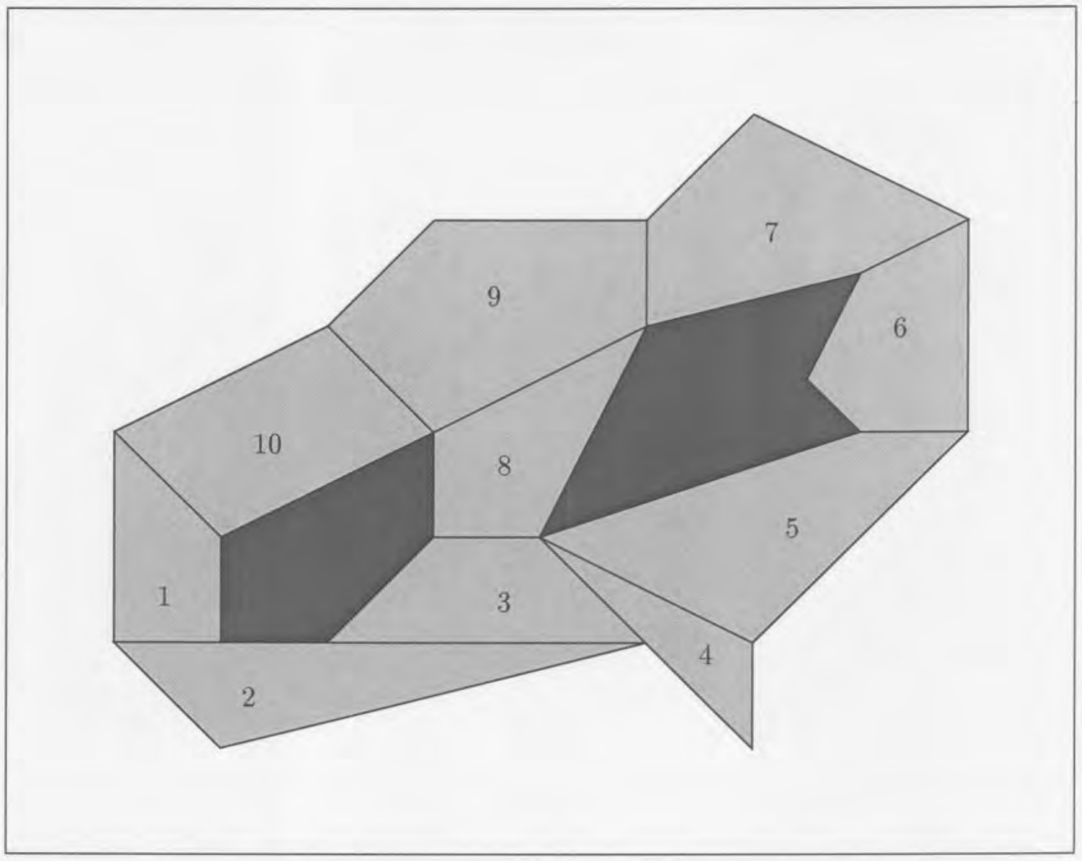

Figure 2.23: A minimum partition with convex polygons

definition is that it is a *partition* of the space into the minimum number of convex polygons. In this case the convex polygons are not allowed to overlap. Figure 2.23 shows a minimum partition of the original polygon into convex polygons. There are other partitions which are also made up of 10 convex polygons but there are no partitions with fewer than 10 such polygons.

In this research the second interpretation of the convex map is adopted. Thus a convex map is a *partition* into the smallest number of convex polygons. The reason for this is that the original work by Hillier *et al.* [1983] clearly intends *partition* rather than *covering*. Mills [1992] also demonstrates that partition is intended. This means that any time the space syntax method is applied to an urban layout, the polygon with holes which represents the layout must be partitioned into the smallest number of convex polygons. This problem has been shown to be NP-Hard [Lingas, 1982] which means that in general the problem cannot be solved exactly in polynomial time and so approximations are required.

Once a convex map (exact or a reasonable approximation) has been found, the next step in space syntax is to place the axial lines which cross all of the shared edges between the convex polygons in the partition. Figure 2.24 shows a placement

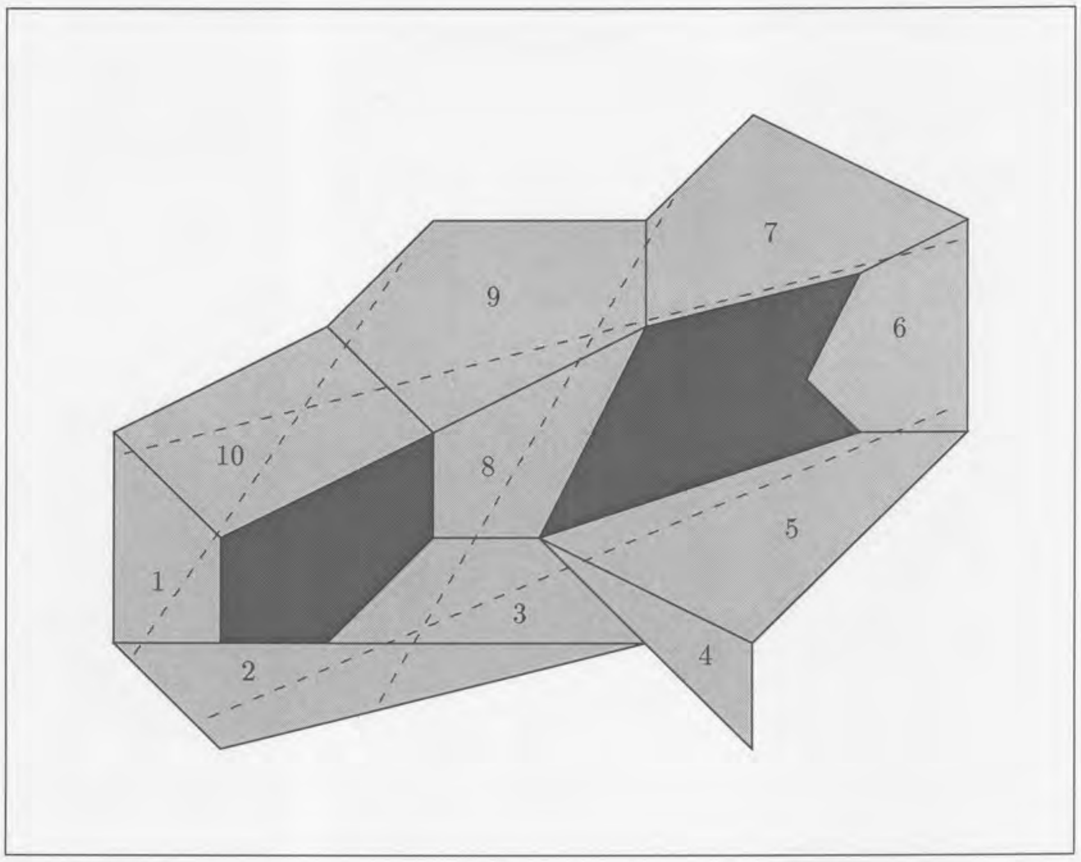

Figure 2.24: An example of placing a minimum number of maximal axial lines

of the minimum number of maximal axial lines to cross all of the shared edges between the convex polygons of the partition in Figure 2.23.

The focus of this research is in finding ways of placing the minimum number of axial lines to cross all of the shared edges in the minimum partition of some urban layout. This is a problem which has not been tackled elsewhere but the problem does benefit from research in the area of visibility in polygons.

A problem which seems similar to axial line placement is that of *stabbing* [Pellegrini and Shor, 1992; Pellegrini, 1993, 1997]. A *stabber* is a line l that intersects every member of a collection $\mathcal{P} = \{P_1, \ldots, P_k\}$ of polyhedral sets. Figure 2.25 shows a two-dimensional simplification of the problem. In this case the problem would be to determine if one line can be found which intersects all of the squares. Here no stabber exists – 3 lines are required to stab the boxes. This problem is different from that of placing axial lines because the squares can overlap; the lines do not have to remain within the squares; and the problem is to determine whether a line exists to stab all of the squares rather than finding the minimum number of lines to stab all of the squares.

In order to determine the minimum number of axial lines, it is necessary at some

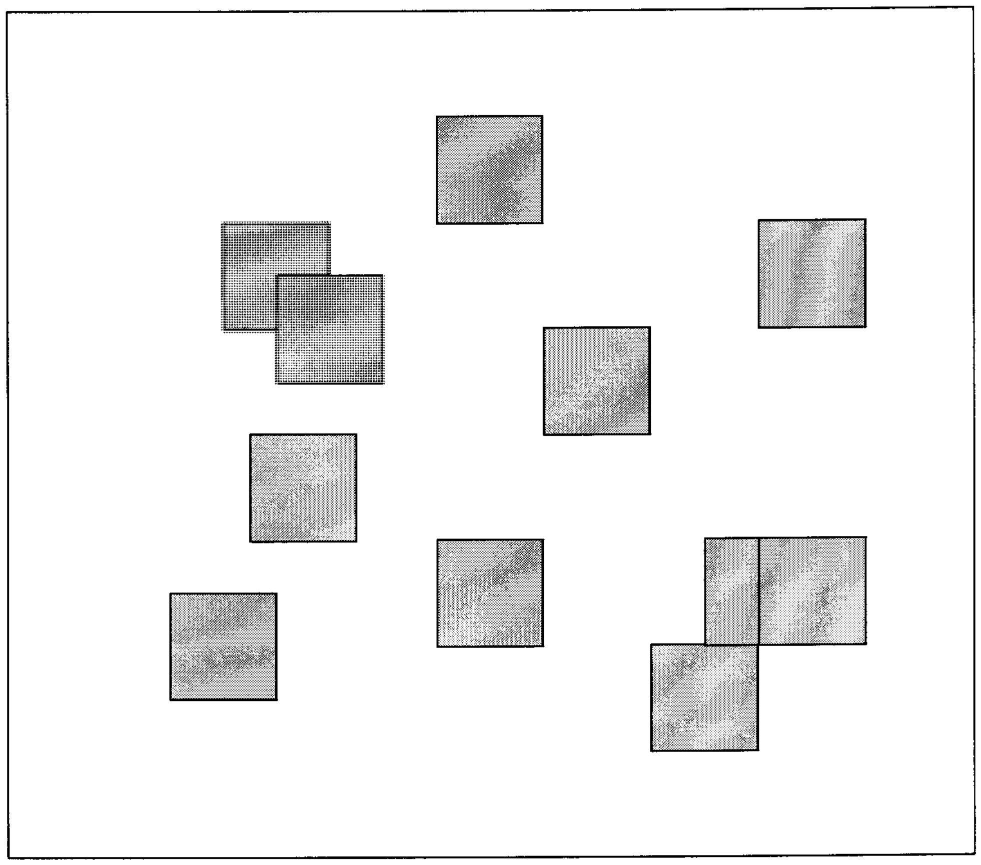

Figure 2.25: An example of stabbing boxes in two-dimensions – no stabbing line exists in this case

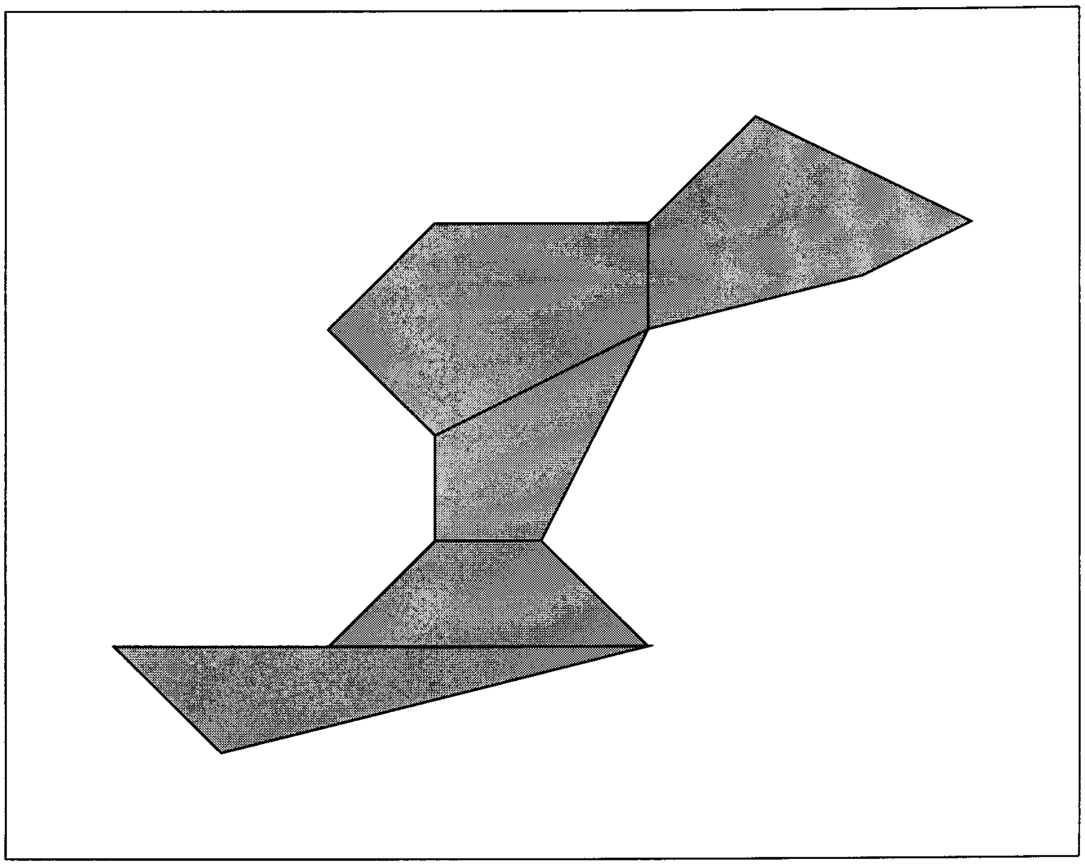

Figure 2.26: A subset of the minimum partition where it is necessary to determine if an axial line can be placed to cross the adjacencies between the convex polygons

point in the process to determine if an axial line can be placed to cross the adjacencies between any subset of the polygons in the partition. Consider the situation where it is necessary to determine whether a single axial line can be placed to cross all of the adjacencies in the subset of convex polygons as shown in Figure 2.26.

The convex polygons in this configuration can be said to form a "chain" of polygons. ***ALP*** requires that any axial line to cross the adjacencies in this chain would have to remain inside the union of the convex polygons concerned. Clearly this problem could be solved by determining if there exists a new line segment which intersects all four of the line segments representing the adjacencies and is always inside the union of the five convex polygons concerned. Solving this problem can also be posed as a visibility problem. Can some part of the edge labeled *first* in the heavily outlined polygon of Figure 2.27 see some part of the edge labeled *last* in this same polygon? The visibility problem would give the answer "yes" if there is a line of sight from some point (or points) on edge *first* to some point (or points) on edge *last*. The definition of visibility means that any line of sight must not leave the polygon. If there is this type of visibility then an axial line could be placed from

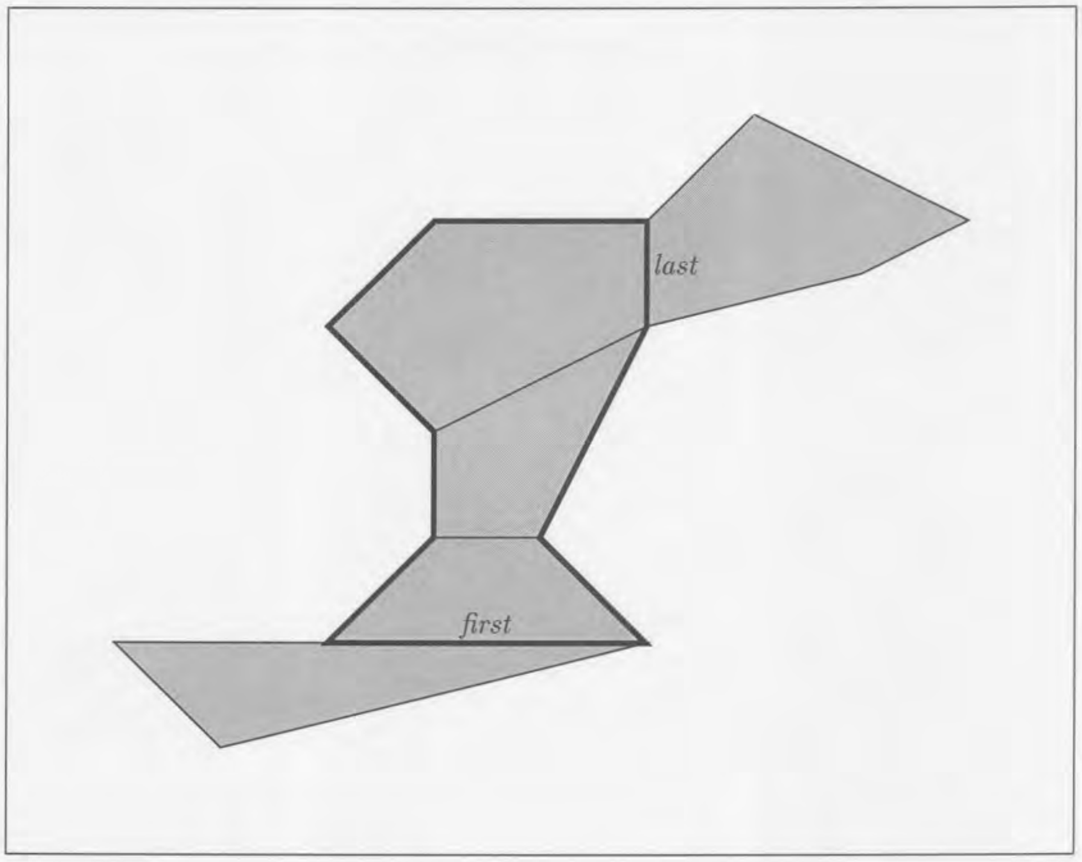

Figure 2.27: The relationship between attempting to place an axial line through a number of adjacencies and edge to edge visibility in a polygon

the first to the last adjacency in the original collection of convex polygons. The fact that the convex polygons concerned form a chain means that this line would also cross all of the other adjacencies in the chain. If there is no visibility then one axial line could not be placed to cross all of the adjacencies and a smaller subset of the original partition would have to be considered. The solution to this visibility problem, which is due to Avis *et al.* [1986], is discussed in detail in Section 2.4.4.3.1 below. Clearly all possible chains would have to be considered to find the minimum number of axial lines but these other chains can be treated in a similar fashion.

In the final analysis stage of space syntax the axial lines are transformed into a "depth graph" which gives information about how easy it is to get from one point in the layout to another point. This ease of movement is based on the number of axial lines a person in the layout would have to move along to get from the start point to the end point. Suppose that someone wanted to know how easy it was to get from point A to point B in Figure 2.28. The starting point A is in the convex polygon 1 which has axial line a passing through it. From the point of view of space syntax, all points in polygon 1 are equivalent so A can be interpreted as a point on axial

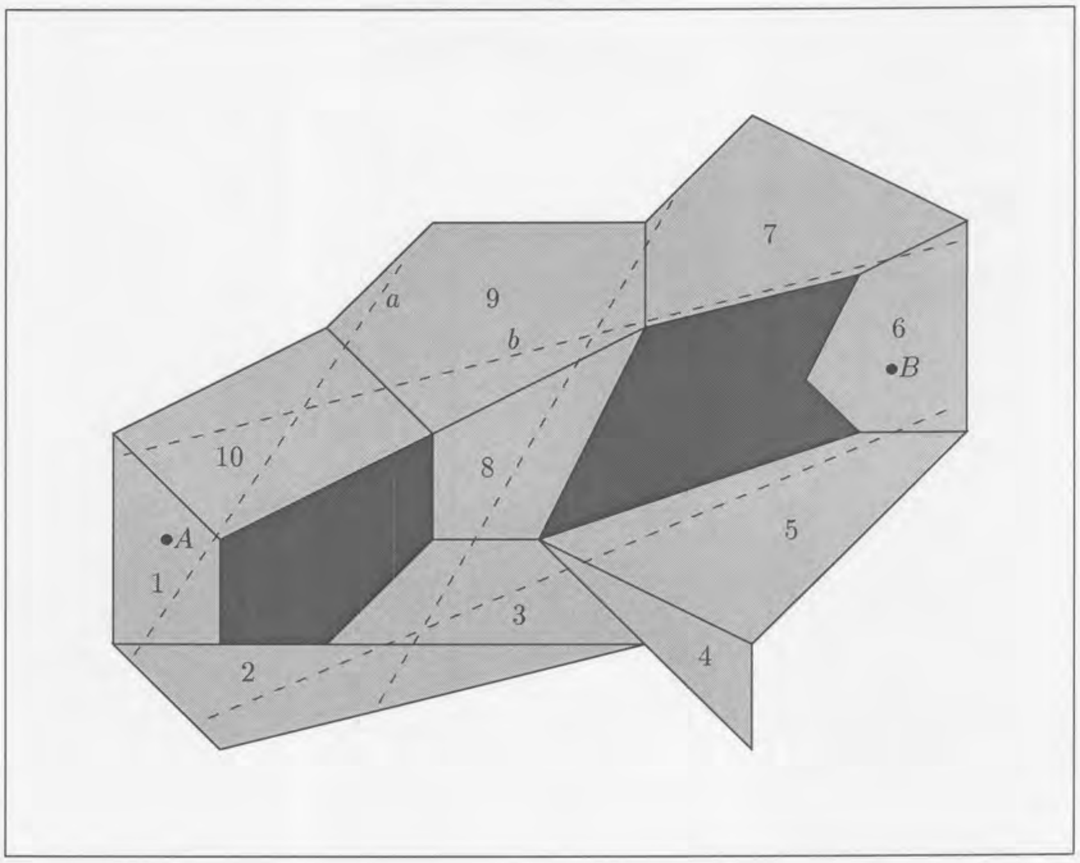

Figure 2.28: Relating L_k visibility to axial lines

line a. The person would then travel along the axial line a until it intersects axial line b. The person would then turn onto line b and travel along that line until convex polygon 6 is reached. This convex polygon contains B so the path is complete. Thus getting from A to B requires two axial lines (and one turn). As discussed earlier A and B are L_2 visible. They can be joined by a path made up of two line segments. There is thus some similarity between the placing of axial lines and determining L_k visibility between points in the polygon.

As many of the problems in the areas of guarding and polygon decomposition have been proven to be NP-hard or NP-Complete, researchers have focussed on restrictions to the general problems. Orthogonal or orthogonally convex polygons have received a lot of attention by researchers. One instance of this is the problem of guarding a *traditional art gallery* [Urrutia, 1999]. Such an art gallery is housed in a rectangular building subdivided into rectangular rooms. Any two rooms have a door connecting them. See Figure 2.29 for an example of such a gallery. In this situation, if a guard is placed in a doorway connecting two rooms then she can guard both rooms at once. Also no guard can completely guard three rooms. This means that at least $\lceil n/2 \rceil$ guards are required if there are n rooms in the gallery. Czyzowicz

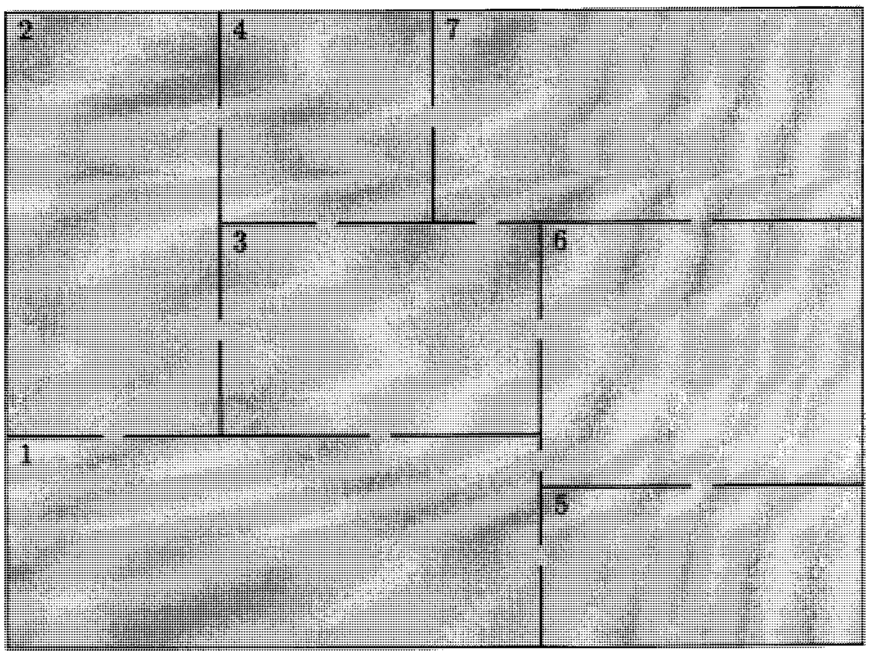

Figure 2.29: A traditional art gallery

et al. [1994] prove that any rectangular art gallery with n rooms can be guarded by exactly $\lceil n/2 \rceil$ guards.

The early work on the axial line placement problem was in some sense quite similar to the idea of guarding a traditional art gallery. This early work in ***ALP*** also focussed on a restriction of the original problem. Instead of attempting to place axial lines to cross the adjacencies between the convex polygons of some minimum partition of a polygon with holes the work focussed on placing axial lines in configurations of adjacent orthogonal rectangles. In the first instance the problem was further restricted by insisting that the axial lines were also orthogonal (see Chapter 4). This restriction was later relaxed and lines of arbitrary orientation were allowed (see Chapter 5). These two problems were originally termed "ray guarding". The idea was that one would place "guards" who could see along some "ray". Each ray guard could be thought of as placing a video camera (or some other monitoring device) which is fixed to point in a certain direction. In an art gallery situation, one could consider this as guarding the flow between the rooms which make up the gallery by monitoring each doorway. ***ALP*** requires that the minimum number of axial lines must be placed to cross the adjacencies between the room rectangles. The

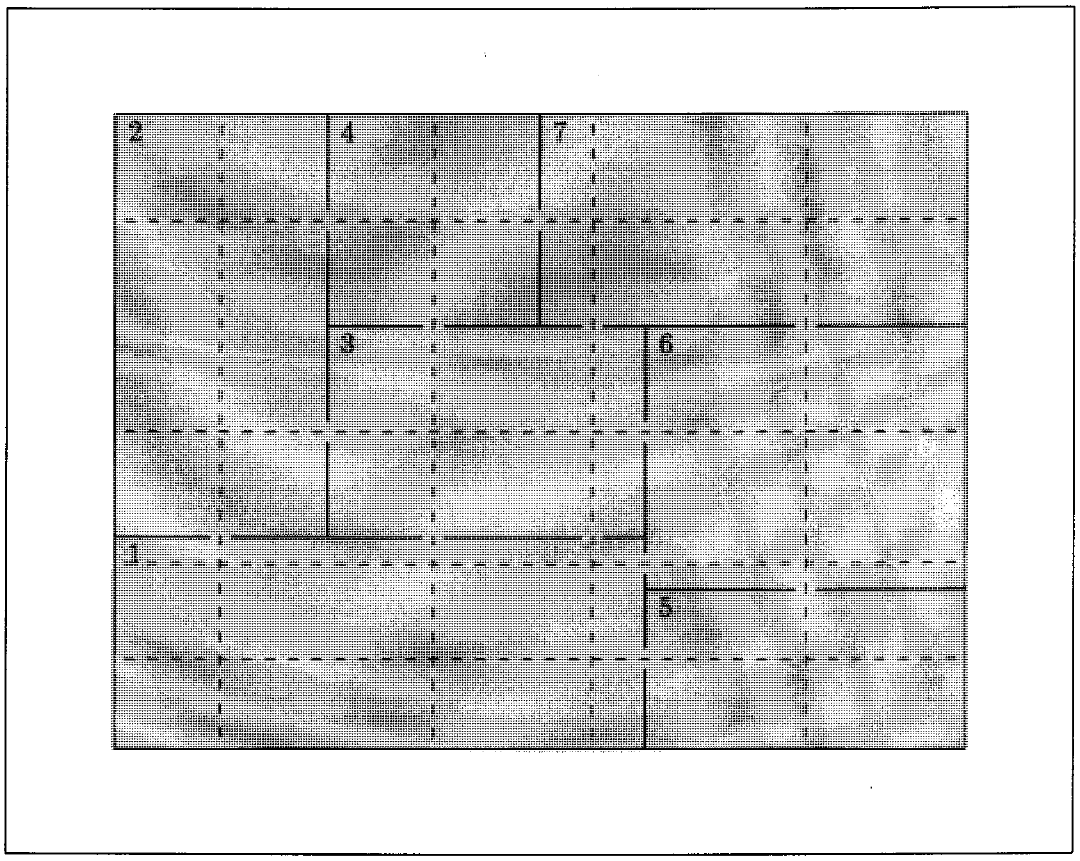

Figure 2.30: "Ray guarding" a traditional art gallery with orthogonal ray guards

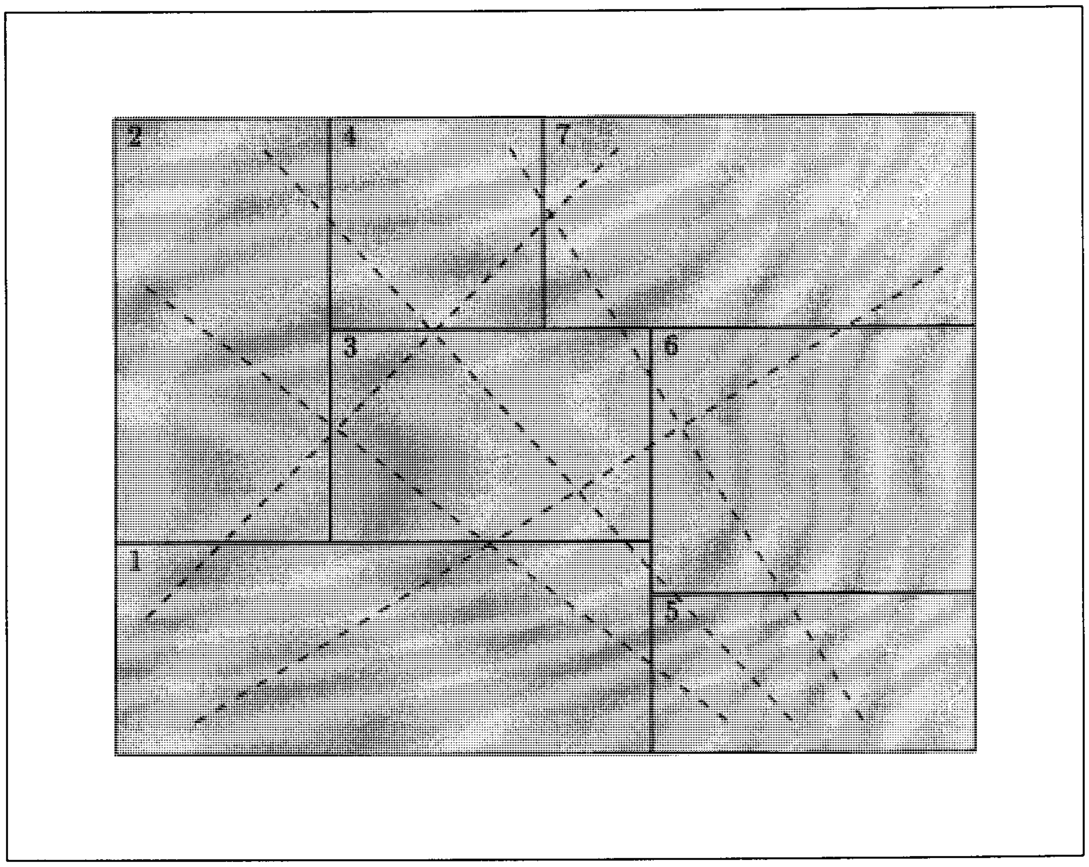

Figure 2.31: A placement of axial lines of arbitrary orientation in a traditional art gallery

point where an axial lines crosses an adjacency can be considered as an appropriate place to position a door between the two adjacent rooms. The doorway can then be guarded by a ray guard placed at the beginning or end of the axial line which crosses the adjacency. As few of these ray guards as possible are required. Placing as few axial lines as possible to cross the adjacencies is then equivalent to determining where to place the doorways between the rooms such that each doorway is guarded by a ray guard. ***ALP*** also requires that each axial line should be as long as possible. This means that any axial lines which is placed to cross a particular adjacency should be extended to cross any other adjacencies that it can even if these adjacencies have already been crossed by other axial lines. In the art gallery scenario this is equivalent to placing additional doors anywhere that a ray guard's vision is obstructed by an interior wall. These additional doorways would still be guarded by at least one ray guard. Figure 2.30 shows the traditional art gallery of Figure 2.29 guarded by ray guards whose lines of sight are restricted to being orthogonal. Note that the wall between rooms 1 and 3 has two doorways in it rather than only one. This arises because an axial line is required to cross the adjacency between rectanges 3 and 4 and an axial line is required to cross the adjacency between rooms 3 and 7. When these axial lines are extended to cross as many adjacencies as possible they both cross the adjacency between rectangles 1 and 3.

If the ray guards' lines of sight are not restricted to being parallel to the coordinate axes then a different positioning of guards results. This would correspond to a placement of not necessarily orthogonal axial lines. A placement of axial lines of arbitrary orientation is shown in Figure 2.31. This placement of axial lines could be converted to a ray guarding situation by placing a door at each point where an axial line crosses an adjacency (a wall between two rooms). Again some of the walls would have more than one doorway.

The discussion in this section has been focussed on some of the similarities and differences between ***ALP*** and the research which has been done in visibility in polygons; guarding of polygons; and decomposition of polygons. ***ALP*** has not been studied before but clearly because the problem areas are not that far removed research into solving ***ALP*** should be informed by work in the other areas. The next section discusses in more detail some specific research which has relevance to ***ALP***.

2.4.4 Results that informed the research on *ALP*

2.4.4.1 Overview

As discussed above there are some commonalities and some differences between ***ALP*** and other research areas in the field of computational geometry. There also general approaches taken and specific techniques used in this other research which informed the research undertaken in this thesis. These ideas are discussed in this section. First some of the general approaches are discussed and then some specific techniques are focussed on.

2.4.4.2 General Approaches

2.4.4.2.1 NP-Hardness results Many of the problems studied in guarding, decomposition or visibility have been shown to be NP-Hard or NP-Complete (see for example the work of Lee and Lin [1986]). As ***ALP*** seems to be a very similar type of problem to these other problems it seemed likely that ***ALP*** too could be NP-Hard. This insight informed the way that the research questions to be tackled in this research were posed. See Chapter 3 for more detail on this. It also meant that the work on ***ALP*** progressed on two fronts – attempting to find a polynomial time algorithm to solve the problem and attempting to prove that the problem was NP-Hard/NP-Complete. In the NP-Completeness proofs the same standard NP-Complete problems used in these other problems were considered (***3SAT*** [Lee and Lin, 1986] and ***vertex cover*** [Ntafos, 1986]). Garey *et al.* [1976], Garey and Johnson [1979] and Lichtenstein [1982] were also consulted.

A specific result that has direct relevance to the possible automation of the space syntax method is the fact that partitioning a simple multiply connected polygon (a polygon with holes) is NP-Hard [Lingas, 1982; O'Rourke and Supowit, 1983]. An urban layout would be modelled as a polygon with holes and this result means that a convex map for the urban layout cannot, in general, be found in polynomial time. There might be cases (urban layouts) where the convex map could be found in polynomial time and determining and describing these cases would be a useful research area. Another interesting research area would be in finding good approximations to the exact solution in reasonable time. However, since it was known that partitioning a polygon with holes is NP-Hard and less was known about the problem of placing axial lines in the urban layout, a decision was made to focus this research on the new problem rather than to extend the research on the known domain. Chapter 3 expands on this decision. Chapter 7 dicusses some work which was done in looking at special cases of urban layouts where the partition can be found in polynomial time but much more work in this area can still be done.

2.4.4.2.2 Restrictions and approximations As mentioned earlier many of the problems which are similar to ***ALP*** have been proven to be NP-Hard (or NP-Complete). This has meant that subsequent work on these problems has taken one of two routes – considering restrictions of the general problem in the hope that these problems can be solved in polynomial time or devising algorithms to find solutions which approximate the optimal ones. Both of these approaches are discussed below.

Restricting the problem A common restriction is to consider orthogonal polygons rather than general poygons. In the area of guarding, the orthogonal art gallery theorem was first formulated and proved by Kahn *et al.* [1983]. It states that $\lfloor n/4 \rfloor$ guards are always sufficient to see the interior of an orthogonal art gallery room and that in some circumstances this number of guards is necessary. Their proof is similar to that of Fisk [1978] using orthogonal comb polygons. The main idea of their proof was to partition an orthogonal polygon into convex quadrilaterals, add the internal diagonals of these quadrilaterals and then four-vertex colour the resulting graph. For orthogonal polygons with holes Shermer conjectured that $\lfloor \frac{n+h}{4} \rfloor$ vertex guards are sufficient to guard any orthogonal polygon with holes [O'Rourke, 1987]. This conjecture is still open [Urrutia, 1999].

The traditional art gallery problem – placing guards to cover rectangular rooms in a rectangular building – is another restriction of the general problem. Restricting the problem in this way has enabled researchers to prove a tight bound on the problem and to determine how to place the guards [Czyzowicz *et al.*, 1994].

The partitioning of orthogonal polygons, with and without holes, has also been studied. Much of the work done in this area is to partition orthogonal polygons into the minimum number of rectangles which generally means that Steiner points are required [Keil, 1999]. If Steiner points are disallowed (which is the case in the space syntax method) then the attention is focussed on partitioning the orthogonal polygon into quadrilaterals. Other work has focussed on partitioning orthogonally convex polygons.

In the area of visibility, work has also focussed on orthogonal polygons particularly with respect to staircase and dent visibility [Asano *et al.*, 1999].

The results in guarding, partitioning and visibility with regard to orthogonal polygons and rectangles show that in some cases these problems are easier to solve than the general problems. This indicated that studying similar restrictions in ***ALP*** would be reasonable approach to take. In particular, a configuration of adjacenct rectangles could be considered as a rectangular partition of some orthogonal polygon representing an urban layout. In ***ALP*** the shared edges between the rectangles must be crossed by the minimum number of axial lines. These axial lines could be orthogonal or have arbitrary orientation. Chapter 3 discusses these restrictions of the general problem in more detail and they are addressed in Chapters 4 and 5 respectively.

Approximations A number of the general problems in guarding, partitioning and visibility, and even a number of the restrictions of these general problems, have been shown to be NP-Hard (or NP-Complete). It is, however, still worthwhile in many instances to have an approximation to the exact solution.

In the problem of guarding an general art gallery Avis and Toussaint [1981] used Fisk's proof as the basis for implementing an $O(n \log n)$ algorithm to assign positions to the guards. Algorithms such this do not guarantee a minimum solution for a given polygon but will place a guard set which is always sufficient (and sometimes necessary) to guard the polygon. Bjorling-Sachs and Souvaine [1995] also used an upper bound proof as the basis of their algorithm to place $\lceil \frac{n+h}{3} \rceil$ guards in a polygon with n vertices and h holes. Other such results can be found in Urrutia [1999].

In partioning polygon without holes many of the early results were approximations to the exact solution (see Schachter [1978]) and it was only later that exact solutions where found [Keil, 1985]. When the polygon contains holes the problem is NP-Hard [Lingas, 1982], and so approximations are required. Keil [1999] discusses some approximation algorithms for various versions of the problem of partitioning a polygon with holes.

This thesis considers the computational geometry problems which arise out of a possible automation of the space syntax method. If ***ALP*** is NP-Hard or NP-Complete (or if variations/restrictions of ***ALP*** are NP-Hard or NP-Complete) then an automated process would not be able to generate the exact axial map of an urban area in a reasonable time and so approaches to find approximate solutions which would be acceptable to someone who wishes to apply the method would have to be found. Chapters 4 and 5 discuss some approaches for finding approximate solutions to some variations of ***ALP***.

In addition, the fact that partitioning a polygon with holes is NP-Hard means that it cannot be guaranteed that the convex map of an urban area could be found in reasonable time. Thus research into finding efficient approximations to the convex map would be a worthwhile research endeavour. This thesis does not focus on the problem of finding the convex map of an urban area but Chapter 7 introduces the idea of a "deformed urban grid" and discusses an algorithm which finds a not necessarily minimum partition of such a polygon with holes.

2.4.4.3 Specific results of importance

2.4.4.3.1 Partial edge visibility ***ALP*** is the problem of creating the axial map of an urban layout given the convex map of the layout. This involves finding the minimum number of axial lines to cross the adjacencies between the convex spaces (convex polygons) which partition an urban area (a polygon with holes). As stated previously, the placing of an axial line to cross the adjacencies in a chain of adjacent convex poloygons can be thought of as determining edge to edge visibility between the adjacency between the first two polygons in the chain and the adjacency between the last two polygons in the chain. In this case, all that is required is that some point on the first adjacency can see some point on the last adjacency. This is partial visibility – edge uv is said to be partially visible from edge xy if there exists a point w on edge uv and a point z on edge xy such that w and z are visible.

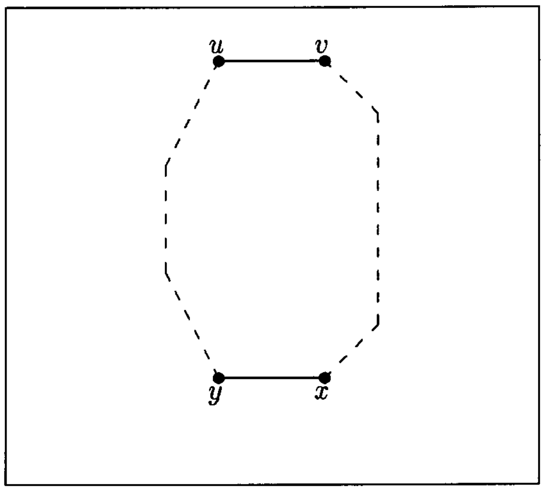

Figure 2.32: The edge to edge visibility algorithm [Avis *et al.*, 1986] – totally facing edges

Avis *et al.* [1986] give a linear time algorithm for determining edge to edge visibility in a simple polygon. This algorithm is important to this work as it can be used in determining whether axial lines can be placed in chains of convex polygons. The algorithm is used in developing the heuristics discussed in Chapter 5.

The algorithm (actually two algorithms) is described in detail by Avis *et al.* [1986]. The approach is quite complicated and so a greatly simplified overview is given below. The reader is referred to the original article for more details.

The input to the algorithm would be a simple polygon $P = (p_1, p_2, \ldots, p_n)$ and two edges in P, uv and xy. For the sake of this explanation assume that the two edges, uv and xy, are as shown in Figure 2.32. The dashed lines connecting the two edges of interest indicate that the boundary of the polygon could have any shape between the two points connected by the dashed lines. Avis *et al.* [1986] define this situation as two edges that *totally face* each other. This is the simplest situation which could occur.

The algorithm works as follows.

1. Construct the quadrilateral $Q(u, v, x, y)$
2. Construct the chains $C(v, x)$ and $C(y, u)$ of the vertices on the path from v to x and y to u respectively.
3. If $C(v, x)$ or $C(y, u)$ cuts through $Q(u, v, x, y)$ then there can be no visibility. Figure 2.33 shows an example of $C(y, u)$ cutting through $Q(u, v, x, y)$.
4. If neither $C(v, x)$ nor $C(y, u)$ cuts through $Q(u, v, x, y)$ then there can be visibility. In this case, find the reduced chains $R(v, x)$ and $R(y, u)$ by determining which parts of the chains $C(v, x)$ and $C(y, u)$ would be in $Q(u, v, x, y)$.

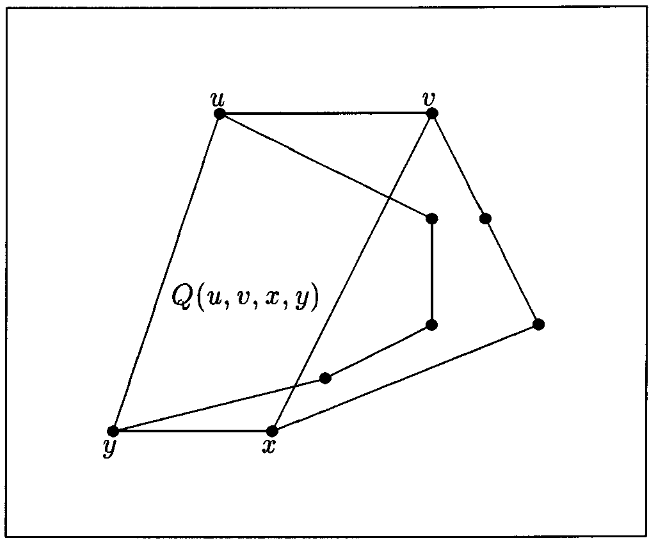

Figure 2.33: The edge to edge visibility algorithm – chain $C(y,u)$ cutting through quadrilateral $Q(u,v,x,y)$, no visibility is possible

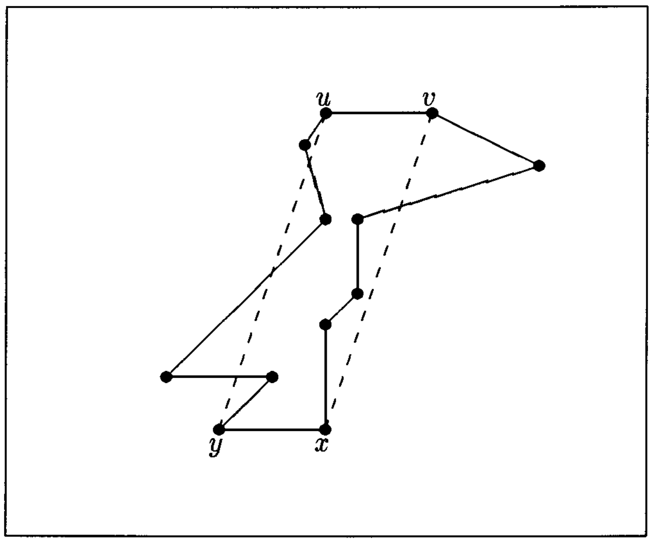

Figure 2.34: An example of the edge to edge visibility algorithm – the input polygon and $Q(u,v,x,y)$

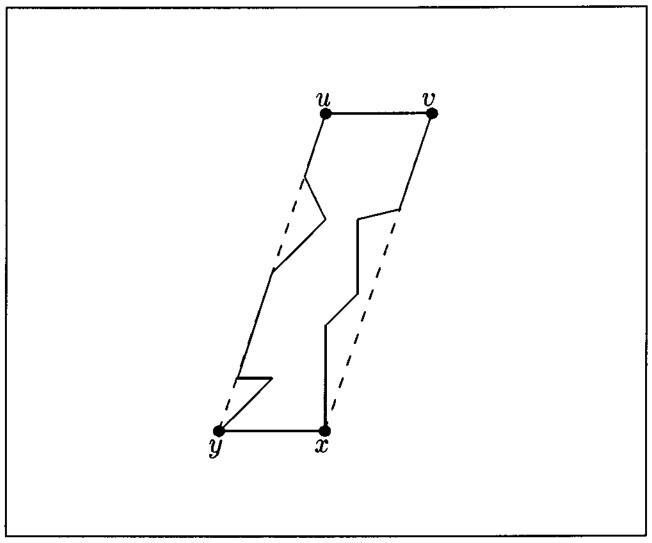

Figure 2.35: An example of the edge to edge visibility algorithm – the reduced chains

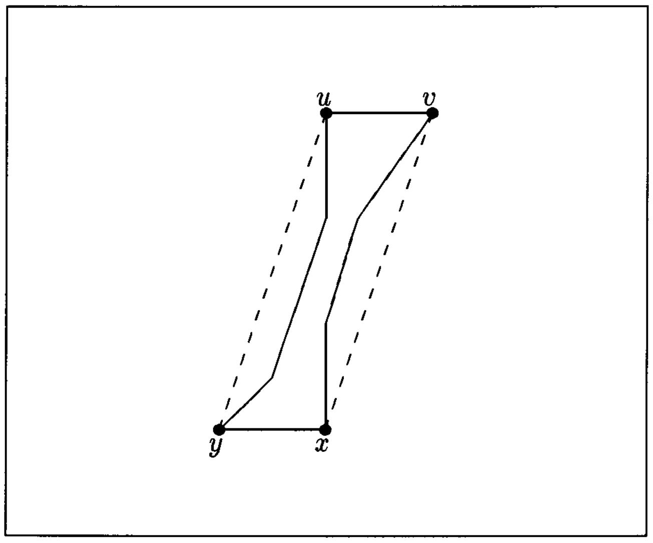

Figure 2.36: An example of the edge to edge visibility algorithm – the inner convex hulls

Figure 2.34 shows an example of an input polygon and Figure 2.35 shows the corresponding reduced chains.

5. Calculate the inner convex hulls $ICH(v,x)$ and $ICH(y,u)$ of $R(v,x)$ and $R(y,u)$. Figure 2.36 shows the inner convex hulls for the example polygon.

6. Construct the polygon $H = ICH(v,x), xy, ICH(y,u), uv$

7. If H is a simple polygon then edge uv is partially visible from edge xy and vice versa. In the example (Figure 2.36), H is a simple polygon so there is partial visibility between the edges uv and xy.

The algorithm also identifies complete visibility, strong visibility and weak visibility in the input polygon but these are not of interest here. More important is that, in the case of partial visibility, the algorithm determines which region of edge uv is visible to edge xy and *vice versa.* This gives the information about where an axial line could be placed in ***ALP***.

2.4.4.3.2 Guarding in grids Ntafos [1986] and Gewali and Ntafos [1993] define the *complete two-dimensional grid* of size n as the graph with vertex set $V = \{1,2,\ldots,n\} \times \{1,2,\ldots,n\}$ and the edge set $E = \{\{(i,j),(k,m)\} : |i-k| + |j-m| = 1\}$ where all edges are parallel to the major axes – see Figure 2.37. In a geometric setting, the grid edges can be thought of as corridors and the grid vertices as intersections of corridors. A (partial) grid is any subgraph of the complete grid. Gewali and Ntafos [1993] also define a *grid segment* as a succession of grid edges along a straight line bounded at either end by a missing edge. A *simple grid* is a grid where all of the endpoints of the grid segments lie on the outer face of the planar subdivision formed by the grid. A *general grid* is a grid which can have holes – some of the endpoints of the segments may lie on the inner face of the planar subdivision.

The star cover or star guard problem in a grid is then to find the minimum number of guards that need to be stationed in the grid so that each point in the grid is visible to some guard [Ntafos, 1986]. If the grid is complete then n guards are necessary and sufficient for a two-dimensional grid of size n. If the grid has obstacles in it – there are missing portions of the grid – then the problem becomes more interesting. Ntafos [1986] shows that a minimum cover for a grid with obstacles can be found in polynomial time by reducing the problem to that of finding a maximum mapping in a bipartite graph.

The idea of a grid is useful in ***ALP*** because part of many cities can be considered as grid-like structures. The result that the star cover can be found in polynomial time suggested that ***ALP*** might also be solvable in polynomial time. Any guard in the star cover guards at most two grid segments – one vertical and one horizontal. It

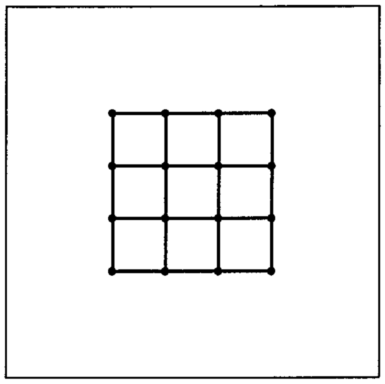

Figure 2.37: A complete grid of size 4

thus seems as though two axial lines could be placed to cover the lines of sight of each star guard. This idea is further developed in Chapter 7.

2.5 Conclusion

In Chapter 1 the idea of space syntax is introduced and the broad areas of computer science research which would arise from attempting to automate the approach are discussed. The range of research questions that could have been addressed is very wide and so it was necessary to choose a smaller area of research for this thesis. The decision was made to focus the research on the problem of finding the axial lines that cross all of the shared boundaries between the convex polygons in the convex map, that is finding the axial map for a given layout – ***ALP***. The problems of separating space from non-space, determining the convex map and the final analysis stage with its associated algorithms were not considered as part of this research.

This chapter begins by introducing some terminology in the fields of computational geometry and graph theory and giving an overview of NP-Completeness. This is done to provide the reader with a framework for understanding the summary of the research which is related to automating the space syntax method and in particular ***ALP***. This related research can be loosely categorised into three areas – guarding of polygons, visibility in polygons and polygon decomposition. This chapter gives an overview of the research in each of these related areas before addressing the commonalities and differences between ***ALP*** and guarding, visibility and polygon decomposition problems. From the dicussion above, it should be clear that ***ALP*** is a new area of research (there are significant differences between ***ALP*** and the related research). A detailed investigation of ***ALP*** is thus warranted. In addition, the fact that many of the related problems are computationally hard indicates that solving this problem or making progress to solving this problem would be a significant research contribution.

The last section of the chapter identifies some related research that informed that research into ***ALP***. First, the fact that many of the problems are NP-Hard or NP-Complete seemed to have relevance for the research in ***ALP***. The fact that these problems are hard means that finding ways of restricting the general problem or of producing reasonable approximate solutions are viable areas of research. This might apply in the case of ***ALP*** as well. Second, some specific results were identified as having direct relevance to this research. The section concludes by discussing two such specific results – partial edge visibility and guarding in grids.

Chapter 3 considers the possible research questions which arise from the decision to focus the work for this research on ***ALP***. The approach taken in posing these questions is based on the related work presented in this chapter.

Chapter 3

Research Questions

3.1 Possible Research Areas

As discussed in the introduction to this thesis (Chapter 1), the range of possible Computer Science research questions that arise from the idea of automating space syntax is very wide. These research questions fall into a number of areas:

- using image processing techniques to separate space from non-space in the town plan or aerial photograph and then using polygon approximation algorithms to "accurately" and "efficiently" represent each area by a bounding polygon,
- studying ways to find the convex map (the minimum number of non-overlapping convex polygons that cover the area of the spaces in the deformed grid) of the area under consideration,
- studying ways to find the axial map (the smallest number of axial lines that will cross all of the shared boundaries between the convex spaces in the convex map) of the area, and
- studying the graph theory and other algorithms used in the final stages of applying space syntax.

This breadth of possible research meant that it was thus necessary to concentrate on a subset of the problems. For this reason, the problems of separating space from non-space, determining the convex map and the final analysis stage with its associated algorithms were not considered as part of this research. The decision was made to focus the research for this PhD on the problem of finding the axial lines that cross all of the shared boundaries between the convex polygons in the convex map, that is finding the axial map for a given layout. This problem has not been previously studied although it is a variation on a number of guarding and visibility problems that have been well studied in the literature (see Chapter 2).

The problem which is the focus of this thesis can then be stated as
Axial Line Placement (ALP): Given a convex map of an urban area determine the axial map required to cover the convex map.

This can be restated as
Axial Line Placement (ALP): Given a collection of adjacent convex polygons representing the convex map of an urban area, find the minimum number of maximum length straight line segments contained wholly inside the convex polygons (axial lines) that will cross every adjacency (shared edge) between the polygons.

This problem has two variations

multiple crossings where each adjacency must be crossed by *at least* one axial line.

single crossing where each adjacency must be crossed by *exactly* one axial line.

Neither of these two variations of ***ALP*** have been previously studied and both were thus candidates for the research in this thesis. The fact that many of the guarding and covering or partitioning problems that appear in the literature have been shown to be NP-Complete or NP-Hard indicates that these problems might also be. The focus of research on the problems should thus be on determining if they are also NP-Complete or NP-Hard. If the problems turn out to be NP-Complete or NP-Hard then later research effort could be focussed finding good heuristics. If they can be solved in polynomial time then later research effort could be focussed on finding good algorithms to solve the problems.

The literature also shows that it is worthwhile studying restrictions of the more general problems because even if the more general problem is NP-Complete or NP-Hard a restriction may not be. A number of restrictions of ***ALP*** could thus be considered as potential research areas. In addition, the problem as stated above assumes that the configuration of adjacent convex polygons represents the convex map of some urban layout. If the configuration of convex polygons does not have to represent an urban layout then a slightly different problem results.

A list of some restrictions or generalisations of the original problem that would be interesting areas of research is given below.

1. Restricting the problem to dealing with orthogonally aligned rectangles rather than general convex polygons and restricting the axial lines to be parallel to the edges of the rectangles. In this case there are two subproblems (see Figure 3.1)

   (a) where each adjacency must be crossed by *at least* one axial line.

   (b) where each adjacency can be crossed by *only* one axial line.

A requirement here is that each axial line is maximal in length. In this work maximal is taken as meaning that the axial line crosses as many adjacencies as possible.

2. Restricting the problem to dealing with orthogonally aligned rectangles where the adjacencies between rectangles are cut by axial lines that are not necessarily orthogonal.

   (a) where each adjacency must be crossed by *at least* one axial line.

   (b) where each adjacency can be crossed by *only* one axial line.

   Again a requirement is that each axial line is maximal.

3. Considering the problem of general convex polygons (not restricted to the form that would occur in the town planning domain) where the adjacencies between polygons are crossed by axial lines that are not necessarily orthogonal.

   (a) where each adjacency must be crossed by *at least* one axial line.

   (b) where each adjacency can be crossed by *only* one axial line.

   Again each axial line must be maximal.

Each of these variations of ***ALP*** poses some interesting questions. Is the general problem solvable in polynomial time? Are only specific instances solvable in polynomial time? Is the problem NP-Complete? If so, are there heuristics that offer acceptable solutions?

The actual problems tackled in the research reported in this thesis are discussed in the next section of the document.

3.2 Scope of this thesis

As discussed above there are many problems in this area that could be addressed but tackling all of them would be too much for a single PhD thesis. Thus, this research only considered some of the problems listed in Section 3.1. In this thesis only the variations of the problems where adjacencies may be crossed by more than one axial line (**multiple crossings** above) was studied. The decision to restrict the study to this case was based on the fact that the original urban design problem allows multiple crossings of adjacencies. The problems actually tackled are discussed below.

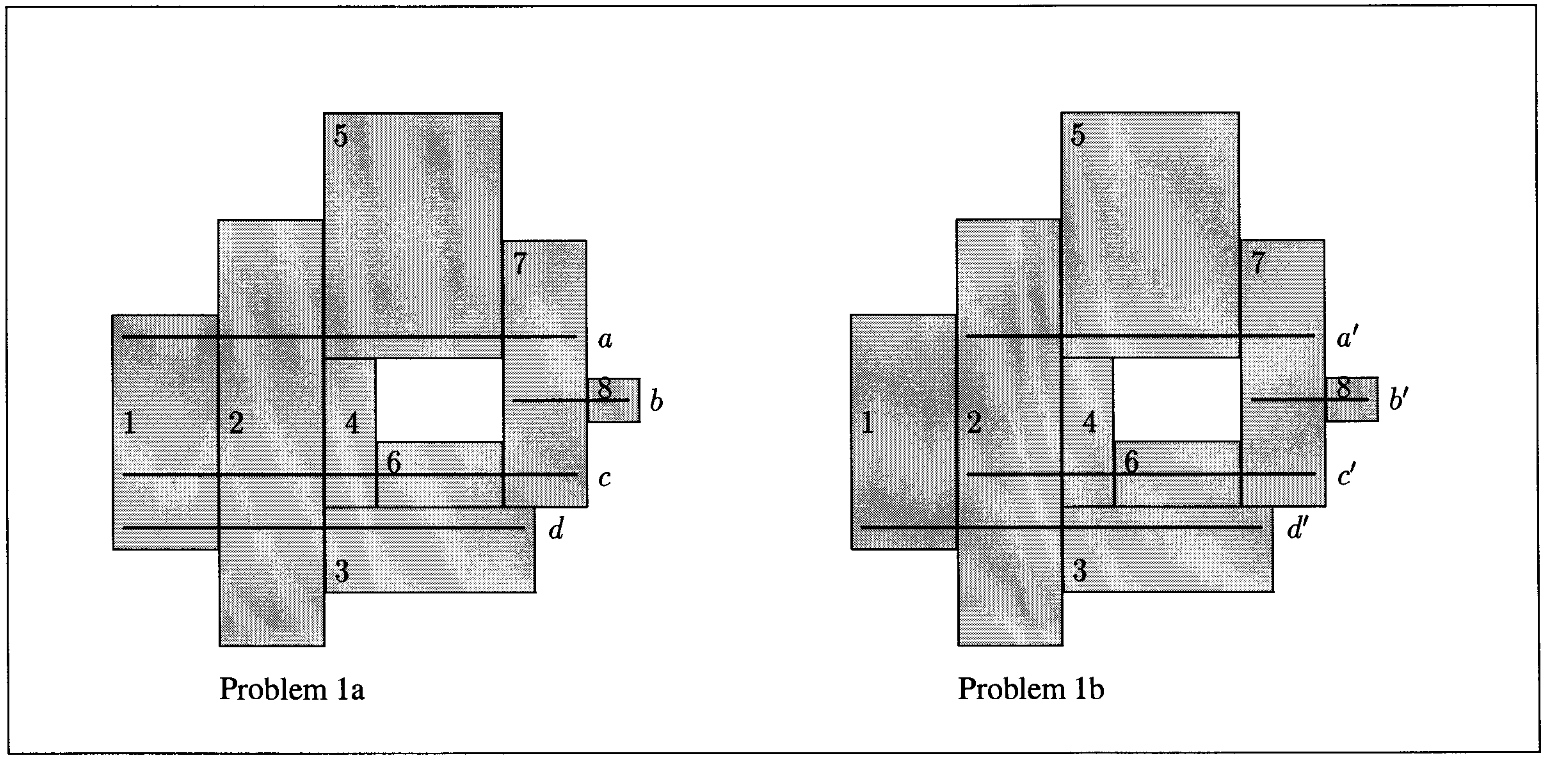

Figure 3.1: A simple configuration showing the two different problems

1. Restricting the problem to dealing with orthogonally aligned rectangles where the adjacencies between rectangles are crossed by axial lines that are restricted to being parallel to the edges of the rectangles (orthogonal) and where each adjacency must be crossed by at least one axial line (1a above).

2. Restricting the problem to dealing with orthogonally aligned rectangles where the adjacencies between rectangles are crossed by lines with arbitrary orientation and where each adjacency must be crossed by at least one axial line (2a above).

3. Considering the problem of general convex polygons (not restricted to the form that would occur in the town planning domain) where the adjacencies between polygons are crossed by axial lines with arbitrary orientation and where each adjacency must be crossed by at least one axial line (3a above).

4. Given a convex map of an urban area determine the minimum number of axial lines required to cover the convex map.

The major emphasis of this thesis is on the first of these problems – orthogonal axial lines crossing the adjacencies between rectangles in configurations of adjacent orthogonal rectangles (see point 1 above). Chapter 4 presents an NP-Completeness proof based on a transformation from vertex cover for planar graphs to show that the problem is NP-Complete and then presents a heuristic algorithm to give a "reasonable" approximation to the exact solution. The chapter concludes by discussing some special cases of the problem that can be solved in polynomial time.

In Chapter 5 the problem of point 2 is considered. Again the problem is shown to be NP-Complete. Future work here would be to develop heuristic algorithms to produce good approximations to the solution. Some heuristics are suggested but they are not fully developed in this research.

Chapter 6 considers the problem in point 3. This problem is a generalisation of 2a and is easily shown to be NP-Complete. Again, future work would be to develop heuristic algorithms to produce good approximations to the solution.

The original ***ALP*** problem (point 4) is a constrained version of that discussed above (point 3) and is considered in Chapter 7 of this thesis.

Chapter 4

Placing orthogonal axial lines to cross adjacencies between orthogonal rectangles

4.1 Introduction

This thesis is concerned with the computational issues that result from attempting to automate the placing of axial lines through the convex spaces in a town plan in the space syntax method [Hillier *et al.*, 1983]. This chapter and chapter 5 concern simplifications of the original problem – the placing of straight lines through collections of orthogonal rectangles. In this chapter the problem is simplified even more, only the problem of placing orthogonal axial lines – axial lines parallel to the Euclidean axis – through a collection of orthogonal rectangles is considered (problem 1a of Chapter 3).

As mentioned earlier (Chapter 2), the problem is similar to many guarding and visibility problems [Bjorling-Sachs and Souvaine, 1991, 1995; Czyzowicz *et al.*, 1994; Gewali and Ntafos, 1993] since axial lines coincide with visibility between two points. The situation can also be envisaged as an art gallery made up of a number of adjacent rooms where the designers wish to position doorways between the rooms in such a way that the minimum number of guards who can only see along a straight line (or laser beams, video cameras, etc.) is required to guard all of the doorways between rooms. To allow easy access between rooms extra doors can be added if a guard's line of sight is blocked by an interior wall. This is equivalent to making the axial lines as long as possible. Another application of this problem is in the design of integrated circuits. Here the problem is the siting of the fewest connecting strips to join all of the components on the chip.

The next section of this chapter presents a formal statement of the problem which is considered. In subsequent sections this problem is shown to be NP-

Complete (Section 4.4) and then a heuristic algorithm that produces a non-redundant solution is presented along with some results of testing this heuristic algorithm on some synthetic test data. The chapter ends with a discussion of some special cases where an exact solution can be found in polynomial time.

4.2 Statement of the Problem

Given a number of adjacent, orthogonally-aligned rectangles, find the fewest orthogonal line segments, contained wholly inside the rectangles, required to cross all of the adjacencies between adjacent rectangles. An additional requirement is that each line segment should cut as many of the adjacencies as possible – the line should be maximal.

The solutions for horizontal line segments (and vertical adjacencies) and vertical line segments (and horizontal adjacencies) are independent and the remainder of this chapter will only discuss the former. The latter problem can be solved by a rotation through 90 degrees.

Depending on how the problem is considered there are 2 similar but distinct problems that can be addressed.

1. The adjacencies between adjacent rectangles can be crossed more than once but every shared boundary must be crossed at least once.

2. Any adjacency between adjacent rectangles has exactly one orthogonal line segment passing through it.

Figure 4.1 shows the difference between these two specifications for a simple configuration of adjacent rectangles. In problem 1 the leftmost adjacency is cut by lines a, c and d. In problem 2, any of a', c' or d' could have cut the leftmost adjacency but only d' actually does. In this thesis only problem 1 is addressed.

In the remainder of this chapter (and thesis) this problem is referred to as ***ALP-OLOR*** Axial Line Placement – Orthogonal Lines and Orthogonal Rectangles.

4.3 Addressing the problem

At first glance this problem would appear to be easy to solve. Given n rectangles, the upper bound on the number of possible adjacencies is $O(n)$ and a simple lower bound for finding the adjacencies can be shown to be $\Omega(n \log n)$.

The upper bound can be easily shown by reducing the rectangles and their adjacencies to the form of a graph where the nodes in the graph represent the rectangles and the edges of the graph represent adjacencies between two rectangles. The graph generated in this way must be planar, thus the maximum number of edges (adjacencies) can be determined from Euler's formula that gives $e \leq 3v - 6$ (e the number

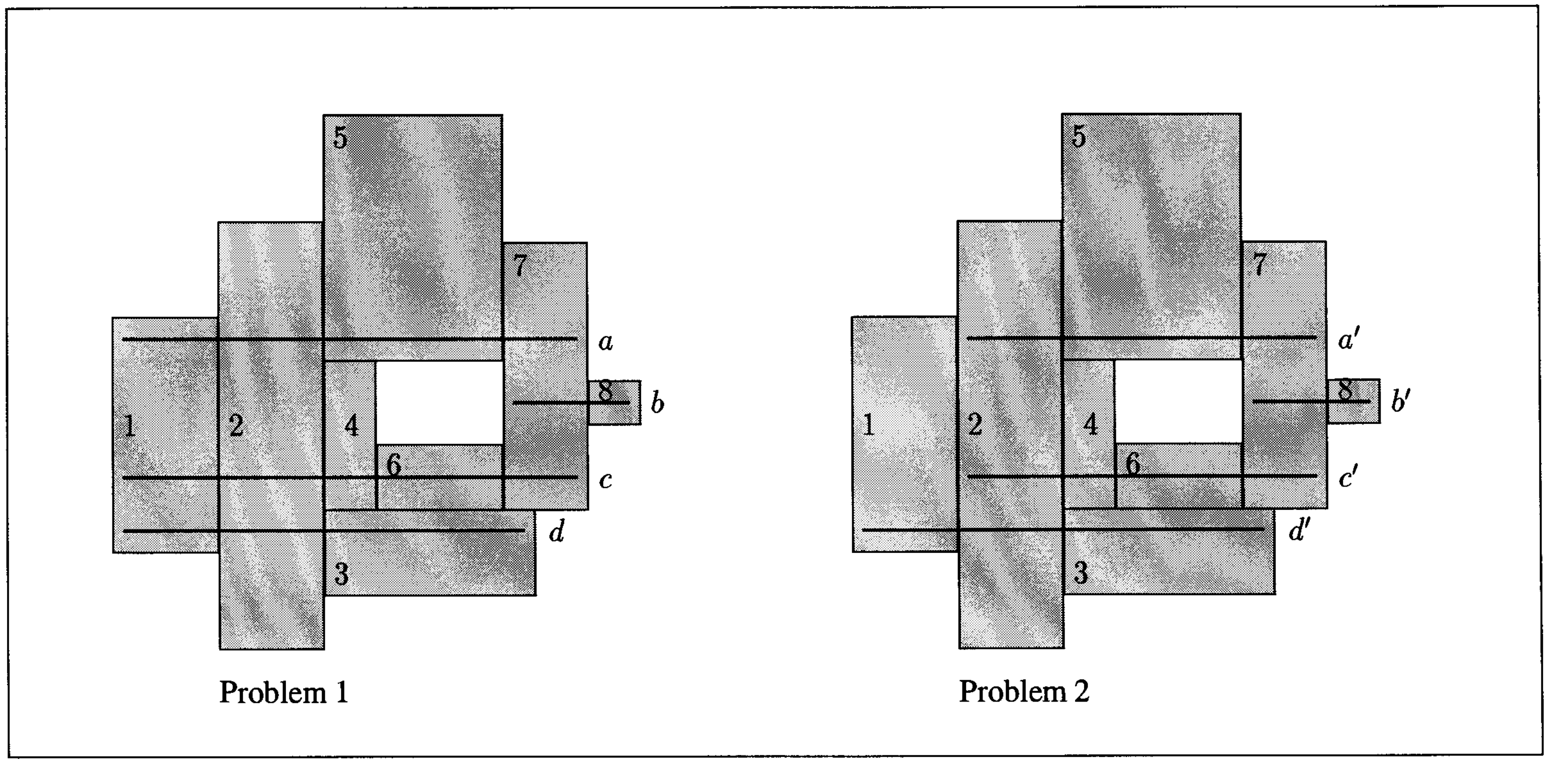

Figure 4.1: A simple configuration showing the two different problems

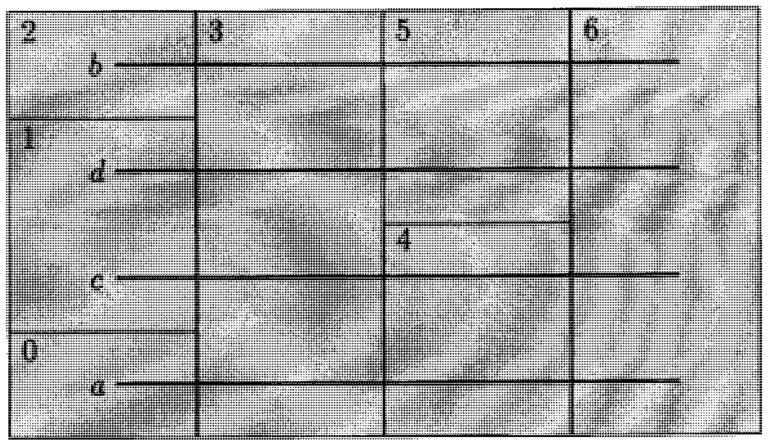

Figure 4.2: A configuration where the solution is not unique

of edges and v the number of vertices). This implies that the number of adjacencies must be $O(n)$.

The lower bound follows by a transformation from ***element uniqueness***. This proof is similar to that by Preparata and Shamos [1985] for intersecting rectangles. ***Element uniqueness*** is defined as: Given n real numbers, decide if any two are equal. ***Element uniqueness*** has an $\Omega(n \log n)$ lower bound [Preparata and Shamos, 1985]. The transformation can be done as follows. Given n real numbers $\{z_1, \ldots, z_n\}$ and an interval $[b, t]$ then for each z_i construct a (degenerate) rectangle defined by bottom left corner (b, z_i) and top right corner (t, z_i). Determining whether any two rectangles are adjacent is now the same as determining whether any two numbers are equal.

The question is then: How easy is it to find the minimum number of axial lines that cross all of the adjacencies in the collection of adjacent rectangles?

The problem is interesting and difficult to solve efficiently because of the issue of choice. The simplest case of choice is illustrated in Figure 4.2. In this case there are seven rectangles and seven adjacencies (0|3, 1|3, 2|3, 3|4, 3|5, 4|6 and 5|6) that must be crossed by axial lines. All but one of the adjacencies can be crossed by the axial lines marked a and b (0-3-4-6 and 2-3-5-6) but the adjacency between rectangles 1 and 3 can be crossed by axial lines c (1-3-4-6) and d (1-3-5-6). Only one of these "choice" axial lines is actually necessary. More complicated choice situations can arise as the number of rectangles to be considered grows. An algorithm to solve the problem must be able to resolve conflicts of this type.

4.4 Proving NP-Completeness of the problem of resolving choice

This section shows ***ALP-OLOR*** is NP-Complete. The proof of this will be accomplished through a transformation from ***vertex cover*** for a planar graph [Garey and Johnson, 1979; Lichtenstein, 1982], to a restricted instance of the problem under consideration, i.e. the problem of choosing the fewest maximal axial lines to cross all the adjacencies in a collection of orthogonal rectangles.

Planar vertex cover is defined as

planar vertex cover
Instance: Planar graph $G = (V, E)$, positive integer $K \leq |V|$.
Question: Is there a vertex cover of size K or less for G, i.e. a subset $V' \subseteq V$ with $|V'| \leq K$ such that for each edge $\{u, v\} \in E$ at least one of u and v belongs to V'?

and ***ALP-OLOR*** can be stated as

ALP-OLOR
Instance: A collection of orthogonal rectangles $R_1 \dots R_n$, where each R_i is adjacent to at least one other rectangle, and a positive integer $O \leq 4n$.
Question: Is there a set P of orthogonal axial lines where each axial line is maximal, each vertical adjacency is crossed at least once by the axial lines in P and $|P| \leq O$?

The transformation from ***planar vertex cover*** [Garey and Johnson, 1979; Lichtenstein, 1982] will be done by mapping vertices in a planar graph to choice axial lines in the problem being considered. Edges in the planar graph will be mapped to adjacencies that are crossed by the choice axial lines. In this mapping an edge between two vertices represents an adjacency that is crossed by two choice axial lines.

This transformation will be done in two steps. First, a planar graph is transformed to a 'stick diagram'. In this 'stick diagram' each vertex in the original graph is mapped to a horizontal line representing a choice axial line and each edge in the original graph is mapped to a vertical line that is cut by the two horizontal lines that represent the two vertices to which the edge is incident. The problem then becomes that of choosing the minimum number of horizontal lines to cut all of the vertical lines.

stick diagram
Instance: A collection H of horizontal lines and U of vertical lines such that each vertical line is cut by exactly two horizontal lines, and a positive integer $S \leq |H|$.

Question: Is there a set of horizontal lines, $H' \subseteq H$, such that every vertical line in U is cut at least once and $|H'| \leq S$?

The reader should note that the idea of converting planar graphs to horizontal and vertical lines has been published elsewhere [Tamassia and Tollis, 1986]. At the time of developing this proof the author of the thesis was not aware of the work of Tamassia and Tollis [1986] whose results are quite similar to those developed in this proof. The transformation below is presented as originally developed because it guarantees that the 'stick diagram' developed by the transformation is in the right form for the second part of the proof. This comment is expanded upon after the transformation process has been presented.

In the second step of the transformation from a planar graph to a configuration of adjacent rectangles, the stick diagram is represented as a collection of adjacent rectangles and horizontal axial lines crossing all of the adjacencies in the collection of rectangles. These axial lines will be of two types "essential lines" which are the only lines to cross a particular adjacency and "choice lines" where a number of lines (none of which are essential) cross some adjacency. Not all of the choice lines are necessary to cross all of the adjacencies in the collection of rectangles. If it is possible to determine in polynomial time a minimal subset of choice lines to cross all of the adjacencies not crossed by essential lines in the diagram then it is possible to solve ***planar vertex cover*** in polynomial time – finding the minimum set of choice axial lines is equivalent to finding the minimum vertex cover of the original graph.

Proving that ***ALP-OLOR*** is NP-Complete is accomplished by means of two theorems – 4.4.1 that shows that ***stick diagram*** is NP-Complete using a transformation from ***planar vertex cover*** and 4.4.2 that shows that ***ALP-OLOR*** is NP-Complete using a transformation from ***stick diagram***.

It is, however, easier to perform the transformation from a planar graph to a stick diagram for a somewhat more restricted form of planar graph – a biconnected planar graph has properties which can be used in the transformation. Therefore it is desirable to prove one other result – vertex cover for a biconnected planar graph is NP-complete. Once it has been shown that this result holds the transformation from ***biconnected planar vertex cover*** to ***stick diagram*** and hence to ***ALP-OLOR*** can be done more easily. This result is addressed in Lemma 4.4.1. The construction presented here is an improvement of a construction first published in Sanders *et al.* [1995] and Sanders *et al.* [1997]. This version of the construction first appeared in Sanders *et al.* [1999] and is similar to that developed by Biedl *et al.* [1997] in proving that vertex cover in cubic triconnected planar graphs is NP-Hard.

biconnected planar vertex cover
Instance: Biconnected planar graph $G = (V, E)$, positive integer $B \leq\mid V \mid$.

```
$G' \leftarrow G$
$j \leftarrow 1$
WHILE there exists a cut vertex $v_j$ in $G'$
    Let $X_j$ and $Y_j$ be any two components created by removing
    $v_j$ from $G'$
    Let $x_j \in X_j$, $y_j \in Y_j$ be such that $x_j$, $y_j$ and $v_j$ lie on a
    face
    Add a triangle graph, $T_j$, to the face in $G'$ containing
    $x_j$, $y_j$ and $v_j$
    Add the edges ($a_j$,$x_j$) and ($b_j$,$y_j$) to $G'$
    $j \leftarrow j+1$
```

Figure 4.3: Creating a biconnected planar graph

Question: Is there a vertex cover of size B or less for G, i.e. a subset $V' \subseteq V$ with $|V'| \leq B$ such that for each edge $\{u, v\} \in E$ at least one of u and v belongs to V'?

Lemma 4.4.1 ***biconnected planar vertex cover*** *is NP-Complete*

Proof

Clearly ***biconnected planar vertex cover*** is in NP. Given a set of vertices V' such that $|V'| \leq B$ it is possible to check in polynomial time that $|V'|$ is a vertex cover.

Now transform ***planar vertex cover*** to ***biconnected planar vertex cover***. Given a planar graph $G(E, V)$, G can be converted to a biconnected planar graph G' using the algorithm given in Figure 4.3. This is accomplished by appropriate addition of instantiations of "triangle graphs" T_j where each T_j is defined as the vertices a_j, b_j and c_j and the edges (a_j, b_j), (b_j, c_j) and (c_j, a_j). Clearly G' is a biconnected planar graph since

1. after each iteration of the algorithm the vertex v_j is no longer a cut vertex with respect to X_j and Y_j and

2. no new cut vertex is ever added during an iteration of the algorithm.

Thus the algorithm terminates with G' free of all cut vertices. In addition, the triangle graphs T_j that are added during each iteration are added to the face containing the vertices to which they are connected thus maintaining the planarity of the graph. Refer to Figure 4.4 for an example of a triangle graph and to Figure 4.5 for an example of how triangle graphs can be added to a planar graph using the algorithm in Figure 4.3 in order to derive a biconnected planar graph.

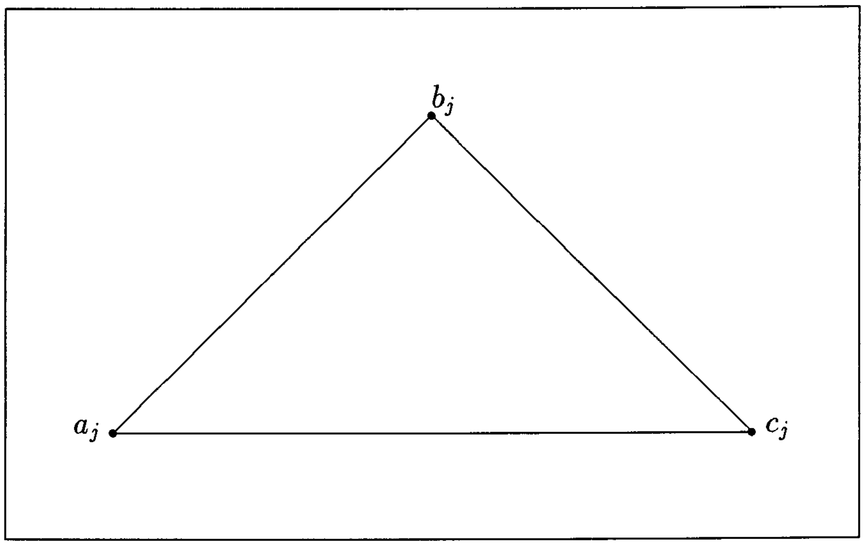

Figure 4.4: An example of a “triangle graph”, T_j

To determine the vertex cover for G', the original graph G plus the new edges and vertices must be considered. The structure of the triangle graphs means that for each triangle graph, T_j, exactly two of a_j, b_j and c_j must be in the vertex cover of G'. If k triangle graphs are added then it is trivial to show that ***planar vertex cover*** for graph G with cover size K is true if and only if ***biconnected planar vertex cover*** is true for G' with cover size $B = K + 2k$.

The transformation from G to G' can clearly be accomplished in polynomial time. A cut vertex (articulation point) can be found in polynomial time [Brassard and Bratley, 1996] and there are at most O(n) cuts (n is the number of vertices in G').

Therefore ***biconnected planar vertex cover*** is NP-Complete.

□

biconnected planar vertex cover can now be used to show that ***ALP-OLOR*** is NP-Complete. The proof is accomplished by means of the following two theorems – Theorem 4.4.1 and Theorem 4.4.2.

Theorem 4.4.1 ***stick diagram*** *is NP-Complete.*

Proof

Clearly ***stick diagram*** is in NP – given a set of horizontal lines H' such that $|H'| \leq S$, it is possible to check in polynomial time that every vertical line in U is cut at least once.

Now transform ***biconnected planar vertex cover*** to ***stick diagram***. If $G(V, E)$ is a biconnected planar graph then G can be embedded in the plane (G is planar) and

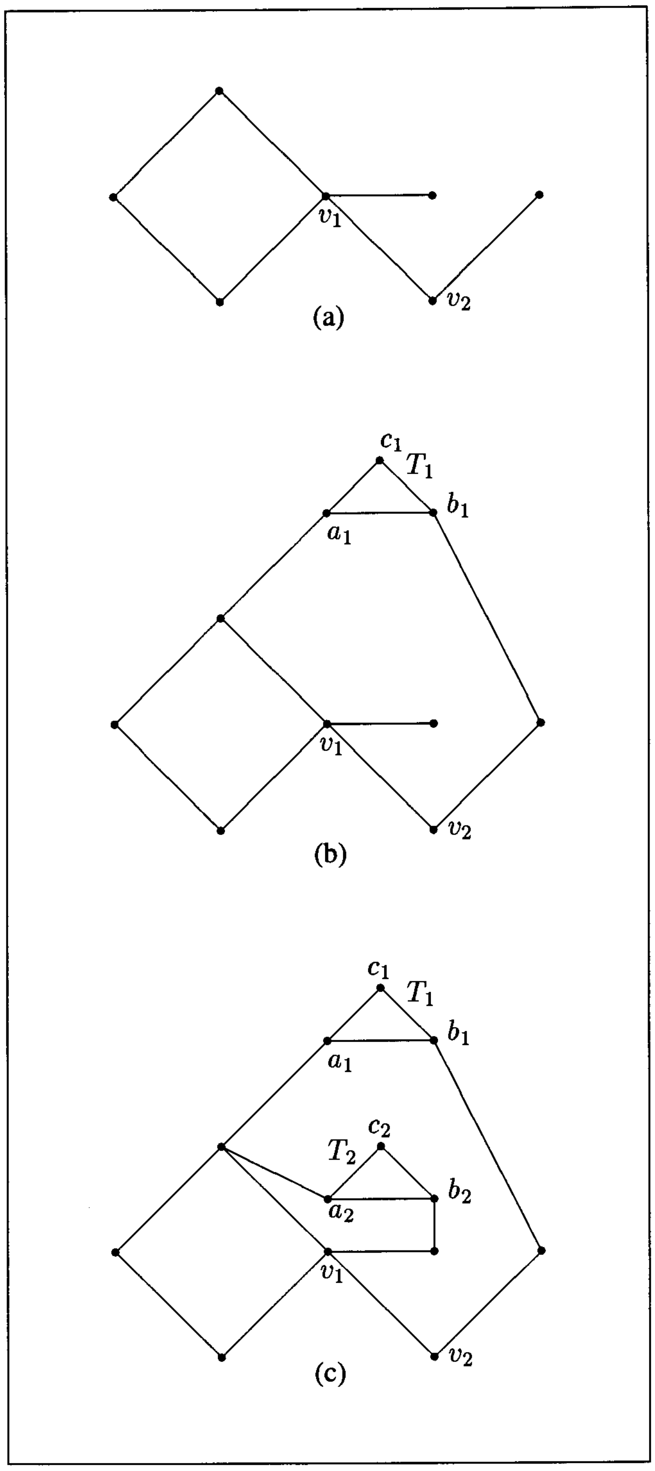

Figure 4.5: An example of adding triangle graphs to a graph to make it biconnected [(a) The original graph, v_1 and v_2 are cut vertices. (b) Graph with T_1 added, v_1 is still a cut vertex. (c) Final biconnected graph]

for every two vertices in G an elementary cycle can be found which contains these vertices (G is biconnected) [Berge, 1962; Harary, 1969]. This implies that there are no vertices of degree 1 in G (other than the case where G is a trivial graph with 2 vertices and 1 edge) and thus that all the faces in G are bounded by cycles [Harary, 1969].

Let F_0 be the exterior face of G and $F_1, \ldots, F_n$ be the interior faces of the graph. The biconnected planar graph G can be transformed to a stick diagram by the following process.

1. Choose C_s as being the cycle bounding any face, F_s of G, that is adjacent to the exterior face F_0 of G.

2. Choose any two vertices x and y joined by an edge A that form part of C_s and are adjacent to the exterior. Represent x and y by horizontal lines in the stick diagram that cut the vertical line representing edge A (see Figure 4.6 (a)).

3. Consider the path from x to y, B, formed by removing edge A from C_s. For the moment, treat B as if it were a simple edge i.e. insert its corresponding vertical line into the stick diagram. This then gives the horizontal lines x and y cutting the vertical lines A and B. The stick diagram is then as shown in Figure 4.6 (b).

4. Break the path B (which was treated as a virtual edge) into its component edges. Let the path B be the sequence of vertices $x, v_0, v_1, \ldots, v_k, y$. On the path B from x to y whenever a vertex v_i is encountered a new horizontal line must be added to the stick diagram. A new vertical line must also be added for each edge encountered. This is done in the following way. Suppose C is the edge joining x to v_0 along the path B. The stick diagram is now altered to include a vertical line for edge C and a horizontal line for vertex v_0. This is shown in Figure 4.6(c).

   In this case, B' represents the original path B minus the edge C which has been included in the stick diagram. A similar operation is applied for all the edges on the path B. Each vertex v_i maps to a horizontal line in the stick diagram and the edge joining it to the previous vertex is a vertical line cut by the two horizontal lines v_{i-1} and v_i. After all the vertices on the path B have been visited, the stick diagram will have the form shown in Figure 4.6 (d). A stick diagram which represents the originally selected closed region F_s of the original graph has now been created.

5. If G only had one interior face then the transformation is complete, otherwise continue with the next step.

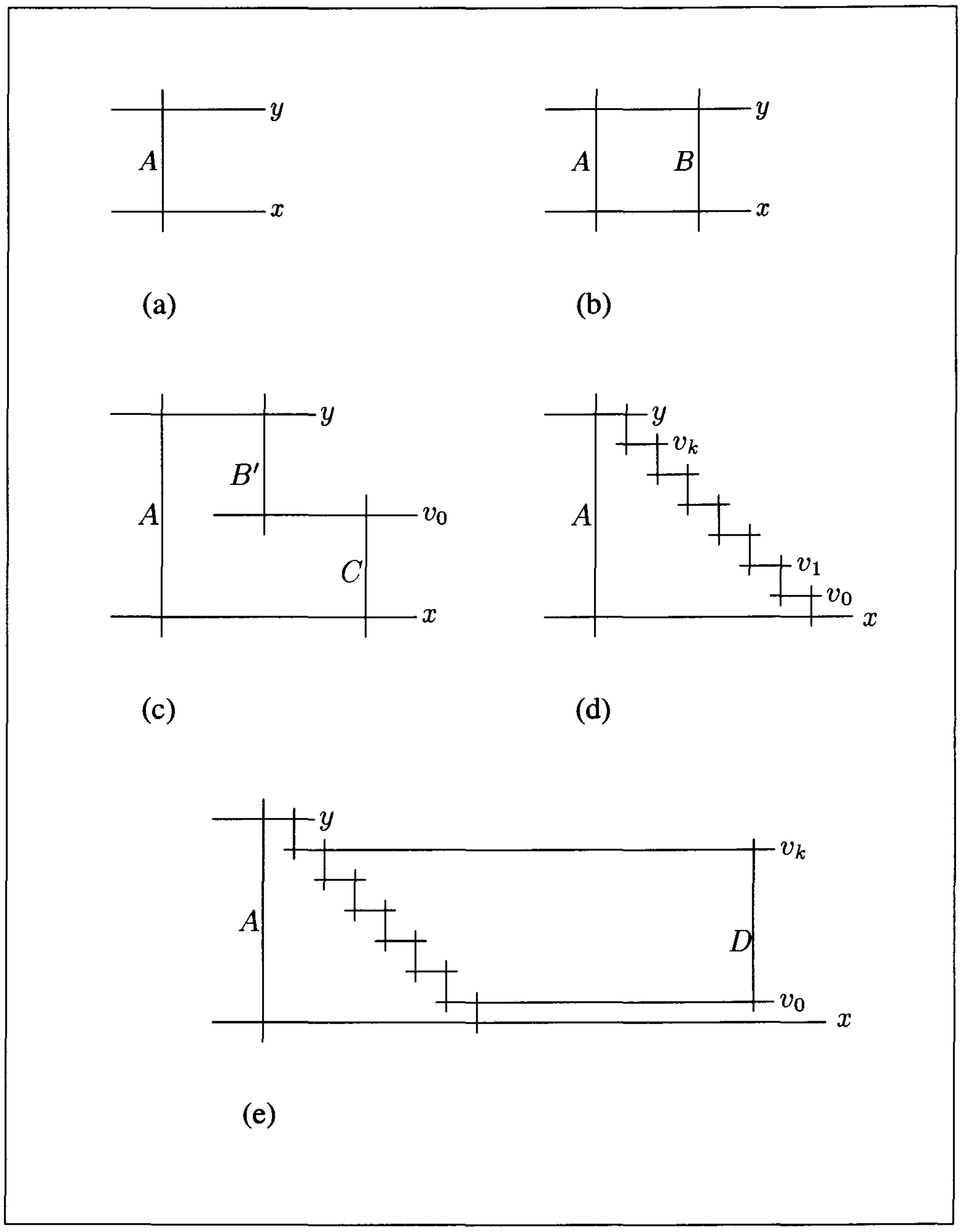

Figure 4.6: Creating a “stick” diagram

6. Let CF (which is used later in the process to store the composite of all the faces considered so far) be F_s. At this stage it will be the first face considered. That is $CF = F_s$ is the face made up of the vertices $x, y, v_0, \dots v_k$ and the edges connecting these vertices

7. While there are still faces in G to consider, repeat the following

   (a) Choose a face $F_m, 1 \leq m \leq n$ that is adjacent to CF. F_m must share a path with CF. C_m, the cycle enclosing F_m is thus made up of two sets of vertices – those that are on the shared path and have already been "visited" (included in the stick diagram) and those that have not yet been visited. The vertices in the latter set make up the path D.

   (b) Treat path D as a single (simple) edge and add it to the stick diagram by extending the horizontal lines representing the start vertex and the end vertex of the shared path to cut a new vertical line representing the path D.

   See Figure 4.6 (e) for an example – in this figure, a new path between v_0 and v_k is being added.

   The path D can then be broken up into its constituent edges in the same fashion as before.

   (c) Grow CF by combining it with F_m and removing the shared path between the faces. In the stage of the process as shown in Figure 4.6 (e), CF would consist of the vertex x, the edge (x, v_0) connecting x to the first vertex in the path D, all of the vertices and edges on the path D, the edge (v_k, y) connecting the last vertex in the path D to y, the vertex y and the edge (y, x). The path from v_0 to v_k through the vertices v_1 to v_{k-1} would have been removed.

This completes the construction of the stick diagram from a biconnected planar graph. A complete example of this is shown in Figure 4.7. First the face represented by $x - y - z - w$ is converted into a stick diagram. Then the face represented by $w - z - p$ is added to the stick diagram and finally the face represented by $w - p - z - y - q$ is added. This gives the complete stick diagram for the original biconnected graph.

Thus it can be seen that if a vertex cover, V', can be found for G then a set of horizontal lines, H', can be found for H – each vertex in G is a horizontal line in H and each edge in G is a vertical line in U. Conversely if a set of horizontal lines H', such that $|H'| \leq S$, could be found to cut each vertical line in U, then a vertex cover, V', for G could be found.

The transformation from ***biconnected planar vertex cover*** to ***stick diagram*** can be accomplished in polynomial time. Each face in the graph G is considered in turn

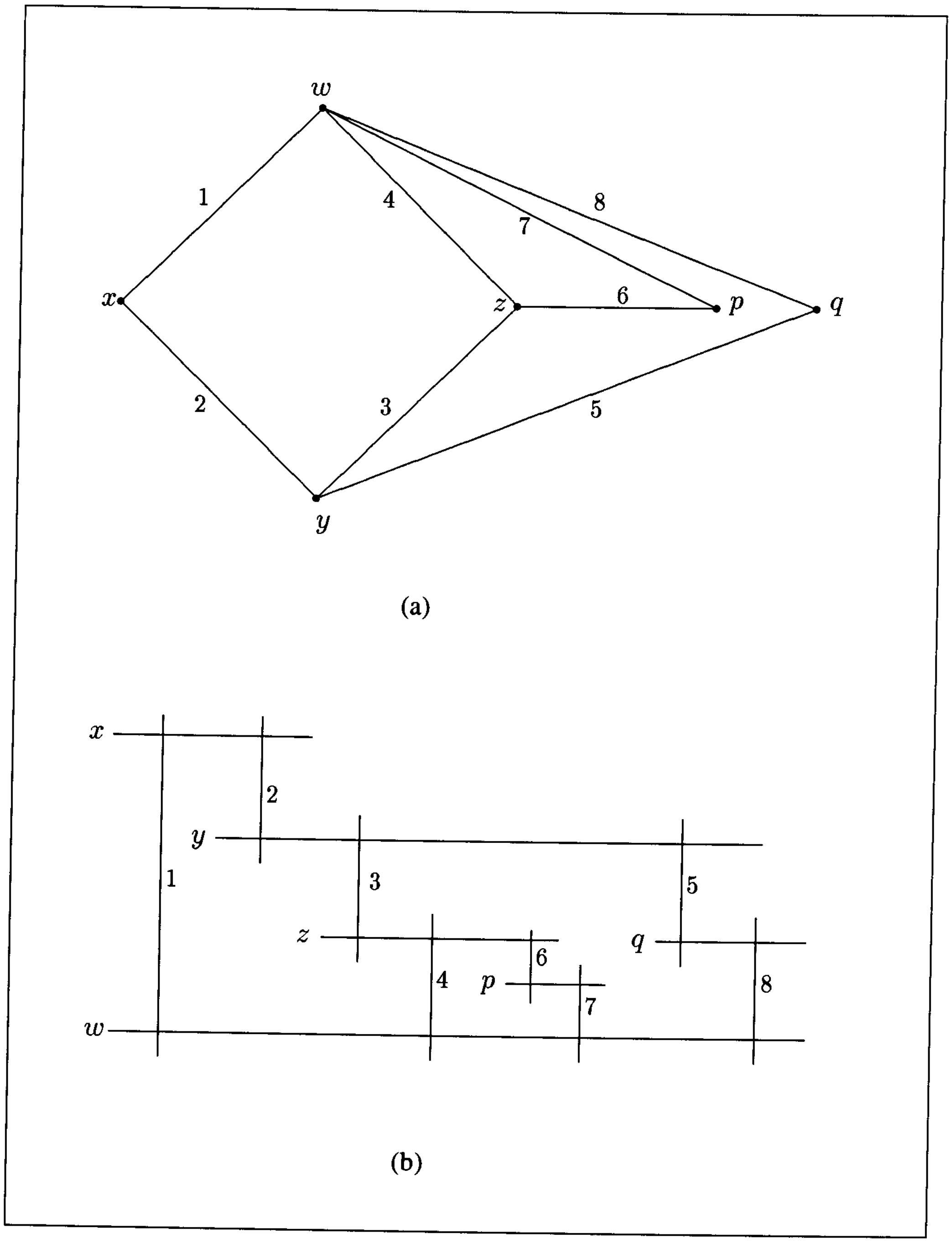

Figure 4.7: An example of the transformation of a biconnected planar graph to a stick diagram

and once only. As the face is considered each edge is added in turn to the stick diagram as a vertical line – this happens once per edge. Horizontal lines are either added to the stick diagram to represent vertices or the horizontal line representing a vertex is extended as necessary. Each vertex can only occur in as many faces as there are in the graph and each vertex in each cycle is only visited once per cycle. Thus the number of operations on vertices is limited by a polynomial expression.

Therefore ***stick diagram*** is NP-Complete.

□

As mentioned earlier Tamassia and Tollis [1986] presented an approach for transforming a planar graph into a configuration of horizontal and vertical line segments – vertices are mapped to horizontal segments and edges to vertical segments. They call this a *visibility representation* of the planar graph where two parallel segments of a set are *visible* if they can be joined by a segment orthogonal to them. They also define *weak-visibility representation* or *w-visibility representation* as a representation where vertices are represented by horizontal segments and edges by vertical segments having only points in common with the pair of horizontal segments representing the vertices they connect. The transformation above produces a representation which is a *w-visibility representation* of the planar graph with the additional property that a left-to-right ordering is imposed on the edges as they are encountered in the faces of the original planar graph (see Figure 4.7). This ordering is important in the second transformation – converting a stick diagram into a collection of adjacent rectangles.

Theorem 4.4.1 shows that a stick diagram can be constructed for any biconnected planar graph. It is now necessary to show that any stick diagram can be represented by a collection of adjacent rectangles whose adjacencies are crossed by essential and choice axial lines. This must be done in a manner that ensures consistency between a minimal selection of choice axial lines crossing rectangle adjacencies and a minimal selection of horizontal lines in the stick diagram. This is considered in Theorem 4.4.2 below. This theorem uses a construction from a stick diagram to produce a collection of adjacent rectangles in which the adjacencies between rectangles are crossed by essential axial lines and choice axial lines. The choice axial lines are directly related to the horizontal lines in the stick diagram. Not all of the choice axial lines are necessary and Theorem 4.4.2 also shows that the problem of choosing the minimum number of such choice axial lines (solving ***ALP-OLOR***) is NP-Complete.

Theorem 4.4.2 ***ALP-OLOR*** *is NP-Complete*

Proof

Clearly ***ALP-OLOR*** is in NP. Given a set of axial lines it is possible to check in polynomial time that each adjacency has been crossed by at least one axial line.

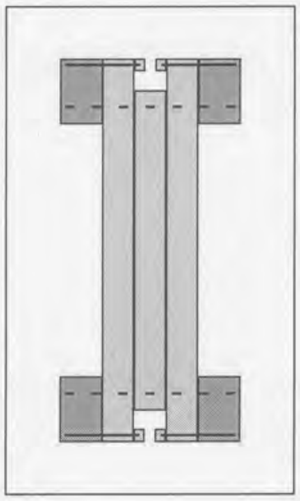

Figure 4.8: The Canonical Unit which produces two choice axial lines (shown as dashed lines)

Now transform ***stick diagram*** to ***ALP-OLOR***.

The transformation from a stick diagram to a collection of adjacent rectangles is done using the canonical choice unit shown in Figure 4.8. In this canonical choice unit, *ccu*, the essential axial lines (shown as solid lines) originating in the four small rectangles and ending in the four darker shaded rectangles are enough to cross all of the adjacencies in the ccu except those between the middle rectangle and the two tall rectangles bordering it. These adjacencies can be crossed by two possible maximal axial lines (shown as dashed lines), only one of which is necessary. Scaling of the canonical choice unit does not change the fact that it always produces these choice axial lines.

This transformation proceeds by replacing each vertical line in the stick diagram by a ccu of an appropriate size. The horizontal lines that crossed through the vertical line are represented by the choice axial lines of the ccu. It is necessary to show that these canonical choice units can be joined together in a fashion which maintains the relation between the horizontal lines in the stick diagram and the choice axial lines in the configuration of adjacent rectangles. There are four ways in which horizontal lines could cut through successive vertical lines or in which choice axial lines could cross successive ccu's (actually there are only two ways, each with a vertical reflection). These are

1. the horizontal line could be the upper (lower) line through one vertical line and the upper (lower) line through the next vertical line (the upper (lower) choice axial line of one ccu is the same as the upper (lower) choice axial line of the next ccu),

2. the horizontal line could be the upper (lower) line through one vertical line

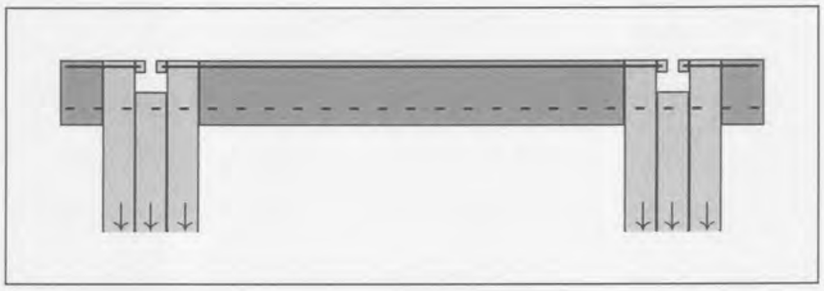
Figure 4.9: Joining the upper choice axial lines of two choice units

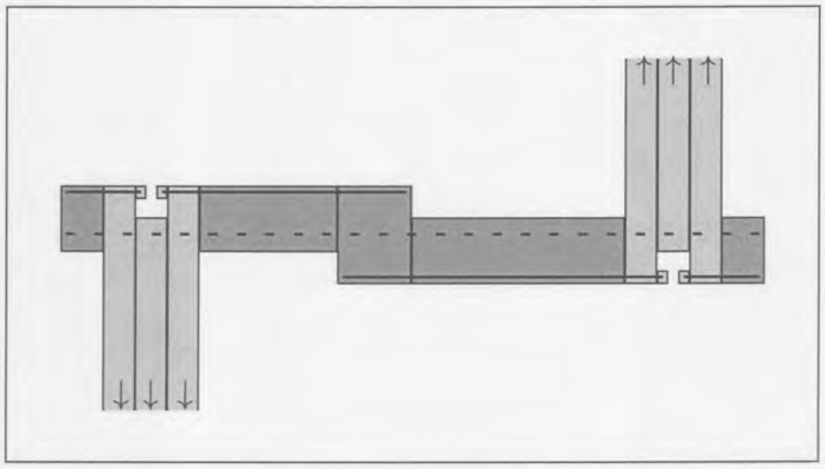
Figure 4.10: Joining the upper choice axial line of one unit to the lower choice axial line of the next unit

and the lower (upper) line through the next vertical line (the upper (lower) choice axial line of one ccu is the same as the lower (upper) choice axial line of the next ccu).

In each of these cases it is possible to connect two ccu's in such a fashion that the relation between lines in the stick diagram and choice axial lines is preserved. This is accomplished by making use of the darker shaded "connector" rectangles of each ccu (see Figure 4.8) and where appropriate making use of "connecting" rectangles. Figure 4.9 shows the construction for case 1 and its reflection. Figure 4.10 shows the construction for case 2 and its reflection.
If all of the vertical lines in the stick diagram are replaced by ccu's, connecting rectangles are added if appropriate and the appropriate changes are made to the connector rectangles then the choices in the original stick diagram can be maintained. An example of converting a stick diagram to a collection of adjacent rectangles is shown in Figure 4.11.

The transformation from ***stick diagram*** to ***ALP-OLOR*** is thus accomplished by inserting an appropriately sized ccu for each vertical line and then joining these up by using the appropriate connecting rectangles working from the leftmost to the rightmost ccu, at each stage connecting the current ccu to those that have already been visited.

This transformation can clearly be done in polynomial time – each vertical line is visited twice, once when it is replaced by a ccu and a second time when it is connected to the ccu(s) to its right in the stick diagram. If the stick diagram can be drawn then a configuration of ccu's can be drawn by scaling the ccu's to be the same size as the vertical lines that they represent. The ccu's (and their connecting rectangles) can thus be drawn as a non overlapping collection of adjacent rectangles – an instance of ***ALP-OLOR***.

It is now necessary to show that it is possible to find a set of horizontal lines H' such that $|H'| \leq S$ for ***stick diagram*** if and only if it is possible to find a set of axial lines axial lines P such that $|P| \leq O$ for ***ALP-OLOR***. Suppose there exists a set of lines H' such that $|H'| \leq S$ for ***stick diagram***. The construction of the collection of adjacent rectangles from the stick diagram changes the horizontal lines in the stick diagram to choice axial lines in the collection of rectangles. It also introduces 4 essential axial lines for every ccu added. These essential axial lines must be in the final solution to ***ALP-OLOR***. There must thus be a solution P ($|P| \leq O$) to ***ALP-OLOR*** with $|P| = |H'| + 4|U| - p$ where p is the number of shared essential axial lines (see Figure 4.9 for an example of how essential axial line sharing can occur). This is because the essential axial lines must be in P and the choice axial lines which correspond to the horizontal lines in H' must also be in P. Conversely if there is a solution P to ***ALP-OLOR*** then there must be a solution $H' = P - \{e \mid e$ is an essential axial line in $P\}$ to ***stick diagram***.

ALP-OLOR is thus NP-Complete.

□

This section shows that the problem of crossing all the adjacencies of a collection of adjacent orthogonal rectangles with the minimum number of maximal-length axial lines is NP-Complete in general. The next section of this chapter (Section 4.5) presents an algorithm that finds an approximate solution to the problem. The next section also includes a discussion of some empirical testing of the heuristic. The following section (Section 4.7) discusses some special cases in which an exact solution to the problem can be found in polynomial time.

4.5 Heuristic Algorithm

Section 4.4 above shows that the problem of crossing all the adjacencies of a collection of adjacent orthogonal rectangles with the minimum number of maximal-length

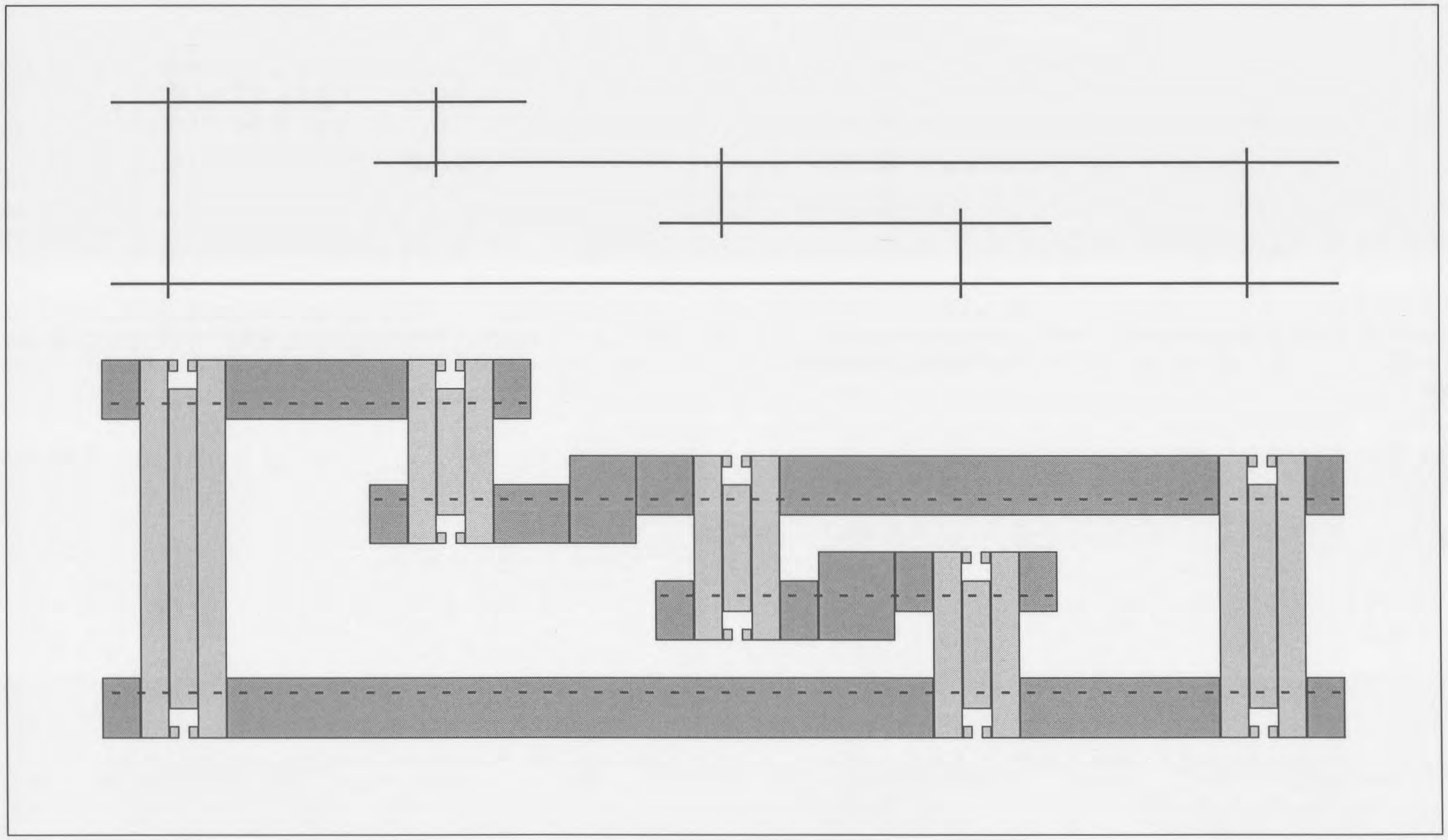

Figure 4.11: An example of converting a stick diagram to a collection of adjacent rectangles

axial lines is NP-Complete in general. A heuristic algorithm to find a non-redundant set of orthogonal axial lines to cross all the adjacencies (a set of lines such that none can be removed without leaving an adjacency uncrossed) is presented in this section. The algorithm has two phases. First the adjacencies among the rectangles are determined (section 4.5.1), and then the axial lines crossing all of the adjacencies are determined (section 4.5.2).

It should be noted that if two rectangles are adjacent then there is an infinite number of orthogonal line segments that could be placed to cross that adjacency. These line segments are all equivalent in the sense that they cross that particular adjacency. Thus in this phase of the research an axial line is defined by a range of y-values through which a line segment parallel to the x-axis could be drawn. The convention used here is that an axial line crossing a given adjacency would be defined by the y-value range of that adjacency and the two rectangles involved. An axial line crossing the adjacencies between a number of rectangles would be given by the common y-value range of the adjacencies between the rectangles and a list of the rectangles concerned.

4.5.1 Determining the adjacencies between the rectangles

In determining the adjacencies between the orthogonal rectangles, horizontal and vertical adjacencies are treated as separate cases. Only the case of vertical adjacencies (and horizontal lines) will be discussed here. Horizontal adjacencies can be treated analogously. An algorithm to determine the adjacencies in a configuration of adjacent orthogonal rectangles is given in Figure 4.12 and discussed below.

Any rectangle R can be defined by the coordinates of its bottom-left and top-right corners. The algorithm thus requires a data structure which contains the rectangle number and these coordinates as well as the ability to keep track of other information calculated in the algorithm. This other information is the number of rectangles which are adjacent to the right hand side of the rectangle being considered and a list of these rectangles. These lists of adjacencies (one list per rectangle) are in fact the required output from this algorithm and are used later in determining the axial lines that must be placed to cross all of the adjacencies. In the algorithm to determine the adjacencies between the rectangles, an array of records is the data structure used. Thus each element of the array, $Rect[i]$, is a record defining a particular rectangle of the configuration and has fields $Rect[i].left$, $Rect[i].bottom$, $Rect[i].right$, $Rect[i].top$, $Rect[i].numadj$ and $Rect[i].adjlist$ to define the bottom-left and top-right corners of the rectangle and to maintain a list of the rectangles which are adjacent to this rectangle on the right.

Left and *Right* are essentially copies of the array *Rect*, but are sorted based on the coordinates of the left and right edges of the rectangles respectively (lines 09 to 12 of the algorithm). These arrays are used to implement a “line sweep” strategy

```
    {Get input}
00  create array $Rect[\,]$ for the rectangles
    {$n$ is the number of rectangles}
01  FOR $i$ from 1 to $n$
02     Input $Rect[i].number$
03     Input $Rect[i].left$
04     Input $Rect[i].right$
05     Input $Rect[i].top$
06     Input $Rect[i].bottom$
07     Set $Rect[i].numadj$ to be 0 {will be calculated}
08     Set $Rect[i].adjlist$ to be Nil {will be calculated}
09  Create an array $Left[\,]$ of all the rectangles {a copy of $Rect[\,]$}
10  Sort $Left[\,]$ in ascending order of $Left[i].left$
       {break ties based on increasing $Left[i].bottom$}
11  Create a list $Right[\,]$ of all the rectangles {a copy of $Rect[\,]$}
12  Sort $Right[\,]$ in ascending order of the $Right[j].right$
       {break ties based on increasing $Right[j].bottom$}
13  Set $i \leftarrow j \leftarrow 1$
14  WHILE $i <= n$ AND $j <= n$
15     CASE
16        $Left[i].left < Right[j].right$
17           increment $i$
18        $Left[i].left > Right[j].right$
19           increment $j$
20        $Left[i].left = Right[j].right$
21           WHILE $Left[i].top <= Right[j].bottom$
             AND $Left[i].left = Right[j].right$
22              increment $i$
23           IF $Left[i].left = Right[j].right$
24              THEN
25                 IF $Left[i].bottom < Right[j].top$
26                    THEN
27                       add $Right[j].number$ to $Left[i].adjlist$
28                       $Left[i].numadj \leftarrow Left[i].numadj + 1$
29                 IF $Left[i].top <= Right[j].top$
30                    THEN
31                       increment $i$
32                    ELSE
33                       increment $j$
```

Figure 4.12: The algorithm for determining the adjacencies between the rectangles

[Manber, 1988; Cormen *et al.*, 1990] for determining which rectangles are adjacent to which other rectangles.

The line sweep (lines 13 to 33) works by comparing the coordinates of the right edges (using *Right*) with the coordinates of the left edges of the other rectangles (using *Left*). Three cases can arise:

1. The left edge of the rectangle being considered in list *Left* is to the left of the right edge of the rectangle being considered in *Right* (lines 16 and 17), in this case these two rectangles cannot be adjacent and so the next rectangle in *Left* must be considered for a potential adjacency. This rectangle's left edge must be further to the right so there is potential for an adjacency to occur.

2. The left edge of the rectangle being considered in list *Left* is to the right of the right edge of the rectangle being considered in *Right* (lines 18 and 19). Again these two rectangles cannot be adjacent and so the next rectangle in *Right* must be considered for a potential adjacency.

3. The left edge of the rectangle being considered in list *Left* and the right edge of the rectangle being considered in *Right* have the same x-coordinate (lines 20 to 33). Thus these two rectangles could be adjacent and it is necessary to determine if they do in fact share a range of y values. This is done by traversing the list *Left* until a rectangle with a top y-value greater than the bottom of the current rectangle from *Right* is found or until the rectangles being considered have different x-values and thus cannot be adjacent (lines 21 and 22).

   In the former case more work needs to be done to test for the adjacency (lines 23 to 33).

   In the latter case no more work will be done in this pass through the loop and the outer `WHILE` will be continued.

   In lines 23 to 33, a test is made to see if the bottom of the left rectangle is below the top of the right rectangle (line 25). If it is, then the right rectangle is adjacent to the left rectangle and this information must be recorded (lines 27 and 28).

   Lines 30 to 33 are then used to determine whether to move along in list *Left* or list *Right*.

The configuration of adjacent rectangles in Figure 4.13 illustrates the working of the algorithm. After input and sorting, the rectangles in the list *Left* would be $\{1,2,3,4,5,6,7,8,9,10,11\}$ and the rectangles in the corresponding list *Right* would be $\{1,2,3,5,7,6,8,4,9,10,11\}$. The `WHILE` loop would begin with $i = 1$ and $j = 1$ and thus would be comparing *Left*[*1*].*left* and *Right*[*1*].*right*. Here

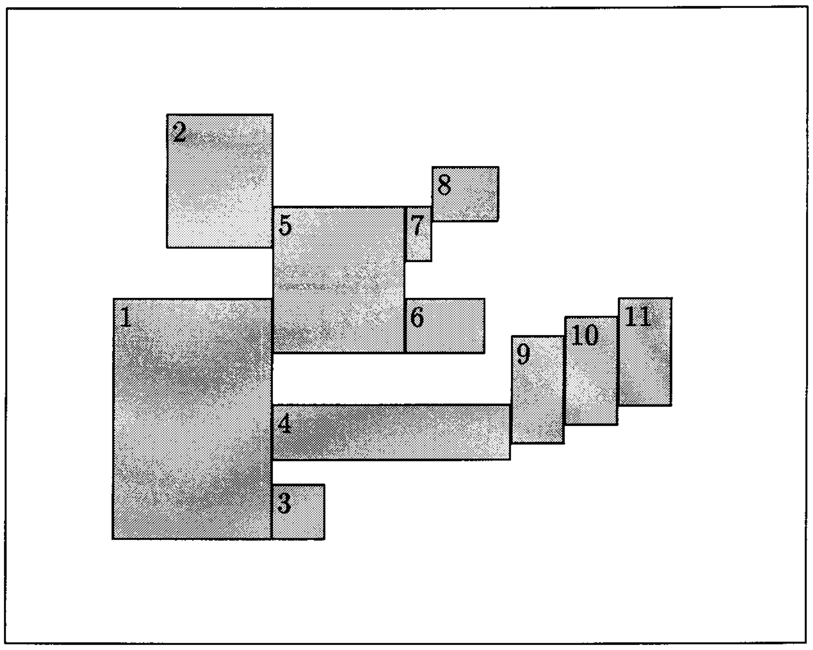

Figure 4.13: A configuration of adjacent orthogonal rectangles

case 1 above holds, the left x-coordinate of rectangle 1 is clearly less than the right x-coordinate of the same rectangle, and i would be incremented. *Left*[*2*].*left* and *Right*[*1*].*right* would then be compared (the left x-coordinate of rectangle 2 is clearly less than the right x-coordinate of rectangle 1) and again i would again be incremented. Next *Left*[*3*].*left* and *Right*[*1*].*right* would be compared. Here the left x-coordinate of rectangle 3 is equal to the right x-coordinate of rectangle 1 and case 3 would be executed. The `WHILE` loop in line 21 would not be executed (*Left*[*3*].*top* is greater than *Right*[*1*].*bottom*) and line 23 would be executed next. *Left*[*3*].*left* = *Right*[*1*].*right* and so line 25 would be executed. Here *Left*[*3*].*bottom* < *Right*[*1*].*top* which means that the two rectangles are adjacent and the adjacency list for rectangle 1 must be updated. Lines 30 to 33 determine whether to increment i or j, in this case there is another rectangle, 4, which has the same left x-coordinate as rectangle 3 and in order to consider it i must be incremented to traverse the list *Left*. The algorithm performs in a similar fashion on the remainder of the two lists.

The sorting phase of this portion of the algorithm is clearly $O(n \lg n)$, where n is the number of rectangles. Determining the adjacencies from the sorted lists is $O(n)$ as each list is simply traversed from beginning to end with no backtracking being necessary. The whole algorithm is then $O(n \lg n)$.

4.5.2 Determining the axial lines

The next step in the process is to determine a non-redundant set of axial lines to cross all of the adjacencies in a configuration of adjacent orthogonal rectangles. The algorithm which generates this non-redundant set uses a number of functions. These functions are given in Figure 4.14. The functions are also used by other algorithms discussed in this chapter. The algorithm itself is given in Figures 4.15, 4.16, 4.17, 4.19 and 4.20. The algorithm generates all the possible orthogonal axial lines which cross the adjacencies between rectangles (Figures 4.15, 4.16), determines which lines are essential (are the only lines which cross a particular adjacency) (Figures 4.17), removes any lines which only cross adjacencies crossed by the essential lines (redundant lines) (Figures 4.19) and then resolves the choice conflict (Figures 4.20). The resolving of the choice is done by repeatedly choosing the choice line which crosses the highest number of previously uncrossed adjacencies. In this algorithm, lines are represented as a list of rectangles. For example, the line $1, 2, 3$ means a line which crosses the adjacency between rectangles 1 and 2 and the adjacency between rectangles 2 and 3. Any line, say l, in this algorithm will be represented a data structure with four fields – $l.line$ which is a list of rectangle numbers, $l.bottom$ which is the bottom y-coordinate of the range of y-coordinates that define the line (the bottom of the common y-coordinate range of the adjacencies between the rectangles) and $l.top$ which is the top y-coordinate of the range of y-coordinates that define the line. The fourth field $l.extended$ is used in the algorithm to record whether a line has been extended.

The algorithm uses the adjacency lists for each rectangle created above (Figure 4.12) and visits the rectangles based on the order in the list *Left* used in Figure 4.12. The algorithm also uses a modified adjacency matrix A. Here $A[i, j]$ indicates whether adjacency $i|j$ (where j is any rectangle which is adjacent to i on the right) exists in the configuration of rectangles. $A[i, j]$ also keeps the track of the first line to cross the adjacency (useful for identifying essential lines), the number of lines crossing the adjacency and also a list of these lines. In addition, a set L, the set of candidate lines at any stage of the execution of the algorithm, must be maintained. A set T of temporary candidate lines is created when lines are extended backwards. The algorithm generates as output a set E of non-redundant orthogonal lines which cover all of the adjacencies between rectangles.

The first phase of the algorithm starts by setting the list of candidate lines L to be empty (line 34 of the algorithm in Figure 4.15 and Figure 4.16). The algorithm then generates every possible longest line which crosses the adjacencies between the adjacent rectangles (lines 35 to 75).

This is done by considering each rectangle, *Rect*$[r]$, in turn from the leftmost to the rightmost (in the order defined by the list *Left*$[\,]$ – see line 35) to find the axial lines which cross the adjacencies between that rectangle and each of its right

---

`||` is used to denote concatenation of two strings or lists

minimum and *maximum* are functions which return the smaller of two numbers and the greater of two numbers respectively

interval(a, b) defines a range of y values between the two arguments a and b

overlap(*interval1*, *interval2*) returns true if the two intervals have an overlapping range of y-values, and returns false otherwise

---

Figure 4.14: Functions used in the algorithms in this chapter

```
Set $L$ to be empty {$L$ is the set of all possible lines}
FOR each rectangle $i$ in $Left[\ ]$
    $p \leftarrow Left[i].number$
    find $r$ such that $Rect[r].number = p$
    FOR each rectangle $j$ in $Left[i].adjlist$
        $q \leftarrow Right[j].number$
        find $s$ such that $Rect[s].number = q$
        $extended \leftarrow$ false
        FOR each line $l$ in $L$
            IF ($overlap(interval(l.low, l.high)$,
            $interval(Rect[s].bottom, Rect[s].top))$ = true) AND
            (rightmost rectangle in $l.line$ = $Rect[r].number$
                THEN
                    $l.extended \leftarrow$ true
                    $lnew.line \leftarrow l.line$ || $Rect[s].number$
                    $lnew.low \leftarrow maximum(l.low, Rect[s].bottom)$
                    $lnew.high \leftarrow minimum(l.high, Rect[s].top)$
                    Add $lnew$ to $L$
                    $extended \leftarrow$ true
```

Figure 4.15: Determining all possible orthogonal axial lines – Phase 1 part a

```
51          IF $extended =$ false
            {no candidate line $l$ can be extended into $Rect[s]$}
52             THEN
53                Set $T$ to be empty
                  {$T$ is the set of lines which can be extended
                   backwards, the longest such line will be chosen}
54                Set $t.line$ to be $Rect[r].number || Rect[s].number$
55                Set $t.low \leftarrow maximum(Rect[r].bottom, Rect[s].bottom)$
56                Set $t.high \leftarrow minimum(Rect[r].top, Rect[s].top)$
57                Add $t$ to $T$
58                For each line $k$ in $L$ which ends in $Rect[r]$
59                   Set $u$ to be equal to $t$
60                   Set $extending$ to true
61                   Set $m$ to point to the second last rectangle in line $k$
62                   WHILE $extending$ AND $m > 1$
63                      find $d$ such that $Rect[d].number =$ $m$th
                        rectangle in line $k$
64                      IF $overlap(interval(u.low, u.high),$
                        $interval(Rect[d].bottom, Rect[d].top))$ = true
65                         THEN
66                            Set $u.line$ to be $Rect[d].number || u.line$
67                            Set $u.low \leftarrow maximum(Rect[d].bottom, u.low)$
68                            Set $u.high \leftarrow minimum(Rect[d].top, u.high)$
69                            Decrement $m$
70                         ELSE
71                            Set $extending$ to false
72                   Add $u$ to $T$
73                Add the longest line, $long$ in $T$ to $L$
74    FOR each $l$ in $L$
75       IF $l.extended =$ true THEN remove $l$ from $L$
```

Figure 4.16: Determining all possible orthogonal axial lines – Phase 1 part b

neighbours, $Rect[s]$, in turn (see line 38). The algorithm determines whether each of the lines coming into $Rect[r]$ from the left can be extended to cross the adjacency between $Rect[r]$ and $Rect[s]$. This is done by testing if there is an overlap of the y-coordinate range of the adjacency and the line being considered and that the line being considered crosses some adjacency into $Rect[r]$ (line 43). Extending the line (lines 44 to 50) involves appending the rectangle number, $Rect[s].number$, to the line and recalculating the y-coordinate range of the line. Figure 4.13 can be used as an example of how lines can be extended. Suppose that the rectangle being considered is the rectangle labeled 5. This rectangle will have two lines coming into it from the left, the line 1–5 and the line 2–5. It also has two right neighbours 6 and 7. The algorithm will attempt to extend each of these lines into each of these neighbours. The result after considering both neighbours will be to have two new lines in L, 1–5–6 and 2–5–7 with the appropriate ranges of y-coordinates.

If no line can be extended into $Rect[s]$ (line 51) then a new line must be started in order to cross the adjacency between $Rect[r]$ and $Rect[s]$. To make sure that this new line is as long as possible it must also be extended as far to the left (backwards) as possible (lines 51 to 73). This extending backwards of the new line is accomplished by considering all lines which cross into $Rect[r]$ in turn (line 58) and looking at these lines an adjacency at a time (lines 58 to 72) to see whether the new line could cross some of the adjacencies crossed by the line being considered. In this fashion a new line is potentially created for each incoming line. Only the longest line which crosses the adjacency under consideration (between $Rect[r]$ and $Rect[s]$) is chosen (line 73). An example of where no line can be extended can be seen in Figure 4.13 when rectangle 10 is being considered. It has the line 1–4–9–10 coming into it from the left and has rectangle 11 as its right neighbour. The line cannot be extended to cross the adjacency between 10 and 11 so a new line must be created to cross this adjacency. This new line must then be extended backwards to cross the adjacency between 9 and 10. If this was not done then the new line crossing the adjacency between 10 and 11 would not be as long as possible. After this phase of extending backwards the line 9–10–11 is added to L.

The last two lines of the algorithm (lines 74 and 75) remove any lines that have been extended forwards during the calculation of the set of lines as these lines are no longer necessary – every adjacency crossed by one of these lines is also crossed by at least one longer line that was generated when the line was extended.

Figure 4.17 gives the algorithm for finding the essential lines from the set of lines calculated by the algorithm in Figures 4.15 and 4.16. This algorithm works by considering each line in L in turn and marking off in an adjacency matrix C the adjacencies that this line crosses (lines 77 to 79). If any adjacency is only crossed by one line from L then that line must be essential and is added to the set E of essential lines (lines 80 to 84). As an example consider the configuration of rectangles in Figure 4.18. The algorithm in Figures 4.15 and 4.16 would have generated

```
76  Set $E$ to be empty
    {$E$ is the set of essential lines}

77  FOR each line $l$ in $L$
78     FOR each pair of adjacent rectangles $i$ and $j$ in $l.line$
79        Mark in $A[i,j]$ that $l$ crosses the adjacency between $i$ and $j$
80  FOR each adjacency $i|j$ in $A$
81     IF $i|j$ is only crossed by one candidate line $e$
82        THEN
83           Add $e$ to $E$
84           Remove $e$ from $L$
```

Figure 4.17: Finding the essential lines – Phase 2

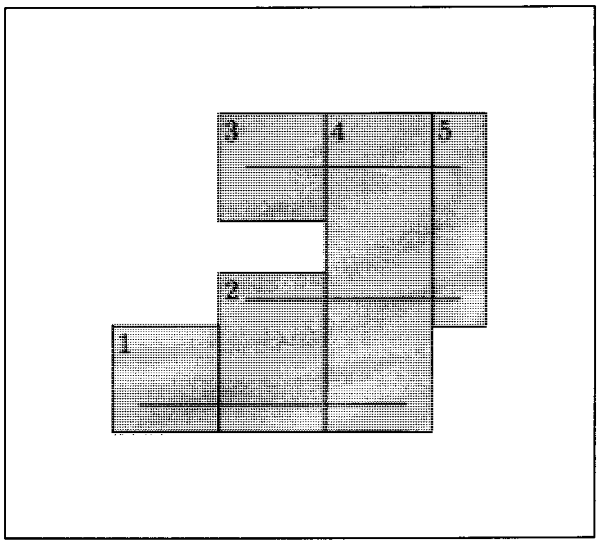

Figure 4.18: A configuration of adjacent orthogonal rectangles

```
85  FOR each line $e$ in $E$
86      FOR each pair of adjacent rectangles $i$ and $j$ in $e.line$
87          Mark in $A$ that $e$ crosses the adjacency between $i$ and $j$
88  FOR each line $l$ in $L$
89      FOR each pair of adjacent rectangles $i$ and $j$ in $l.line$
90          Check if adjacency $i|j$ is crossed by an essential line
91      IF all adjacencies in $l$ are crossed by lines in $E$
92          THEN Remove $l$ from $L$
```

Figure 4.19: Removing Redundant lines – Phase 3

3 lines (1–2–4, 3–4–5 and 2–4–5). The line 2–4–5 would have been created when considering rectangle 4 and attempting to extend the line 1–2–4 into rectangle 5 – this would not be possible and the line 2–4–5 would have been created by first creating a new line 4–5 and then extending that line backwards. In this situation the adjacency between rectangle 1 and rectangle 2 ($C[1, 2]$) is only crossed by the line 1–2–4 so this line must be an essential line. A similar observation applies for line 3–4–5 and the adjacency between rectangles 3 and 4. The line 2–4–5 is in fact redundant and the algorithm in Figure 4.19 describes how this line is identified and removed.

The algorithm considers each essential line in turn and marks off the adjacencies crossed by that line (lines 85 to 87). It then considers each line in L (the set of all possible lines) in turn and determines if all the adjacencies in a given line have been crossed by one or other of the essential lines (lines 88 to 90). If this is the case then the line is redundant. This can be seen in Figure 4.18. The line 2–4–5 crosses the adjacency between 2 and 4 which is also crossed by the essential line 1–2–4 and the adjacency between 4 and 5 which is also crossed by the line 3–4–5. Thus the line 2–4–5 is redundant and is removed from L (lines 91 and 92).

Once the essential lines have been identified and the redundant lines have been removed then there could still be some lines L which are choice lines (see the discussion in Section 4.3 and the Figures 4.2 and 4.11) and only some of these lines are necessary. Phase 4 of the algorithm (Figure 4.20) resolves this choice.

The algorithm first considers each of the remaining lines in L – these are the choice lines – and counts how many adjacencies, which have not already been crossed by essential lines, each line crosses (lines 93 and 94). In Figure 4.2 lines c and d each cross one such adjacency (the adjacency 1|3). In Figure 4.11 two of the choice lines cross 6 previously uncrossed adjacencies and the other two lines cross 4 each. The algorithm then repeatedly applies the heuristic of choosing the line that

```
93  FOR each remaining line $z$ in $L$ {These are the choice lines}
94     Determine how many adjacencies not crossed by essential
       lines that this line crosses
95  REPEAT
96     Choose the line $y$ which crosses the most previously uncrossed
       adjacencies
97     Add $y$ to $E$
98     FOR each adjacency $a|b$ crossed by $y$
99        FOR each line $t$ which crosses $a|b$
100          Decrement the number of adjacencies crossed by $t$
101          IF the number of adjacencies crossed by $t$ is equal to 0
102             THEN Remove $t$ from $L$
103    Remove $y$ from $L$
104 UNTIL all of the adjacencies have been crossed.
```

Figure 4.20: Resolving the issue of choice – Phase 4

crosses the most previously uncrossed adjacencies and making that an "essential line" (actually adds it to the set of non-redundant lines) (lines 96 and 97). Any lines that cross previously uncrossed adjacencies crossed by the chosen line have their counts reduced appropriately and the process is repeated (lines 98 to 103). When all the previously uncrossed adjacencies have been crossed by lines chosen in this fashion then a non-redundant set of lines has been generated and the algorithm terminates. The heuristic applied means that a minimal set of axial lines is not guaranteed by this algorithm but it seems that the heuristic does produce reasonable approximations in some cases (see Section 4.6.4).

4.5.3 The Correctness of the method

The algorithm in Figures 4.15, 4.16, 4.17, 4.19 and 4.20 generates all the possible orthogonal axial lines that cross the adjacencies. It also extends all lines as far as possible to the left and right. Clearly any line that is the only line to cross a particular adjacency must be in the final set of lines otherwise there would be at least one adjacency that has not been crossed. Also any line that only crosses adjacencies which are crossed by lines which are essential should not be in the final set of lines.

It remains to show that the method for dealing with choice lines does give a non-redundant set of lines. The algorithm repeatedly chooses the choice line that crosses the highest number of adjacencies which are previously uncrossed. This means that the selected line can be treated as essential provided no line selected previously crosses any of those adjacencies. This is clearly the case. In addition, no line selected later can cross only those adjacencies and adjacencies already crossed by the essential lines. A line chosen later has to cross at least one previously uncrossed adjacency. Thus each line selected from the set of choice lines crosses at least one previously uncrossed adjacency and is thus necessary.

It should be noted that this algorithm is not guaranteed to give an optimal solution to the problem. It does, however, give a non-redundant solution from the point of view that removing any line from the final set of lines would leave at least one adjacency uncrossed. Figure 4.21 shows a configuration of rectangles where the heuristic would pick line a first as it crosses the most adjacencies not crossed by essential lines. The algorithm would then pick one of lines b and e, one of lines c and f and one of lines d and g. Thus a solution with four lines would be returned. The optimal solution would be lines b, c and d – only three lines.

4.6 Complexity Argument

4.6.1 Time

If all the rectangles were arranged such that one line could cross all the adjacencies between them then it is clear that this instance of the problem can be solved in linear time. Each rectangle has one adjacent rectangle and one candidate line passes from the rectangle to its neighbour.

The worst case for the algorithm could potentially occur when each rectangle (assuming n rectangles) in the configuration has $O(n)$ neighbours on its right-hand side and also has $O(n)$ lines coming into it from its left-hand side. In this case, if each incoming line could be extended into every right neighbour then $O(n^3)$ work would be required. If none of the incoming lines can be extended into a right neighbour and if for every right neighbour all of the lines have to be extended backwards to find the maximal line then potentially $O(n^4)$ work would be required. It is, however, not possible to construct configurations of rectangles which correspond to these cases.

A configuration of rectangles which would force a lot of work to be done in extending lines forward is shown in Figure 4.22. In this case the rectangles on the left edge of the collection will give rise to $n/2$ lines that must be extended through another $n/2$ rectangles. This means that the portion of the algorithm which generates the lines (lines 35 to 50) is $O(n^2)$ – order $O(n)$ rectangles have $O(n)$ incoming lines which can be extended into 1 neighbour on the right.

The configuration of rectangles in Figure 4.23 shows a situation where $O(n)$ lines would have to be extended forwards and then extended backwards. This configuration would force the algorithm to do a similar amount of work to the case above for generating the lines going into the last large vertical rectangle. This rect-

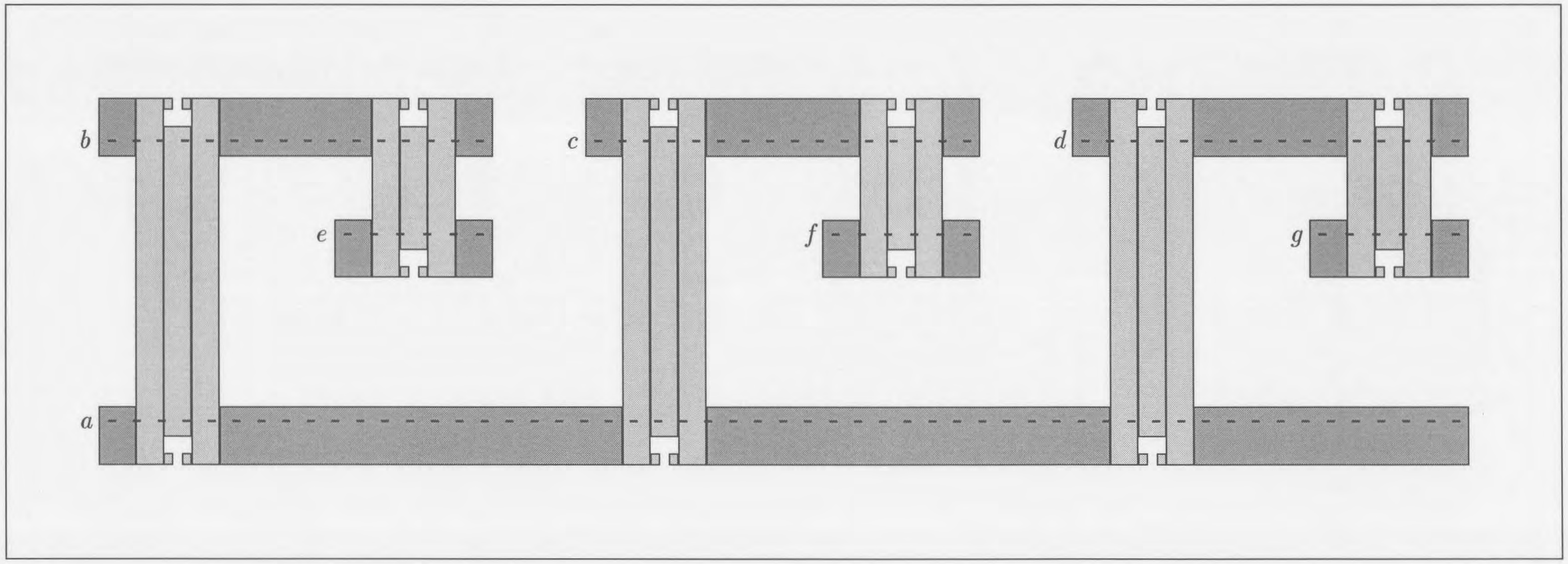

Figure 4.21: An example where the heuristic algorithm would not return an optimal solution

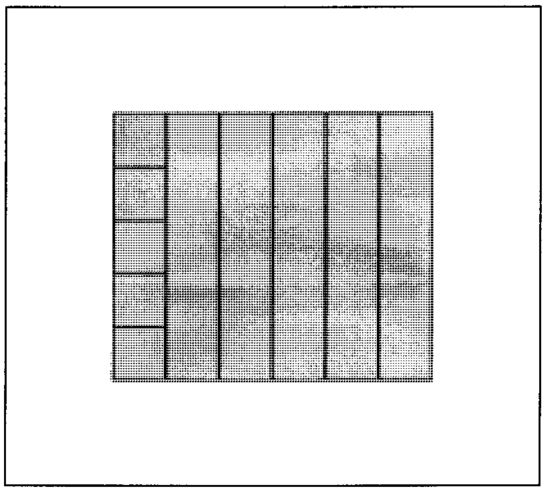

Figure 4.22: A configuration in which there are $O(n^2)$ adjacency crossings

angle then has $O(n)$ incoming lines (the lines originating from the small rectangles on the left of the configuration) and $O(n)$ right neighbours. None of these lines can be extended into any of the small rectangles on the right-hand side of the configuration and so lines must be extended backwards from each of the adjacencies between the right-hand side small rectangles and the rightmost tall rectangle. An additional $O(n^2)$ work would have to be done. Overall the work done is still $O(n^2)$.

The adjacencies which are crossed by only a single line can be found in $O(n^2)$ time by traversing each line and marking off in the adjacency matrix each adjacency as it is crossed. The first time it is crossed it is marked with the identity of the line that crosses it and subsequent crossings are marked as such (setting the first line field to some flag value). As the number of lines must be less than or equal to the number of adjacencies and the number of adjacencies crossed by any line must be less than or equal to the number of adjacencies, this is clearly $O(n^2)$. Having done this it is easy, $O(n^2)$, to determine which adjacencies are only crossed once and thus to determine the essential lines.

Removing redundant lines can also be done in $O(n^2)$ time – by first marking the adjacencies crossed by the essential lines and then determining which lines in the set of candidate lines only cross adjacencies already crossed by essential lines.

The issue of resolving the choice lines is potentially the most expensive part of the algorithm but, in fact, is also $O(n^2)$. The first step in this phase is to calculate how many adjacencies which are not crossed by essential lines are crossed by each choice line. This is clearly $O(n^2)$ – each adjacency ($O(n)$) in each line ($O(n)$) is considered. On each pass through the loop (lines 94 to 104) a number of steps are performed – the line which crosses the highest number of these previously uncrossed adjacencies is selected ($O(n)$); this line is added to E ($O(1)$); each adjacency of the selected line is considered and other lines crossing this adjacency have

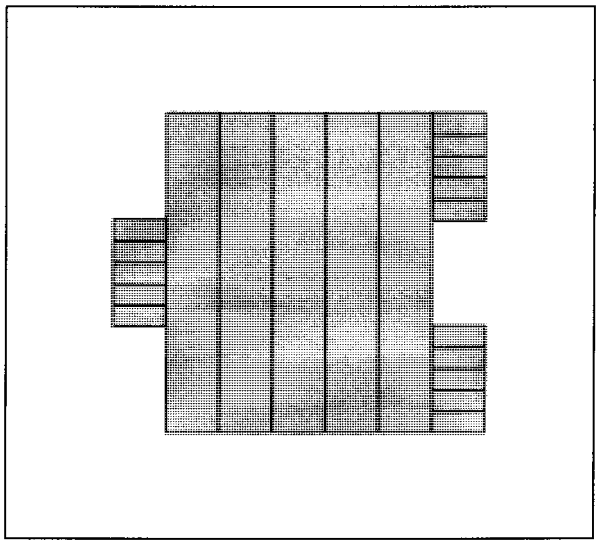

Figure 4.23: A configuration which forces the algorithm to extend O(n) lines backwards

their counts decremented and are removed from L if they no longer cross uncrossed adjacencies (this could be as expensive as O(n^2) if the line selected has O(n) adjacencies and each adjacency is crossed by O(n) lines); finally the selected line is removed from L (O(1)). The work done inside the loop could potentially be as expensive as O(n^2) (lines 98 to 102). This would mean that resolving choice could be as expensive as O(n^3). Again it seems unlikely that a configuration of rectangles which forces this amount of work could be created.

The overall time complexity of the algorithm to produce a non-redundant set of orthogonal axial lines is thus potentially as bad as O(n^4) but this situation is unlikely to occur because of the geometry of the problem.

4.6.2 Space

For each candidate line we store the list of rectangles which are crossed by the line, the list of top and bottom coordinates for each of the rectangles in the line and the final interval for the whole candidate line to date. The final interval can be used each time the line is tested for extending but recalculation must be done when the line is extended backwards. This is O(n^2) space (O(n) lines by O(n) possible crossings of adjacencies). In addition the adjacency matrix could require O(n^2) space.

4.6.3 Bounding the heuristic

The greedy heuristic of choosing the line with the most previously uncrossed adjacencies is analogous to the heuristic of choosing the vertex with the most edges in the original vertex cover problem. This algorithm has been shown not to be an ϵ-approximation [Papadimitriou, 1994] (for any $\epsilon < 1$ the error ratio grows as $\log n$ and thus no ϵ smaller than 1 is valid). The best known approximation algorithm for this problem is based on choosing any edge, say (u, v), in the set of edges, adding both u and v to the set of vertices and repeating until all edges are covered [Papadimitriou, 1994]. This algorithm has an approximation threshold of at most $1/2$ – a solution which is at most twice the optimum solution. Hochbaum [1982] also discusses a heuristic that gives a value that does not exceed twice the optimal value. For unweighted graphs he guarantees a bound strictly less than 2 – a solution strictly less than twice an optimal solution.

4.6.4 Experimental Results

Three algorithms were implemented and tested: the greedy algorithm (most uncrossed adjacencies heuristic); an algorithm based on the random selection of lines and an algorithm that produces the minimum number of orthogonal axial lines required. The algorithm that produces the minimum solution to the problem did an exhaustive search of all the possible ways of selecting subsets of choice lines to find a minimum sized subset. This algorithm was made as efficient as possible by partitioning the choice axial lines into subsets which have in common adjacencies not crossed by essential lines. The solutions for these subsets can then be found independently. This solution works well on average but it is still possible that all the choice lines are in one subset – there are uncrossed adjacencies that are common to all of the choice axial lines.

The test data were generated by first randomly generating a number of rectangles – typically about 30 – placed one on top of another. This configuration was then grown from left to right by randomly generating rectangles which are adjacent to the right hand edges of those that had already been placed in the configuration. The advantage of generating the data in this fashion was that a large number of adjacencies between rectangles in the horizontal direction was guaranteed. All rectangles in the configuration had breadth and height randomly chosen in the range from 5 to 15 units. The final configuration of rectangles was tightly packed and each rectangle could have as many as 4 rectangles adjacent to its right hand edge.

Fifty cases of configurations of 1000 rectangles each were tested, as were 20 cases of configurations of 1500 and 2000 rectangles. In the tests performed the greedy algorithm performs as well as the exact solution in most cases but there were instances of the greedy algorithm requiring an extra line to cross all the adjacencies. The random algorithm ranged in accuracy from producing the same result as the exact solution to requiring six extra lines to cross all adjacencies. Table 4.1 shows the results of the testing of the heuristics on configurations of rectangles of this form.

Number of rectangles	1000	1500	2000
Number of tests	50	20	20
Minimum number of lines	321	477	655
Maximum number of lines	365	527	703
Average number of lines	341	510	680
Standard deviation	10.86	10.81	13.44
Minimum number of essential lines	277	426	569
Maximum number of essential lines	322	476	616
Average number of essential lines	300	450	592
Standard deviation	9.69	14.19	14.24
Minimum number of choice lines	4	4	18
Maximum number of choice lines	27	36	52
Average number of choice lines	14	20	32
Standard deviation	5.83	8.44	8.77
Minimum error for most uncrossed adjacencies heuristic	0	0	0
Maximum error for most uncrossed adjacencies heuristic	1	1	1
Average error for most uncrossed adjacencies heuristic	0.02	0.10	0.05
Standard deviation	0.14	0.31	0.22
Minimum error for random choice heuristic	0	0	0
Maximum error for random choice heuristic	4	6	5
Average error for random choice heuristic	1.26	1.30	1.90
Standard deviation	0.94	1.66	1.55

Table 4.1: Experimental results

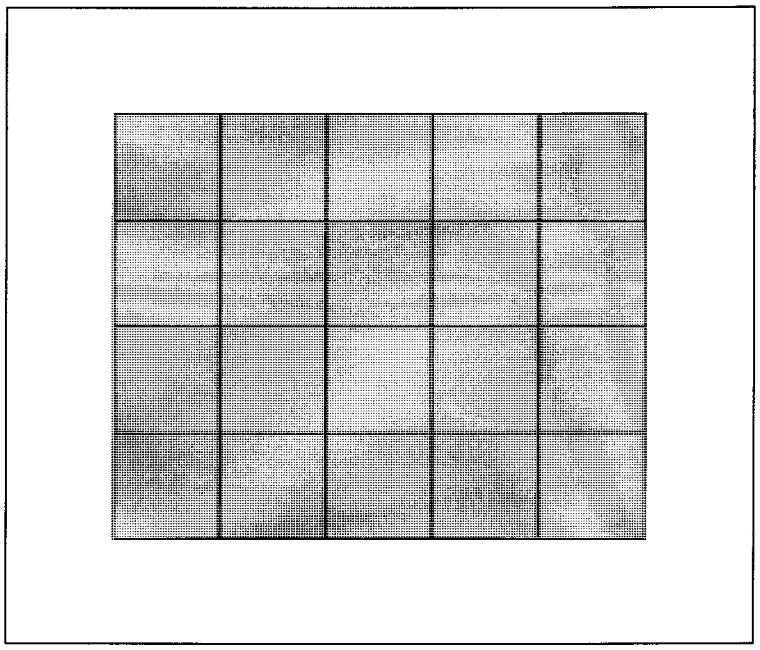

Figure 4.24: A "chequerboard" collection of rectangles

From these results it can be seen that the heuristic of choosing the choice axial line that crosses the most previously uncrossed adjacencies at any stage resulted in a good approximation for the configurations tested. For much larger numbers of rectangles or a different packing method the heuristic might not work as well but configurations of this form were chosen as a reasonable approximation to the type of collections of rectangles that could occur in the problem being studied.

This experimental work showed that although ***ALP-OLOR*** is in general NP-Complete it is possible to get good approximations to the exact solution – at least in the cases tested. The next section of this thesis considers special cases of ***ALP-OLOR*** where exact solutions can be found in polynomial time.

4.7 Special Cases that can be solved exactly in polynomial time

4.7.1 Mapping to interval graphs

ALP-OLOR is in general NP-Complete but there are some cases for which polynomial time algorithms can be obtained. In this section some of these special cases are discussed.

It is clear that any "chequerboard" collections of rectangles (Figure 4.24) can be solved exactly in polynomial time even if there are holes in the chequerboard

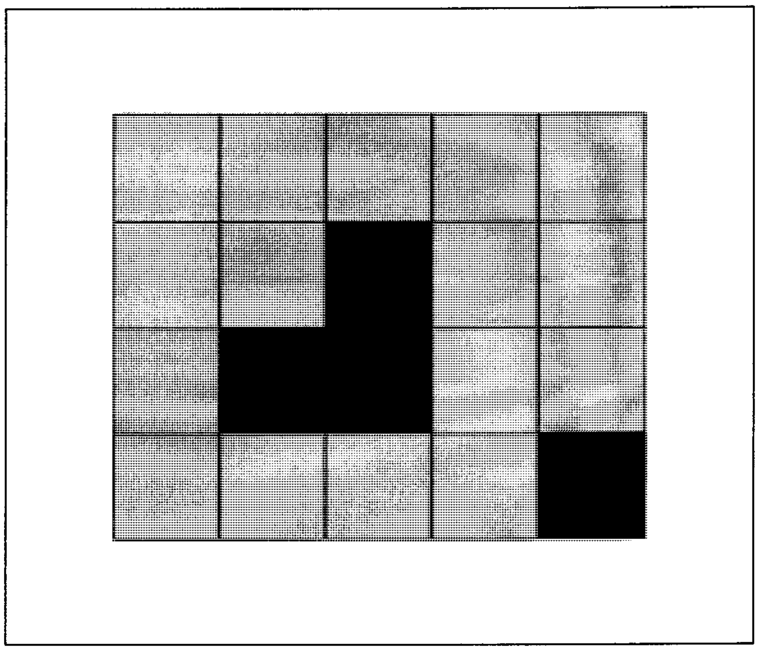

Figure 4.25: A "chequerboard" with holes

(Figure 4.25) [Hedetniemi, 1996]. In fact, this condition can be extended to a more general collection of rectangles.

Suppose that the union of the adjacent rectangles is itself a rectangle as in Figure 4.2 (of Section 4.3), Figure 4.22, Figure 4.24 and Figure 4.26, then the axial line placement problem for orthogonal axial lines and orthogonal rectangles can be solved in polynomial time. This can be shown as follows. First, project each vertical adjacency to a corresponding interval on the vertical line L (see Figure 4.27). Then the problem of finding the minimum number of horizontal lines that intersect the vertical adjacencies (***ALP-OLOR***) is equivalent to that of finding the minimum number of points on L needed such that each interval contains at least one point. This is the problem of finding the independent set of an interval graph which can be solved in linear time [Gavril, 1972; Golumbic, 1980]. The mapping from ***ALP-OLOR*** to vertex cover for interval graphs is also possible for other configurations of adjacent rectangles provided that any vertical adjacencies that produce overlapping intervals when projected onto L can be crossed by a horizontal line that does not leave the union of the rectangles. For example for the configuration of rectangles in Figure 4.28 the mapping would produce a correct answer but for the configuration in Figure 4.29 (and in fact in Figure 4.25) it would not.

There are other configurations of rectangles where solutions to ***ALP-OLOR*** can be found in polynomial time. Some of these are discussed below.

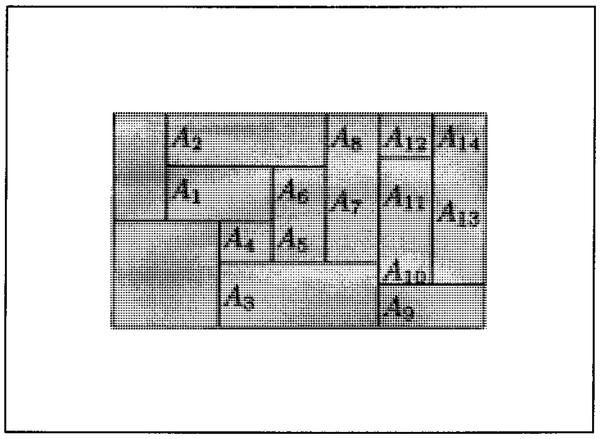

Figure 4.26: A simple configuration of rectangles with a rectangular union

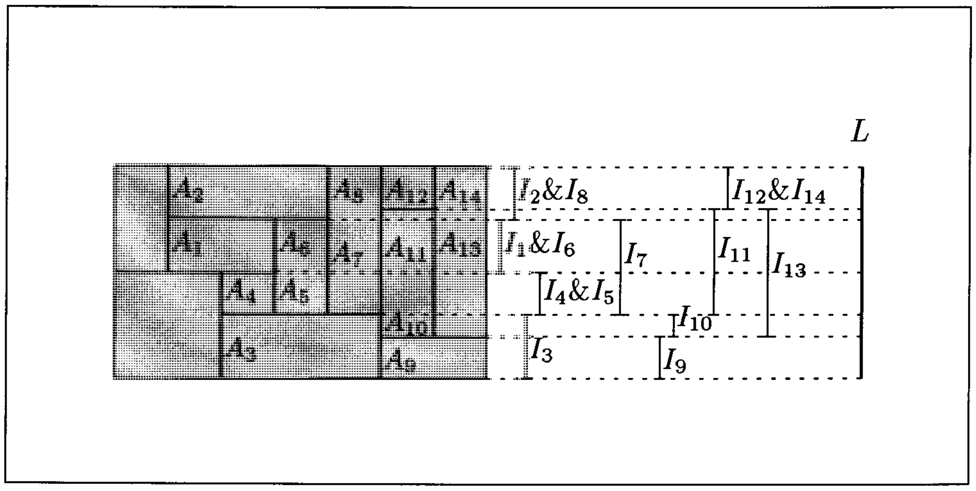

Figure 4.27: Projecting Adjacencies onto Intervals on the line L

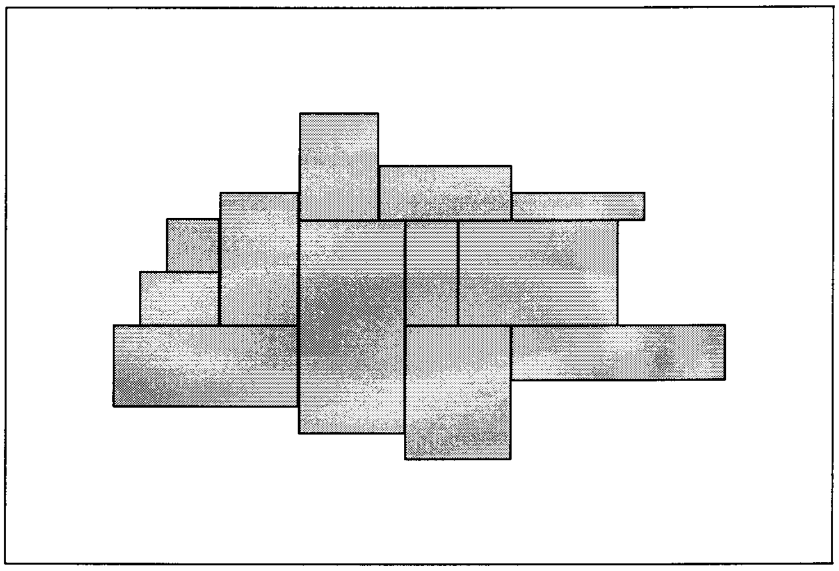

Figure 4.28: A simple configuration of rectangles that can be used in the production of an interval graph

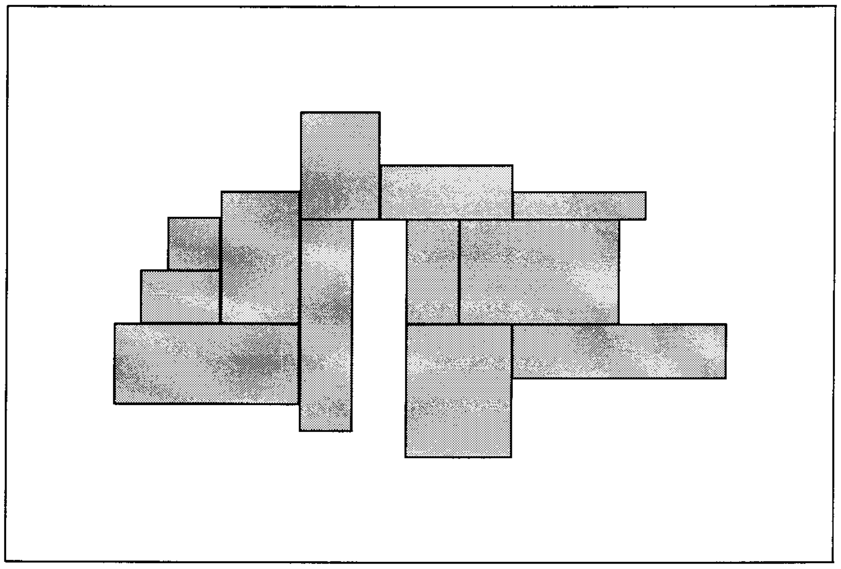

Figure 4.29: A simple configuration of rectangles that cannot be used in the production of an interval graph

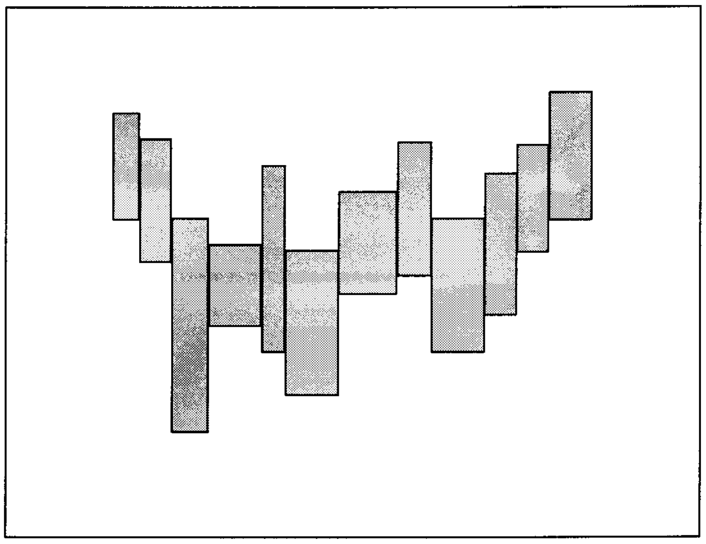

Figure 4.30: An example of a chain

4.7.2 Chains and trees of orthogonal rectangles

This section of the thesis considers algorithms to solve two restricted cases of ***ALP-OLOR*** – chains and trees of rectangles (see Section 4.7.2.1 for definitions of these). These problems cannot be solved by a mapping to an interval graph because the layout of the rectangles could lead to adjacencies that cannot be crossed by a single axial line which remains inside the union of the rectangles being mapped to the same interval. These problems can, in fact, be solved using the heuristic algorithm discussed in Section 4.5 (and discussed briefly in Section 4.7.2.2 below) but this algorithm does a lot of unnecessary work in these cases so better ways of solving these problems are required. In this section of the thesis an $O(n)$ algorithm for orthogonal axial line placement in chains of orthogonal rectangles is given (Section 4.7.2.3) and an $O(n^2)$ algorithm for trees of rectangles is presented (Section 4.7.2.4.

4.7.2.1 Terminology

Definition 4.7.1 *A chain of orthogonal rectangles is any collection of orthogonal rectangles where every rectangle is horizontally (vertically) adjacent to at most one other rectangle at each end.*

An example of a chain is shown in Figure 4.30.

Definition 4.7.2 *A tree of orthogonal rectangles is a collection of adjacent orthogonally aligned rectangles, where each rectangle is joined on the left (right) end to at most one rectangle and on the right (left) to zero or more rectangles.*

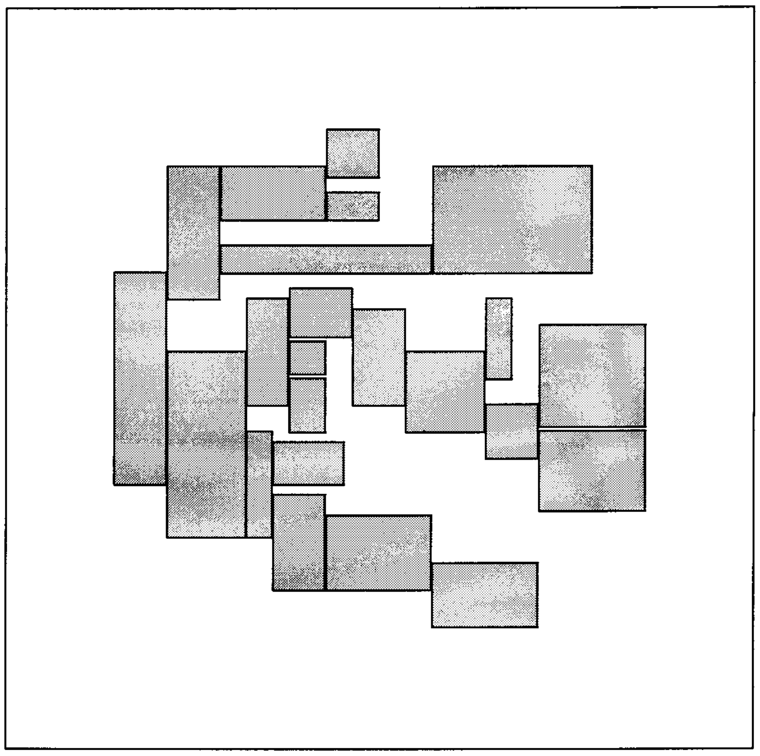

Figure 4.31: An example of a tree of rectangles

A tree is thus a generalisation of a chain – each branch of the tree can be considered as a chain of rectangles. An example of a tree of rectangles is shown in Figure 4.31.

4.7.2.2 The naive algorithm

An O(n^2) algorithm to return a non-redundant set of maximal orthogonal axial lines for ***ALP-OLOR*** was presented in Section 4.5 above. This algorithm has four phases (after determining which rectangles are adjacent). It generates all the possible straight lines which cross the adjacencies between rectangles; it determines which lines are essential (i.e. are the only lines which cross a particular adjacency); it removes any lines which only cross adjacencies crossed by the essential lines (redundant lines); and then it resolves the choice conflict. The resolving of the choice is done by repeatedly choosing the choice line that crosses the highest number of previously uncrossed adjacencies. Each phase is O(n^2) in the worst case.

This algorithm can be applied directly to place the minimal number of axial lines in chains and trees of orthogonal rectangles but it does more work than is necessary – it generates lines which are redundant and then have to be removed to get a minimal set of maximal lines. This is shown in Figure 4.32 where 5 lines are generated in the first part of the algorithm but only 2 lines are actually needed – the lines a–b–c–d–e–f and e–f–g–h–i–j. The reason that these extra lines are

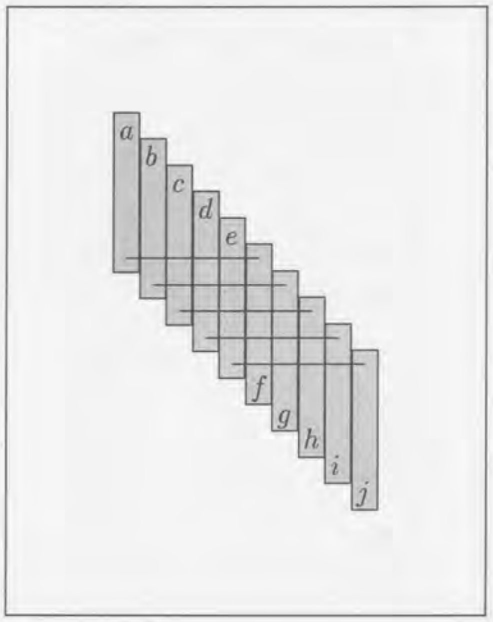

Figure 4.32: A case where more axial lines than necessary are generated

generated is that the algorithm starts from the left and tries to extend any existing line into any rectangle adjacent to the current one. If this can be done it is, but if it cannot be done then a new line is created crossing from the current rectangle into the next rectangle and this new line is then extended as far as it can be to the left before the algorithm continues working towards the right. In Figure 4.32 the line a–b–c–d–e–f cannot be extended into g so a new line f–g is created and this is extended back as far as possible giving the line b–c–d–e–f–g. Similarly for the other lines shown. Phase 3 of the algorithm would identify the two lines which have to be in the solution – the line a–b–c–d–e–f is the only line to cross the adjacency between rectangles a and b and the line e–f–g–h–i–j is the only line to cross the adjacency between i and j. These lines are thus essential. The unnecessary lines would then be removed from the final set of lines in phase 3 of the algorithm. Phase 4 of the algorithm would be unnecessary for both chains and trees of rectangles as no choice lines will ever be generated.

The remainder of this section of the thesis looks at algorithms that are more efficient for solving this problem in the case of chains and trees of orthogonal rectangles. Section 4.7.2.3 gives an $\mathrm{O}(n)$ algorithm for placing orthogonal axial lines in chains of orthogonal rectangles and Section 4.7.2.4 gives an $\mathrm{O}(n^2)$ algorithm for placing orthogonal axial lines in trees of orthogonal rectangles.

```
{Stage 0}
{Get input}
create array $Rect[\ ]$ of all given rectangles
{$n$ is the number of rectangles}
FOR $i$ from 1 to $n$
    Input $Rect[i].left$
    Input $Rect[i].right$
    Input $Rect[i].top$
    Input $Rect[i].bottom$
    Set $Rect[i].adjbottom$ to be undefined {will be calculated}
    Set $Rect[i].adjtop$ to be undefined

{Stage 1:}
{Define order of rectangles in the chain}
sort $Rect[\ ]$ according to left value of each rectangle
{i.e.  based on $Rect[\ ].left$}

{Stage 2:}
{Determine the extent of the adjacency between each rectangle
 and its right neighbour.}
FOR $i$ from 1 to $n-1$
    $Rect[i].adjtop \leftarrow$ minimum($Rect[i].top$, $Rect[i+1].top$)
    $Rect[i].adjbottom \leftarrow$ maximum($Rect[i].bottom$, $Rect[i+1].bottom$)
```

Figure 4.33: The algorithm for placing orthogonal axial lines in chains of orthogonal rectangles – Stages 0 to 2

4.7.2.3 Orthogonal axial line placement in chains of rectangles

In this section of the thesis an algorithm is presented to solve the orthogonal axial line placement problem in a chain of orthogonal rectangles. The algorithm is shown in Figures 4.33, 4.34, 4.35 and 4.36.

The algorithm takes as input a set of rectangles that are known to represent a chain. Each rectangle is defined by the coordinates of its bottom left and top right corner. The data structure used is an array of records *Rect*[]. Here each record has fields for *left*, *right*, *top* and *bottom* to represent the rectangle's coordinates (minimum x-coordinate, maximum x-coordinate, minimum y-coordinate, maximum y-coordinate) and *adjtop* and *adjbottom* to represent the lowest and highest y-values of the range of y-values that defines the adjacency with the next rectangle in the

```
{Stage 3:}
{Determining the set of forward lines}
{ForwardLines is a list of the lines that have been found in
 this stage}
{The forward sweep is started by initialising the smallest
 common adjacency}
{This is called CurrentAdj}
{Initially this is the adjacency between rectangles 1 and 2}
{Currentline is a list of adjacencies that are crossed by the
 line being worked on}

CurrentAdj.top $\leftarrow$ Rect[1].adjtop
CurrentAdj.bottom $\leftarrow$ Rect[1].adjbottom
Add adjacency 1|2 to the list CurrentLine

FOR $i$ from 2 to $n$
   IF (Rect[$i$].adjtop < CurrentAdj.bottom) OR
   (Rect[$i$].adjbottom > CurrentAdj.top)
      THEN
         Add the list CurrentLine to the end of list ForwardLines
         CurrentAdj.top $\leftarrow$ Rect[$i$].adjtop
         CurrentAdj.bottom $\leftarrow$ Rect[$i$].adjbottom
         Set CurrentLine to be empty
      ELSE
         IF (Rect[$i$].adjtop < CurrentAdj.top)
            THEN CurrentAdj.top $\leftarrow$ Rect[$i$].adjtop
         IF (Rect[$i$].adjbottom > CurrentAdj.bottom)
            THEN CurrentAdj.bottom $\leftarrow$ Rect[$i$].adjbottom
         Add the adjacency i|i+1 to the end of list CurrentLine
IF CurrentLine is not empty
   THEN
      Add the list CurrentLine to the end of list ForwardLines
```

Figure 4.34: The algorithm for placing orthogonal axial lines in chains of orthogonal rectangles – Stage 3

```
    {Stage 4:}
    {Determining the set of reverse lines to be stored in list
     $ReverseLines$}

29  Set $CurrentLine$ to be empty
30  $CurrentAdj.top \leftarrow Rect[n-1].adjtop$
31  $CurrentAdj.bottom \leftarrow Rect[n-1].adjbottom$
32  Add the adjacency n-1|n to the end of list $CurrentLine$

33  FOR $i$ from $n-2$ downto 1
34     IF ($Rect[i].adjtop$ < $CurrentAdj.bottom$) OR
       ($Rect[i].adjbottom$ > $CurrentAdj.top$)
35        THEN
36           Add the list $CurrentLine$ to the front of list $ReverseLines$
37           $CurrentAdj.top \leftarrow Rect[i].adjtop$
38           $CurrentAdj.bottom \leftarrow Rect[i].adjbottom$
39           Set $CurrentLine$ to be empty
40        ELSE
41           IF ($Rect[i].adjtop$ < $CurrentAdj.top$)
                THEN $CurrentAdj.top \leftarrow Rect[i].adjtop$
42           IF ($Rect[i].adjbottom$ > $CurrentAdj.bottom$)
                THEN $CurrentAdj.bottom \leftarrow Rect[i].adjbottom$
43           Add the adjacency i|i+1 to the front of list $CurrentLine$
44  IF $CurrentLine$ is not empty
45     THEN
46        Add the list $CurrentLine$ to the front of list $ReverseLines$
```

Figure 4.35: The algorithm for placing orthogonal axial lines in chains of orthogonal rectangles – Stage 4

```
    {Stage 5:}
    {Merge the lines to get the final set of lines}
47  Set $m$ to the number of lines in $ForwardLines$ or $ReverseLines$
    {these will be equal}
48  FOR $i$ from 1 to $m$
49     set $FinalLines[i]$ to be empty
50     WHILE $ForwardLines[i]$ is not empty AND
       $ReverseLines[i]$ is not empty
51        IF (the first adjacency in $ForwardLines[i]$ is to the left
          of the first adjacency in $ReverseLines[i]$)
52           THEN
53              remove first adjacency from $ForwardLines[i]$
54              add this adjacency to end of $FinalLines[i]$
55           ELSE
56              IF the adjacencies are the same
57                 THEN
58                    remove first adjacency from $ForwardLines[i]$
59              remove first adjacency from $ReverseLines[i]$
60              add this adjacency to end of $FinalLines[i]$
61     IF $ForwardLines[i]$ is not empty
          THEN add all remaining adjacencies to $FinalLines[i]$
62     IF $ReverseLines[i]$ is not empty
          THEN add all remaining adjacencies to $FinalLines[i]$
```

Figure 4.36: The algorithm for placing orthogonal axial lines in chains of orthogonal rectangles – Stage 5

chain. The algorithm sorts these rectangles according to the x-coordinate of the left edge of each rectangle. This first stage is dominated by the sorting and thus has a complexity of $\mathrm{O}(n \lg n)$. Clearly this stage is unnecessary if the rectangles are given in sorted order as a chain of rectangles.

From this sorted list of rectangles the right adjacencies of each rectangle are found – lines 9 to 11 of the algorithm – by comparing the y-coordinate range of any rectangle with the y-coordinate range of its neighbour on the right – the largest bottom y-value and the smallest top y-value define this adjacency. This information is stored in $Rect[i].adjtop$ and $Rect[i].adjbottom$ for rectangle i. This process takes linear time and forms the second stage of the algorithm.

The third stage involves determining the lines that move forward through the chain. This stage of the algorithm proceeds by traversing the chain of rectangles from left to right keeping track of any common range of y-values ($CurrentAdj$) of the adjacencies that have been considered so far. The existence of such a range implies that an axial line could be placed to cross the adjacencies that share this range. The algorithm also keeps track of which adjacencies between rectangles could be crossed by such a line by maintaining a list of these adjacencies ($CurrentLine$ is the list of adjacencies that share a common range of y-values and each adjacency is given by the rectangles in it in the form *left*|*right*). The common range starts out as the range of y-values of the adjacency between the first and second rectangles (see lines 12 and 13 of the algorithm). $CurrentLine$ is initialised to *1*|*2* (line 14) as rectangle *1* is adjacent to rectangle *2* and a line could be drawn crossing the adjacency between them. This range and the current line are then updated when the next adjacency is encountered (see lines 15-25). If there is an overlap in y-values between the common range and the next adjacency then the current line can be extended (lines 23-25). If there is no overlap then a new line must be started (18 to 21). In this case the common range is reset as the range of the adjacency being considered. Lines 16 to 25 are repeated until the end of the chain is reached i.e. until all the adjacencies have been crossed by a line in the set of forward lines. Lines 26 to 28 make sure that the last line is also added to the set of forward lines. Since each rectangle is visited once by a single line, this stage of the algorithm also takes linear time.

The fourth stage is the same the third stage, except one finds the reverse lines by moving from right to left, instead of left to right (see lines 29 to 46 which are very similar to lines 12 to 28). Note that the set of reverse lines are still arranged so that the leftmost lines come first in the ordering. The adjacencies in any line are also arranged in order from left to right. Clearly the complexity for this stage is the same as the previous stage.

Lastly the forward lines and the reverse lines are merged together to obtain the *maximal* lines that cross every adjacency. This merging is accomplish by noting that any forward line crosses each adjacency that it can exactly once and extends as far

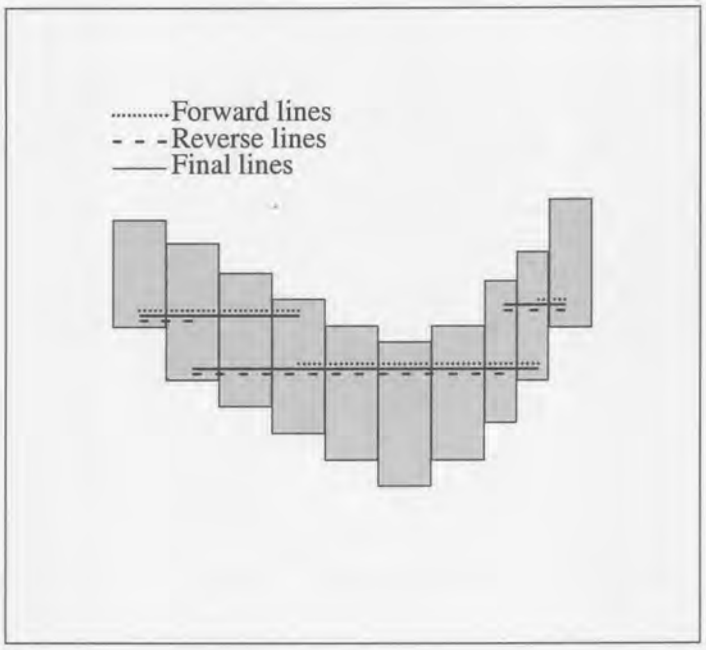

Figure 4.37: A chain of orthogonal rectangles showing the forward and reverse lines and the final maximal lines.

as possible to the right. The process of placing the forward lines thus generates a set containing the minimum number of *non-maximal* lines to cross all of the adjacencies in the chain – any fewer lines and some adjacencies would be left uncrossed, any more lines and one or more adjacencies would have to be crossed more than once. Similarly any reverse line crosses each adjacency that it can exactly once and extends as far as possible to the left. The set of reverse lines is also the minimum number of non-maximal lines required to cross all of the adjacencies. Clearly the two sets must contain the same number of lines. As each forward line extends as far as possible to the right, the forward lines will thus define the rightmost end of some maximal line. Similarly the reverse lines extend as far as possible to the left and define the leftmost extent of the maximal lines. Thus merging any forward and reverse lines that have ranges of y-values which overlap and that cross some common adjacencies will result in a maximal line, through the resulting overlap of y-values, that crosses all the adjacencies crossed by both lines.

The sets of forward and reverse lines are arranged from leftmost to rightmost starting x-value. The merging begins by considering the first (leftmost) lines in each set and generating a new line that crosses all of the adjacencies crossed by those two lines. No other lines need to be considered as no other lines will have an overlap of y-values and share some common adjacencies. Lines 47 to 62 show the detail of how this is accomplished. The merging begins by considering the first adjacency in $ForwardLines[1]$ and comparing it with the first adjacency in $ReverseLines[1]$

(line 51). The leftmost of these adjacencies (found by looking at the x-coordinates of the rectangles concerned) gives the leftmost adjacency in the new final line. This adjacency is removed from the list that it occurred in (both if this was the case) and added to final line which is being created (lines 53 and 54 or lines 56 to 60). The process is the repeated with the first adjacencies of the two new lists. If either list becomes empty then the remaining adjacencies in the non-empty list are copied to the final line which is being built up (lines 61 and 62). This new line is then the first line in the final set of lines – it is maximal because it extends as far as it can to the left and to the right. The other lines in the forward and reverse sets are handled in a similar fashion to produce final lines. An example of a chain of rectangles with forward lines, reverse lines and the resulting final lines is shown in Figure 4.37. This final stage of merging the forward and reverse lines takes $O(n)$ time to complete – each adjacency can appear once in a forward line and once in a reverse line so at most $2n$ adjacencies will be considered for addition into one of the final lines.

The complexity of the entire algorithm is thus $O(n)$.

Empirical tests were done on some different configurations of chains of rectangles [Watts and Sanders, 1997]. The data for the running time of the algorithm, excluding the sorting done in the first stage, verified the theoretical analysis – that is, the data confirmed that the last four stages of the algorithm are indeed linear.

This restricted instance of the problem is, in fact, a special case of the problem to be considered in the next section but it was presented as a separate problem in order to make the algorithm presented in the next section easier to understand.

4.7.2.4 Orthogonal axial line placement in trees of rectangles

The algorithm to find the minimal set of orthogonal axial lines to cross the adjacencies in a tree of orthogonal rectangles is presented in Figures 4.38, 4.39, 4.40 and 4.41. The algorithm takes as input a list of rectangles that represents a tree of orthogonal rectangles and produces a minimal set of maximal orthogonal axial lines.

The algorithm is split into six stages: inputting the rectangle data, finding the adjacencies between the rectangles; defining the order in which the rectangles will be visited; finding the forward lines; finding the leaf lines; and merging the forward lines and leaf lines into the final lines.

The main data structure in the algorithm (as given in Figure 4.38) is an array $Rect$ of records to represent the rectangles. Each rectangle is represented by a record with 8 fields – $left$, $right$, top and $bottom$ to define the rectangle, $parent$ to keep track of the rectangle to the left of the current rectangle, $numadj$ to keep track of the number of rectangles adjacent to the right end of any rectangle, $adjlist$ which is a list of these rectangles and finally $LeafLineNo$ which is used in stages 4 and 5 to keep track of which leaf line crosses the rectangle.

```
    {Stage 0:}
    {Get input}
00  create array Rect[ ] of all given rectangles
    {$n$ is the number of rectangles}
01  FOR $i$ from 1 to $n$
02      Input Rect[$i$].left
03      Input Rect[$i$].right
04      Input Rect[$i$].top
05      Input Rect[$i$].bottom
06      Set Rect[$i$].parent to be undefined {will be calculated}
07      Set Rect[$i$].numadj to be undefined {will be calculated}
08      Set Rect[$i$].adjlist to be Nil {will be calculated}
09      Set Rect[$i$].LeafLineNo to be undefined
        {will be used in stages 4 and 5}

    {Stage 1:}
10  Find adjacencies {using algorithm discussed before}
    {Rect[$i$].parent, Rect[$i$].numadj and Rect[$i$].adjlist are calculated here}

    {Stage 2:}
    {Define the order in which the rectangles will be visited}
11  create array RightList[ ] sorted according to right value of
    each rectangle {i.e.  based on Rect[ ].right}
    {This will be an array of the numbers of the rectangles in
     the order in which they will be visited.}
```

Figure 4.38: The algorithm for placing orthogonal axial lines in trees of orthogonal rectangles – Stages 0 to 2

```
{Stage 3:}
{Find forward lines}

{Set values in root vertex of Interval tree}
$high \leftarrow Rect[RightList[1]].top$
$low \leftarrow Rect[RightList[1]].bottom$
$line \leftarrow RightList[1]$
FOR $p$ from 2 to $n$ do {traverse rectangles in order of right edge}
    $k \leftarrow RightList[p]$
    IF $Rect[k].numadj$ = 0
        THEN
            add $k$ to $LeafList$
        ELSE
            find vertex $z$ in interval tree such that
            $overlap(interval(z.low, z.high),$
            $interval(Rect[k].bottom, Rect[k].top))$ = true
            AND last rectangle in $z.line$ = $k$
            set $extended$ to be false
            FOR each rectangle $r$ in $Rect[k].adjlist$ do
                IF $overlap(interval(z.low, z.high),$
                $interval(Rect[r].bottom, Rect[r].top))$ = true
                    THEN
                        {Extend an existing line}
                        Insert a child of $z$ in interval tree with
                            $high \leftarrow$ top of overlap
                            $low \leftarrow$ bottom of overlap
                            $line \leftarrow z.line$ || $r$
                        set $extended$ to be true
                    ELSE
                        {Start a new line}
                        Insert a child of $z$in interval tree with
                            $high \leftarrow Rect[r].top$
                            $low \leftarrow Rect[r].bottom$
                            $line \leftarrow r$
            IF $extended$ = true THEN delete vertex $z$
{traverse interval tree to produce a list of forward lines}
$i \leftarrow 0$
FOR each vertex $z$ in the interval tree
    $i \leftarrow i+1$
    $ForwardLines[i].rects \leftarrow z.line$
    $ForwardLines[i].top \leftarrow z.high$
    $ForwardLines[i].bottom \leftarrow z.low$
```

Figure 4.39: The algorithm for placing orthogonal axial lines in trees of orthogonal rectangles – Stage 3

```
{Stage 4:}
{Find leaf lines}

FOR each leaf do
    $k \leftarrow RectNo$ of current leaf
    $CurrentT \leftarrow Rect[k].top$
    $CurrentB \leftarrow Rect[k].bottom$
    $p \leftarrow Rect[k].parent$
    $currentline.rects \leftarrow k$
    $i \leftarrow 1$
    WHILE $p$ is still defined
        IF $overlap(interval(Rect[p].top, Rect[p].bottom),$
        $interval(CurrentT, CurrentB))$ = true
            THEN
                {extend an existing leaf line}
                $currentline.rects \leftarrow$ p || $currentline.rects$
                $currentline.top \leftarrow$ top of overlap
                $currentline.bottom \leftarrow$ bottom of overlap
                $CurrentT \leftarrow$ top of overlap
                $CurrentB \leftarrow$ bottom of overlap
            ELSE
                {add current line to set of leaf lines}
                $LeafLines[i].rects \leftarrow currentline.rects$
                $LeafLines[i].top \leftarrow currentline.top$
                $LeafLines[i].bottom \leftarrow currentline.bottom$
                {start a new leaf line}
                $i \leftarrow i+1$
                $currentline.rects \leftarrow p$
                $currentline.top \leftarrow Rect[p].top$
                $currentline.bottom \leftarrow Rect[p].bottom$
                $CurrentT \leftarrow Rect[p].top$
                $CurrentB \leftarrow Rect[p].bottom$
        $Rect[k].LeafLineNo \leftarrow i$
        $k \leftarrow p$
        $p \leftarrow Rect[k].parent$
```

Figure 4.40: The algorithm for placing orthogonal axial lines in trees of orthogonal rectangles – Stage 4

```
    {Stage 5:}
    {Merge forward lines and leaf lines to produce final lines}

71  Set $m$ to the number of lines in $ForwardLines$
72  FOR $i$ from 1 to $m$
73     $FinalLines[i].top \leftarrow ForwardLines[i].top$
74     $FinalLines[i].bottom \leftarrow ForwardLines[i].bottom$
75     set $FinalLines[i].line$ to be empty
76     FOR each rectangle $j$ in $ForwardLines[i].line$ from right end
       to left end
77        $FinalLines[i].line \leftarrow j$ || $FinalLines[i].line$
78        $k \leftarrow Rect[j].LeafLineNo$
79        IF $overlap(interval(ForwardLines[i].top, ForwardLines[i].bottom),$
          $interval(LeafLines[k].top, LeafLines[k].bottom))$ = true
80           THEN
81              $FinalLines[i].top \leftarrow$ top of overlap
82              $FinalLines[i].bottom \leftarrow$ bottom of overlap
83              Add all rectangles in $LeafLines[k].line$ to
                $FinalLines[i].line$
84              Break out of for loop
```

Figure 4.41: The algorithm for placing orthogonal axial lines in trees of orthogonal rectangles – Stage 5

The adjacencies between the rectangles are found using the algorithm presented in Section 4.5.1. In this stage of the algorithm *parent*, *numadj* and *adjlist* are given values.

In stage 2 of the algorithm an additional array, *RightList*[], is created which stores a list of the indices of the array *Rect*[] arranged according to the x-coordinate of the right end of the rectangles. This list defines the order in which the rectangles will be visited in stage 3 – finding the forward lines. Stages 0 to 2 are essentially preprocessing to define a tree of orthogonal rectangles.

Stage 3 of the algorithm (as given in Figure 4.39) finds the *forward lines*. These are the lines found by starting at the root rectangle (in this case the leftmost rectangle – with the smallest x-coordinate of its right edge) and working towards the leaf rectangles of the tree (these are rectangles with no right neighbours) considering each rectangle in turn. An interval tree [Cormen *et al.*, 1990] is used to maintain the y-value ranges that can be considered at any stage. An interval tree is a red-black tree where each vertex x contains an (open or closed) interval defined by its low and high endpoints and where the *key* of the vertex is the low endpoint. Insertions into the tree are based on the low endpoint of the interval to be inserted. Thus an inorder traversal of an interval tree would return the intervals in sorted order by low endpoint. In the algorithm above every vertex in the interval tree stores an interval defined by the variables *low* to *high* and, in addition, a candidate line (*line*) represented by a list of the rectangles such a line could cross. Each vertex in the interval tree thus represents a line that might need to be considered when attempting to extend lines from the root to the leaf rectangles. Lines 12 to 14 initialise the root vertex of the interval tree to contain the interval represented by the y-value range of the root rectangle and a line that crosses only that rectangle but could be extended into the root rectangle's right neighbours.

After the initialisation each rectangle is considered in turn (lines 15 to 36 of Figure 4.39). Each such rectangle can have at most one rectangle adjacent to it on the left and could have a number of other rectangles adjacent to it on the right. If it has no rectangles adjacent to it on the right then it is a leaf rectangle and its number is added to a list of such leaf rectangles (line 17 to 19). If it has adjacent rectangles on the right then the forward line coming into the rectangle could potentially be extended into these adjacent rectangles (lines 21 to 36). The algorithm searches the interval tree for the line which comes into the rectangle from the left – it will have an interval which has some overlap with the y-value range of the rectangle and the last rectangle through which the line passes will be the current rectangle (line 21). The algorithm then considers each right neighbour of the current rectangle in turn (lines 22 to 35) and determines which adjacencies can be crossed by extensions of this line. There could be more than one possible extension of the line depending on the y-coordinates of the adjacencies being considered and thus the one line coming into a rectangle from the left could become more than one line (see lines 26 to 30) –

one line going into each adjacent rectangle where there is an overlap of the interval covered by the line and bottom and top y-values of the adjacent rectangle. Each of these new lines would result in a new vertex being inserted into the interval tree to store the new interval and detail which rectangles are crossed by the new line. The algorithm also determines for which adjacencies (if any) new forward lines have to be started (lines 32 to 35), i.e. the existing line cannot be extended to cross the new adjacencies (there is no overlap of y-coordinates). Again new vertices are created in the interval tree.

After all the right neighbours have been considered then the vertex in the interval tree representing the line coming into the current rectangle is deleted if the line has been extended (it is no longer needed in this case). If the line has not been extended then it is one of the forward lines and is thus not deleted. The last step in this stage of the algorithm (lines 37 to 42) is to traverse the interval tree and produce a list of the forward lines, represented by a range of y values and a list of rectangles crossed, to cross all of the adjacencies in the tree of rectangles.

Figure 4.42 shows the forward lines generated for part of the tree of Figure 4.31. Note that the line that crosses the adjacencies between rectangles a and b and b and c can be extended in d but not into e. A new forward line is created to cross the adjacency between rectangles c and e. This line can be extended into f but not into g.

Stage 4 of the algorithm (as given in Figure 4.40) works from the leaves of the tree to the root rectangle, hence finding the *leaf lines*. This case is easier than that of finding the forward lines. It is only necessary to check if one of the lines coming into any rectangle from the right can be extended to cross the one adjacency on the left or whether a new leaf line must be created. The algorithm considers each leaf rectangle (from line 19 in stage 3) in turn (lines 43 to 70). The leaf line for any leaf rectangle is first initialised (lines 44 to 48) – this involves finding the interval to be considered, finding the parent of that rectangle and setting the first rectangle in the current line. The algorithm then works up the rectangle tree until the root rectangle is reached (lines 48 to 68). For each rectangle on the path from leaf rectangle to root rectangle it determines if the leaf line coming into a rectangle can be extended into the current rectangle's parent (lines 53 to 57) or if the current leaf line must be terminated and a new leaf line started (lines 59 to 67). This stage of the algorithm also records the number of the last leaf line that crosses from the current rectangle into its parent rectangle (*LeafLineNo* – line 68) – this could be a new leaf line (starts in that rectangle) or one which crosses the rectangle from right to left. This information is used in stage 5 of the algorithm which is described below. Figure 4.42 shows the leaf lines generated for part of the tree of Figure 4.31. Note that the leaf line from rectangle g cannot be extended from e into c and a new "leaf" line has to be created here.

The final stage of the algorithm (stage 5 as given in Figure 4.41) merges the

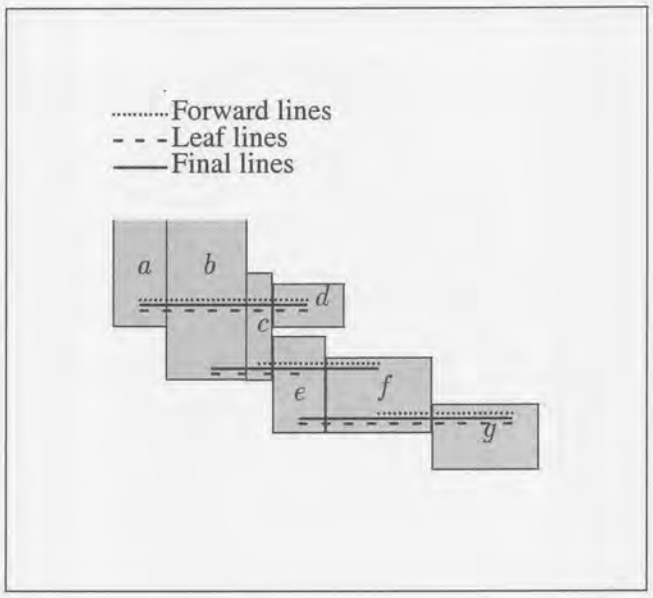

Figure 4.42: An example of placing orthogonal axial lines in a tree of orthogonal rectangles

forward and leaf lines to produce the set of maximal *final lines* which cross all of the adjacencies in the tree of rectangles. Section 4.7.2.3 explained why these lines need to be merged in order to obtain maximal axial lines. The merging here is necessarily more complicated than when working with chains of rectangles because any rectangle could be crossed by more than one forward line and more than one leaf line. Here the algorithm considers each forward line in turn (lines 71 to 84 in Figure 4.41). A new final line is created for each forward line (lines 73 to 75) and the algorithm then considers each rectangle in the forward line in turn starting from the rightmost end of the line – as before forward lines define the rightmost extent of any final line (lines 76 to 84). The leaf line which corresponds to the forward line is found by considering the leaf line (determined in stage 4, line 68) associated with each of the rectangles which make up the forward line in turn, until a leaf line that has an overlapping range of y-values is found (line 79). This leaf line then defines the leftmost extent (leaf lines are always extended as far as possible to the left). The final line is created by using the adjacencies from the forward line and its associated leaf line – the adjacencies from the forward line are used (lines 77 and 78) to define the right end of the final line and when the associated leaf line is found its adjacencies are used to define the left extent of the line (lines 81-84). Figure 4.42 shows the final lines (and forward and leaf lines) generated for part of the tree of Figure 4.31. The forward line c–e–f and the leaf line b–c–e which together result in the final line $b - c - e - f$ is an example of how the merging works. Rectangle f is considered first, the leaf line through f does not have an overlapping y-value range so rectangle e is considered. Here the leaf line which was stored in stage 4 of the algorithm is the line that starts in e and crosses into c. This line does have an overlap so the final line can be generated using the forward line and this leaf line.

This example also illustrates why it is sufficient in stage 4 (line 68) of the algorithm to record only the last leaf line which either starts in or crosses a given rectangle. Leaf lines define the left extent of any final line. If two or more leaf lines start in or cross a given rectangle then the associating of a leaf line and a forward line will not be done when that rectangle is considered. In Figure 4.42, rectangle c has 2 leaf lines which cross it – the line a–b–c–d and the line b–c–e. These leaf lines are paired with their forward lines when rectangle d of forward line a–b–c–d and rectangle e of forward line c–e–f respectively are being considered. Similar arguments apply for other combinations of more than one leaf line either crossing or starting in a given rectangle.

Sanders *et al.* [2000a] show another example of the applying the algorithm to a specific tree.

The algorithm essentially consists of four parts: finding the adjacencies and defining the order of processing, finding the forward lines, finding the leaf lines, and merging the forward lines and leaf lines. To analyse the algorithm each of these stages are considered in turn.

Stage 1 and 2: Find adjacencies and define order The algorithm used to find the adjacencies of the rectangles in stage 1 of the algorithm is as in Section 4.5.1. The algorithm also determines the array *RightList*. This algorithm is dominated by sorting and thus has a complexity of $O(n \lg n)$ where n represents the number of rectangles in the tree.

Stage 3: Find forward lines In essence this phase of the algorithm determines which adjacencies overlap with intervals that have already been inserted into the interval tree – an interval in the interval tree means that there is a forward line which can be considered. There are $n - 1$ adjacencies in a tree of n rectangles and each adjacency only needs to be considered once – only one forward line can cross any adjacency. Also each interval tree operation (adding or deleting here) takes $O(\lg n)$ time [Cormen *et al.*, 1990]. Therefore stage 3 of the algorithm has a complexity of $O(n \lg n)$.

This stage of the algorithm could be done more efficiently as regards time by noting that each rectangle can only have one line coming into it from the left and using a matrix of size $n - 1$ to store the y intervals of these lines. This has the advantage of direct lookup but the disadvantage of always requiring $O(n)$ space – the best case for the interval tree could be much less.

Stage 4: Find leaf lines In this phase of the algorithm the lines starting in the leaves of the tree are extended back towards the root of the tree. Here testing

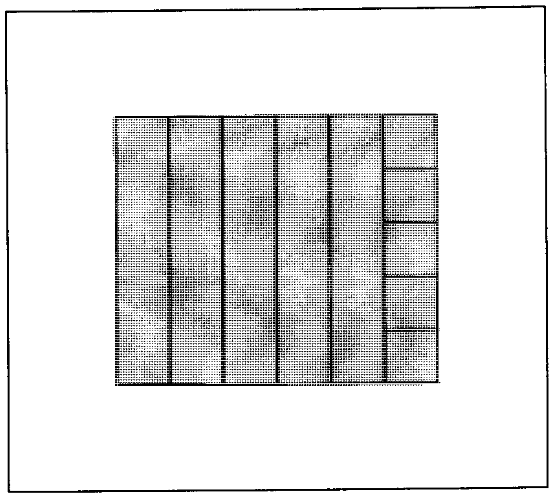
Figure 4.43: A tree with $n/2$ leaves and height also $n/2$

whether the line can be extended is $O(1)$ (a simple test for overlap) but it is possible that there are $O(n)$ leaves and that for each of these leaves it is necessary to extend the line through $O(n)$ rectangles (a tree with approximately $n/2$ leaves and of height also approximately $n/2$ – see Figure 4.43 for a simple example of this case). So the complexity for this part of the algorithm is $O(n^2)$ in the worst case.

Stage 5: Merge forward lines and leaf lines For a forward line to exist, it must cross at least one adjacency between two rectangles in the tree, and each adjacency is crossed exactly once by the definition of the axial line placement problem. Therefore since there are n rectangles, there are exactly $n - 1$ of these adjacencies and there can be a maximum of $O(n)$ forward lines.

The merging phase of the algorithm considers each forward line in turn and in the worst case works from the rightmost rectangle of the line until a leaf line with overlap is encountered. The algorithm then merges these two lines. Effectively this is merging two lines in a chain of rectangles which is $O(n)$ (as shown in Section 4.7.2.3).

So since there are a maximum of $O(n)$ forward lines and merging a forward line with its associated leaf line is $O(n)$, this part of the algorithm is $O(n^2)$ in the worst case.

The complexity of the entire algorithm is thus $O(n^2)$ in the worst case but better performance can be expected from some configurations of input rectangles.

The algorithm was implemented and tested on various configurations of trees. The results of this empirical analysis confirmed the theoretical analysis and also demonstrated that different shapes of the trees of rectangles affect the complexity quite dramatically. In some cases the sorting and finding the forward lines dominate and in other cases finding the leaf lines and merging the forward and leaf lines are the more costly operations. Sanders *et al.* [2000a] presents detailed results of the empirical analysis.

Section 4.7.1 presented some variations of the orthogonal axial line placement problem that can be solved in polynomial time. In this section polynomial time algorithms for the orthogonal axial line placement problem for chains of orthogonal rectangles and trees of orthogonal rectangles have been presented. It is now interesting to consider finding other arrangements of rectangles that are more general than a tree of rectangles that can be solved in polynomial time. This is discussed in the next section of this thesis.

4.7.3 More general cases

The issue to be considered now is whether the results above can be extended to cover the case where each rectangle is joined to at most two rectangles at its left end and two rectangles at its right end (analogously for two rectangles above and two rectangles below). It turns out that this is not the case. This can be seen if the configuration of rectangles shown in Figure 4.44 is considered. This is a configuration which meets the restrictions – each rectangle is adjacent to at most two neighbours to the left and two to the right. Also this configuration of rectangles produces choice – the adjacency between rectangles 4 and 6 can be crossed by the line through $1-3-4-6$ or the line through $1-2-4-6$. Lines $0-1-3-4$ and $1-2-4-5$ are essential lines.

If this configuration of rectangles is treated as a basic building block it is clear that a configuration of rectangles that meets the restriction of at most two adjacent rectangles at each end can be generated and this configuration of rectangles could offer global choice. See for example Figure 4.45. A proof similar to that shown in Theorems 4.4.1 and 4.4.2 can be used to show that this restricted case of the problem is also NP-Complete in the general case.

4.8 Future research

There are a number of interesting research questions that arise from the research discussed in this chapter. Tackling all of these problems is outside the scope of this research. This section of the thesis highlights some of these questions. A more complete coverage can be found Chapter 8 of this document.

As extensions to the work done in this research and discussed in this chapter the following problems are interesting areas of research.

- Reducing the amount of work required by the heuristic algorithm to calculate

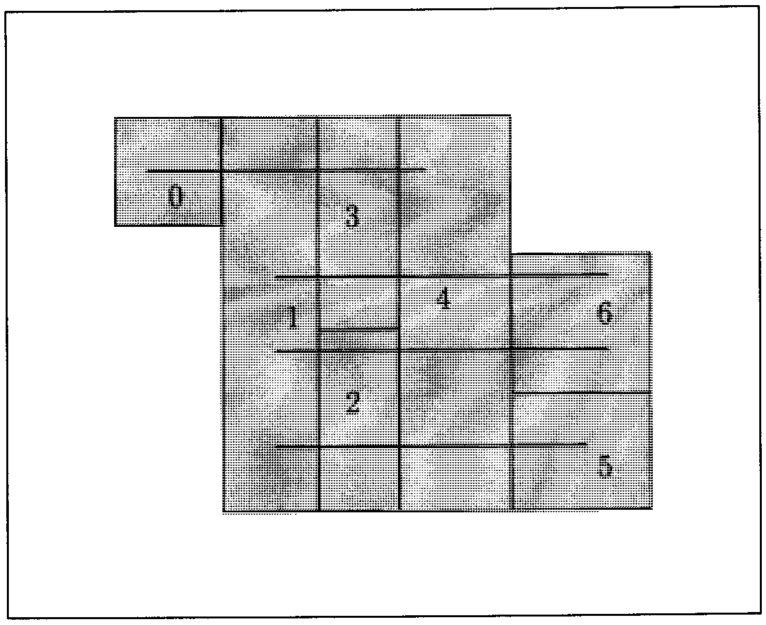

Figure 4.44: Choice introduced where each rectangle has at most 2 left and 2 right neighbours

the non-redundant set of orthogonal axial lines that covers all of the adjacencies in a configuration of adjacent orthogonal rectangles. In particular, attempting to address the issue of redundant calculations which are made for the collection of rectangles shown in Figure 4.22 or similar configurations.

- Developing other heuristics to produce approximate solutions to the exact solution.

- Considering other special cases of the problem that can be solved in polynomial time.

An interesting, but not directly related, area of further research is in the generation of test data. An efficient algorithm for generating configurations of non-overlapping adjacent orthogonal rectangles would be useful in order to test any heuristics that are developed. In addition, generating non-trivial trees of adjacent but non-overlapping orthogonal rectangles proved to be relatively complex in this research and a more efficient way of doing so could be useful for any work at improving the algorithm.

4.9 Conclusion

This chapter addresses the problem of finding the fewest longest orthogonal axial lines that pass through all of the shared adjacencies between adjacent orthogo-

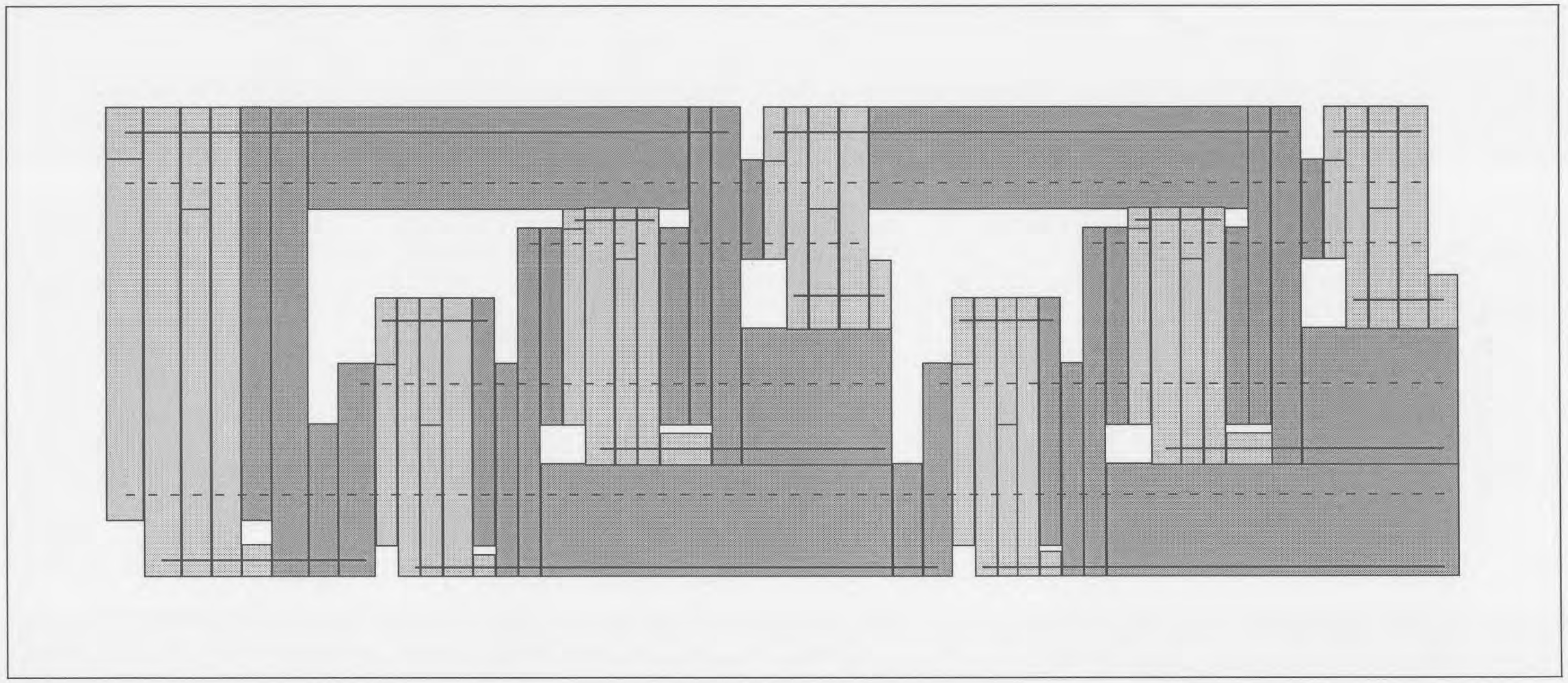

Figure 4.45: An example of choice in configuration where each rectangle has at most 2 left and 2 right neighbours

nal rectangles – ***ALP-OLOR***. The problem is made NP-Complete by the fact that various instances of choice can arise. This chapter of the thesis presents an NP-Completeness proof based on a reduction from ***biconnected planar vertex cover*** to ***stick diagram*** and hence to ***ALP-OLOR***. An $O(n^2)$ algorithm is presented that finds a non-redundant set of lines to cross all of the adjacencies of a collection of adjacent orthogonal rectangles. The algorithm has been shown to produce a very good approximation to the exact solution in all cases tested.

This chapter also presents some restrictions of the orthogonal axial line placement problem for which polynomial time solutions can be obtained – configurations that are mappable to interval graphs, chains of orthogonal rectangles ($O(n)$) and trees of orthogonal rectangles ($O(n^2)$). It is likely that other restrictions exist that are solvable in polynomial time and future research will investigate this area.

The next chapter of this thesis considers the case where the lines which pass through the shared adjacencies between adjacent orthogonal rectangles are no longer restricted to being parallel to the axes – axial lines with arbitrary orientation are acceptable. This problem is also shown to be NP-Complete and once again the use of a heuristic approach to solving the problem is discussed.

Chapter 5

Placing axial lines with arbitrary orientation to cross the adjacencies between orthogonal rectangles

5.1 Introduction

In Chapter 3 the aim and focus of this thesis are discussed. This chapter addresses one of the questions raised in Chapter 3 – the problem of finding the minimum number of longest axial lines with arbitrary orientation which cross all of the adjacencies of a collection of adjacent orthogonal rectangles (***ALP-ALOR*** Axial Line Placement – Arbitrary Lines and Orthogonal Rectangles). In Chapter 4 ***ALP-OLOR*** (Axial Line Placement – Orthogonal Lines and Orthogonal Rectangles) was shown to be NP-Complete by a transformation from biconnected planar vertex cover to stick diagram and then to ***ALP-OLOR***. In this chapter, a similar process is used to show that ***ALP-ALOR*** is NP-Complete. In the next chapter of this thesis (Chapter 6) the NP-Completeness proof is extended to show that ***ALP-ALCP*** (Axial Line Placement – Arbitrary Lines and Convex Polygons) is also NP-Complete.

In the remainder of this chapter (***ALP-ALOR***) is discussed in more detail. The problem is restated in detail in Section 5.2 below. In Section 5.3 ***ALP-ALOR*** is shown to be NP-Complete using a similar transformation to that discussed in Chapter 4 for ***ALP-OLOR***. Then, because ***ALP-ALOR*** is NP-Complete, some ideas for heuristics to find approximate axial maps are discussed in Section 5.5. In Section 5.7 some ideas for future research are briefly discussed.

5.2 Statement of the Problem

ALP-ALOR can be formally stated as follows:
Given a number of adjacent, orthogonally-aligned rectangles find the fewest axial

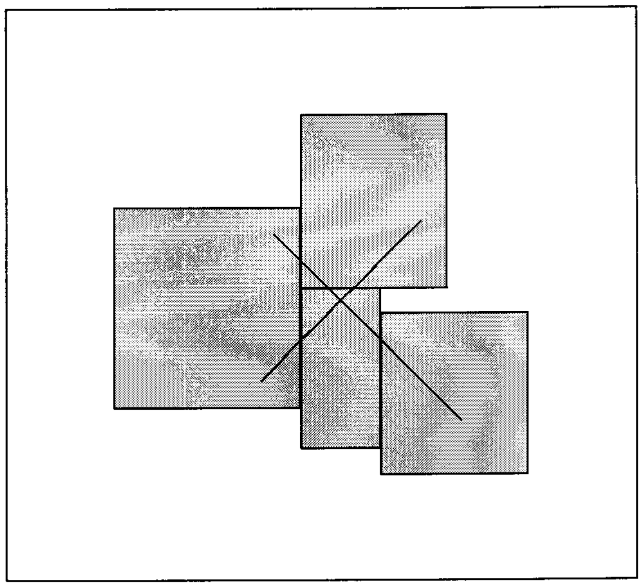

Figure 5.1: An example of the problem

lines (line segments), contained wholly inside the rectangles, required to cross all of the boundaries shared between adjacent rectangles. An additional requirement is that each axial line should cross as many of the shared boundaries as possible – a *maximal* axial line.

As in Chapter 4 for ***ALP-OLOR***, depending on how the problem is considered there are two similar but distinct problems which can be addressed – adjacencies can be crossed more than once but every adjacency must be crossed at least once; and any adjacency must be crossed exactly once. In this thesis only the first variation is addressed. Figure 5.1 shows an example of this.

The decision problem can then be stated as below.

ALP-ALOR
Instance: A collection of orthogonal rectangles $R_1 \ldots R_n$, where each R_i is adjacent to at least one other rectangle, and a positive integer $O \leq 4n$.
Question: Is there a set P of (possibly non-orthogonal) axial lines where each axial line is maximal in length, each axial line is contained wholly within the rectangles, each adjacency is crossed at least once by the axial lines in P and $|P| \leq O$?

In section 5.3 ***ALP-ALOR*** is shown to be NP-Complete.

5.3 Proving the problem is NP-Complete

In Chapter 4 it is shown that the problem of finding the minimum number of maximal *orthogonal* axial lines to cross all of the adjacencies in a configuration of adjacent orthogonal rectangles (***ALP-OLOR***) is NP-Complete. The proof is accomplished through a transformation from ***biconnected planar vertex cover***. This transformation is accomplished by mapping vertices in a biconnected planar graph to choice axial lines. In this mapping an edge between two vertices represents an adjacency which is crossed by two choice axial lines.

The transformation is done in two steps. First, a biconnected planar graph is transformed to a stick diagram where each vertex in the original graph is mapped to a horizontal line and each edge in the original graph is mapped to a vertical line which is crossed by the two horizontal lines which represent the two vertices to which the edge is incident. The problem then becomes that of choosing the minimum number of horizontal lines to cross all of the vertical lines. ***Stick diagram*** is thus NP-Complete – if it is possible to determine in polynomial time which of the set of horizontal lines cross all of the vertical lines in the stick diagram then it is possible to solve ***biconnected planar vertex cover*** in polynomial time. Finding the minimum set of horizontal lines is equivalent to finding the minimum vertex cover of the original graph. Second, the stick diagram is represented as a collection of adjacent orthogonal rectangles and horizontal axial lines crossing all of the adjacencies in the collection of rectangles. These horizontal axial lines are of two types "essential axial lines" which are the only axial lines to cross a particular adjacency and "choice axial lines" where a number of axial lines (none of which are essential) cross some adjacency. Not all of the choice axial lines are necessary to cross all of the adjacencies in the collection of rectangles. If it is possible to determine in polynomial time which of the set of choice axial lines cross all of the adjacencies in the diagram then it is possible to solve ***stick diagram*** in polynomial time – finding the minimum set of choice axial lines is equivalent to finding the minimum set of horizontal lines. Thus ***ALP-OLOR*** has been shown to be NP-Complete.

In this chapter the fact that ***stick diagram*** is NP-Complete is used to show that ***ALP-ALOR*** is also NP-Complete. This is done by transforming an instance of ***stick diagram*** to an instance of ***ALP-ALOR***. Once again this is accomplished by using "choice units" although the units used here are somewhat different to those used in Chapter 4 and different choice axial lines with arbitrary orientation are generated. Note that, in the remainder of this chapter any reference to an axial line should be taken to mean an axial line which is not necessarily orthogonal.

Theorem 5.3.1 ***ALP-ALOR*** *is NP-Complete*

Proof

Clearly ***ALP-ALOR*** is in NP. Given a set of axial lines with arbitrary orientation it

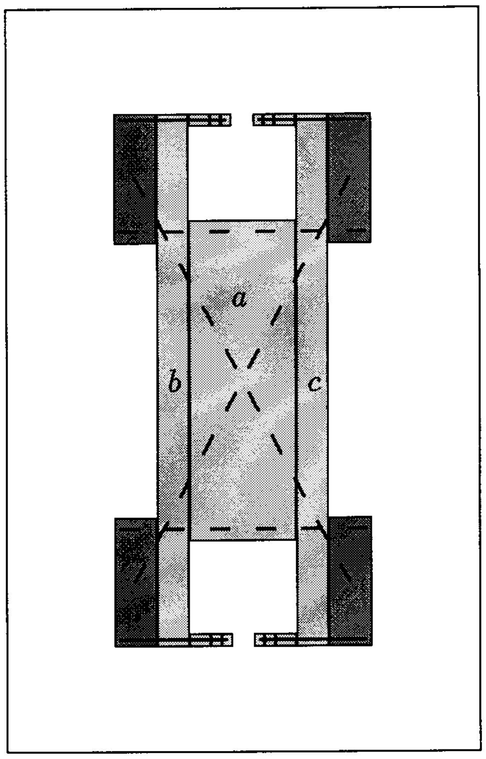

Figure 5.2: The Canonical Choice Unit which produces choice axial lines with arbitrary orientation

is possible to check in polynomial time that each adjacency has been crossed by at least one axial line.

Now transform ***stick diagram*** to ***ALP-ALOR***.

A collection of rectangles which create choice axial lines with arbitrary orientation can be represented by a canonical choice unit, *ccu*, shown in Figure 5.2. In this ccu, the adjacencies between the middle rectangle a and the rectangles b and c can be crossed by four "sets" of axial lines with arbitrary orientation (the upper, lower and two diagonal sets). Figure 5.2 shows as a dashed line a representative axial line from each of the four sets. Only one of these axial lines is actually necessary to cross the adjacencies between rectangles a, b and c. All the other adjacencies are crossed by the axial lines which originate in the "horns" of the ccu. These axial lines *do not* have to be horizontal but the size and position of the rectangles in the horns means that the axial lines are restricted to a small range of different slopes. Scaling of the canonical choice unit does not change the fact that it can/does produce choice axial lines.

The transformation proceeds by replacing each vertical line in the stick diagram by a ccu of an appropriate size. The horizontal lines which cross through the vertical line are represented by a subset of the choice axial lines of the ccu. It is then

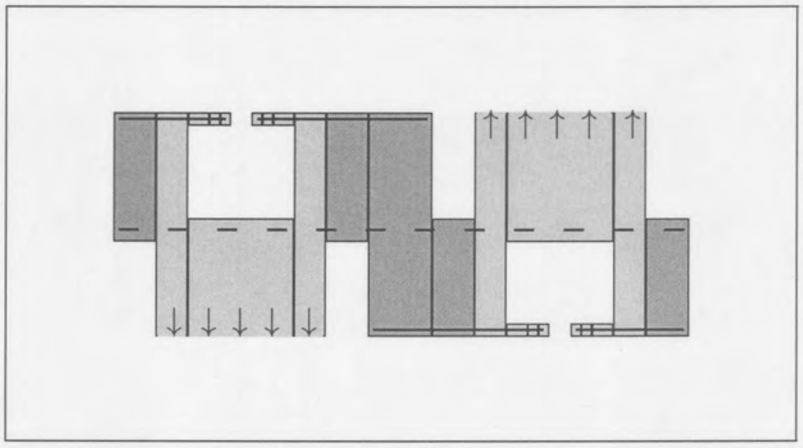

Figure 5.3: Connecting the upper portion of one ccu to the lower portion of the next

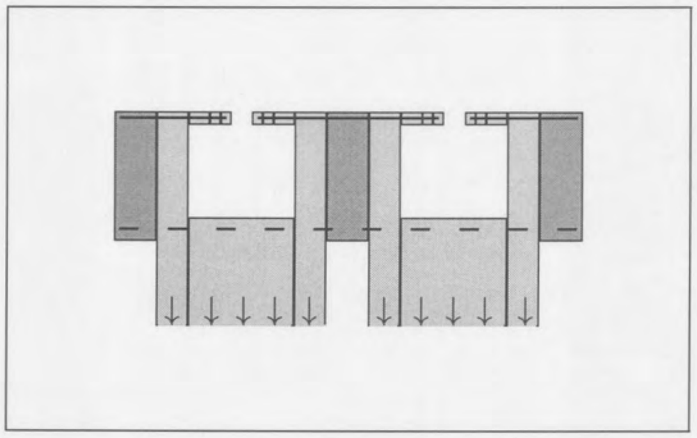

Figure 5.4: Connecting the upper portions of two ccus

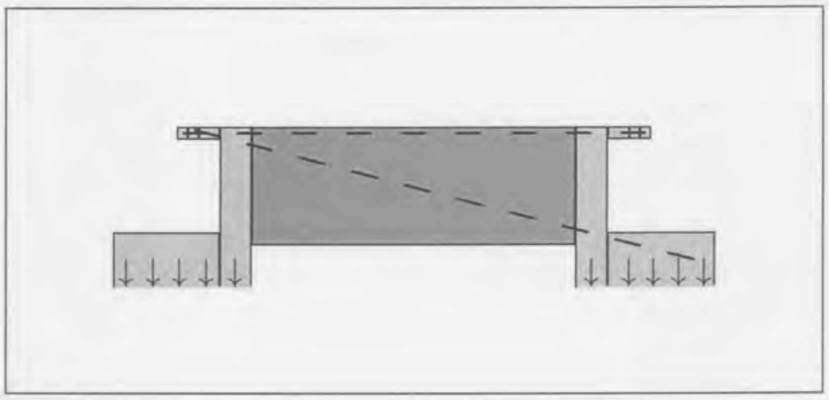

Figure 5.5: Possible rays from a "horn"

necessary to show that these canonical choice units can be joined together in a fashion which maintains the relation between the horizontal lines in the stick diagram and the choice axial lines in the configuration of adjacent rectangles. The situation here is somewhat different from Chapter 4 for ***ALP-OLOR***. There the fact that the choice and essential axial lines had to be horizontal could be used to control the stopping or continuing of axial lines. In this case the size and length to breadth ratio of the ccu's and the connecting rectangles are more critical.

As in Chapter 4, there are four ways in which horizontal lines could cross through successive vertical lines. (Actually there are only two ways, each with a vertical reflection.) These are

- the horizontal line could be the upper (lower) line through one vertical line and the upper (lower) line through the next vertical line,
- the horizontal line could be the upper (lower) line through one vertical line and the lower (upper) line through the next vertical line.

It must be shown that in each of these cases it is possible to connect two ccu's in such a fashion that the choice is preserved.

These cases are now considered in turn.

- *upper to lower*
  Here the two ccu's are connected by placing a rectangle of appropriate size into the position indicated in Figure 5.3. This "connecting rectangle" ensures that the upper choice axial line through the left ccu and the lower choice axial line through the right ccu are the same choice axial line. After adding the connecting rectangles, the adjacencies between rectangles a, b and c in each ccu are still only crossed by choice axial lines. The sets of choice axial lines which cross each ccu from the bottom left to the top right (and top left to bottom right) are still possibilities for crossing the adjacencies between

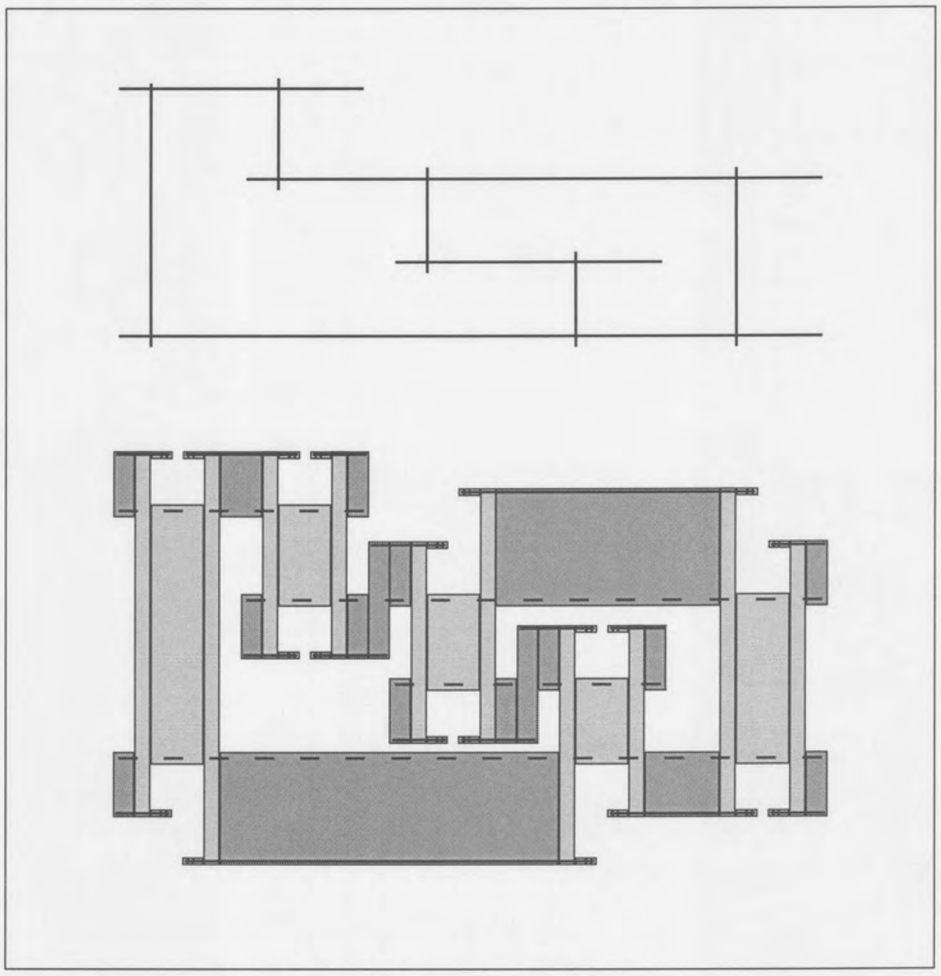

Figure 5.6: An example of converting a stick diagram to a collection of adjacent rectangles

a, b and c in each ccu but there are longer axial lines which cross the same adjacencies so axial lines from these sets will not be in the final set of axial lines for the collection of rectangles. The axial lines from the horns in the left ccu cannot be extended to cross the adjacencies between a, b and c in the right ccu. The only axial line which could be extended into the right ccu in this case is the upper choice axial line from the lefthand ccu. After connecting the two ccu's, the adjacencies in both ccu's which were only crossed by choice lines are still only crossed by choice lines. All other adjacencies are crossed by essential axial lines. In this sense, choice is maintained.

- *lower to upper*
  This is a mirror image of the case above about the x-axis, see Figure 5.3.
- *upper to upper*
  In this case no connecting rectangles are required, it is enough to simply merge the appropriate connector rectangles. This is shown in Figure 5.4. Again the adjacencies previously crossed only by choice lines are still only crossed by choice lines and the essential axial lines cross all other adjacencies.
- *lower to lower*
  Again this case is a mirror image of the case above, see Figure 5.4.

A potential problem could arise when a number of ccu's are joined together to make up a collection of adjacent rectangles which represents a stick diagram. This could happen because it is possible for the axial line through the rectangles in one of the horns to cross one of the adjacencies previously crossed only by choice axial lines. This can be seen in Figure 5.5 where the range of axial lines from the horn in the left ccu includes axial lines which cross an adjacency which was previously only crossed by choice axial lines. This problem can be overcome by some very simple changes to the structure of the ccu. The horn could be made longer – this would be accomplished by keeping the two smaller rectangles in the horn the same size and making the longer rectangle still longer. This would have the effect of reducing the angle at which axial lines could leave the horn and thus no essential axial lines could cross the adjacency previously crossed only by choice axial lines. The horn could also be made narrower by changing the vertical extent of the rectangles in the horn. This would have the same effect of reducing the angles at which axial lines could leave the horn. Other ways of addressing the problems could be to increase the height of the b and c type rectangles and move the horns higher up these rectangles. In this case the range of angles at which axial lines could leave the horn is kept constant but the required range is increased. A similar approach would be to make the connector rectangle shorter.

The transformation from ***stick diagram*** to ***ALP-ALOR*** is thus accomplished by inserting an appropriately sized ccu for each vertical line and then joining these up by using the appropriate connecting rectangles working from the leftmost to the rightmost ccu. ccu's should also be scaled as necessary to ensure that the axial lines from the horns (which are essential) cannot interfere which the issue of choice. Figure 5.6 shows the final result after taking an instance of ***stick diagram*** and converting it to an instance of ***ALP-ALOR***. Note that in this diagram some ccu's have been scaled to ensure that the essential axial lines from the horns do not interfere with the choice.

It is now necessary to show that there is a solution to ***stick diagram*** if and only if there is a solution to ***ALP-ALOR***. The construction of the collection of adjacent rectangles from the stick diagram changes the horizontal lines in the stick diagram to choice axial lines in the collection of rectangles. It also introduces 4 essential axial lines for every ccu added (note, some of these axial lines could be shared across ccu's – see Figure 5.6 for an example of this). These essential axial lines must be in the final solution to ***ALP-ALOR***. Suppose there is a solution for ***stick diagram***, i.e. there exists a set of lines H' such that $|H'| \leq S$. Then there must be a solution P to ***ALP-ALOR*** with $|P| = |H'| + 4|U| - p$, where U is the number of ccu's and p is the number of shared essential axial lines. This is because the essential axial lines must be in P and the choice axial lines which correspond to the selected horizontal lines in S must also be in P. Conversely if there is a solution P to ***ALP-ALOR*** then there must be a solution $S = P - \{e \mid e \text{ is an essential line in } P\}$.

This transformation can clearly be done in polynomial time – each vertical line is visited twice, once when it is replaced by a ccu and a second time when it is connected to the ccu(s) to its right in the stick diagram. If the stick diagram can be drawn then a configuration of ccu's can be drawn by scaling the ccu's to be the same size as the vertical lines that they represent. The ccu's (and their connecting rectangles) can thus be drawn as a non-overlapping collection of adjacent rectangles – an instance of ***ALP-ALOR***.

ALP-ALOR is thus NP-Complete.

□

The result which has been proved above can be extended to axial lines with arbitrary orientation crossing the adjacent edges between convex polygons ***ALP-NLCP*** – rectangles are a special case of convex polygons. This is discussed in more detail in Chapter 6.

The fact that ***ALP-ALOR*** is NP-Complete means that in general only exact solutions for small problem instances can be found and that investigating heuristics is a worthwhile area of research. In Section 5.5 of this thesis some ideas for possible heuristics to be used in finding approximate solutions for ***ALP-ALOR*** are discussed. Fully investigating these heuristics is outside of the scope of this thesis but they are presented to give the reader some idea of approaches which could be taken.

A problem which arises, whether an exact solution is being calculated or a heuristic is being applied to find an approximate solution, is to be able to efficiently decide whether an axial line can be placed to cross a given set of adjacencies. The next section of this thesis (Section 5.4) addresses this problem before heuristics which are assumed to use the idea proposed here are presented.

5.4 Determining whether axial lines can be placed in chains of adjacent rectangles

In Chapter 4 which deals with placing orthogonal lines to cross the adjacencies between orthogonal rectangles, it is very easy to determine whether an axial line crosses a number of adjacencies. If the adjacent rectangles which give rise to the adjacencies are treated as a chain of rectangles (i.e. the rectangles are adjacent and each rectangle has at most one left neighbour and one right neighbour) and if there is an overlap in the y-values of the adjacencies then an axial line can be placed to cross those adjacencies. Note that, as discussed before, an axial line is actually a representative of a set of lines all of which fall within the same range of y-values and cross the same adjacencies between rectangles. In the algorithms discussed in Chapter 4 (Sections 4.5, 4.7.2.3 and 4.7.2.4) each time a line is extended forwards or backwards in effect an axial line is being extended through a chain of rectangles.

In the case of arbitrary lines and orthogonal rectangles, a single axial line could cross both vertical and horizontal adjacencies and these thus have to be considered at the same time. This makes determining whether an axial line can be placed to cross all of the adjacencies in a "chain" of adjacent rectangles more complicated. Here a chain of rectangles is defined as a sequence of adjacent rectangles where each rectangle has at most one "incoming" neighbour (left, top or bottom) and one "outgoing" neighbour (right, top or bottom). In Figure 5.7 the rectangles 1, 2, 4, 5 and 7 can be said to make up a chain of rectangles. As can the rectangles 1, 2, 4, 6 and 8. At some stage of an algorithm which finds the axial lines for the configuration of rectangles in Figure 5.7 it might be necessary to determine whether an axial line can be placed to cross the adjacencies 1–2, 2–4, 4–5 and 5–7 in the chain of rectangles 1, 2, 4, 5 and 7. As can be seen in this example such a line can, in fact, be found – the difficulty is in deciding how to place the line or how to determine the range of possible lines. Clearly this could be accomplished by solving sets of simultaneous equations to find the lines with the minimum and maximum slopes which intersect the adjacencies. Another approach is to use the idea of visibility in polygons (as discussed in Chapter 2, Section 2.4.4.3.1).

The idea behind this approach is to convert the problem of determining whether an axial line can be placed to cross the adjacencies in a given chain of rectangles to the problem of determining partial visibility of two edges.

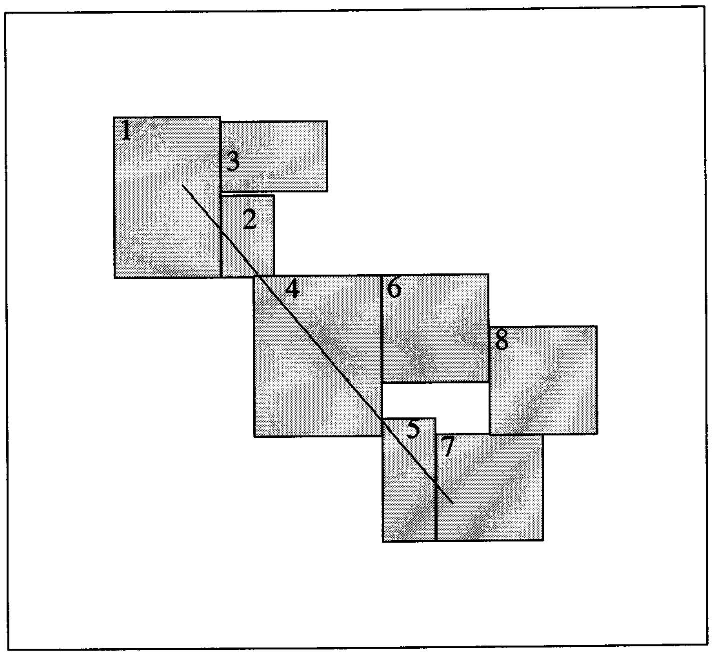

Figure 5.7: Placing a arbitrary axial line in a chain of rectangles

The process is as follows. Given a chain of rectangles, convert the chain into an adjacency polygon – an adjacency polygon is a polygon which uses the first and last adjacency in the chain of rectangles as two of the edges of the resultant polygon and remaining edges of the adjacency polygon are the edges or the parts of the edges in the remaining rectangles in the chain. Figure 5.8 shows the chain of rectangles discussed previously and its associated adjacency polygon. If it were possible for an axial line to be placed to cross all of the adjacencies in the chain of rectangles then there will be partial edge visibility between the two edges of the adjacency polygon which were the first and last adjacencies in the chain (edges a_f and a_l in Figure 5.8). Bilbrough and Sanders [1998] discusses an expected time linear algorithm for determining this partial visibility.

The algorithm can be converted to a linear algorithm by making use of an approach suggested by Avis *et al.* [1986] of generating the "Inner Convex Hull" of the adjacency polygon. This is done by finding the inner convex hulls of the top chain and the bottom chain respectively and using these to create the "reduced" top and bottom chains. See Figure 5.9 for an example of how this is accomplished. du Plessis and Sanders [2000] discuss in detail how the reduced adjacency polygon can be calculated and from that how partial edge visibility can be determined. Again, if it were possible for an axial line to be placed to cross all of the adjacencies in the chain of rectangles then there will be partial edge visibility between the two edges of the reduced adjacency polygon which were the first and last adjacencies in the chain (edges a_f and a_l in Figure 5.9). This approach is used in the heuristics

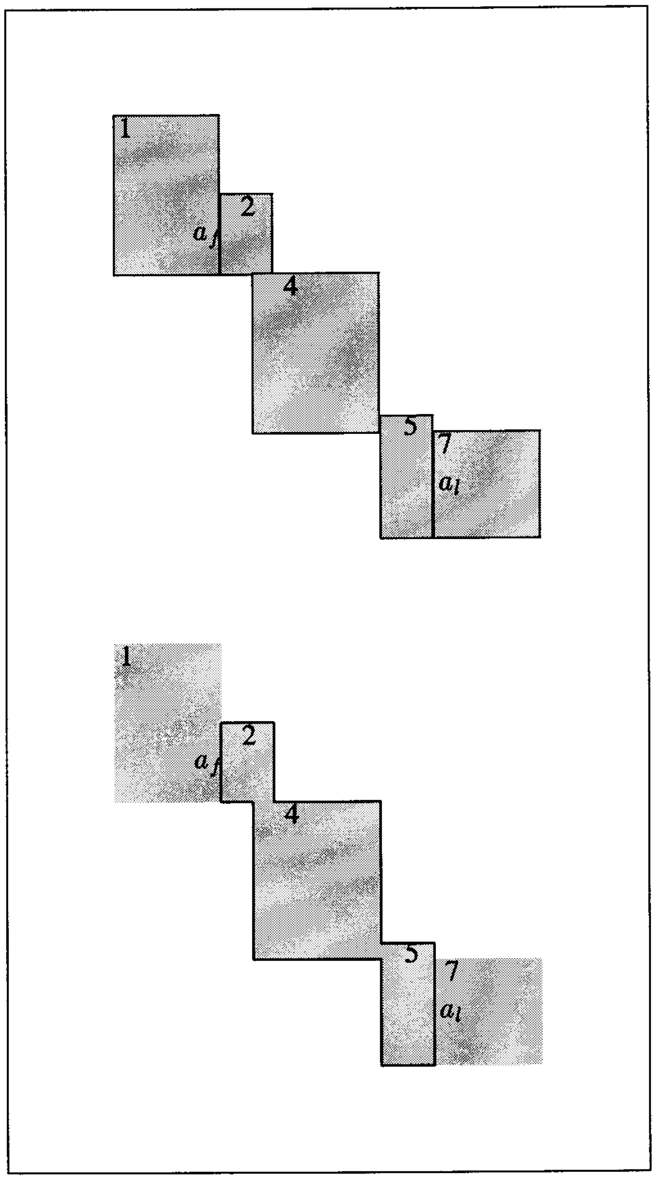

Figure 5.8: Converting a chain of rectangles into an adjacency polygon

discussed below.

5.5 Heuristics to find approximate solutions for *ALP-ALOR*

5.5.1 Overview

ALP-ALOR has been shown to be NP-Complete (Section 5.3) and thus in the general case finding a minimal solution could take an unreasonable amount of time. It is thus necessary to consider heuristics to find approximate solutions in reasonable time. In this section some ideas for heuristics are presented but the actual testing of the heuristics is beyond the scope of this thesis and thus will be future work which arises from this thesis.

5.5.2 Extending lines into all neighbours

In Section 4.5 an O(n^2) algorithm to return a non-redundant set of maximal orthogonal axial lines for the problem of placing orthogonal axial lines to cross the adjacencies between orthogonal rectangles (***ALP-OLOR***) was presented. This algorithm has four phases (after determining which rectangles are adjacent). It generates all the possible orthogonal axial lines which cross the adjacencies between rectangles; it determines which axial lines are essential (i.e. are the only lines which cross a particular adjacency); it removes any lines which only cross adjacencies crossed by the essential lines (redundant lines); and then it resolves the choice conflict. The resolving of the choice is done by repeatedly choosing the choice line which crosses the highest number of previously uncrossed adjacencies. One possible approaching to finding an approximation to ***ALP-ALOR*** is to adopt a similar approach to that used before. The main difference would be that the restriction on the axial lines being orthogonal would be removed – lines with arbitrary orientation would be allowed. This would mean that the algorithm would need to consider *all* neighbours rather than just the right neighbours of a particular rectangle. All of these adjacencies would have to be determined and thus the algorithm for calculating the adjacencies between the rectangles would be more complicated – the algorithm developed in Chapter 4 will no longer suffice. In addition, there seem to be two potential problems with adapting this algorithm to work in this case.

First, the algorithm for ***ALP-OLOR*** works by considering each rectangle in turn from left to right at each step attempting to extend any lines which cross into a rectangle from the left into its right neighbours. In the arbitrary case it would be necessary to determine if the incoming lines could be extended into all the current rectangle's neighbours. The effect of this change could mean that many more lines

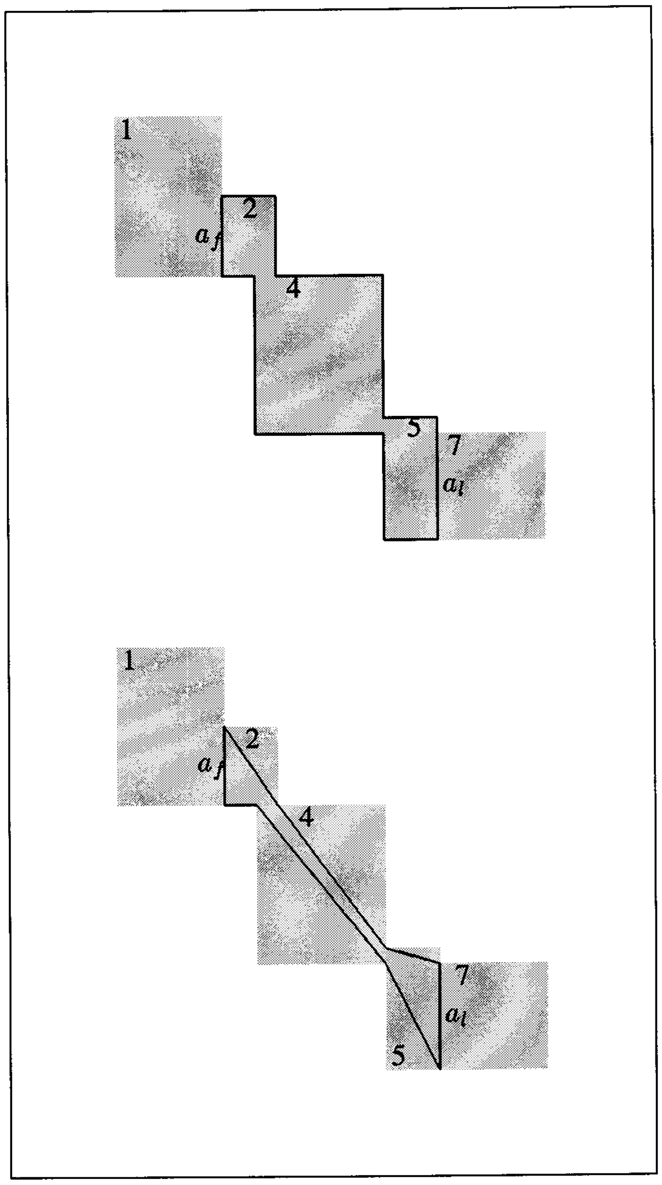

Figure 5.9: Converting an adjacency polygon into a reduced adjacency polygon

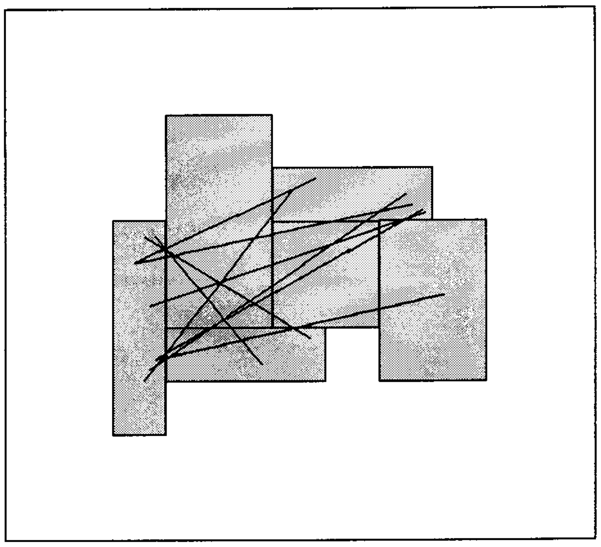

Figure 5.10: An example of using the algorithm extend lines into all neighbours

than are actually necessary are generated in the first phase of the algorithm and many of these lines would have to be removed later – either as redundant lines or in the phase of resolving choice. Figure 5.10 shows an example of the axial lines which would be generated by extending any axial lines coming into a rectangle into all of its neighbours.

Second, it seems that the lines which are generated are very dependent on the initial configuration of the rectangles and some lines which could be part of a minimal set of axial lines would not be generated by this slightly modified algorithm. Figure 5.11 shows some lines which would not be generated by this changed algorithm. This happens because, in this particular example, as each rectangle is considered it is possible to extend one (or more) of the incoming lines into each of its neighbours. There is never the necessity to start a new line.

Some way of addressing these problems would be required if this approach was to be used. Alternatively a different approach should be considered. Some other approaches are suggested in the following sections of this thesis.

5.5.3 Separating top-bottom and left-right lines

Another heuristic for finding an approximate solution for ***ALP-ALOR*** would be to calculate a solution in two passes – a horizontal sweep followed by a vertical sweep. In the horizontal sweep the algorithm works from the leftmost rectangle to the rightmost rectangle in the configuration. For each rectangle, any line which enters the rectangle from the left is checked to see if it can be extended into one or more of the rectangle's right neighbours. If any line can be extended, this new line is added to the set of lines and the line which was extended is removed. This is analogous

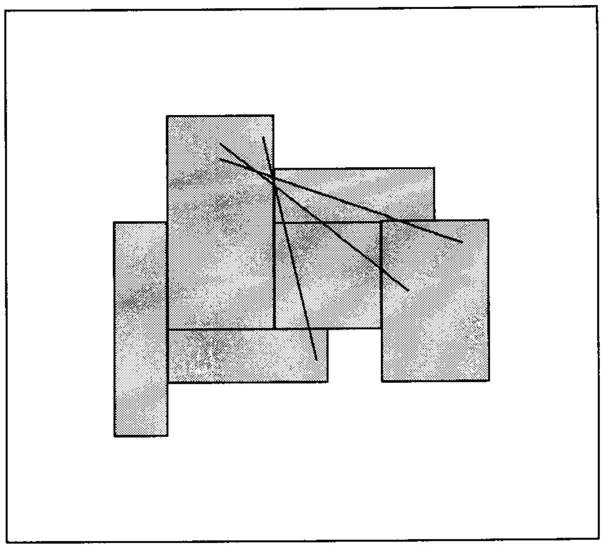

Figure 5.11: An example showing some lines which would not be generated

to the solution for ***ALP-OLOR*** except here the lines need not be orthogonal. The vertical sweep works from the highest rectangle (based on the y-value of the top edges of the rectangles) to the lowest rectangle (the rectangle which has the lowest y-value for its top edge). This second pass extends lines which enter the rectangle being considered from the top into its lower neighbours (if this can be done). Each of these passes would (as before) generate all possible lines (while obeying the left-right or top-bottom constraint); determine essential lines; remove redundant lines; and resolve any choice. These two passes then generate two sets of lines – a set which crosses the vertical adjacencies and a set which crosses the horizontal adjacencies. Combining the final two sets of lines would give an approximation to ***ALP-ALOR***. Figure 5.12 shows an example of the lines which would be generated when extending lines from left to right and top to bottom.

This approach seems unlikely to give an approximation which is close to the exact solution but at least it is more efficient in terms of the number of unnecessary lines than the approach of extending all possible lines (see Section 5.5.2).

An advantage of this approach is that the left-right and top-bottom neighbours can be determined separately and thus the algorithm from Chapter 4 can be used directly.

5.5.4 Longest Chains

A slightly different approach for finding an approximate solution for ***ALP-ALOR*** could be to attempt to find the longest possible axial line at any stage of the algorithm. Clearly to do this would require first generating all possible lines which is inefficient (see the argument in Section 5.5.2). An approach which goes some

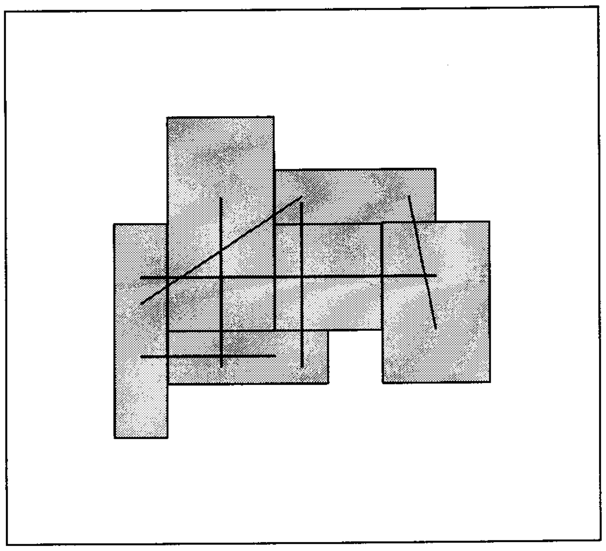

Figure 5.12: An example showing the lines which would be generated in a top-bottom and left-right manner

way towards using this idea is to identify "extreme" rectangles, generate the chains which include these rectangles and then successively choose the longest of such chains until all of the adjacencies have been crossed. The motivation for doing this is that lines which cross the adjacencies of these extreme rectangles could potentially cross a number of other adjacencies as well and so might be a good "greedy" choice. An algorithm to find an approximation based on this idea is sketched below.

1. Determine all of the adjacencies between the orthogonal rectangles. Store this information in an adjacency matrix.

2. Determine all of the extreme rectangles – those which do not have neighbours on both of the top and bottom and the left and right sides. In Figure 5.13 rectangles 1, 2, 5 and 6 are extreme rectangles (the extreme rectangles are more lightly shaded than the rectangles which are not characterised as extreme). Rectangle 3 has left and right neighbours and is thus not extreme. Rectangle 4 has left and right and top and bottom neighbours and is thus not extreme.

3. As long as there are still some unvisited extreme rectangles.

   (a) Pick the next extreme rectangle

   (b) For each adjacency of the extreme rectangle which has not yet been crossed

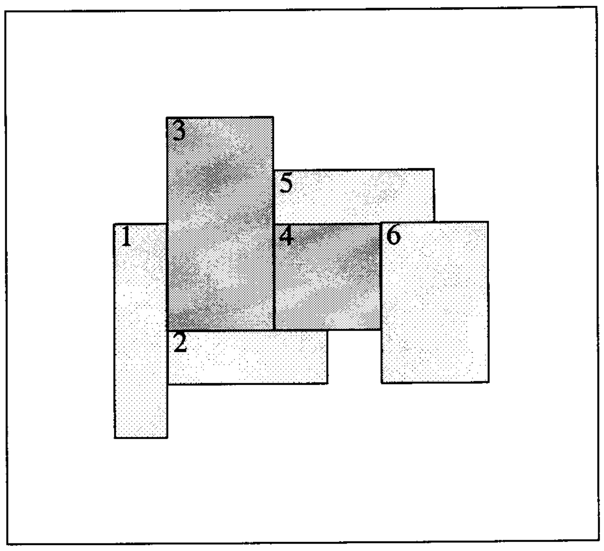

Figure 5.13: Longest chains – identifying "extreme" rectangles

i. using the information stored in the adjacency matrix determine all of the chains of rectangles starting from the extreme rectangle selected in 3a, crossing the current adjacency and ending in another extreme rectangle.

ii. select the longest chain – the chain with the most rectangles in it – that includes that adjacency

iii. determine whether all of the adjacencies in this longest chain can be crossed by one line. To answer this question the approach of determining partial visibility between two edges in a polygon (see Section 5.4) can be used.

   If all of the adjacencies in this chain can be crossed by one axial line

   - include that line as one of the final set of lines
   - mark all of the adjacencies in that chain as crossed

   Or else if all of the adjacencies in this chain *cannot* be crossed by one axial line then

   - select the next longest chain of rectangles and repeat the process until an axial line can be placed or there are no more chains

(c) Repeat this process until all the extreme rectangles have been considered.

4. If there are still uncrossed adjacencies then place an axial line to cross each of those adjacencies.

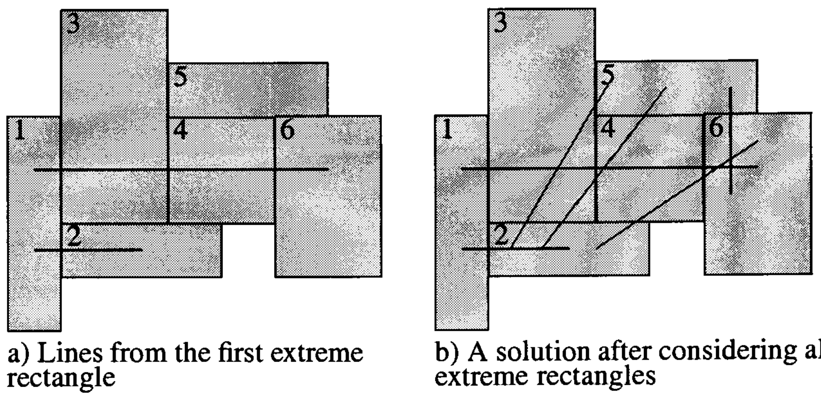
3
5
1
4
6
2
a) Lines from the first extreme rectangle
3
5
1
4
6
2
b) A solution after considering all extreme rectangles

Figure 5.14: The longest chain heuristic

5. The resulting set of lines is an approximation to the solution for ***ALP-ALOR***

If this heuristic was applied to the configuration of rectangles in Figure 5.13 then the first extreme rectangle to be considered would be rectangle 1. This gives rise to the chains 1–2, 1–3–2, 1–3–5, 1–3–4–2, 1–3–4–5 and 1–3–4–6. These chains all start and end with one of the extreme rectangles. A chain like 1–2–3–4–5 would not be generated as it has an extreme rectangle 2 which is not at the beginning or end of the chain. The algorithm would stop generating a chain as soon as rectangle 2 was encountered. In the list of valid chains originating from rectangle 1, the chain 1–2 is the only chain that includes adjacency 1|2 so this chain is selected and the line 1–2 can be placed. There are three longest chains which include the adjacency 1|3 and any could be chosen. If the chain 1–3–4–6 was chosen then the line 1–3–4–6 could be placed. Figure 5.14.a shows the lines placed for the first extreme rectangle. Figure 5.14.b shows a possible solution once all of the extreme rectangles have been visited. This figure also demonstrates a problem with the longest line heuristic – the line crossing adjacency 1|2 is not "as long as possible" – the line should have been extended to cross adjacencies 2|3, 3|4 and 4|6.

Clearly other solutions could occur if different chains are selected when there is a choice of "longest chains". This local choice can, in fact, lead to another problem with this heuristic. This is shown in Figure 5.15. In this case the line chosen to cross the adjacency 1|3 was the line 1–3–4–5. This would leave the adjacency 4|6 uncrossed until the extreme rectangle 6 was considered. Then the line 1–3–4–6 would be placed. This would result in a solution with one more line than if 1–3–4–6

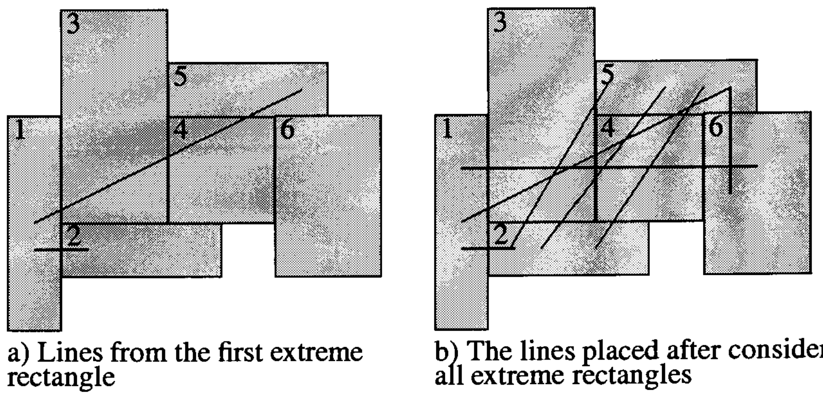
3
5
1
4
6
2
a) Lines from the first extreme rectangle
3
5
1
4
6
2
b) The lines placed after considering all extreme rectangles

Figure 5.15: The longest chain heuristic: A problem with the heuristic – there are redundant lines in the final set of lines

had been chosen when the adjacency 1|3 was being considered.

Another problem with this "longest chains" heuristic (related to a problem discussed earlier) could occur if the configuration of rectangles is as shown in Figure 5.16.a. Here the configuration has only two extreme rectangles 1 and 7. There are two chains from 1 to 7 (and from 7 to 1) but in neither chain can one line cross all of the adjacencies between the rectangles in the chain. Thus no line will be placed while considering each extreme rectangle in turn. Thereafter all the adjacencies are still uncrossed and lines will be placed to cross each adjacency – see the example in Figure 5.16.b. Clearly this results in far more lines than are actually necessary and each line is not "as long as possible".

This problem could possibly be solved by pre-processing to determine that the configuration of adjacent rectangles is of this form but more complicated configurations of rectangles could display similar behaviour. A heuristic should not be as susceptible to the actual configurations of input rectangles as this one is. The heuristic in Section 5.5.5 goes some way to addressing the problems which could arise with the heuristic discussed here.

5.5.5 Crossing one adjacency at a time

Another idea for a heuristic to find an approximation to ***ALP-ALOR*** is to consider crossing each as yet uncrossed adjacency in turn with a line which is "as long as possible". The detail of this idea is expanded upon below.

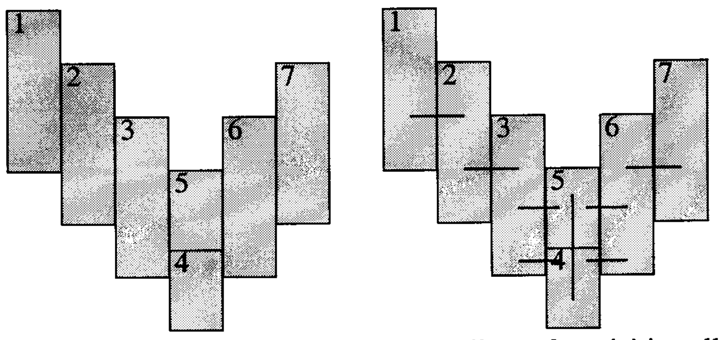
1
2
3
5
4
6
7
a) A configuration with two extreme rectangles
1
2
3
5
4
6
7
b) The lines after visiting all extreme rectangles

Figure 5.16: The longest chain heuristic: A second problem with the heuristic – all the adjacencies are uncrossed after all the extreme rectangles have been considered

1. Determine all of the adjacencies between the orthogonal rectangles. Store this information in an adjacency matrix.

2. Sort the adjacencies based on their minimum x-coordinate, break ties based on minimum y-coordinate.

3. As long as there are still some uncrossed adjacencies.

   (a) Pick the next adjacency.

   (b) Using the information stored in the adjacency matrix and starting with the two rectangles which define the adjacency selected in point (3a) successively add rectangles to this chain until adding another rectangle would cause the chain to "kink'.

   The sweep works from left to right – so any rectangle to be added to the chain would have to be added to the right side or the top or bottom of the last rectangle in the chain. Kinking then occurs if

   Case 1 if the rectangle which is being considered for adding to the chain is adjacent to the left side of the last rectangle in the chain

   Case 2 if the rectangle which is being considered for adding to the chain is adjacent to the bottom of the last rectangle in the chain and the last rectangle in the chain was added to the top of the penultimate rectangle in the chain

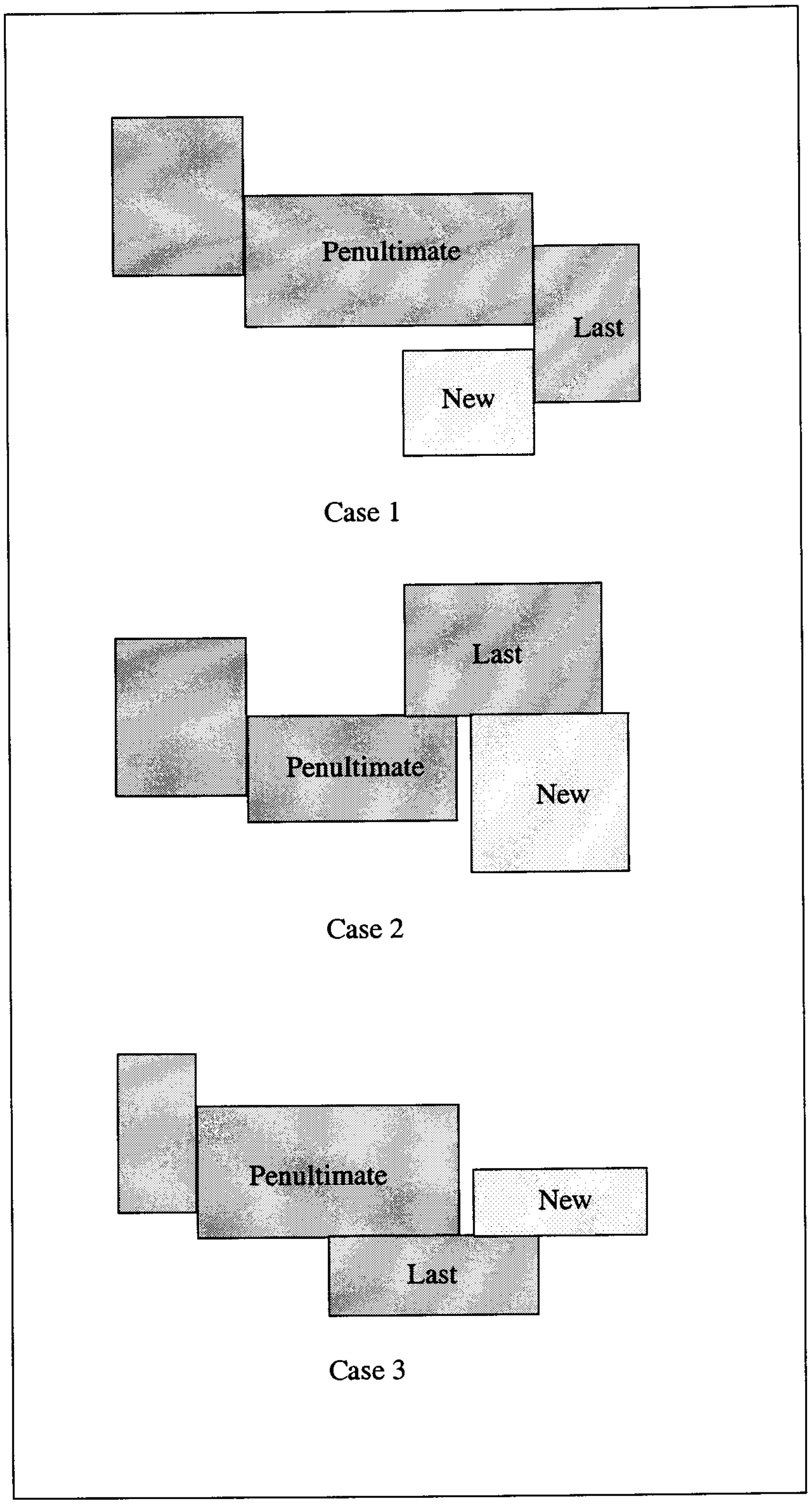

Figure 5.17: Crossing one adjacency at a time: Cases which cause “kinking ” in a chain of rectangles

Case 3 if the rectangle which is being considered for adding to the chain is adjacent to the top of the last rectangle in the chain and the last rectangle in the chain was added to the bottom of the penultimate rectangle in the chain

Figure 5.17 shows examples of these three cases.

(c) Determine if a line can be placed to cross the adjacencies in the chain. (This can be done using the idea of partial edge visibility discussed in section 5.4.)

If a line can be placed then add this line to the final set of lines.

If a line cannot be placed then repeatedly

i. generate a different chain starting with the same adjacency and following the same constraints

ii. determine whether a line can be placed to cross all of the adjacencies in that chain

until either a line is found or all the possible chains have been considered.

If a line cannot be placed through any chain starting at that adjacency and generated as above then make a line which crosses just that adjacency.

(d) Mark off all the adjacencies crossed by the line generated above.

4. The resulting set of lines is an approximation to the solution for ***ALP-ALOR***

The application of this idea is illustrated by considering the configuration of rectangles in Figure 5.18. The first adjacency considered is 1|2. The chain thus starts as being 1–2. Then suppose that rectangle 3 is considered, this rectangle can be added to the chain which now becomes 1–2–3. This chain can be extended to 1–2–3–4–5. If an attempt is made to add rectangle 6 to the chain then kinking occurs. Rectangles 4 and 6 are on the same side of rectangle 5 and it would thus be impossible to put a straight line through the chain of rectangles 1–2–3–4–5–6. It might however be possible to place a line through the chain 1–2–3–4–5. This is, in fact, the case so the line 1–2–3–4–5 is created. This line crosses the adjacencies 1|2, 2|3, 3|4 and 4|5 – see Figure 5.19.a.

The next adjacency to be considered would be 1|3 – this is the leftmost as yet uncrossed adjacency. A number of possible chains (without kinking) start with this adjacency: 1–3–2, 1–3–5, 1–3–4–5 and 1–3–4–6. Suppose that 1–3–2 was the first one considered. A line can be placed through the adjacencies in this chain and so this line is added to the final set of lines – see Figure 5.19.b.

The next adjacency to be considered would then be 3|5 (1|2, 1|3, 2|3 and 3|4 have already been crossed) and the process would continue. A possible final set of

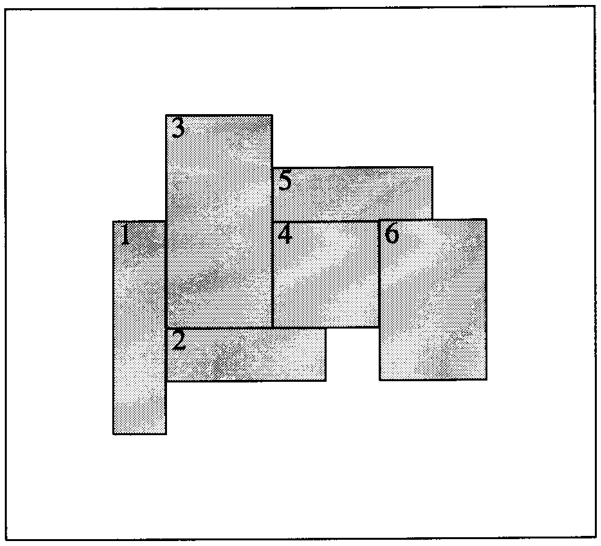

Figure 5.18: Crossing one adjacency at a time – example input

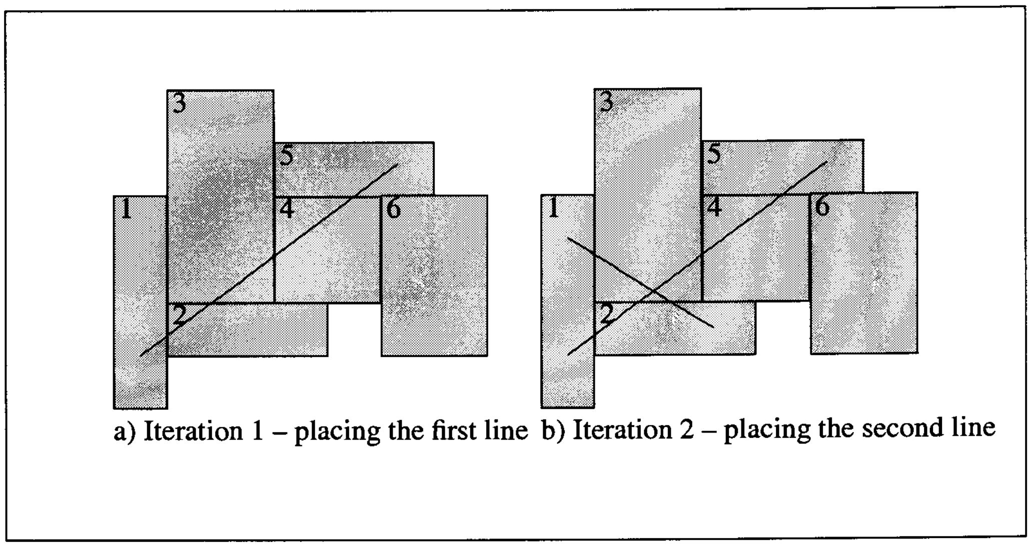
3
5
1
4
6
2
a) Iteration 1 – placing the first line
3
5
1
4
6
2
b) Iteration 2 – placing the second line

Figure 5.19: Crossing one adjacency at a time – the first two passes of the algorithm

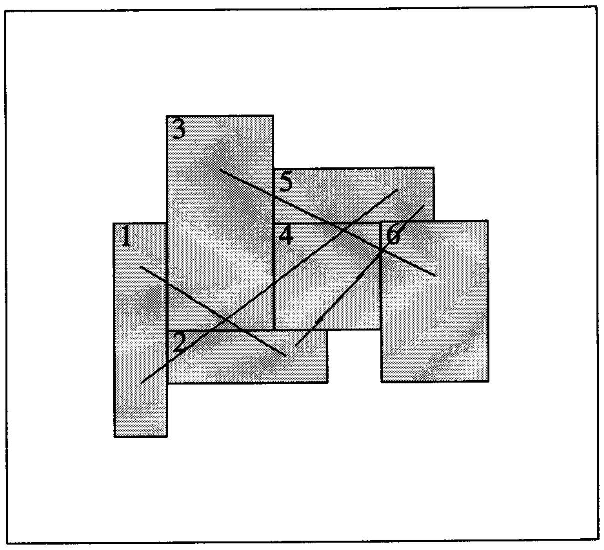

Figure 5.20: Crossing one adjacency at a time – a possible solution

lines is shown in Figure 5.20. This is in fact a minimal solution for this instance of the problem.

There are, however, some problems with this algorithm. Suppose that a different chain was selected when attempting to cross adjacency 1|2 and that after two iterations the lines 1–2–4–6 and 1–3–2 had been placed – see Figure 5.21.a. The next adjacency to be crossed would be 3|4. Three chains of rectangles are possible 3–4–2, 3–4–5 and 3–4–6. A line can be placed in each of these chains. Assume (for the sake of the discussion) that the second chain was chosen (the argument applies to the other two chains as well). Then the situation after placing this line (3–4–5) would be as shown in Figure 5.21.b. This last line is clearly not "as long as possible" – it should cross the adjacency 1|3 as well. A second pass through the final set of lines would be required to extend such lines backwards as far as possible.

Another problem which could occur with this heuristic is that it does not take into account "curves" in chains – see Figure 5.22.a. In a case like this where the adjacencies being considered is 1|2, the rectangles 1, 2, 3, 4, 5 and 6 form a chain (1–2–3–4–5–6) and although there is no point at which the chain wraps around (none of the cases identified above occur), no line can be placed through the whole chain. The heuristic would thus generate the line 1–2 and move on to consider the adjacency 2|3 where a similar situation would occur. The final set of lines (shown in Figure 5.22.b) would thus not be minimal.

Again it is possible that this problem could be resolved by a "post-processing" pass over the "final" set of lines to attempt to extend any lines which only cross a single adjacency as far as possible.

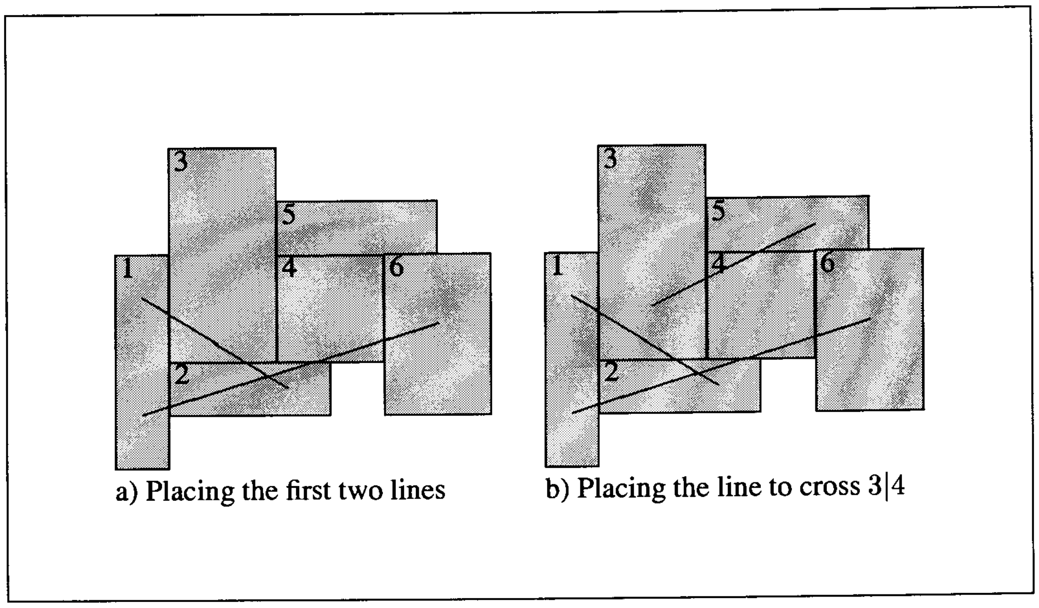
3
5
1
4
6
2
a) Placing the first two lines
3
5
1
4
6
2
b) Placing the line to cross 3|4

Figure 5.21: Crossing one adjacency at a time: A problem with the heuristic – lines that don't extend far enough to the left

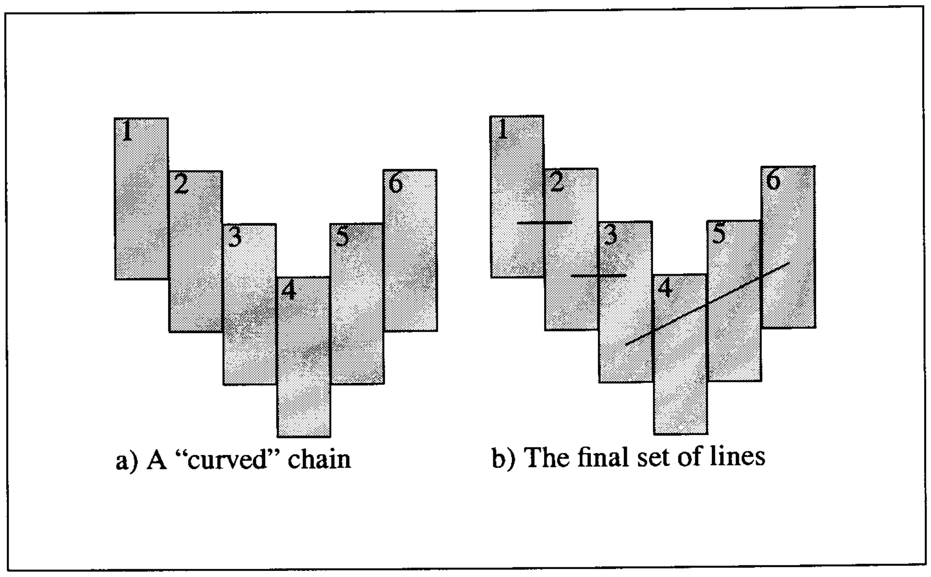

Figure 5.22: Crossing one adjacency at a time: A second problem with the heuristic – lines which only cross a single adjacency

5.5.6 Extending forwards and then backwards

This approach is similar to the heuristic suggested above and was developed by Kenny [2000] working under the supervision of the author of the thesis. Again for each adjacency that is still uncrossed a new chain working from left to right is constructed. When the chain kinks the forward phase is complete and it is necessary to determine whether a straight line can be placed in the chain. If a line can be placed then the next stage of the process – extending the chain to the left – follows. However, if a line cannot be placed then one rectangle at a time will be removed from the right hand side of the chain until a line can be placed or until there are only two rectangles in the chain. These two remaining rectangles are adjacent to each other and clearly a line can be placed to cross the adjacency between them.

This first stage ensures that the lines extend as far as possible to the right. However, it is possible that they don't extend as far to the left as possible. The second stage of the heuristic addresses this.

The chain is now extended backwards (to the left) as far as possible by successively adding rectangles to the chain until kinking would occur. Then a similar process of testing for line placement and removing rectangles from the chain is followed.

The resulting line is now also extends as far to the left as possible and so is "as long as possible". It is thus included in the final set of lines.

Parts a) to i) of Figure 5.23 show the application of this heuristic to a simple configuration of adjacent rectangles. The first uncrossed adjacency to be considered is adjacency A. The two rectangles comprising this adjacency, namely rectangles 1 and 2 form the start of the chain as shown in a) . Rectangle 3 is adjacent to 2 on the right and is added to the chain. Similarly, rectangle 4 is adjacent to rectangle 3 and is also added to the chain. Rectangle 5 is on the left of rectangle 4 and therefore is not considered for inclusion in the chain since it would result in kinking of the chain. The chain comprising rectangles 1, 2, 3, and 4 is now checked for visibility. The chain clearly does not allow visibility so rectangle 4 is removed from the chain. The new chain shown in d) consists of rectangles 1, 2 and 3. This chain is does allow visibility so the forward part of the algorithm is complete and an attempt is made to extend this possible line backwards. Rectangle 1 does not have any left or bottom adjacencies so the chain cannot be extended backwards. Thus a line is placed crossing adjacencies A and B. The next uncrossed adjacency is C. The first two rectangles of the chain are 3 and 4 as shown in e). The chain cannot be extended in the forward direction since this leads to kinking. However, we can extend the chain backwards into rectangles 2 and 1, as shown in f) and g). The chain in g) consisting of rectangles 1, 2, 3, and 4 does not allow visibility. Since the chain is now being extended backwards, rectangle 1 is removed from the left hand side of the chain. The resulting chain shown in h) does allow visibility and thus a

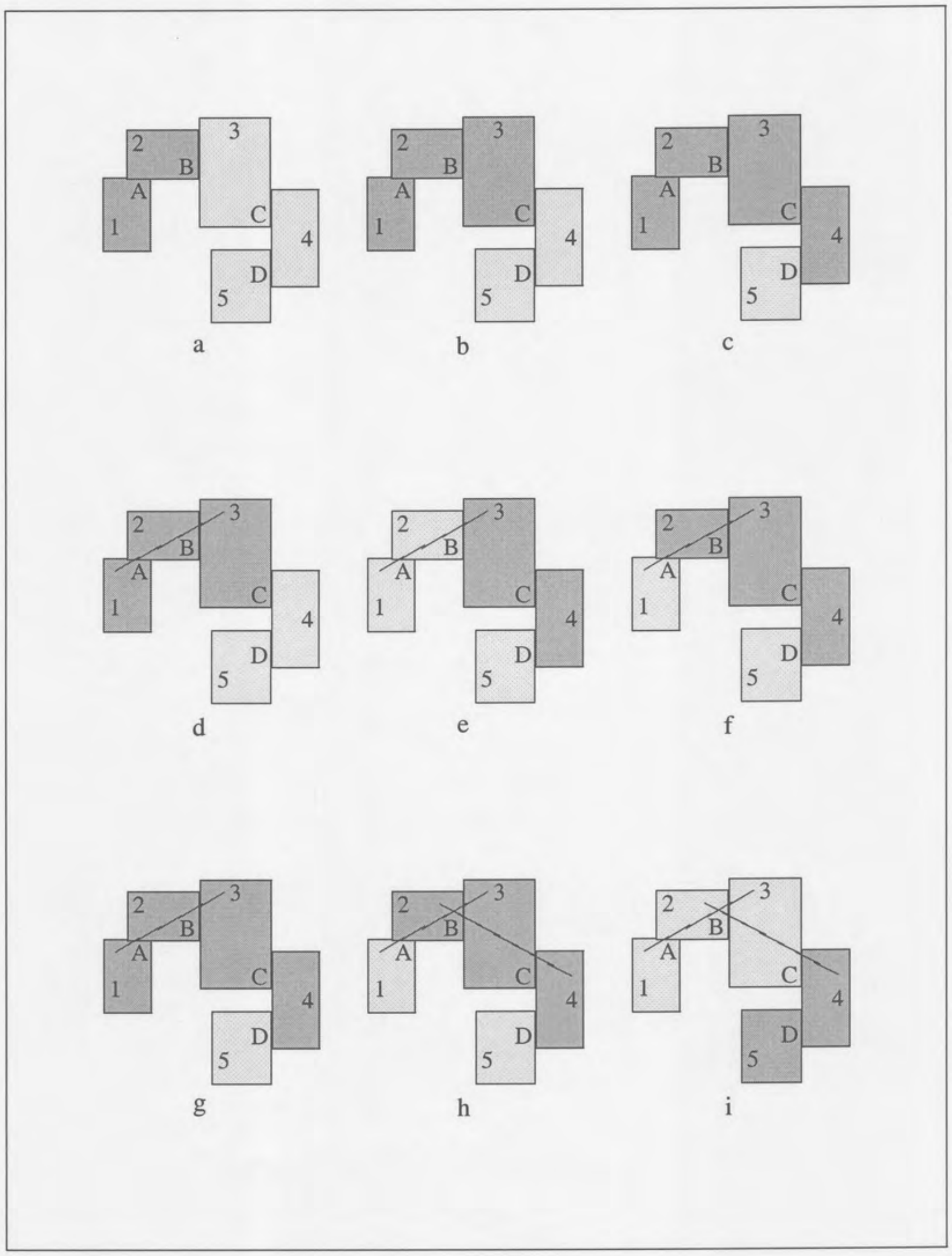

Figure 5.23: Extending forwards and then backwards – the different stages of determining chains, modifying the chains and placing axial lines.

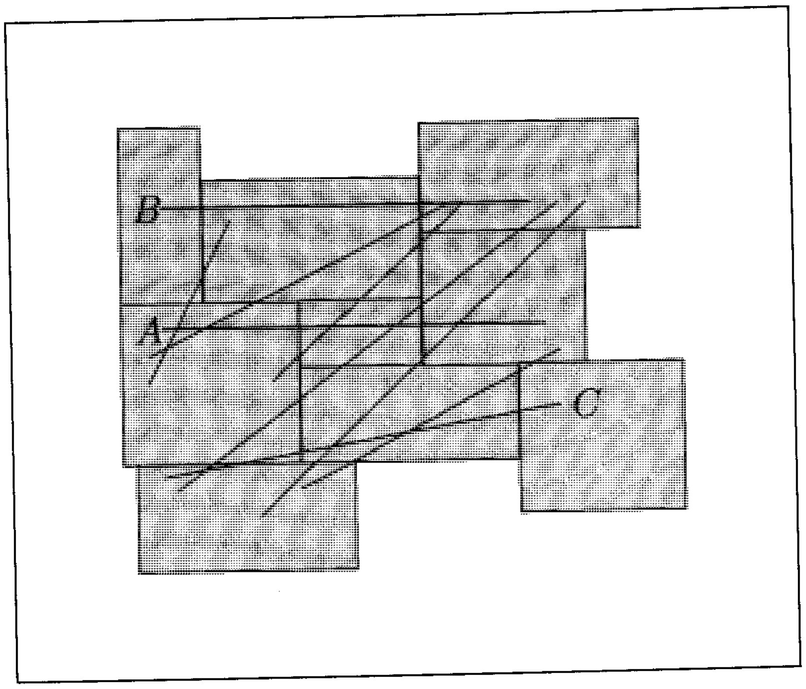

Figure 5.24: Extending forwards and then backwards – redundant lines can be generated.

line can be placed that crosses adjacencies B and C. The only remaining uncrossed adjacency is D. The start of the new chain comprises rectangles 4 and 5 as shown in i). The chain cannot be extended forwards (there are no more rectangles) and cannot be extended backwards into rectangle 3 since that would lead to kinking. Therefore a line can be placed to cross adjacency D. All of the adjacencies have now been crossed.

This heuristic has been implemented and tested on some configurations of adjacent rectangles. The heuristic seems to offer a reasonable approximation to the exact result in the cases tested. There are, however, some cases where the heuristic produces redundant lines – lines which only cross adjacencies which are crossed by other lines generated later in the process (see Figure 5.24 where lines A, B and C are redundant). The removal of redundant lines could be achieved by a post processing phase – identifying "essential lines" (axial lines that cross some adjacencies that are not crossed by any other axial lines) and then removing the redundant lines.

Another problem with the implementation of this heuristic is that it is biased in favour of crossing vertical adjacencies (the first rectangle considered to extend a chain of rectangles is a right neighbour) and this can affect the accuracy of the approximation.

5.5.7 Summing Up

This section of the thesis has considered some heuristics which could be used to produce approximate solutions to ***ALP-ALOR***. Clearly more work in this area is required to clarify the ideas and find a good heuristic. This additional work is outside the scope of this thesis but some work is currently being undertaken (under the author's supervision) by an Honours student in the School of Computer Science at the University of the Witwatersrand.

5.6 Special cases which can be solved in polynomial time

ALP-OLOR is in general NP-Complete but it seems likely that there are some special cases of the general problem for which polynomial time algorithms can be obtained. In particular it seems likely that an exact solution for configurations of rectangles which form chains, as defined in Section 4.7.2.3, could be found in polynomial time. In this variation of the problem a more generalised form of chain where any rectangle in the chain can have at most two neighbours (adjacent on different sides) as shown in Figure 5.25 could also be considered. It seems likely that exact solutions for this type of configuration could also be found in polynomial time.

Whether exact polynomial-time solutions for more complicated configurations of rectangles can be found is still an open question.

5.7 Future Research

Section 5.3 of this thesis has shown that ***ALP-ALOR*** is NP-Complete. This points to two fruitful areas for future research – heuristic algorithms for approximate solutions and special cases where the exact solution can be found in polynomial time.

A previous section of this thesis (Section 5.5) addressed the first of these areas to some extent and suggested some heuristics which could be used to produce approximate solutions to ***ALP-ALOR*** but future work could be focussed on

- developing more, and hopefully better, heuristics to solve the problem,
- testing of the heuristics presented here and any new heuristics which are developed.
- looking at other approaches, for example genetic algorithms, for finding approximate solutions.

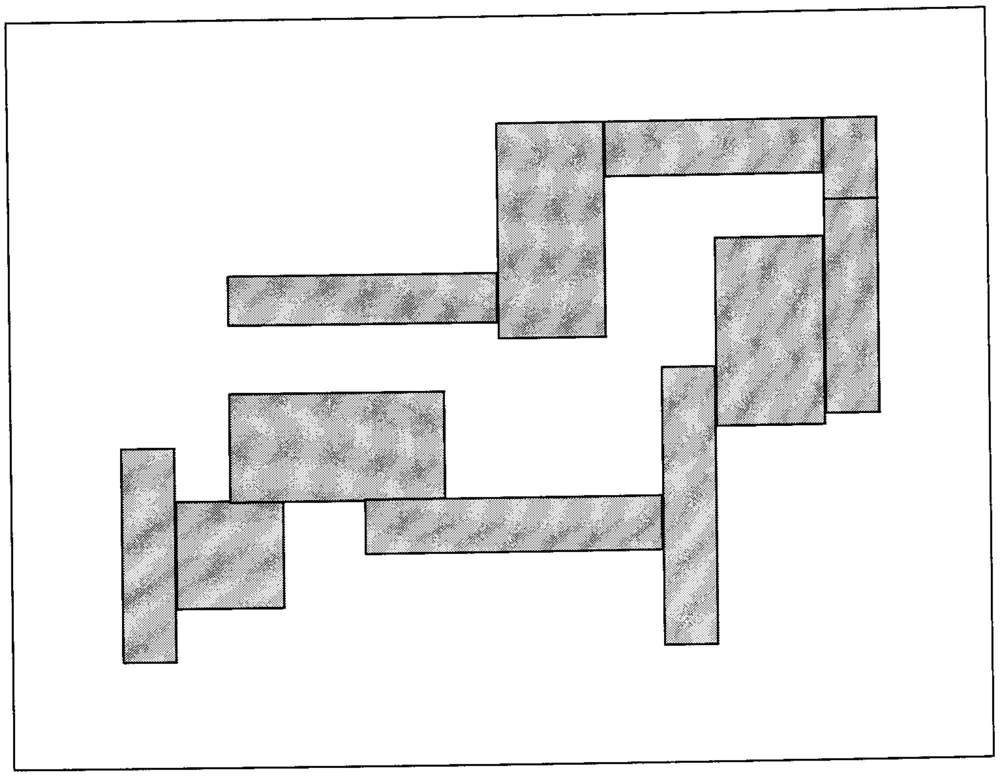

Figure 5.25: A more general chain of rectangles

The area of special cases which can be solved in polynomial time is very briefly addressed in this thesis. It is conjectured that simple chains of adjacent rectangles and even more complicated chains could be solved exactly in polynomial time. Attention should be focussed on proving these claims and should also be focussed on attempting to determine other, more complicated configurations of adjacent rectangles for which the problem can be solved exactly in polynomial time.

5.8 Conclusion

The axial line placement problem for axial lines with arbitrary orientation and orthogonal rectangles (***ALP-ALOR***) has been shown to be an NP-Complete problem as is ***ALP-OLOR***. Also in this chapter some heuristics to find approximations for ***ALP-ALOR*** have been suggested but these heuristics still have to be tested to determine how good the approximations are likely to be. The idea of special cases which can be solved in polynomial time has been briefly touched upon but this is essentially left as future work.

The NP-Completeness result presented in this chapter can be extended to a slightly more general problem – axial lines with arbitrary orientation crossing the adjacent edges between convex polygons (***ALP-ALCP***). This extension of the result is discussed in the next chapter (Chapter 6) of this thesis.

Chapter 6

Placing axial lines with arbitrary orientation to cross the adjacencies between convex polygons

6.1 Introduction

The problem which is being considered in this thesis is that of placing the axial lines through the convex spaces in an urban layout. Chapters 4 and 5 discuss simplifications of this problem. In both cases the convex spaces (convex polygons) are restricted to being rectangles and the problem is that of finding the minimum number of axial lines that cross all of the adjacencies between rectangles. In Chapter 4 the axial lines are restricted to being parallel to the Euclidean axes while in Chapter 5 this restriction no longer applies. This chapter discuss the generalisation of the problem where the aim is to find the minimum number of axial lines to cross all of the shared boundaries between adjacent convex polygons. Both of the simplifications, ***ALP-OLOR*** and ***ALP-ALOR***, are NP-Complete. In this chapter ***ALP-ALCP*** (Axial Line Placement – Arbitrary Lines and Convex Polygons) is also shown to be NP-Complete.

In the remainder of this chapter (***ALP-ALCP***) is discussed in more detail. The problem is restated in detail in Section 6.2 below. In Section 6.3 ***ALP-ALCP*** is shown to be NP-Complete by demonstrating that ***ALP-ALOR*** is a restriction of ***ALP-ALCP***. Then, because ***ALP-ALCP*** is NP-Complete, some ideas for heuristics to find approximate axial maps are discussed in Section 6.4 and some comments are made about special cases which can be solved in polynomial time are made in Section 6.5. In Section 6.6 some ideas for future research are briefly discussed.

6.2 Statement of the Problem

Given a number of adjacent convex polygons find the fewest axial lines contained wholly inside the polygons which will cross all of the shared boundaries (adjacencies) between adjacent polygons. An additional requirement is that each axial line should cross as many of the shared boundaries as possible – a *maximal* axial line.

As in Chapters 4 and 5, depending on how the problem is considered there are 2 similar but distinct problems which can be addressed.

1. Adjacencies can be crossed more than once but every adjacency must be crossed *at least* once.

2. Any adjacency has *exactly* one line crossing it.

In this chapter and in this thesis only problem 1 is addressed. An example of the problem is shown in Figure 6.2.

6.3 Proving the Problem is NP-Complete

The problem can be formally stated as

ALP-ALCP
Instance: A collection of convex polygons $P_1 \ldots P_n$, where each polygon is adjacent to at least one other polygon, and a positive integer M.
Question: Is there a set L of axial lines where each axial line is maximal in length, each line is wholly contained in the collection of polygons, each shared boundary between adjacent polygons is crossed at least once and $\mid L \mid \leq M$?

Theorem 6.3.1 *ALP-ALCP is NP-Complete.*

Proof
Clearly ***ALP-ALCP*** is in NP. Given a set of axial lines with arbitrary orientation it is possible to check in polynomial time that each adjacency has been crossed by at least one axial line.

The problem ***ALP-ALOR*** which was proved to be NP-Complete in Chapter 5 is a restricted instance of ***ALP-ALCP***. It is a special case of ***ALP-ALCP*** where the convex polygons are restricted to being orthogonal rectangles. Thus using the approach of proof by restriction (see Section 2.3) Theorem 6.3.1 has been proved.

□

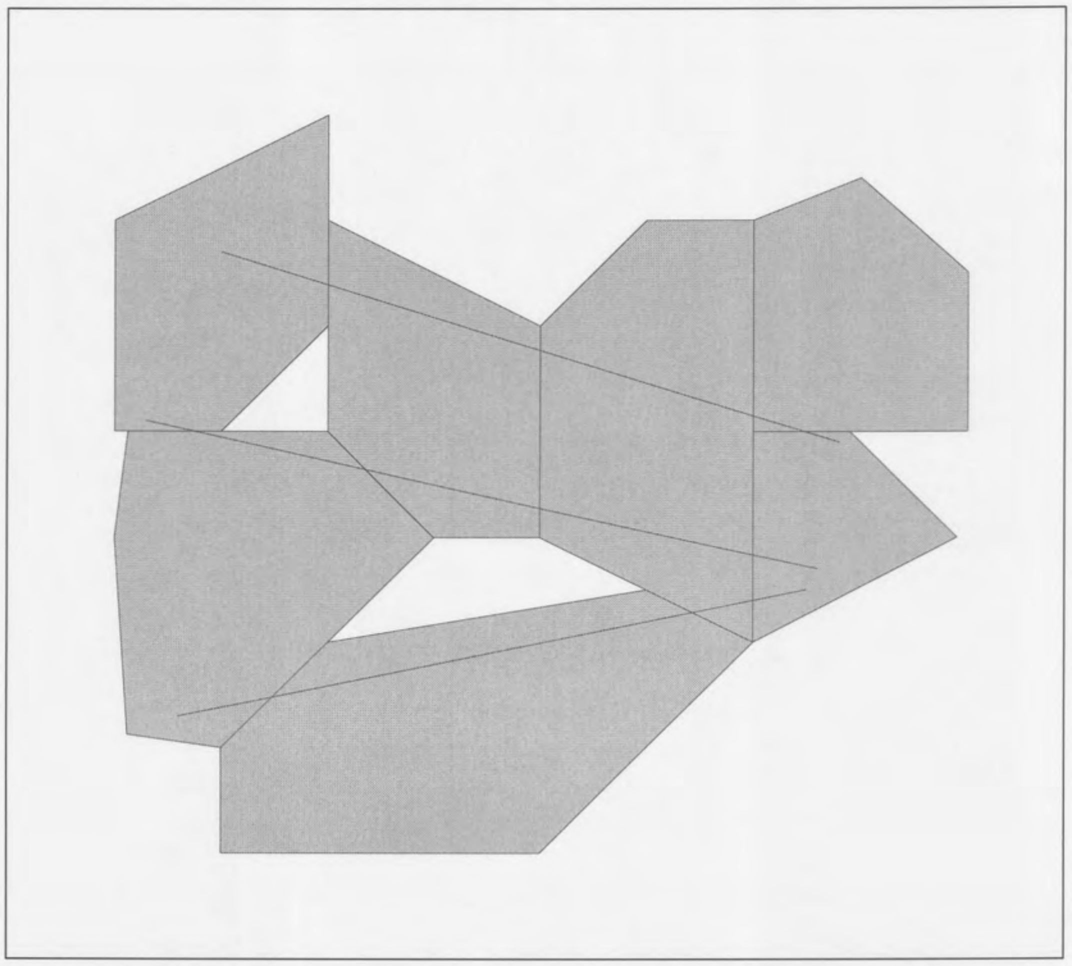

Figure 6.1: An example of placing axial lines to cross all of the adjacencies in a configuration of adjacent convex polygons

6.4 Heuristics to find approximate solutions for *ALP-ALCP*

ALP-ALCP has been shown to be NP-Complete (Section 6.3) and thus in the general case finding a minimal solution could take an unreasonable amount of time. It is thus necessary to consider heuristics to find approximate solutions in reasonable time. Developing the heuristics is beyond the scope of this thesis and thus will be future work which arises from this thesis. However some of the heuristics which are suggested in Section 5.5 could possibly be modified to produce approximate solutions to ***ALP-ALOR***.

6.5 Special cases of *ALP-ALCP* which can be solved in polynomial time

ALP-OLOR is in general NP-Complete but it seems likely that there are some special cases of the general problem for which polynomial time algorithms can be obtained. In particular it seems possible that an exact solution for configurations of convex polygons which form chains (analogous to the more general chains presented in Section 5.6) could be found in polynomial time. See Figure 6.2 for an example of such a chain.

Whether exact polynomial-time solutions for more complicated configurations of convex polygons can be found is still an open question and is outside the scope of this thesis.

6.6 Future Work

Section 6.3 of this thesis shows that ***ALP-ALCP*** is NP-Complete. This points to two fruitful areas for future research – heuristic algorithms for approximate solutions and special cases where the exact solution can be found in polynomial time. Neither of these areas has been addressed in this thesis but both certainly warrant attention.

6.7 Finding the adjacencies in a configuration of adjacent convex polygons

In order to develop heuristics to find reasonable approximations to the exact solution for ***ALP-ALCP*** or to consider special cases for ***ALP-ALCP***, it is necessary to be able to efficiently find the adjacencies in a configuration of adjacent convex polygons.

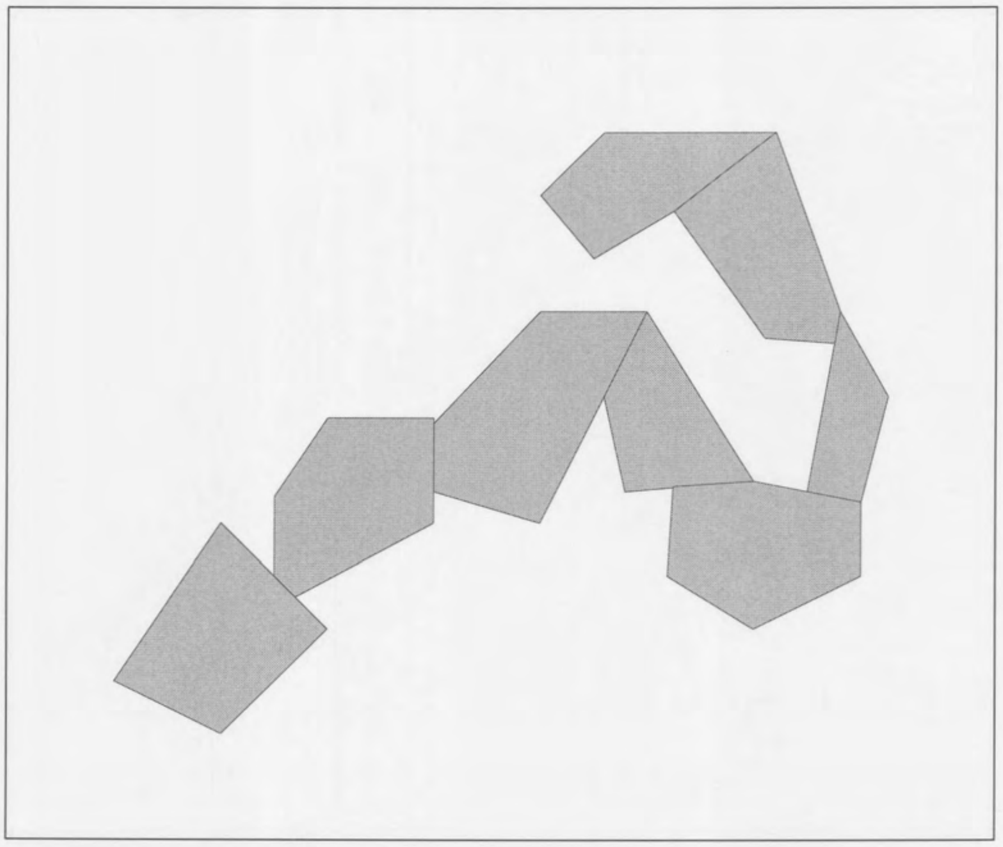

Figure 6.2: A chain of convex polygons

Recently Adler *et al.* [2001] produced a solution to this problem[1].

6.8 Conclusion

In this chapter of the thesis the axial line placement problem for axial lines with arbitrary orientation and convex polygons (***ALP-ALCP***) has been shown to be an NP-Complete problem as are ***ALP-OLOR*** and ***ALP-ALOR***. The result in this chapter (***ALP-ALCP***) follows because ***ALP-ALOR*** is a restriction of ***ALP-ALCP*** – orthogonal rectangles are a subclass of convex polygons. This result raises that question of whether or not the original axial line placement problem – placing the minimum number of axial lines to cross the adjacencies in a convex map of a town or urban layout – is also NP-Complete. In the next chapter of the thesis this question is addressed.

[1]The author of this thesis set the problem as an assignment for the 2001 University of the Witwatersrand Computer Science Honours class in their Analysis of Algorithms course. The resulting assignment solutions were then combined to produce a paper which was submitted to the SAICSIT 2001 Research Symposium and accepted as an "electronic publication"

Chapter 7

Placing Axial Lines in Town Plans

7.1 Introduction

The original problem from the town planning domain which is considered in this thesis is to find the axial map for an urban area given the convex map of area. A convex map is composed of the minimum number of convex spaces (convex polygons) which partition the urban area (represented as a polygon with holes). In this thesis only partitions which do not make use of Steiner points (see Section 2.2) are allowed. An axial map consists of the minimum number of "axial lines" (straight line segments) necessary to cross the adjacencies between the convex polygons which make up the convex map. In this thesis only the problem where adjacencies can be crossed more than once but *must* be crossed at least once is considered. An additional requirement is that the axial lines should cross as many adjacencies as possible. These constraints are chosen to reflect the constraints in the real world problem as closely as possible.

As was seen in Chapter 2 the first of these problems – covering or partitioning a general polygon by the smallest number of convex polygons (or other types of polygons) – has been quite well studied in the general case. Many of these covering and partitioning problems have been shown to be NP-Complete or NP-Hard. In particular O'Rourke and Supowit [1983] have shown that the problem of partitioning a general polygon with holes into convex polygons is NP-Hard. This thesis concentrates on the second of these problems – crossing the adjacencies between convex spaces by the minimum number of axial lines. In Chapter 4 the problem of crossing the adjacencies between orthogonal rectangles by orthogonal lines is shown to be NP-Complete. Likewise in Chapter 5 the problem of crossing the adjacencies between orthogonal rectangles by lines which are not necessarily orthogonal is shown to be NP-Complete. The proof in Chapter 5 can be extended to show that the problem of crossing the adjacencies between general convex polygons by lines with arbitrary orientation is NP-Complete (Chapter 6).

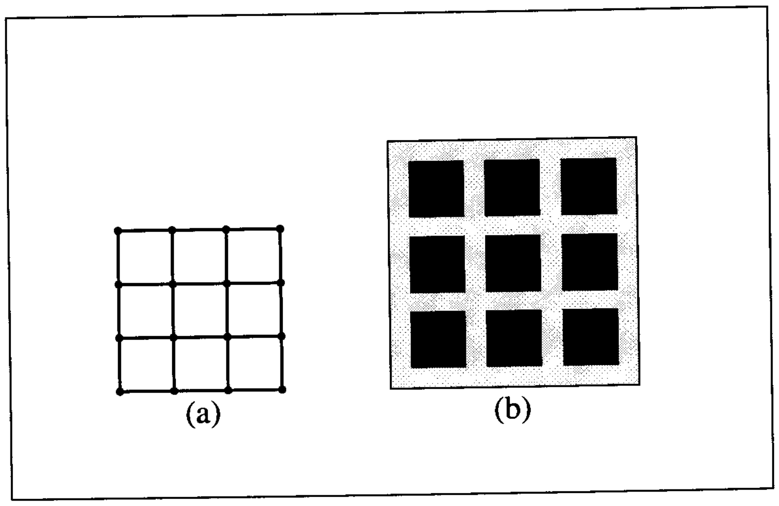

Figure 7.1: (a) A complete grid of size 4, (b) The corresponding complete urban grid of size 4

In this chapter the emphasis is on finding sets of lines to cross the adjacencies between convex polygons in a situation which is representative of the real world problem, i.e. covering the convex map of some town with an axial map. The chapter begins by considering some special cases where it is possible to prove that the minimal axial line cover can be found in polynomial time (Section 7.2). The problem in somewhat less restricted cases is then considered (Section 7.3) and finally the more general cases are briefly considered.

7.2 Urban Grids

Gewali and Ntafos [1993] define the *complete two-dimensional grid* of size n as the graph with vertex set $V = \{1, 2, \ldots, n\} \times \{1, 2, \ldots, n\}$ and the edge set $E = \{\{(i,j),(k,m)\} : |i-k| + |j-m| = 1\}$ where all edges are parallel to the major axes. In a geometric setting, the grid edges can be thought of as corridors and the grid vertices as intersections of corridors. A (partial) grid is any subgraph of the complete grid.

In this thesis the grid definitions of Gewali and Ntafos [1993] are extended to define the concept of *urban grids*.

Definition 7.2.1 *A* complete urban grid *of size n is a complete two-dimensional grid of size n where each grid edge or corridor $E_p, 1 \leq p \leq 2n(n-1)$ is an orthogonal rectangle with length l and width w, $l > w > 0$, and each grid vertex or intersection, $I_{i,j}, i, j = 1, 2, \ldots n$, is an orthogonal square of size $w \times w$. Each end of any corridor rectangle must be a side of an intersection.*

Figure 7.1 shows in (a) an example of a complete grid of size 4 and in (b) a complete urban grid of size 4.

Definition 7.2.2 *An urban grid is any subgraph of the complete urban grid.*

Gewali and Ntafos [1993] also define a *grid segment* as a succession of grid edges along a straight line bounded at either end by a missing edge. A *simple grid* is a grid where all of the endpoints of the grid segments lie on the outer face of the planar subdivision formed by the grid. A *general grid* is a grid which can have holes – some of the endpoints of the segments may lie on the inner face of the planar subdivision.

Definition 7.2.3 *A thoroughfare is a grid segment in an urban grid.*

Definition 7.2.4 *A simple urban grid is an urban grid where all of the endpoints of the thoroughfares lie on the outer face of the planar subdivision formed by the grid.*

Definition 7.2.5 *A general urban grid is an urban grid which can have holes – some of the endpoints of the thoroughfares may lie on the inner face of the planar subdivision.*

In order to place the axial lines in a complete urban grid it is necessary first to be able to find the convex map of the urban grid, i.e. to be able to partition the complete urban grid into the smallest number of convex polygons. The complete urban grid of size n has n^2 intersections and $2n(n-1)$ corridors. The number of corridors is easily seen by observing that each of the n horizontal thoroughfares have $n-1$ corridors, giving a total of $n(n-1)$ corridors lying horizontally. Similarly, there are $n(n-1)$ corridors lying vertically. The convex map must have fewer convex polygons than the sum of the corridors and intersections as the convex polygons representing these can be merged into a smaller number of larger polygons.

The minimum partition for the corridors and intersections of corridors which make up the four outer thoroughfares will be comprised of 4 convex polygons. Any attempt to partition the urban grid by combining the partitioning of the inner and outer regions would lead to more than 4 convex polygons in the outer region without resolving the partitioning of the inner region. There are a number of partitions of size 4 – see Figure 7.2 and Figure 7.3 for examples of the partitioning of the outer region of a complete urban grid of size 4. In Figure 7.2 the partition is accomplished by drawing a diagonal across each corner of the grid and closing off each corridor leading into the interior of the grid. In Figure 7.3 the corner points of the corner intersection are used to define the shared edges of the partition.

Partitioning the inner region will involve the placement of appropriately sized rectangles. This is essentially accomplished by determining where the adjacencies

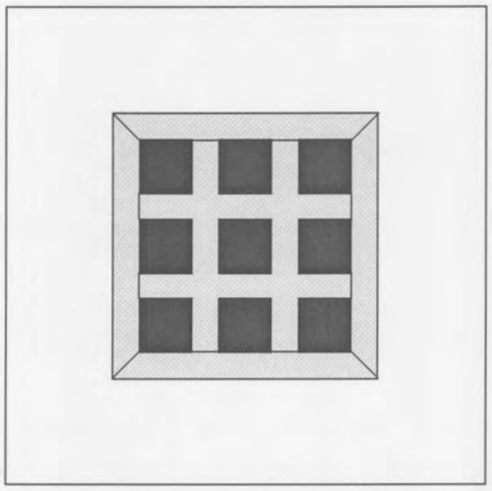

Figure 7.2: An example of partitioning the outer thoroughfares of a complete urban grid of size 4

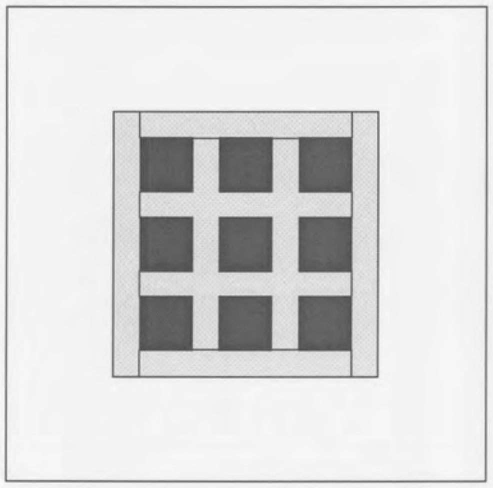

Figure 7.3: Another partitioning of the outer thoroughfares of a complete urban grid of size 4 – other similar partitionings exist

between rectangles can occur. These adjacencies are essentially edges connecting vertices on the boundary of the inner region. The geometry of the problem means that the best place to put these edges will be at the intersections of corridors. There are 4 cases which can occur – see Figure 7.4. Case 1 involves the fewest convex polygons so it would make sense to use as many of these types of partitions as possible. Clearly it is possible to place Case 1 adjacencies at each intersection. If this is done so that these adjacencies are all parallel to the x-axis then the interior region is partitioned into $n-2$ long rectangles (1 for each horizontal thoroughfare in the interior region) and $(n-2)(n-1)$ short rectangles ($n-1$ of them for each vertical thoroughfare in the interior region). The total number of convex polygons required is thus $4+(n-2)+(n-2)(n-1)$. The same number of convex polygons would be required if the long rectangles were aligned with the y-axis. Figure 7.5 shows an example of a partition using Case 1 adjacencies for a complete urban grid of size 4. In the case of an urban grid of size 4 the minimum number of convex polygons required is $4+2+(2)(3)=12$.

The question remains as to whether or not this is the minimum number of convex polygons which can partition a complete urban grid. Theorem 7.2.1 below addresses this question.

Theorem 7.2.1 *A minimum of* $4+(n-2)+(n-2)(n-1)$ *convex polygons are required to partition any complete urban grid of size* $n \geq 2$

Proof

As has been shown above the outer region cannot be partitioned by fewer than 4 convex polygons. Thus it is necessary to show that no partition of the inner region can have fewer than $(n-2)+(n-2)(n-1)$ convex polygons.

Any solution can only be achieved by merging two or more of the short rectangles representing the corridors, plus the squares representing the intersections, in the same thoroughfare (merging across thoroughfares will not result in convex polygons). There are $2(n-2)(n-1)$ of these short rectangles and $(n-2)^2$ intersections in the inner region. The intersections must be merged with the short rectangles to form longer rectangles or convex polygons. As argued above, the best way to do this is to use Case 1 (from Figure 7.4) – using any of the other forms of merging will result in more convex polygons in the final partition. Case 1 merging involves 3 rectangles, Case 2 involves 4 rectangles and Cases 3 and 4 involve 4 convex polygons.

In Case 1, merging each intersection with the corridors around it forces the closing of the ends of two rectangles – these two rectangles cannot now be further merged. This result applies whether horizontal or vertical adjacencies are placed to close off the rectangles. Each intersection must be addressed and so $2(n-2)^2$ rectangle ends are closed when merging the intersections with the corridors around them. The partitioning of the outer region forces the closing of the ends of $4(n-2)$

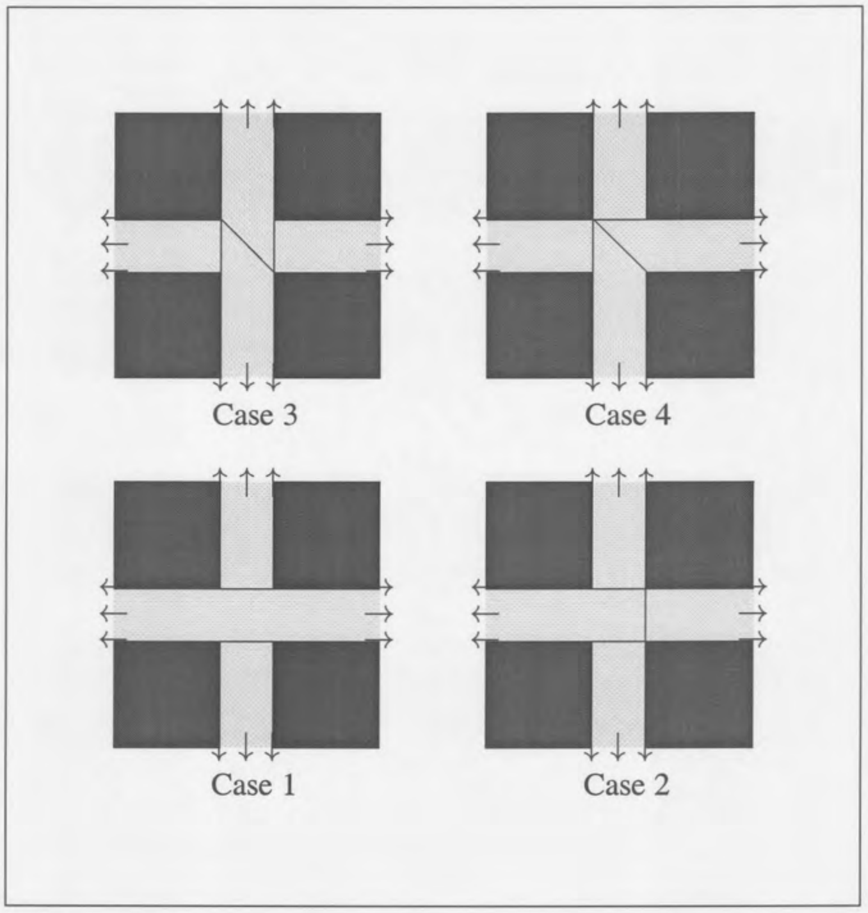

Figure 7.4: Possible adjacencies which could occur in a corridor intersection. Case 1 – Through the intersection (3 convex polygons involved) Case 2 – Rectangular ending at the intersection (4 convex polygons involved) Case 3 – Diagonal ending at the intersection (4 convex polygons involved) Case 4 – L-shaped diagonal ending at the intersection (4 convex polygons involved)

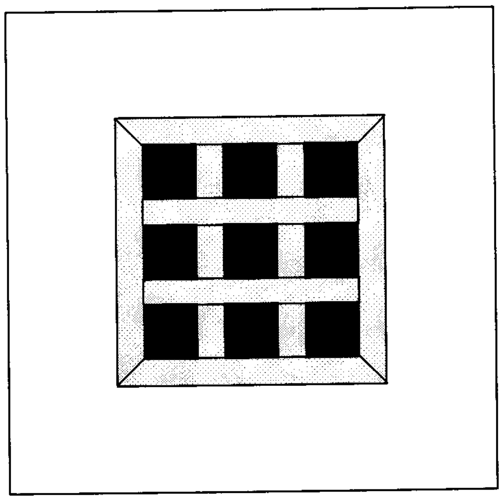

Figure 7.5: A complete partitioning of a complete urban grid of size 4

rectangles – each of the short rectangles which are adjacent to the outer region. Thus in total $4(n-2)+2(n-2)^2$ rectangle ends are closed. This means there must be $(4(n-2)+2(n-2)^2)/2$ or $(n-2)+(n-2)(n-1)$ rectangles or convex polygons in any partition of the inner region $((4(n-2)+2(n-2)^2)/2 = 2(n-2)+(n-2)^2 = 2n-4+n^2-4n+4 = (n-2)+(n^2-3n+2) = (n-2)+(n-2)(n-1))$. The result follows.

□

The next step in the process would be to determine the axial map for the complete urban grid. This is considered in Theorem 7.2.2 below.

Theorem 7.2.2 *The minimal set of axial lines to cross the adjacencies in a minimally partitioned complete urban grid can be found in polynomial time.*

Proof
A minimum partition (convex map) of a complete urban grid consists of $4+(n-2)+(n-2)(n-1)$ convex polygons. Irrespective of the exact form of the convex polygons which make up the partition the form of adjacencies which can occur in any partition are limited. Adjacencies can be

1. diagonals across the intersections in the outer region
2. the edges of the corner intersections in the outer region
3. the shared edge between two rectangles in any thoroughfare (see case 1 in Figure 7.4)
4. at the ends of rectangles between the inner and outer regions

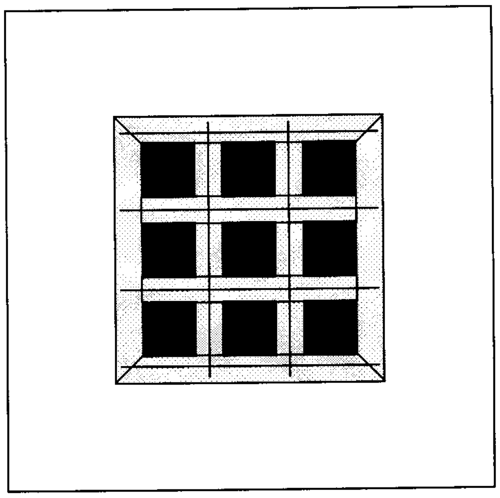

Figure 7.6: The axial map for a complete urban grid of size 4

All of these adjacencies can be crossed as below.

- By placing exactly 2 axial lines to cross the adjacencies in the outer region. The axial lines do not have to be orthogonal and there are only 4 adjacencies to be crossed in any minimally partitioned outer region. The axial map shown in the size 4 version of the problem in Figure 7.6 is one example which shows that only two axial lines are required to cross all of the adjacencies in the outer region. The axial lines in this case could be orthogonal. Figure 7.7 shows an example where axial lines which are non-orthogonal are required. The adjacencies in all other partitions of the outer region can be crossed with similar arrangements of axial lines.

- By placing an axial line in each thoroughfare. These axial lines must cross the adjacencies between the inner and outer regions and all the other adjacencies (if there are any) in the thoroughfare. Note that these axial lines do not necessarily have to be orthogonal.

Clearly this set of axial lines is the minimum to cross the adjacencies in the convex map. Each line through an interior thoroughfare is required. Any such thoroughfare will have at least two adjacencies which must be crossed by axial lines – one at each end of the thoroughfare between the inner and outer regions. One axial line will cross both of these adjacencies and any other adjacencies which appear in that thoroughfare. However, that axial line cannot cross the adjacencies in any other thoroughfare or in the outer region.

Clearly determining these axial lines can be done in polynomial time. If the complete urban grid is of size n then the number of thoroughfares is $2n$ (this includes the thoroughfares in the outer region). There are $2(n-2)$ interior thoroughfares. Placing the axial lines in all of the interior thoroughfares can thus be done

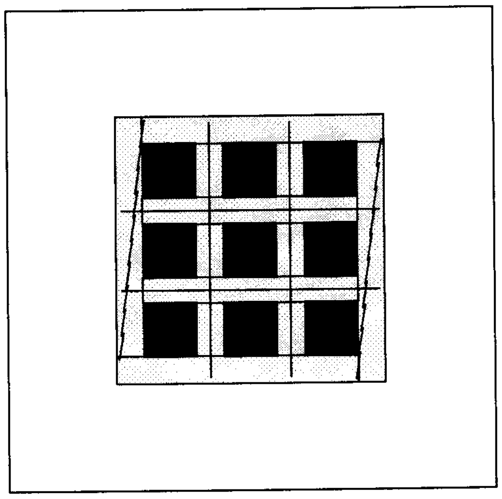

Figure 7.7: The axial map for a complete urban grid of size 4

in $\Theta(n)$ time – placing each line can be done in constant time. Determining which axial lines are necessary in the outer region can clearly be accomplished in $O(1)$ time – it is only necessary to determine what adjacencies are used at each corner intersection of the outer region.

□

The importance of Theorem 7.2.2 is that, irrespective of the convex polygons which make up the convex map of the complete urban grid, determining the axial map of the grid can be done in polynomial time. This holds even though determining the convex map is an NP-Hard problem in general. This theorem means that if the space in town plan can be modelled as a complete urban grid then the axial map for the town can be found in polynomial time. Clearly a central city area with a grid arrangement of streets could be modelled in this fashion. Typically in such a situation the ratio between the length and width of a corridor $l : w$ would be in the range $[2, 10]$. The results above would, however, still hold as all that was required by the definition of an urban grid was that $l > w$.

A further important result follows from Theorem 7.2.2.

Corollary 7.2.1 *The size of the axial map of a minimally partitioned complete urban grid of size n is equal to $2(n-2)+2$.*

In addition to this result a number of other results follow as well.

Corollary 7.2.2 *The minimal set of axial lines to cross all of the adjacencies in a minimally partitioned simple urban grid can be found in polynomial time.*

Proof

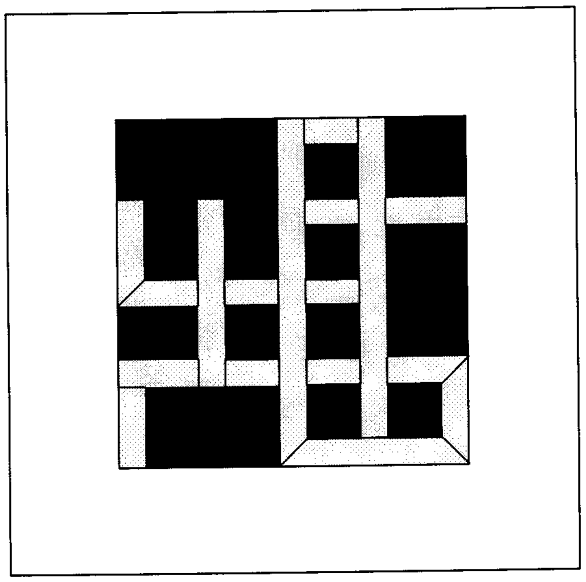

Figure 7.8: An example of minimally partitioning a simple urban grid

A minimally partitioned simple urban grid can be covered with convex polygons which are of similar form to those that cover the complete urban grid. This means that the adjacencies which have to be crossed will be orthogonal at all intersections except where only two corridors meet at right angles at an intersection – a *corner*. In this case the adjacency could be one of the interior edges of the intersection or a diagonal through the intersection. See Figure 7.8 for an example of a possible minimal partition of a simple urban grid.

The partitions which result from placing the different types of adjacencies at corners are equivalent in terms of the size of the partition. Using the diagonals rather than the orthogonal edges to construct the partitions is, however, preferable for the next phase of the process – placing the axial lines. There are two reasons for this. First, if orthogonal adjacencies are used then there are situations where more axial lines are required to cover all of the adjacencies in the partition than if diagonals are used. See Figure 7.9 where the partition using orthogonal adjacencies (a) requires two axial lines and the partition using diagonal adjacencies (b) only requires one axial line. A second, and subsidiary, reason is that the process of placing the axial lines is simplified if only diagonal adjacencies are used in the corners. In the remainder of this section diagonal adjacencies will be used at corners. Figure 7.10 shows a partition of Figure 7.8 under this constraint.

If any thoroughfare has orthogonal adjacencies in it then one axial line must be placed to cross all of the adjacencies in this thoroughfare. Such a thoroughfare may also have diagonal adjacencies in it. These will be crossed by the axial line placed to cross the orthogonal adjacencies.

If any thoroughfare only has diagonal adjacencies in it then, because of the

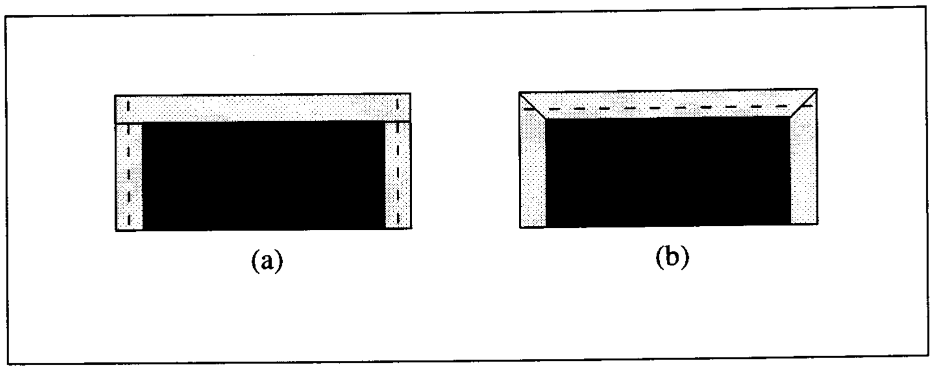

Figure 7.9:

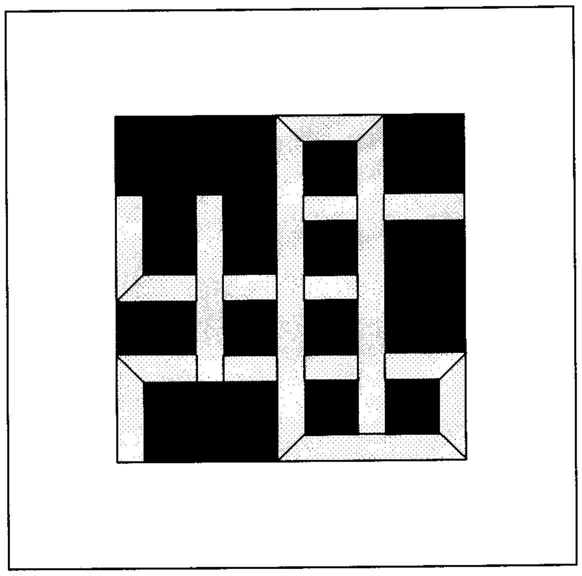

Figure 7.10: An example of minimally partitioning a simple urban grid using only diagonal adjacencies at corners

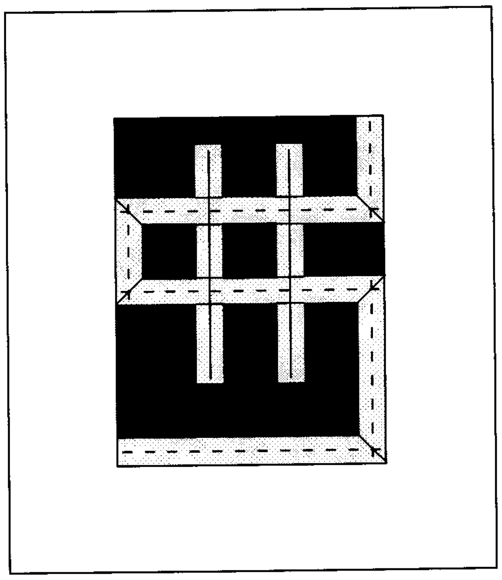

Figure 7.11: An example of limited choice in placing axial lines in a simple urban grid – only 3 of the dashed lines are necessary

geometry of the problem, it can have either one or two such adjacencies – more diagonal adjacencies would result in a non-minimal partition. If a number of thoroughfares with only diagonal adjacencies in them are connected then limited cases of choice can occur. See Figure 7.11 for an example of this. In these situations an axial line through each thoroughfare is sufficient to cross all of the adjacencies but fewer axial lines are actually necessary. In the example in Figure 7.11, 6 axial lines cross all of the diagonal adjacencies but only 3 lines are necessary. In this situation the lines through thoroughfares with only one diagonal adjacency will never be selected and so the problem becomes choosing 3 lines out of 4. Other similar cases could occur but they all have the common property that they would form a chain, or in some cases disjoint chains, of convex polygons. In these chains an axial line crosses the adjacency between a convex polygon and its predecessor in the chain (if it is not the first polygon in the chain) and the adjacency between the convex polygon and its successor (if it is not the last polygon).

It is also possible to get cycles and not just chains of thoroughfares with only diagonal adjacencies. See Figure 7.12 for an example of this situation. Here 4 choice axial lines cross the four diagonal adjacencies but only two are necessary.

Any algorithm to place the minimum number of axial lines in a simple grid would have to be able to determine which of the possible lines to include in the final set of lines.

If choice such as that described above can be resolved in polynomial time, then the problem of finding the axial map can be solved in polynomial time by the following process.

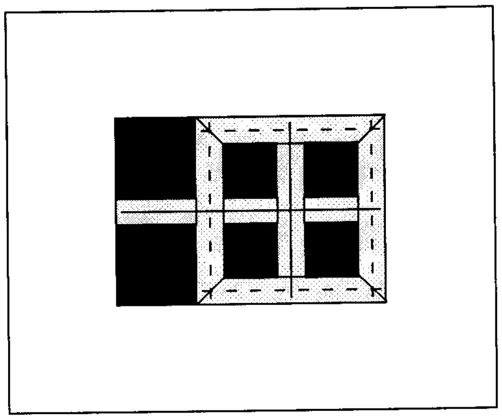

Figure 7.12: An example of the limited choice in placing axial lines in a simple urban grid resulting in a cycle of choice axial lines – only 2 of the dashed lines are necessary

1. Placing an axial line through each thoroughfare which has orthogonal adjacencies.
2. Placing an axial line through each thoroughfare which has two diagonal adjacencies in it.
3. Placing an axial line to cross any diagonal adjacency which has not been crossed in earlier phases of the process. In order to make such a line maximal it should be placed to cross as many adjacencies as possible. In this situation the only adjacencies it could cross would be adjacencies on the boundaries of the two thoroughfares which make up the corner. An axial line which crosses the diagonal adjacency cannot cross more than one such adjacency because of the geometry of the problem. See Figure 7.13 for an example of this situation.
4. Resolving the choice.

Clearly steps 1, 2 and 3 can be done in polynomial time – there are a polynomial number of thoroughfares to consider. To see that step 4 can also be done in polynomial time, note that each axial line in one of these chains or cycles crosses exactly two adjacencies. This means that it is simple to transform each chain or cycle of convex polygons which are adjacent at the diagonals of corners to an instance of ***edge cover*** by mapping adjacencies to vertices and axial lines to edges. ***Edge cover*** gives the minimum number of edges to cover all the vertices in a graph. This maps to the minimum number of axial lines to cross all of the adjacencies in any chain or cycle. ***Edge cover*** can be solved in polynomial time [Garey and Johnson, 1979]. The result follows.

□

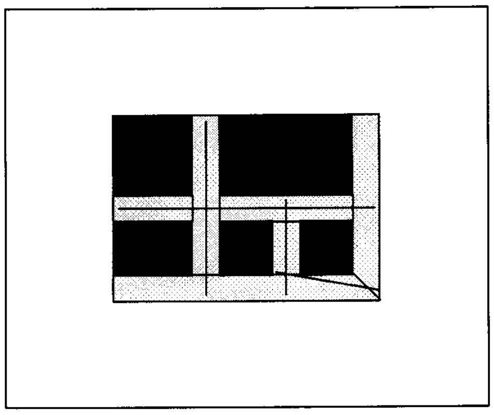

Figure 7.13: An example of placing an axial line to cross a single diagonal adjacency in two thoroughfares in a simple urban grid

Corollary 7.2.3 *The minimal set of axial lines to cross all of the adjacencies in a minimally partitioned general urban grid can be found in polynomial time.*

Proof

The proof in this instance follows from the proof of Corollary 7.2.2. Figure 7.14 shows an example of this version of the problem including an instance of choice. The method of resolving the choice, as discussed above, applies in this case as well. □

From the proofs of corollaries 7.2.2 and 7.2.3 it follows that in both simple and general urban grids the number of axial lines in the axial map is less than or equal to the number of thoroughfares in the grid. The results of these corollaries also mean that if the space in town plan can be modelled as a simple or general urban grid then the axial map for the town can be found in polynomial time. Clearly some towns with grid-like arrangements of streets could be modelled in this fashion.

7.3 Deformed Urban Grids

The results in the previous section show that axial maps can be found in polynomial time for town layouts which can be modelled by urban grids. Unfortunately all town plans cannot be modelled by urban grids – many town plans are not that regular. More general polygons would be needed to model town plans which are not regular grids. These more general polygons could make the problems of finding the convex map and the axial map more difficult. In this section slightly less restricted layouts than in Section 7.2 are considered. The aim here is attempt to determine whether relaxing the restrictions in this manner means that the problems of finding

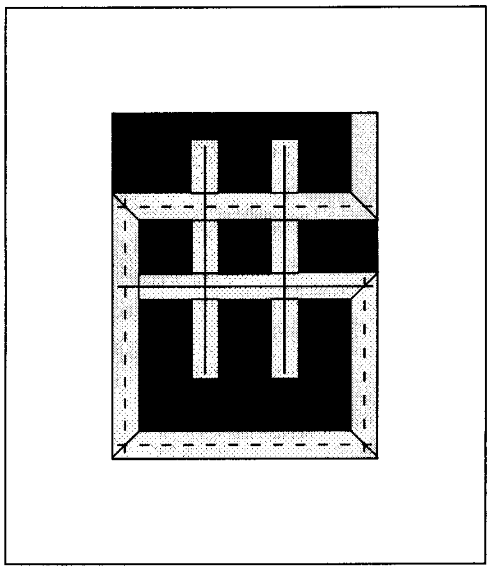

Figure 7.14: An example of choice in placing axial lines in a general urban grid – only 3 of the dashed lines are necessary

the convex and axial maps are still solvable in polynomial time or whether they are more like the general cases – NP-Hard or NP-Complete.

Definition 7.3.1 *A* complete deformed urban grid *of size n is a generalisation of a complete urban grid of size n with the following properties.*

- *The intersections $I_{i,j}, i, j = 1, 2, \dots n$ and corridors $E_p, 1 \leq p \leq 2n(n-1)$ in the grid may be convex quadrilaterals rather than orthogonal squares and rectangles respectively.*
- *Each corridor E_p must be adjacent to exactly two intersections on opposing sides.*
- *Where a corridor E_t is adjacent to an intersection $I_{q,r}$, they must share an edge.*

A deformed urban grid is any subset of a complete deformed urban grid.

In this definition the deformation could very small and the resulting deformed urban grid would be quite regular – similar to an urban grid. The deformation could also be quite large in which case the deformed urban grid could be something more like a general configuration of adjacent rectangles. In the first instance finding the convex map and from that finding the axial map would be similar to solving the corresponding urban grid problems in Section 7.2. In the other extreme the problems could become that of partitioning a general polygon with holes which has been shown to be NP-Hard [Lingas, 1982; O'Rourke and Supowit, 1983] and then solving ***ALP-ALCP*** which is shown to be NP-Complete in Chapter 6. In the remainder of this section deformed urban grids where the deformation is not very great are considered. The interest here is in considering deformed urban grids which could model real world urban layouts which are often grid-like but have to take into account the geographical environment. Figure 1.1 in Chapter 1 gives an idea of the type of layout which it would be useful to model using a deformed urban grid. Another example of a deformed urban grid (which could be an urban layout) is shown in Figure 7.15.

In the deformed urban grids which will be used to model these "grid-like" urban layouts some additional restrictions will be added to the definition of deformed urban grids given above. First, the length of each side, $d_{i,j_a}, a = 1, 2, 3, 4$, of an intersection quadrilateral, $I_{i,j}$, is such that $0 < d_{i,j_a} < 2w$. Second, each corridor E_p must have two opposing sides which are long compared to the other two sides. The ratio of the length of the shortest side to the longest side in any corridor should be approximately $1 : 2$. As stated previously, each of the two short sides of any corridor will also be a side of some intersection.

In the remainder of the thesis any mention of a deformed urban grid should be taken to mean a "grid-like" deformed urban grid. That is, a deformed urban grid with the additional length constraints.

Conjecture 7.3.1 *The problem of finding the minimal set of convex polygons to partition a deformed urban grid is NP-Hard.*

This conjecture is based on the results of Lingas [1982] and O'Rourke and Supowit [1983] that partitioning a polygon with holes is NP-Hard. A deformed urban grid seems to offer enough choice to make the problem difficult to solve exactly. Figure 7.16 shows a possible partition of the deformed urban grid of Figure 7.15. This partition was created by merging corridors and intersections to form larger convex polygons. If the merging allowed parts of intersections to be merged with parts of corridors then there would be more scope for choice and the problem would be harder to solve.

Clearly future work should be focussed on proving or disproving Conjecture 7.3.1.

The algorithm in Figures 7.17 and 7.18 calculates a partition of a deformed urban grid in polynomial time. The algorithm could return a minimal partition but is not guaranteed to do so – this depends on the shapes of the intersections and corridors. If Conjecture 7.3.1 is true then future research can be focussed on determining configurations of intersections and corridors which will allow this algorithm to return minimal solutions. Future research in improving this algorithm or developing a better algorithm is also worthwhile. If Conjecture 7.3.1 is not true then an algorithm which is guaranteed to find a minimal solution in all cases should be sought.

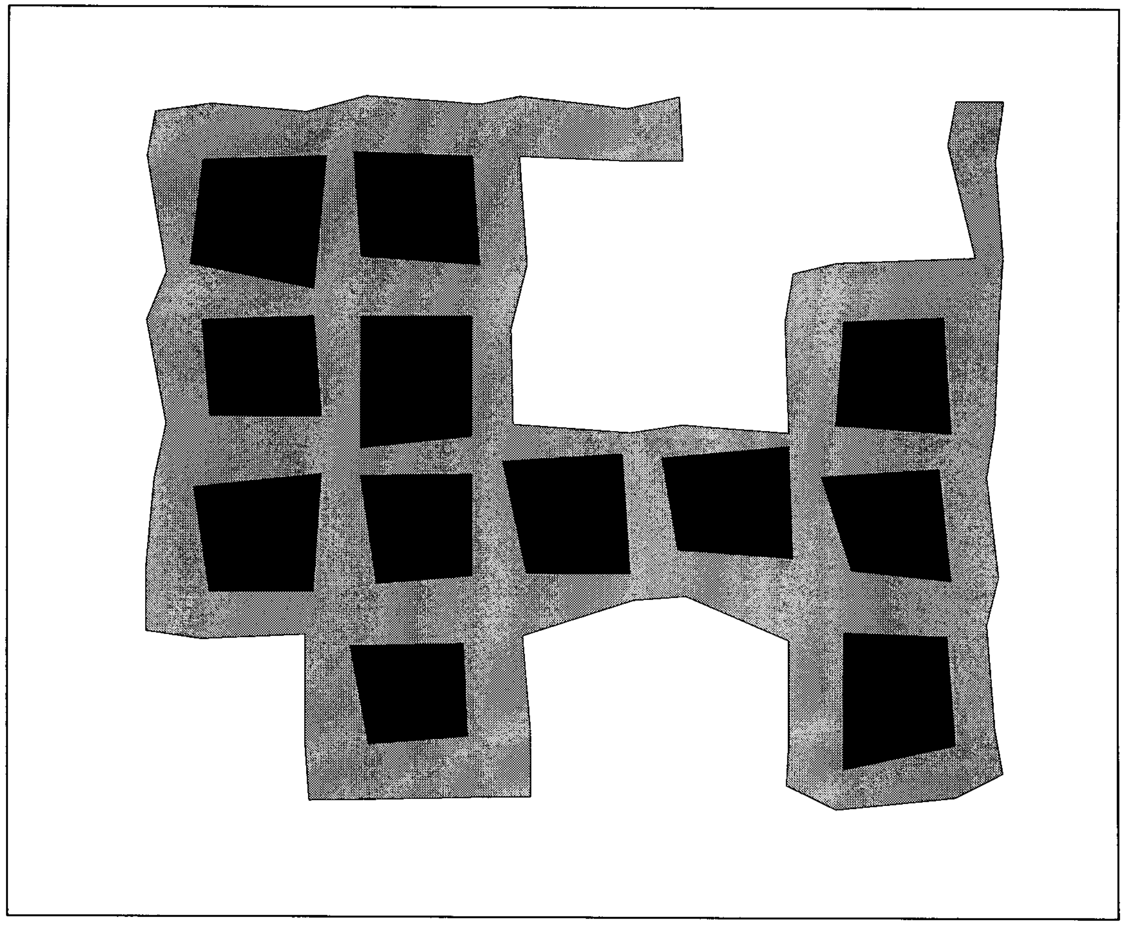

Figure 7.15: An example of a deformed urban grid

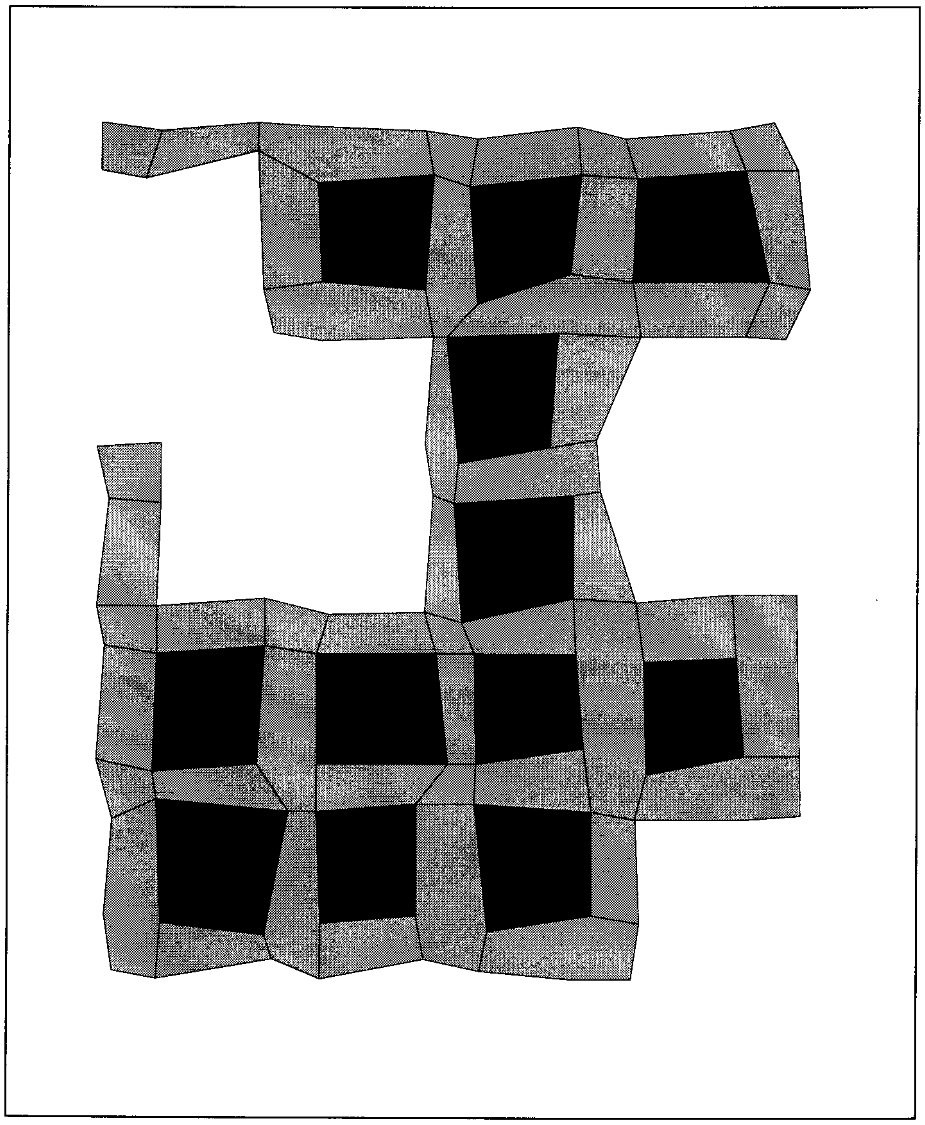

Figure 7.16: A partition of a deformed urban grid

```
{This algorithm assumes that the deformed urban grid is input in
 the form of the thoroughfares defined by the intersections and
 corridors of which they are comprised.}

{Each thoroughfare is an array, where each element in the array
 contains
  information as to whether the polygon is an intersection or a
  corridor and
  the number of the intersection or corridor in the appropriate
  list
 Each element in the corridor or intersection lists contains
  the coordinates of the vertices in that corridor or
  intersection and
  a flag to indicate its status as regards inclusion in the
  final partition}

{$Included(polygon)$ is a function which takes a polygon
 determines if the polygon is an intersection or a corridor and
 checks a flag to see if this polygon has been included in a
 polygon which is already in the partition.
 It returns True or False.}
{$Combine(polygon1, polygon2)$ forms a new polygon which is a
 merging of two convex polygons}
{$Convex(polygon)$ is a function which determines whether a
 given polygon is convex.
 It returns True or False.}
```

Figure 7.17: Partitioning a deformed urban grid – The description of the input into the algorithm and the functions used in the algorithm

```
FOR $i$ from 1 to number of thoroughfares
   IF $included(thoroughfare[i].polygon[1])$
      THEN
         $WorkingPolygon \leftarrow thoroughfare[i].polygon[2]$
         $j \leftarrow 3$
      ELSE
         $WorkingPolygon \leftarrow thoroughfare[i].polygon[1]$
         $j \leftarrow 2$
   WHILE $j \leq$ number of polygons in thoroughfare
      IF $NOT(included(thoroughfare[i].polygon[j]))$
         THEN
         {It might be possible to merge some polygons}
            $TestPolygon \leftarrow Combine(WorkingPolygon,$
            $thoroughfare[i].polygon[j]))$
            IF $Convex(Testpolygon)$
               THEN
                  $WorkingPolygon \leftarrow TestPolygon$
                  $j \leftarrow j + 1$
               ELSE
                  IF $WorkingPolygon$ is not a single intersection
                     THEN
                        Add $WorkingPolygon$ to $Partition$
                        Mark all corridors and intersections in
                        $WorkingPolygon$ as included
                        $WorkingPolygon \leftarrow thoroughfare[i].polygon[j]$
                        $j \leftarrow j + 1$
                     ELSE
                        $WorkingPolygon \leftarrow thoroughfare[i].polygon[j]$
                        $j \leftarrow j + 1$
         ELSE
         {The next polygon is an intersection which was already
          included in a cross thoroughfare.
          This intersection cannot be used again.}
            Add $WorkingPolygon$ to $Partition$
            Mark all corridors and intersections in $WorkingPolygon$
            as included
            $WorkingPolygon \leftarrow thoroughfare[i].polygon[j]$
            $j \leftarrow j + 1$
FOR $k$ from 1 to Number of Intersections
   IF $NOT(Included(intersection[k]))$
      Add $Intersection[k]$ to $Partition$
```

Figure 7.18: Partitioning a deformed urban grid

To see how the algorithm works consider the portion of a deformed urban grid shown in Figure 7.19. In this example only two thoroughfares are shown but this is enough to illustrate how the algorithm computes a partition.

Assume that the horizontal thoroughfare is considered first. This thoroughfare is made up of intersection a, corridor A, intersection b, corridor B and intersection c. The first polygon in this thoroughfare is a. This polygon has not yet been included in a polygon in the final partition (line 01) so the *WorkingPolygon* becomes that polygon and j is initialised appropriately (lines 06 and 07). This is the beginning of the phase of attempting to merge the polygons in that horizontal thoroughfare into a smaller number of convex polygons.

The `WHILE` loop (lines 08 to 30) is then entered. In the `WHILE` loop the first step is to test whether the next polygon in the thoroughfare has already been included in a polygon in the final partition (line 09). In this case the next polygon is A. A is a corridor and could not have been included in a polygon in the final partition – this is the first and only time it is considered. The *WorkingPolygon*, a, and A are then combined to form the *TestPolygon* (line 11). *TestPolygon* is then tested to see if it is convex (line 12). It is convex, a and A make a convex polygon, and so *WorkingPolygon* becomes this new polygon (line 14) and the pointer to the next polygon to be considered is incremented (line 15). The `WHILE` loop will then be re-entered to consider the next polygon in the thoroughfare.

This next polygon is the intersection b. b has not been included – this is the first time it is being considered – and so is combined with *WorkingPolygon* to make a new *TestPolygon*. As before *TestPolygon* is convex and so this polygon becomes the new *WorkingPolygon*.

When the algorithm considers the next polygon in the thoroughfare, now B, *TestPolygon* is not convex. This means that *WorkingPolygon* will be saved as a component of the final partition (provided it is not made up of only a single intersection (line 17)). Lines 19 to 22 save the current *WorkingPolygon*, mark the appropriate intersection and corridor polygons as included and make the new *WorkingPolygon* the next polygon in the thoroughfare. In this case *WorkingPolygon* becomes B.

In the case where *WorkingPolygon* was a single intersection then lines 24 and 25 would be executed. Any *WorkingPolygon* which is only a single intersection would not be immediately saved as part of the final partition as it could later be considered for combining with other polygons when the vertical thoroughfares are considered.

In this example the algorithm would continue and c would be merged with B. This first thoroughfare would thus be partitioned into two convex polygons.

The algorithm would proceed in a similar fashion for all other horizontal thoroughfares before considering the thoroughfare consisting of intersection a, corridor C, intersection d, corridor D and intersection e. In this case the first intersection

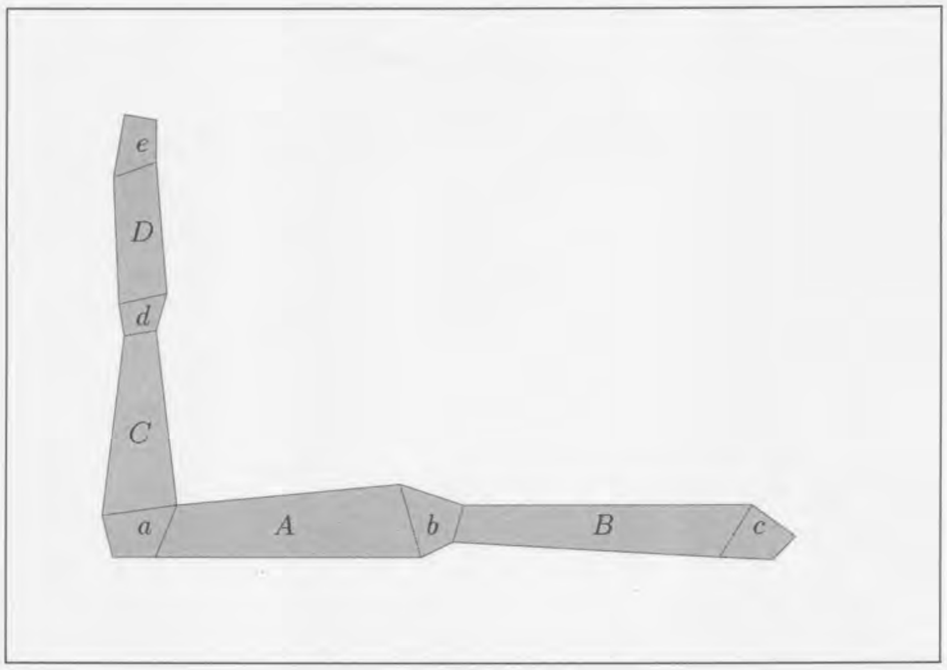

Figure 7.19: A portion of a deformed urban grid – to illustrate the algorithm

in the thoroughfare has already been included in some other larger convex polygon and cannot be reused. Lines 03 and 04 are executed – $WorkingPolygon$ becomes C.

The algorithm would then continue as before to process this thoroughfare and all of the other vertical thoroughfares.

The final stage of the algorithm (lines 31 to 33) would be to ensure that any intersections which were not included in larger convex polygons when they were considered as part of a horizontal thoroughfare and later as part of vertical thoroughfare are added to the final partition.

Figure 7.20 shows the convex polygons (in gray) which would have been generated by the algorithm in Figures 7.17 and 7.18 working on the deformed urban grid in Figure 7.15 once the lower thoroughfares have been considered.

Using the algorithm in Figures 7.17 and 7.18 a partition for a deformed urban grid can be found. The next step in the process would be to place the minimum number of axial lines to cross all of the adjacencies between the convex polygons in the partition.

Conjecture 7.3.2 *The minimal set of axial lines to cross the adjacencies in the convex map of a deformed urban grid can be found in polynomial time.*

This conjecture seems to be true because the convex polygons which make up the convex map of the area are likely to be aligned along deformed thorough-

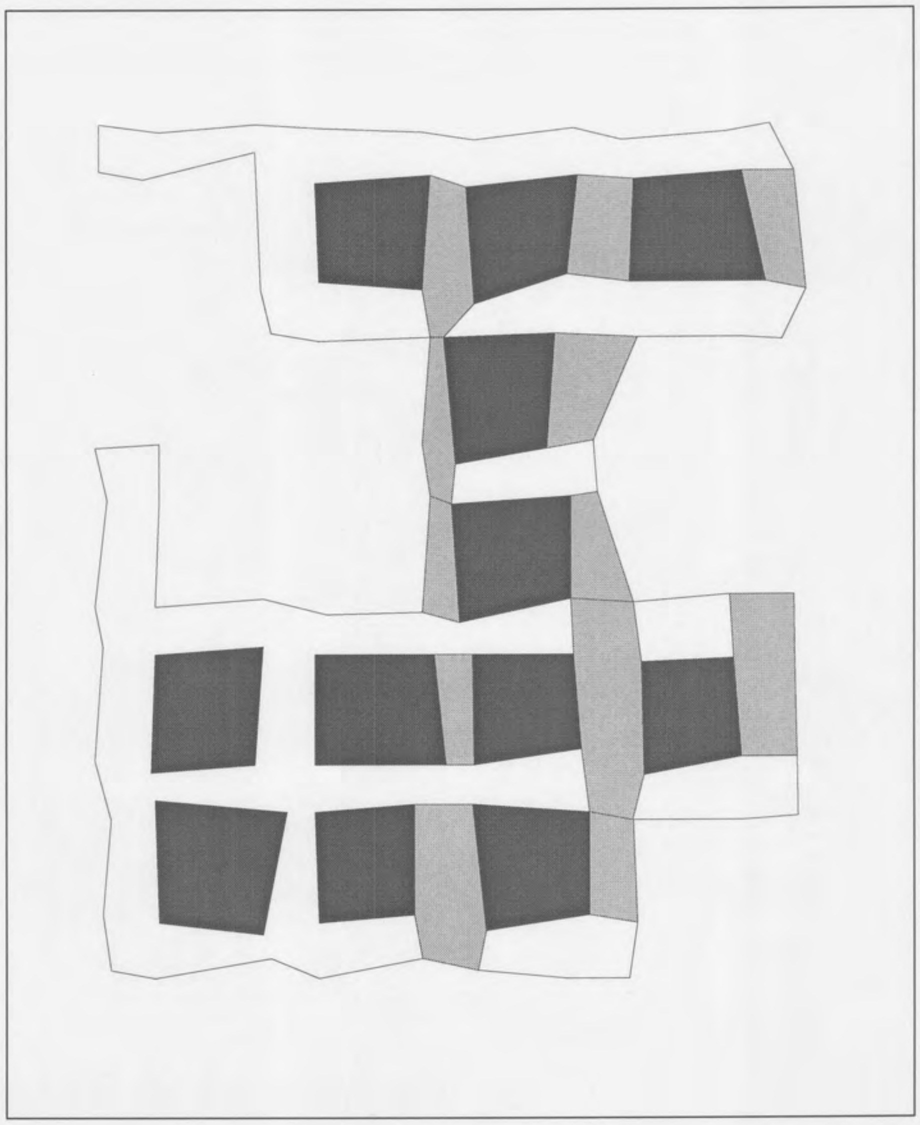

Figure 7.20: A partial solution from the algorithm to partition a deformed urban grid

fares. The shapes of the corridors (deformed rectangles) and intersections (deformed squares) mean that each intersection can potentially be merged with the corridors in one thoroughfare but it is unlikely that an intersection could be merged with corridors from more than one thoroughfare to form a convex polygon. Thus the number of axial lines needed to cover each thoroughfare can be found and the final solution can be determined by removing any unnecessary lines from this set. Figure 7.21 shows the axial lines which would be placed to cross the adjacencies in the (not necessarily) minimal partition of Figure 7.16. This figure supports the idea that there is little choice in placing the axial lines and that a solution in polynomial time is likely.

Future research can be focussed on proving Conjecture 7.3.2 by finding a polynomial time algorithm to place a minimal set of lines or on disproving the conjecture by extending the NP-Completeness proofs presented above to cover this situation as well.

The results presented in this section of the thesis are important because they show that if the space in the original town plan can be modelled as a deformed urban grid then it is likely (if Conjecture 7.3.2 is true) that an axial map can be found in polynomial time.

7.4 More general urban polygons

In reality it is not likely to be possible to model all urban layouts (polygons with holes) as deformed urban grids – the layouts are more irregular than this. Here the results of Lingas [1982] and O'Rourke and Supowit [1983] show that the partitioning problem is NP-Hard. In Chapter 5 of this thesis it is shown that the axial line placement problem restricted to rectangles and lines of arbitrary orientation is NP-Complete and in Chapter 6 the axial line placement problem for convex polygons and axial lines of arbitrary orientation is shown to be NP-Complete. These results indicate strongly that finding the convex map of an urban layout and the placing of the minimum number of axial lines to cross the adjacencies in the convex map of some urban layout are likely to be NP-Complete. The first step in future research in this regard would be to prove these claims. Subsequent research could then be concentrated on approximation algorithms to give "reasonable" approximations to the convex map and axial map for urban layouts of this form.

7.5 Conclusion

This chapter shows that the axial line placement problem can be solved exactly in polynomial time for some restricted cases – a complete urban grid, a simple urban grid and a general urban grid. It also seems (but is not proven here) that an

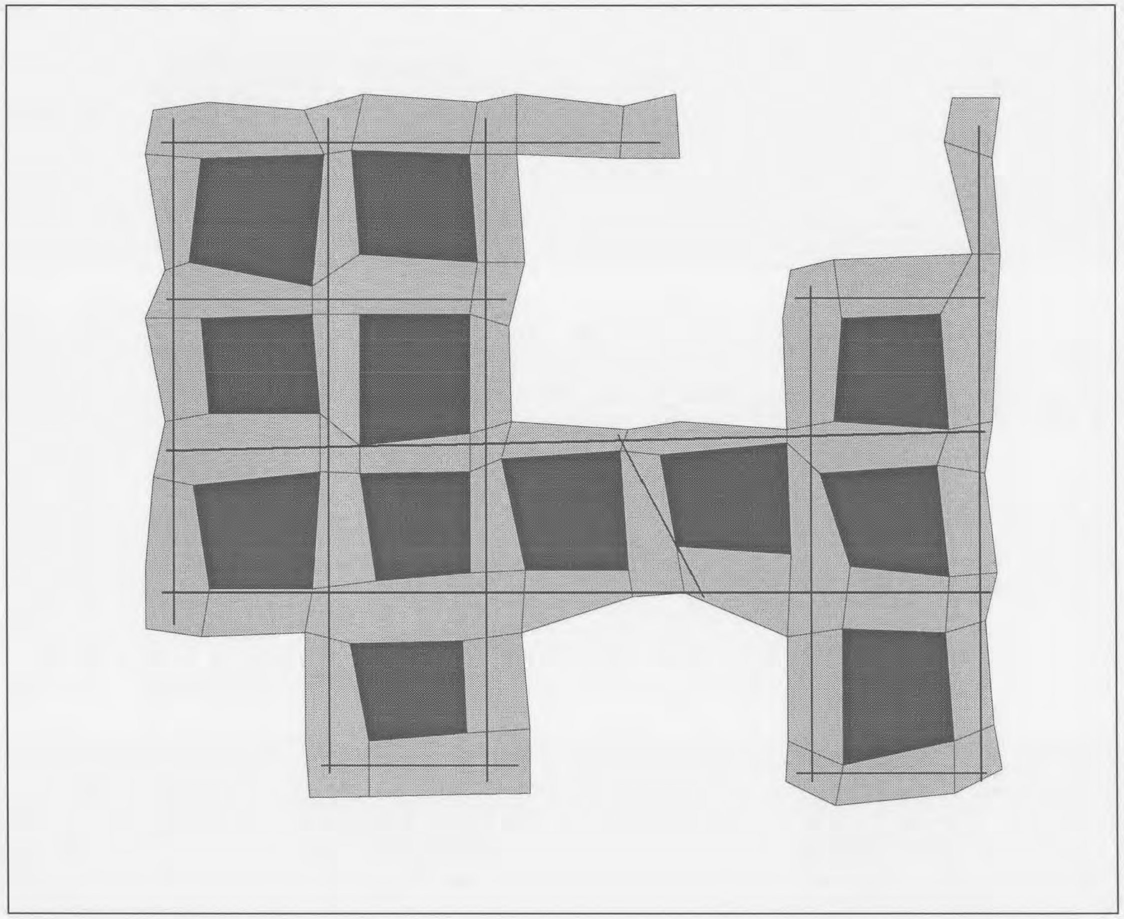

Figure 7.21: Placing the axial lines to cross the adjacencies in a partitioned deformed urban grid

exact solution is possible in polynomial time for a deformed urban grid. The more general problem – placing a minimal set of axial lines to cross all of the adjacencies between arbitrary convex polygons – was shown to be NP-Complete in Chapter 6 of this thesis and this suggests that finding polynomial time solutions in general town plans could also be NP-Complete and heuristics will have to be used to find acceptable approximations.

The next chapter of the thesis presents some ideas for further research to build on the results of this work.

Chapter 8

Future Research

8.1 Introduction

This thesis addresses a number of computational problems which arise from the potential automation of the space syntax method [Hillier *et al.*, 1983]. The research focussed specifically on the problems associated with the placing of axial lines in a convex map. It was, however, impossible to address all of the possible research questions which were originally identified in this area. In addition, as the work for this thesis progressed other problems were raised. The future research which stems from the work discussed in this thesis is thus of two types

1. problems which were presented in Chapter 3 but were not tackled at all in this research,
2. problems which arose in the course of pursuing this research but were not addressed (or at least not fully addressed) at the time.

Sections 8.2 and 8.3 of this chapter discuss the problems which fall into these two groups in more detail.

8.2 Open problems

This thesis considers a number of axial line placement problems which have derived from the original idea of automating the Space Syntax Analysis method. In all cases the problems which have been considered are where multiple crossings of adjacencies are permitted. If the restriction is made that each adjacency is crossed *exactly* once then the list of problems below are still open questions.

1. Restricting the problem to dealing with orthogonally aligned rectangles and axial lines which are parallel to the edges of the rectangles and where each adjacency must be crossed by *exactly* one axial line.

2. Restricting the problem to dealing with orthogonally aligned rectangles but where the axial lines are not necessarily orthogonal and where each adjacency must be crossed by *exactly* one axial line.

3. Considering the problem of general convex polygons (not restricted to the form which would occur in the town planning domain) where the adjacencies between polygons are crossed by non-orthogonal axial lines and where each adjacency must be crossed by *exactly* one axial line.

4. Considering the problem of convex polygons, which represent the deformed grid in the town planning domain, where the adjacencies between polygons are crossed by non-orthogonal lines and where each adjacency must be crossed by *exactly* one axial line.

Based on the results given in Chapters 4, 5 and 6 there is evidence to suggest that each of these problems is NP-Complete. Future research would have to substantiate this claim. Once this has been done heuristics will have to be derived and tested to produce approximations to the solutions to these problems.

Heuristics which are likely to work well in the case of (1) (and possibly (2), (3) and (4)) above are

- choosing the longest line first.
- choosing the line with the highest number of single crossings first.
- choosing the line with the highest number of grouped single crossings first.
- choosing essential lines first (as was done for the heuristic discussed in Chapter 4) then using one or all of the other approaches suggested here. This makes sense as the adjacency which is crossed only by the essential line will generate a line which must be in the final set of lines.

Again a profitable area of research could be to study special cases of these problems where exact solutions can be found in polynomial time.

8.3 Future research arising from this thesis

The problems here are related to the various problems which have been solved as part of this work for this thesis. They are given below grouped according to the variation of the axial line placement problem in which they occurred.

- ***ALP-OLOR***

- In Chapter 4 an algorithm which finds a non-redundant set of axial lines for ***ALP-OLOR*** is discussed. In some instances this algorithm does extra work. Future work could focus on reducing the amount of work done by this "most uncrossed adjacencies" heuristic algorithm. In particular, attempting to address the issue of redundant calculations which are made for the collection of rectangles shown in Figure 4.22 or similar configurations.
- Developing other heuristics to produce approximate solutions to the exact solution could also be a fruitful area of research.
- Section 4.7.1 presents some variations of ***ALP-OLOR*** which can be solved in polynomial time. Also polynomial time algorithms for the orthogonal axial line placement problem for chains of orthogonal rectangles and trees of orthogonal rectangles are presented. Future research can be focussed on finding other arrangements of rectangles which are more general than a tree of rectangles which can be solved in polynomial time.
- An interesting, but not directly related, area of further research is in the generation of test data. Generating non-trivial trees of adjacent but non-overlapping rectangles proved to be relatively complex in this research.

• ***ALP-ALOR***

- In Chapter 5 some ideas for heuristics to produce approximate solutions to the exact solution for ***ALP-ALOR*** are presented. These ideas need to be more fully developed or better heuristics derived. Once this has been done, any resulting heuristics should be implemented and tested for some carefully selected test cases.
- A related area of further research would be looking at other approaches, for example genetic algorithms, for finding approximate solutions.
- The idea of special cases of the problem which can be solved in polynomial time was also very briefly raised in Chapter 5. This is an area in which a lot of additional work could be done.

• ***ALP-ALCP***

- In Chapter 6 ***ALP-ALCP*** is shown to be NP-Complete. This means that finding reasonable approximations to the exact solution in polynomial time becomes important. Thus heuristics for this problem should be developed and tested.
- As part of the work in developing heuristics for this problem it would be interesting to consider other approaches, for example genetic algorithms, to generate approximate solutions.

  - The idea of special cases of the problem which can be solved in polynomial time was also very briefly raised in Chapter 6. This is an area in which a lot of additional work could be done.

- Placing Axial Lines in Town Plans

  - Conjecture 7.3.1 of Chapter 7 states that "The problem of finding the minimal set of convex polygons to partition a deformed urban grid is NP-Hard." This conjecture is based on the results of Lingas [1982] and O'Rourke and Supowit [1983] that partitioning a polygon with holes is NP-Hard. A deformed urban grid seems to offer enough choice to make the problem difficult to solve exactly. Clearly future work should be focussed on proving or disproving this conjecture.
  - Conjecture 7.3.2 states that "The minimal set of axial lines to cross the adjacencies in the convex map of a deformed urban grid can be found in polynomial time." Future research can be focussed on proving this conjecture by finding a polynomial time algorithm or on disproving the conjecture by extending the NP-Completeness proofs to cover this situation as well.
  - Not all urban layouts can be modelled by deformed urban grids as defined in Chapter 7. A fruitful area of research would be considering more general configurations of polygons with holes which could represent urban layouts. The focus here should be on determining whether the problem remains NP-Complete or whether the restriction to modelling urban layouts is enough to remove the choice which complicates the problem.

8.4 Conclusion

This thesis has considered a number of Axial Line Placement problems but the area is still new and there are lots of other problems which should be of interest to computer scientists. Some of these problems are discussed above but the list is not complete and it is likely that the list will grow rather than shrink as more work is done in this area.

Chapter 9

Conclusion

9.1 Introduction

Computers can be used to automate many mundane and boring tasks and can thus free the professional to concentrate on the more intellectually demanding aspects of many jobs. The original intent of this thesis was to consider the possibility of automating some or all of the task of applying the Space Syntax method to an urban layout. An architect or town planner would apply the space syntax method ([Mills, 1992]) to a town or city in 4 main steps.

1. Finding the "deformed grid" of the urban layout.
2. Creating the convex map of the area.
3. Creating the axial map of the area from the convex map.
4. Combining the information from the convex map and the axial map to produce an integration factor for the town/city.

Currently the town planner or architect does all but the final analysis phase by hand. The initial aim of this thesis was thus to determine which phases of this work could potentially be automated in order to free the architect or town planner to concentrate on the more intellectually stimulating design phases.

In the introduction to this thesis it is shown that all of these phases could benefit from automation and that the potential automation of space syntax gives rise to many areas of research. These areas include image processing, computational geometry and algorithms. However, tackling all of these areas would be an immense tasks. For this reason, the problems of separating space from non-space, determining the convex map and the final analysis stage with its associated algorithms were not considered as part of this research. The decision was made to focus the research for this PhD on the problem of finding the axial lines which cross all of the shared boundaries between the convex polygons in the convex map, that is finding the axial map for a given layout – the *Axial Line Placement* problem or ***ALP***. This problem on its own is still very big and could not be solved entirely. This thesis only tackled a small subset of the research questions proposed in Chapter 3 but the contributions of this research (discussed in Section 9.2) can go some way towards answering the original research question and are interesting problems in computational geometry in their own right.

9.2 Contributions of this thesis

As discussed in Section 9.1 above, this thesis only considered a small subset of the possible problems which arise in Axial Line Placement. The process which was applied in the research was to start with simplifications of ***ALP*** and then to build towards solving the general problem. The specific contributions that were made by this research are presented below.

- This research proved that the axial line placement problem for orthogonal axial lines and collections of adjacent orthogonal rectangles (***ALP-OLOR***) is NP-Complete. This proof was accomplished by a transformation from ***biconnected planar vertex cover*** to ***stick diagram*** and hence to ***ALP-OLOR***. As ***ALP-OLOR*** is NP-Complete the research then focussed on approximations and the thesis presents a heuristic algorithm which produces a non-redundant set of maximal axial lines to cross all of the adjacencies in a collection of adjacent orthogonal rectangles. This heuristic algorithm was implemented and tested on a number of configurations of adjacent rectangles and was shown to produce reasonable approximations in polynomial time. The research then focussed on special cases of ***ALP-OLOR*** which can be solved in polynomial time. Specifically it was shown that exact polynomial-time solutions can be found for configurations of adjacent orthogonal rectangles which can be converted to interval graphs; chains of adjacent orthogonal rectangles and trees of adjacent orthogonal rectangles.

- The next phase of the research involved relaxing the restriction that the axial lines had to be orthogonal. The research proved that the axial line placement problem for axial lines with arbitrary orientation and collections of adjacent orthogonal rectangles (***ALP-ALOR***) is NP-Complete. This proof was also accomplished by a transformation from ***biconnected planar vertex cover*** to ***stick diagram*** and hence to ***ALP-ALOR***. Some heuristics for ***ALP-ALOR*** were derived and are presented in this thesis.

- The next step in the research was to allow general convex polygons instead of orthogonal rectangles. The research proved that the axial line placement problem for axial lines with arbitrary orientation and collections of adjacent convex polygons (***ALP-ALCP***) is NP-Complete.

- After considering the simplifications discussed above, the research emphasis was shifted to focus on finding axial lines to cross the adjacencies between convex polygons in a situation which is representative of the real world problem, i.e. covering the convex map of some town with an axial map. In this phase of the research it was proved that the minimum partition of complete, simple and general urban grids can be found in polynomial time. It was also proved that finding a minimal set of maximal axial lines for complete, simple and general urban grids can be done in polynomial time. It was conjectured that finding the convex map of a deformed urban grid is NP-Complete or NP-Hard (based on the result of Culberson and Reckhow [1994]). Assuming that this conjecture is true a polynomial-time heuristic algorithm to produce a partition of a deformed urban grid was developed. It was further conjectured that the axial map of a deformed urban grid can be found in polynomial time.

In summary this research has shown that ***ALP-OLOR***, ***ALP-ALOR*** and ***ALP-ALCP*** are NP-Complete. It has also shown that placing axial lines in some simplified configurations of polygons can be done in polynomial-time. There are, however, still many open problems in this area. These are discussed in some detail in Chapter 8 and discussed briefly below.

9.3 Future work

The results of the research discussed in this thesis do not answer all the questions raised in ***ALP***. In fact, as the research progressed more problems were raised than were solved. Chapter 8 discusses in some detail the open problems which were not tackled in this research and the subsidiary problems which arose out of the problems which were tackled.

The biggest area of future research is in considering the groups of problems (similar to those discussed in Section 9.2) where any adjacency must be crossed by exactly one axial line – in contrast to this research where multiple crossings of adjacencies were permitted. It has been conjectured [O'Rourke, 1999] that these problems can be solved in polynomial time but as yet this has not been proved.

Another important area of research is in extending the results of this research to address the original problem of finding the convex map and axial map of some urban layout. The partitioning problem for polygons with holes is NP-Hard so heuristic algorithms will have to be found to return acceptable approximations to the exact solution. It seems that when the architect or urban designer applies the Space Syntax Analysis technique that she/he uses heuristics in her/his solution for the convex map. This means that an approximate solution is likely to be acceptable to the urban designer especially if the solution is similar to her/his own. This suggests an artificial intelligence approach to designing the heuristic could be a fruitful area of research. A similar comment applies for determining the axial map which covers the convex map of some urban layout.

9.4 Overall Conclusions

The original aim of this thesis – automating the Space Syntax method – was clearly too big a task to be undertaken in a single PhD. With this in mind the research was focussed only on the computational problems which arise with the placing of axial lines. Even with this scaling down, the research area was still very big and only some of the problems which were identified were tackled. The work which was completed is, however, important in that a number of new NP-Complete problems (***ALP-OLOR***, ***ALP-ALOR*** and ***ALP-ALCP***) were identified and also because some new and interesting problems in the area of computational geometry have been introduced.

References

[Adler *et al.* 2001] J. Adler, G. D. Christelis, J. A. Deneys, G. D. Konidaris, G. Lewis, A. G. Lipson, R. L. Phillips, D. K. Scott-Dawkins, D. A. Shell, B. V. Strydom, W. M. Trakman, and L. D. Van Gool. Finding adjacencies in non-overlapping polygons. In *Electronic Proceedings of South African Institute of Computer Scientists and Information Technologists Annual Research Symposium*, 2001.

[Andreae 1992] T. Andreae. Some results on visibility graphs. *Discrete Applied Mathematics*, 40:5–17, 1992.

[Asano *et al.* 1986] T. Asano, T. Asano, and H. Imai. Partitioning a polygonal region into trapezoids. *Journal of the ACM*, 33(2):290–312, April 1986.

[Asano *et al.* 1999] T. Asano, S. K. Ghosh, and T.C. Shermer. Visibility in the plane. In J.-R. Sack and J. Urrutia, editors, *Handbook on Computational Geometry*, chapter 19. Elsevier Science, 1999.

[Ashman 1999] J. Ashman. The design of different input arrangements to achieve predictable performance of a ray guarding heuristic. Honours Research Report, School of Computer Science, University of the Witwatersrand, 1999.

[Avis and Toussaint 1981] D. Avis and G. T. Toussaint. An optimal algorithm for determining the visibility of a polygon from an edge. *IEEE Transactions on Computers*, C-30(12):910–914, December 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 1997] S. Baase. *A Gift of Fire: Social, Legal, and Ethical Issues in Computing*. Prentice Hall, Upper Saddle River, New Jersey, 1997.

[Barzohar and Cooper 1996] M. Barzohar and D. B. Cooper. Automatic finding of main roads in aerial images by using geometric-stochastic models and estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(7):707–721, 1996.

[Berge 1962] C. Berge. *The Theory of Graphs And Its Applications.* Methuen & Co, London, 1962. Translated by A. Doig.

[Bhattacharya and Ghosh 1998] B. K. Bhattacharya and S. K. Ghosh. Characterizing LR-visibility polygons and related problems. In Mike Soss, editor, *Proceedings of the 10th Canadian Conference on Computational Geometry*, pages 48–49, Montréal, Québec, Canada, 1998. School of Computer Science, McGill University.

[Biedl *et al.* 1997] T. Biedl, G. Kant, and M. Kaufmann. On triangulating planar graphs under the four-connectivity constraint. *Algorithmica*, 19:427–446, 1997.

[Bilbrough and Sanders 1998] J. Bilbrough and I. D. Sanders. A linear algorithm for partial edge visibility. In *Proceedings of the 1998 SAICSIT Research and Development Symposium*, pages 200–210. South African Institute of Computer Scientists and Information Technologists, November 1998.

[Bilbrough 1998] J. Bilbrough. Partial edge visibility in chains of orthogonal rectangles. Honours Research Report, School of Computer Science, University of the Witwatersrand, 1998.

[Bjorling-Sachs and Souvaine 1991] I. Bjorling-Sachs and D. L. Souvaine. A tight bound for guarding general polygons with holes. Technical Report LCSR-TR-165, Laboratory for Computer Science Research, Hill Centre for the Mathematical Sciences, Busch Campus, Rutgers University, New Brunswick, New Jersey, 1991.

[Bjorling-Sachs and Souvaine 1995] I. Bjorling-Sachs and D. L. Souvaine. An efficient algorithm for guard placement in polygons with holes. *Discrete & Computational Geometry*, 13:77–109, January 1995.

[Bose *et al.* 1993] P. Bose, L. Guibas, A. Lubiw, M. Overmars, D. Souvaine, and J. Urrutia. The floodlight problem. In *Proceedings of the 5th Canadian Conference on Computational Geometry*, pages 399–404, Waterloo, ON, Canada, August 1993. University of Waterloo.

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

[Bukovska 2000] D. Bukovska. Partitioning a deformed urban grid. Honours Research Report, School of Computer Science, University of the Witwatersrand, 2000.

[Carlsson and Jonsson 1993] S. Carlsson and H. Jonsson. Guarding a treasury. In *Proceedings of the 5th Canadian Conference on Computational Geometry*, pages 85–90, Waterloo, ON, Canada, August 1993. University of Waterloo.

[Carlsson and Jonsson 1995] S. Carlsson and H. Jonsson. Computing a shortest watchman path in a simple polygon in polynomial-time. In *Proceedings of the 4th Workshop on Algorithms and Data Structures (WADS'95)*, volume 955 of *Lecture Notes in Computer Science*, pages 122–134. Springer-Verlag, 1995.

[Castleman 1996] K. R. Castleman. *Digital Image Processing*. Prentice Hall, Upper Saddle River, New Jersey, 1996.

[Chazelle and Dobkin 1985] B. Chazelle and D. P. Dobkin. Optimal convex decompositions. In G. T. Toussaint, editor, *Computational Geometry*, pages 63–133. North-Holland, 1985.

[Chazelle and Guibas 1989] B. Chazelle and L. J. Guibas. Visibility and intersection problems in plane geometry. *Discrete & Computational Geometry*, 4:551–581, 1989.

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

[Conn and O'Rourke 1987] H. Conn and J. O'Rourke. Some restricted rectangle covering problems. In *Proceedings of the 1987 Allerton Conference*, 1987.

[Contreras *et al.* 1998a] F. Contreras, J. Czyzowicz, N. Fraiji, and J. Urrutia. Illuminating triangles and quadrilaterals with vertex floodlights. In *Proceedings of the Tenth Canadian Conference on Computational Geometry*, pages 58–59, Montreal, Quebec, Canada, August 1998. School of Computer Science, McGill University.

[Contreras *et al.* 1998b] F. Contreras, J. Czyzowicz, E. Rivera-Campo, and J. Urrutia. Optimal floodlight illumination of stages. In *Proceedings of the Fourteenth Annual Symposium on Computational Geometry (SCG'98)*, pages 409–410, New York, June 1998. Association for Computing Machinery.

[Cormen *et al.* 1990] T. H. Cormen, C. E. Leierson, and R. L. Rivest. *Introduction to Algorithms*. The MIT Press, Cambridge, Massachusetts, 1990. Sixteenth Printing, 1996.

[Culberson and Reckhow 1988] J. Culberson and R. A. Reckhow. Covering polygons is hard. Preliminary Abstract TR 88-6, Department of Computing Science, The University of Alberta, 1988.

[Culberson and Reckhow 1989a] J. Culberson and R. A. Reckhow. Othogonally convex coverings of orthogonal polygons without holes. *Journal of Computer and System Sciences*, 39(2):166–204, 1989.

[Culberson and Reckhow 1989b] J. Culberson and R. A. Reckhow. A unified approach to orthogonal polygon covering problems via Dent diagrams. Technical Report TR 89-6, Department of Computing Science, The University of Alberta, 1989.

[Culberson and Reckhow 1994] J. Culberson and R. A. Reckhow. Covering polygons is hard. *Journal of Algorithms*, 17:2–44, 1994.

[Czyzowicz *et al.* 1993] J. Czyzowicz, E. Rivera-Campo, and J. Urrutia. Optimal floodlight illumination of stages. In *Proceedings of the Fifth Canadian Conference on Computational Geometry*, pages 393–398, Waterloo, ON, Canada, August 1993. University of Waterloo.

[Czyzowicz *et al.* 1994] J. Czyzowicz, E. Rivera-Campo, N. Santoro, J. Urrutia, and J. Zaks. Guarding rectangular art galleries. *Discrete Applied Mathematics*, 50:149–157, 1994.

[Das *et al.* 1993] G. Das, P. J. Heffernan, and G. Narasimhan. LR-Visibility in polygons. In *Proceedings of the Fifth Canadian Conference on Computational Geometry*, pages 303–307, Waterloo, ON, Canada, August 1993. University of Waterloo.

[Deene and Joshi 1992] L. L. Deene and S. Joshi. Treasures in an art gallery. In *Proceedings of the Fourth Canadian Conference on Computational Geometry)*, pages 17–22, 1992.

[Doh and Chwa 1993] J.-I. Doh and K.-Y. Chwa. An algorithm for determining visibility of a simple polygon from an internal line segment. *Journal of Algorithms*, 14(1):139–168, January 1993.

[du Plessis and Sanders 2000] N. du Plessis and I. D. Sanders. Partial Edge Visibility in Chains of Orthogonal Rectangles. Technical Report TR-Wits-CS-2000-15, Department of Computer Science, University of the Witwatersrand, September 2000.

[du Plessis 1999] N du Plessis. Partial edge visibility in chains of orthogonal rectangles. Honours Research Report, School of Computer Science, University of the Witwatersrand, 1999.

[Eastman 1972] C. M. Eastman. Preliminary report on a system for general space planning. *Communications of the ACM*, 15(2):76–87, February 1972.

[Eidenbenz *et al.* 1998] S. Eidenbenz, C. Stamm, and P. Widmayer. Inapproximability of some art gallery problems. In Mike Soss, editor, *Proceedings of the 10th Canadian Conference on Computational Geometry*, pages 64–65, Montréal, Québec, Canada, 1998. School of Computer Science, McGill University.

[ElGindy and Avis 1981] H. A. ElGindy and D. Avis. A linear algorithm for computing the visibility polygon from a point. *Journal of Algorithms*, 2:186–197, 1981.

[Feng and Pavlidis 1975] H.-Y. Feng and T. Pavlidis. Decomposition of polygons into simpler components: Feature generation for syntatic pattern recognition. *IEEE Transactions on Computers*, C-24:636–650, June 1975.

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

[Franzblau and Kleitman 1984] D. S. Franzblau and D. J. Kleitman. An algorithm for covering polygons with rectangles. *Information and Control,*, 63:164–189, 1984.

[Franzblau 1989] D. S. Franzblau. Performance guarantees on a sweep-line heuristic for covering rectilinear polygons with rectangles. *SIAM Journal on Discrete Mathematics*, 2(3):307–321, August 1989.

[Füredi and Kleitman 1994] Z. Füredi and D. J. Kleitman. The prison yard problem. *Combinatorica*, 14(3):287–300, 1994.

[Garey and Johnson 1979] M. R. Garey and D. S. Johnson. *Computers and Intractability, A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York, 1979.

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

[Gavril 1972] F. Gavril. Algorithms for minimum colorings, maximum clique, minimum coverings by cliques, and maximum independent set of a chordal graph. *SIAM Journal of Computing*, 1:180–187, 1972.

[Geman and Jedynak 1996] D. Geman and B. Jedynak. An active testing model for tracking roads in satellite images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(1):1–14, 1996.

[Gewali and Ntafos 1993] L. Gewali and S. Ntafos. Covering grids and orthogonal polygons with periscope guards. *Computational Geometry: Theory and Applications*, 2:309–334, 1993.

[Gewali *et al.* 1992] L. Gewali, M. Keil, and S. Ntafos. On covering orthogonal polygons with star-shaped polygons. *Information Sciences*, 65:45–63, 1992.

[Gewali 1993] L. P. Gewali. On minimum and maximum visibility problem. In *Proceedings of the Fifth Canadian Conference on Computational Geometry*, pages 241–245, Waterloo, ON, Canada, August 1993. University of Waterloo.

[Ghosh *et al.* 1993] S. K. Ghosh, A. Maheshwari, S. P. Pal, S. Saluja, and C. E. Veni Madhavan. Characterizing and recognising weak visibility polygons. *Computational Geometry: Theory and Applications*, 3:213–233, 1993.

[Ghosh 1991] S. K. Ghosh. Computing the visibility polygon from a convex set and related problems. *Journal of Algorithms*, 12(1):75–95, March 1991.

[Ghosh 1996] S. K. Ghosh. Corrigendum: A note on computing the visibility polygon from a convex chain. *Journal of Algorithms*, 21(3):657–662, November 1996.

[Ghosh 1997] S. K. Ghosh. On recognizing and characterizing visibility graphs of simple polygons. *Discrete & Computational Geometry*, 17:143–162, 1997.

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

[Gonzalez and Wintz 1987] R. C. Gonzalez and P. Wintz. *Digital Image Processing*. Addison Wesley, Cambridge, Massachusetts, second edition, 1987.

[Gonzalez and Woods 1992] R. C. Gonzalez and R. E. Woods. *Digital Image Processing*. Addison Wesley, Reading, Massachusetts, 1992. 1993 Printing.

[Györi *et al.* 1996] E. Györi, F. Hoffman, K. Kriegel, and T. Shermer. Generalized guarding and partitioning for rectilinear polygons. *Computational Geometry: Theory and Applications*, 6:21–44, 1996.

[Hagger 2001] L. Hagger. Partitioning of a deformed urban grid. Honours Research Report, School of Computer Science, University of the Witwatersrand, 2001.

[Hall 1999] A. Hall. Ray guarding trees of rectangles. Honours Research Report, School of Computer Science, University of the Witwatersrand, 1999.

[Harary 1969] F. Harary. *Graph Theory.* Addison-Wesley, Reading, Massachusetts, 1969.

[Harel 1992] D. Harel. *Algorithmics: The Spirit of Computing.* Addison-Wesley, Reading, Massachusetts, 2nd edition, 1992. With technical assistance of Roni Rosner.

[Hedetniemi 1996] S. T. Hedetniemi. Personal communication, 1996.

[Heffernan and Mitchell 1995] P. J. Heffernan and J. S. B. Mitchell. An optimal algorithm for computing visibility in the plane. *SIAM Journal of Computing*, 24(1):184–201, February 1995.

[Herbert *et al.* 1994] T. S. Herbert, G. Mills, and I. D. Sanders. African shape grammar: A language of linear Ndebele homesteads. *Environment and Planning B: Planning and Design*, 21(4):453–476, 1994.

[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.

[Hochbaum 1982] D. S. Hochbaum. Approximation algorithms for the set covering and vertex cover problems. *SIAM Journal of Computing*, 1982.

[Hoffmann *et al.* 1991] F. Hoffmann, M. Kaufmann, and K Kriegel. The art gallery theorem for polygons with holes. In *Proceedings of the 32nd Symposium of Foundations of Computer Science*, pages 39–48, 1991.

[Huertas and Nevatia 1988] A. Huertas and R. Nevatia. Detecting buildings in aerial images. *Computer Vision, Graphics, and Image Processing*, 41(2):131–152, February 1988.

[Imai and Asano 1986] H. Imai and T. Asano. Efficient algorithms for geometric graph search problems. *SIAM Journal of Computing*, 15:478–494, 1986.

[Jähne 1997] B. Jähne. *Digital Image Processing: Concepts, algorithms and scientific applications.* Springer-Verlag, Berlin, 1997.

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

[Ke 1989] Y. Ke. *Polygon visibility algorithms for weak visibility and link distance problems*. PhD thesis, Johns Hopkins University, Baltimore, 1989.

[Keil and Sack 1985] J. M. Keil and J.-R. Sack. Minimum decompositions of polygonal objects. In G. T. Toussaint, editor, *Computational Geometry*, pages 197–216. North-Holland, 1985.

[Keil and Snoeyink 1998] J. M. Keil and J. Snoeyink. On the time bound for convex decomposition of simple polygons. In *Proceedings of the 10th Canadian Conference on Computational Geometry*, pages 53–54, Montréal, Québec, Canada, August 1998. School of Computer Science, McGill University.

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

[Keil 1999] J. M. Keil. Polygon decomposition. In J.-R. Sack and J. Urrutia, editors, *Handbook on Computational Geometry*, chapter 11. Elsevier Science, 1999.

[Kenny 2000] L.-A. Kenny. Non-orthogonal axial line placement in orthogonal rectangles. Honours Research Report, School of Computer Science, University of the Witwatersrand, 2000.

[Konidaris 2001] G. D. Konidaris. Axial line placement in deformed urban grids. Honours Research Report, School of Computer Science, University of the Witwatersrand, 2001.

[Koning and Eizenberg 1981] H. Koning and J. Eizenberg. The language of the prairie: Frank Lloyd Wright's prairie houses. *Environment and Planning B: Planning and Design*, 8:295–323, 1981.

[Kooshesh *et al.* 1990] A. A. Kooshesh, B. M. E. Moret, and L. A. Székely. Improved bounds for the prison yard problem. In *Congressus Numerantium 76*, pages 145–149, 1990.

[Ku and Leong 1995] L. Ku and H.W. Leong. Optimum partitioning of a rectilinear layout. Technical Report TRC2/95, Department of Information Systems and Computer Science, National University of Singapore, February 1995.

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

[Levitt and Dwolatzky 1999] S. P. Levitt and B. Dwolatzky. BuRS: A building recognition system. *South African Computer Journal*, (24):68–76, 1999.

[Liaw *et al.* 1993] B.-C. Liaw, N. F. Huang, and R. C. T. Lee. The minimum cooperative guards problem on k-spiral polygons. In *Proceedings of the Fifth Canadian Conference on Computational Geometry*, pages 97–102, Waterloo, ON, Canada, August 1993. University of Waterloo.

[Lichtenstein 1982] D. Lichtenstein. Planar formulae and their uses. *SIAM Journal of Computing*, 11(2):329–393, May 1982.

[Lingas and Soltan 1996] A. Lingas and V. Soltan. Minimum convex partition of a polygon with holes by cuts in given directions. *Lecture Notes in Computer Science*, 1178:315–325, 1996.

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

[Liou *et al.* 1989] W. Liou, J. Tan, and R. Lee. Minimum partitioning simple rectilinear polygons in o($n \log \log n$)-time. In *Proceedings of the Fifth ACM Symposium on Computational Geometry*, pages 344–353, Saarbruchen, 1989.

[Liow and Pavlidis 1990] Yuh-Tay Liow and Theo Pavlidis. Use of shadows for extracting buildings in aerial images. *Computer Vision, Graphics, and Image Processing*, 49(2):242–277, February 1990.

[Lu *et al.* 1998] B-K. Lu, F-R. Hsu, and C. Y. Tang. Guarding in a simple polygon. In Mike Soss, editor, *Proceedings of the 10th Canadian Conference on Computational Geometry*, pages 46–47, Montréal, Québec, Canada, 1998. School of Computer Science, McGill University.

[Lubiw 1990] Anna Lubiw. The Boolean basis problem and how to cover some polygons by rectangles. *SIAM Journal on Discrete Mathematics*, 3(1):98–115, February 1990.

[Manber 1988] U. Manber. *Introduction to Algorithms: A Creative Approach*. Addison Wesley, Reading, MA, 1988.

[Masek] W. J. Masek. Some NP-Complete set covering problems. Referenced in Garey and Johnson, 1979, p. 232.

[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.

[Mitchell 1990] W. J. Mitchell. *The Logic of Architecture: Design, Computation and Cognition*. The MIT Press, Cambridge, Massachusetts, 1990.

[Motwani *et al.* 1990a] R. Motwani, A. Raghunathan, and H. Saran. Covering orthogonal polygons with star polygons: The perfect graph approach. *Journal of Computer and System Sciences*, 40(1):19–48, February 1990.

[Motwani *et al.* 1990b] R. Motwani, A. Raghunathan, and H. Saran. Perfect graphs and orthogonally convex covers. *Journal of Computer and System Sciences*, 40(1):19–48, February 1990.

[Ntafos and Tsoukalas 1994] S. Ntafos and M. Tsoukalas. Optimum placement of guards. *Information sciences*, 76:141–150, 1994.

[Ntafos 1986] S. Ntafos. On gallery watchman in grids. *Information Processing Letters*, 23:99–102, 1986.

[O'Rourke and Supowit 1983] J. O'Rourke and K. J. Supowit. Some NP-Hard polygon decomposition problems. *IEEE Transactions on Information Theory*, 29:181–190, 1983.

[O'Rourke 1983] J. O'Rourke. Galleries need fewer mobile guards: A variation on Chvátal's theorem. *Geometriae Dedicata*, 14:273–283, 1983.

[O'Rourke 1987] J. O'Rourke. *Art Gallery Theorems and Algorithms*. Number 3 in The International Series of Monographs on Computer Science. Oxford University Press, New York, 1987.

[O'Rourke 1993] J. O'Rourke. *Computational geometry in C*. Cambridge University Press, Cambridge, 1993. 1995 printing.

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

[O'Rourke 1999] J. O'Rourke. Personal communication, 1999. At the 11th Canadian Conference on Computational Geometry.

[Papadimitriou 1994] C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley Publishing Company, Reading, Massachusetts, 1994.

[Park *et al.* 1993] J-H. Park, S. Y. Shin, K-Y. Chwa, and T. C. Woo. On the number of guard edges of a polygon. *Discrete & Computational Geometry*, 10:447–462, 1993.

[Pavlidis and Horowitz 1974] T. Pavlidis and L. Horowitz. Segmentation of plane curves. *IEEE Transactions on Computers*, C-23(8):860–869, August 1974.

[Pellegrini and Shor 1992] M. Pellegrini and P. W. Shor. Finding stabbing lines in 3-space. *Discrete & Computational Geometry*, 8:191–208, 1992.

[Pellegrini 1993] M. Pellegrini. Lower bounds on stabbing lines in 3-space. *Computational Geometry: Theory and Applications*, 3:53–58, 1993.

[Pellegrini 1997] M. Pellegrini. Ray shooting and lines in space. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, pages 467–479. CRC Press, Boca Raton, 1997.

[Perez and Vidal 1994] J.-C. Perez and E. Vidal. Optimum polygonal approximation of digitized curves. *Pattern Recognition Letters*, 15:743–750, 1994.

[Phillips 2001] R. Phillips. Special cases for axial line placement in orthogonal rectangles. Honours Research Report, School of Computer Science, University of the Witwatersrand, 2001.

[Preparata and Shamos 1985] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction.* Texts and Monographs in Computer Science. Springer-Verlag, New York, 1985.

[Ramer 1972] U. Ramer. An iterative procedure for the polygonal approximation of plane curves. *Computer Graphics and Image processing*, 1:244–256, 1972.

[Rinsma *et al.* 1990] I. Rinsma, J. W. Giffin, and D. F. Robinson. Orthogonal floorplans from maximal planar graphs. *Environment and Planning B: Planning and Design*, 17:57–71, 1990.

[Ruskin 1997] S. G. Ruskin. Polygonal approximation algorithms with application to aerial photographs. Master's thesis, Department of Computer Science, University of the Witwatersrand, 1997.

[Russ 1999] J. C. Russ. *The Image processing handbook.* CRC Press, Boca Raton, Florida, 1999.

[Sack and Suri 1990] J.-R. Sack and S Suri. An optimal algorithm for detecting weak visibility of a polygon. *IEEE Transactions on computers*, 39(10):1213–1219, October 1990.

[Sanders and Kenny 2001a] I. D. Sanders and L-A. Kenny. Heuristics for placing non-orthogonal axial lines to cross the adjacencies between orthogonal rectangles. In *Abstracts for the Thirteenth Canadian Conference on Computational Geometry*, pages 153–156. University of Waterloo, August 2001.

[Sanders and Kenny 2001b] I. D. 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, August 2001.

[Sanders *et al.* 1995] I. D. Sanders, D. J. Lubinsky, and M. Sears. Ray guarding configurations of adjacent rectangles. In *Proceedings of the 25th Annual Southern African Computer Lecturers' Association Conference*, pages 104–116. Rhodes University, July 1995.

[Sanders *et al.* 1997] I. D. Sanders, D. J. Lubinsky, and M. Sears. Ray guarding configurations of adjacent rectangles. In *Proceedings of The 1997 National Research and Development Conference (SAICSIT 97)*, pages 221–238. Potchefstroom University for Higher Christian Education, November 1997. This paper was subsequently submitted to the South African Computer Journal.

[Sanders *et al.* 1999] I. D. Sanders, D. J. Lubinsky, M. Sears, and D. Kourie. Orthogonal ray guarding of adjacencies between orthogonal rectangles. *South African Computer Journal*, (23):18–29, 1999.

[Sanders *et al.* 2000a] I. D. Sanders, D. C. Watts, and A. Hall. Orthogonal axial line placement in chains and trees of orthogonal rectangles. Technical Report TR-Wits-CS-2000-8, Department of Computer Science, University of the Witwatersrand, June 2000.

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

[Sanders 1998a] I. D. Sanders. Non-orthogonal ray guarding. In *Proceedings of the 28th Annual SACLA Conference*, pages 133–137. Stellenbosch University, June 1998. Work in progress.

[Sanders 1998b] I. D. Sanders. Non-orthogonal ray guarding. In *Proceedings of the 1998 SAICSIT Research and Development Symposium*, pages 230–235. South African Institute of Computer Scientists and Information Technologists, November 1998. Work in progress.

[Sanders 1999] I. D. Sanders. Non-orthogonal ray guarding. In J. Snoeyink, editor, *Abstracts for the Eleventh Canadian Conference on Computational Geometry*, pages 80–83. University of British Columbia, Vancouver, August 1999. The full paper is available in the electronic proceedings – `http://www.cs.ubc.ca/conferences/CCCG/elec_proc/elecproc.html`.

[Sanders 2000] I. D. Sanders. Placing axial lines in urban grids. *South African Computer Journal*, (26):145–153, 2000. This issue of SACJ is a Special Issue which constitutes the *Proceedings of the 2000 SAICSIT Research and Development Symposium*, Cape Town, 1-3 November 2000. Included in this Special Issue are Research articles, Experience papers and New Ideas papers. This is a Research article.

[Sarkar 1993] D. Sarkar. A simple algorithm for detection of significant vertices for polygonal approximation of chain-coded curves. *Pattern Recognition Letters*, 14:959–964, 1993.

[Schachter 1978] B. Schachter. Decomposition of polygons into convex sets. *IEEE Transactions on computers*, C-27(11):1079–1082, November 1978.

[Scott-Dawkins 2001] D. Scott-Dawkins. Genetic algorithm for the ALP-ALOR problem. Honours Research Report, School of Computer Science, University of the Witwatersrand, 2001.

[Shermer 1989] T. C. Shermer. Hiding people in polygons. *Computing*, 42(2-3):109–131, 1989.

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

[Soares 1997] R. Soares. Guarding rectilinear polygons with holes using vertex guards. Honours Research Report, School of Computer Science, University of the Witwatersrand, 1997.

[Soltan and Gorpinevich 1993] V. Soltan and A. Gorpinevich. Minimum dissection of a rectilinear polygon with arbitrary holes into rectagles. *Discrete and Computational Geometry*, 9:57–79, 1993.

[Soltész 2000] É. Soltész. The axial line placement problem in deformed urban grids. Honours Research Report, School of Computer Science, University of the Witwatersrand, 2000.

[Srinivasaraghavan and Mukhopadhyay 1993] G. Srinivasaraghavan and A. Mukhopadhyay. On the notion of completeness for reconstruction algorithms on visibility graphs. In *Proceedings of the 5th Canadian Conference on Computational Geometry*, pages 315–320, Waterloo, ON, Canada, August 1993. University of Waterloo.

[Stilla *et al.* 1996] U. Stilla, E. Michaelsen, and K. Luetjen. Automatic extraction of buildings from aerial images. In F. Leberl, R. Kalliany, and M. Gruber, editors, *Methods for extracting and mapping buildings, roads and other man-made structures from images*. Oldenburg, 1996.

[Subramaniyam and Diwan 1991] R. V. Subramaniyam and A. A. Diwan. A counterexample for the sufficiency of edge guards in star polygons. *Information Processing Letters*, 40:97–99, 1991.

[Sugihara *et al.* 1990] K. Sugihara, I. Suzuki, and M. Yamashita. The searchlight scheduling problem. *SIAM Journal on Computing*, 19(6):1024–1040, December 1990.

[Suri 1997] S. Suri. Polygons. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, pages 429–444. CRC Press, Boca Raton, 1997.

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

[Ton *et al.* 1991] J. Ton, J. Sticklen, and A. K. Jain. Knowledge-based segmentation of landsat images. *IEEE Transactions on Geoscience and Remote Sensing*, 29(2):222–232, March 1991.

[Urrutia 1999] J. Urrutia. Art gallery and illumination problems. In J.-R. Sack and J. Urrutia, editors, *Handbook on Computational Geometry*, chapter 22. Elsevier Science, 1999.

[Venkatasubramanian and Cullum 1993] R. Venkatasubramanian and A. Cullum. Grazing inside a convex polygon. In *Proceedings of the Fifth Canadian Conference on Computational Geometry*, pages 228–233, Waterloo, ON, Canada, August 1993. University of Waterloo.

[Viswanathan 1993] S. Viswanathan. The edge guard problem for spiral polygons. In *Proceedings of the Fifth Canadian Conference on Computational Geometry*, pages 103–108, Waterloo, ON, Canada, August 1993. University of Waterloo.

[Watts and Sanders 1997] D. C. Watts and I. D. Sanders. Ray guarding a chain of adjacent rectangles. Technical Report TR-Wits-CS-1997-02, University of the Witwatersrand, 1997.

[Watts 1997] D. C. Watts. Ray guarding chains of rectangles. Honours Research Report, School of Computer Science, University of the Witwatersrand, 1997.

[Wilson 1997] D. Wilson. Guard placement in orthogonal towns. Honours Research Report, School of Computer Science, University of the Witwatersrand, 1997.

[Wood and Yamamoto 1993] D. Wood and P. Yamamoto. Dent and staircase visibility. In *Proceedings of the Fifth Canadian Conference on Computational Geometry*, pages 297–302, Waterloo, ON, Canada, August 1993. University of Waterloo.

[Wood 1985] D. Wood. An isothetic view of computational geometry. In G. T. Toussaint, editor, *Computational Geometry*, pages 429–459. North-Holland, 1985.

[Zarganakis 1997] D. Zarganakis. Lowering the upper bound for guarding general polygons with and without holes. Honours Research Report, School of Computer Science, University of the Witwatersrand, 1997.

[Zhu and Seneviratne 1997] Y. Zhu and L. Seneviratne. Optimal polygonal approximation of digitised curves. *IEE Proceedings. Vision, Image and Signal Processing*, 144(1):8–14, 1997.