The South African Institute of Computer Science and Information Technology

The 1997 National Research and Development Conference

Riverside Sun
Vanderbijlpark
13 & 14 November

Hosted by

Potchefstroomse Universiteit
vir Christelike Hoër Onderwys

The Department of Computer Science and Information Systems
Potchefstroom University for Christian Higher Education
Vaal Triangle Campus

PROCEEDINGS

Edited by L.M. Venter & R.R. Lombard
The South African Institute of Computer Science
and
Information Technology

Proceedings
of the
The 1997 National Research and Development Conference
Towards 2000

Riverside Sun
Vanderbijlpark
13 & 14 November

Edited by
L.M. Venter
R.R. Lombard
©1997 Copyrights reside with the original authors who may be contacted directly

**ISBN 1-86822-300-0**

Printed and Binded by Xerox Printers, Potchefstroom

The views expressed in this book are those of the individual authors
Foreword

This book contains a collection of papers presented at a Research and Development conference of the South African Institute of Computer Scientists and Information Technologists (SAICSIT). The conference was held on 13 & 14 November 1997 at the Riverside Sun, Vanderbijlpark. Most of the organization for the conference was done by the Department of Computer Science and Information Technology of the Vaal Triangle Campus, Potchefstroom University for Christian Higher Education.

The programming committee accepted a wide selection of papers for the conference. The papers range from detailed technical research work to reports of work in progress. The papers originate mainly from Academia, but also describe work done in and for Industry. It is hoped that the papers give a true reflection of the current research scene in Computer Science and Information Technology in South Africa. Since one of the aims of the conference is Research development, the papers were not subjected to a refereeing process.

A number of people spent numerous hours helping with the organization of this conference. In this regard, we wish to thank the members of the Organizing committee, and the Programming committee who had very little time to screen the abstracts and compile the program. A special thanks goes to the secretary of the department, Mrs Helei Jooste, whose very able work was interrupted by the birth of her first child.
Organizing Committee

Conference General Chairs
Prof. J.M. Hattingh (PU for CHE)

Organizing Chair
Prof. Lucas Venter (PU for CHE)

Organizing Committee
Mrs. S. Gilliland
Mr. J.P. Jooste
Mr. R.R. Lombard
Mrs. M. Huisman

Secretariat
Mrs. H. Jooste

Program Chair
Prof. A de Waal (PU for CHE)

Program Committee
Prof. D. Kourie (UP)
Prof. C. Bornman (UNISA)
Prof. L.M. Venter (PU for CHE)
# Table of Contents

<table>
<thead>
<tr>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Foreword</td>
<td>i</td>
</tr>
<tr>
<td>Organizing Committee</td>
<td>ii</td>
</tr>
<tr>
<td>List of Contributors</td>
<td>vii</td>
</tr>
<tr>
<td><strong>Software Objects Change: Problems and Solution</strong></td>
<td>1</td>
</tr>
<tr>
<td>S.A. Ajila</td>
<td></td>
</tr>
<tr>
<td><strong>Liming-like Curve Constructions</strong></td>
<td>26</td>
</tr>
<tr>
<td>M.L. Baart and R. McLeod</td>
<td></td>
</tr>
<tr>
<td><strong>A Model for Evaluating Information Security</strong></td>
<td>27</td>
</tr>
<tr>
<td>L. Barnard and R. von Solms</td>
<td></td>
</tr>
<tr>
<td><strong>Integrating Spatial Data Management and Object Store Technology</strong></td>
<td>31</td>
</tr>
<tr>
<td>S. Berman, S. Buffler and E. Voges</td>
<td></td>
</tr>
<tr>
<td><strong>Metamodelling in Automated Software Engineering</strong></td>
<td>32</td>
</tr>
<tr>
<td>S. Berman and R. Figueira</td>
<td></td>
</tr>
<tr>
<td><strong>Using Multimedia Technology for Social Upliftment in Deprived Communities of Southern Africa</strong></td>
<td>33</td>
</tr>
<tr>
<td>L. Bester and E. de Preez</td>
<td></td>
</tr>
<tr>
<td><strong>Extending the Client-Server Model for Web-based Execution of Applications</strong></td>
<td>36</td>
</tr>
<tr>
<td>L. Botha, J.M. Bishop and N.B. Serbedzija</td>
<td></td>
</tr>
<tr>
<td><strong>Access Control Needs in an Electronic Workflow Environment</strong></td>
<td>45</td>
</tr>
<tr>
<td>R.A. Botha</td>
<td></td>
</tr>
<tr>
<td><strong>The Use of the Internet in an Academic Environment to Commercially Supply and Support Software Products</strong></td>
<td>51</td>
</tr>
<tr>
<td>B. Braude and A.J. Walker</td>
<td></td>
</tr>
<tr>
<td><strong>Explanation Facilities in Expert Systems Using Hypertext Technology</strong></td>
<td>63</td>
</tr>
<tr>
<td>T. Breetzke and T. Thomas</td>
<td></td>
</tr>
<tr>
<td><strong>Theoretical Computer Science: What is it all about, and is it of any relevance to us?</strong></td>
<td>75</td>
</tr>
<tr>
<td>C. Brink</td>
<td></td>
</tr>
<tr>
<td><strong>Representing Quadrics on a Computer</strong></td>
<td>76</td>
</tr>
<tr>
<td>M.A. Coetzee and M.L. Baart</td>
<td></td>
</tr>
<tr>
<td>Title</td>
<td>Page</td>
</tr>
<tr>
<td>----------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>The Generation of Pre-Interpretations for Detecting Unsolvable Planning Problems</td>
<td>77</td>
</tr>
<tr>
<td>D.A. de Waal, M. Denecker, M. Bruynooghe and M. Thielscher</td>
<td></td>
</tr>
<tr>
<td>The Emerging Role of the Chief Information Officer in South Africa</td>
<td>87</td>
</tr>
<tr>
<td>B. Dekenah</td>
<td></td>
</tr>
<tr>
<td>A Java-Implemented Remote Respiratory Disease Diagnosis System on a High Bandwidth Network</td>
<td>88</td>
</tr>
<tr>
<td>A. Foster</td>
<td></td>
</tr>
<tr>
<td>Early Results of a Comparative Evaluation of ISO 9001 and ISO/IEC 15504 Assessment Methods Applied to a Software Project</td>
<td>89</td>
</tr>
<tr>
<td>C. Gee and A.J. Walker</td>
<td></td>
</tr>
<tr>
<td>A Neural Network Model of a Fluidised Bed</td>
<td>99</td>
</tr>
<tr>
<td>M. Hajek</td>
<td></td>
</tr>
<tr>
<td>The Effects of Virtual Banking on the South African Banking Industry</td>
<td>100</td>
</tr>
<tr>
<td>M.L. Hart and M. Dunley-Owen</td>
<td></td>
</tr>
<tr>
<td>Linear Response Surface Analysis and Some Applications</td>
<td>118</td>
</tr>
<tr>
<td>J.M. Hattingh</td>
<td></td>
</tr>
<tr>
<td>Model Checking Software with Symbolic Trajectory Evaluation</td>
<td>120</td>
</tr>
<tr>
<td>A. Hazelhurst</td>
<td></td>
</tr>
<tr>
<td>A Risk Model to Allocate Resources to Different Computerized Systems</td>
<td>137</td>
</tr>
<tr>
<td>H.A. Kruger and J.M. Hattingh</td>
<td></td>
</tr>
<tr>
<td>Returns on the Stock Exchange</td>
<td>144</td>
</tr>
<tr>
<td>J.W. Kruger</td>
<td></td>
</tr>
<tr>
<td>Cardinality Constrained 0-1 Knapsack Problems</td>
<td>150</td>
</tr>
<tr>
<td>M.F. Kruger, J.M. Hattingh and T. Steyn</td>
<td></td>
</tr>
<tr>
<td>An Investigation in Software Process Improvement in the Software Development of a large Electricity Utility</td>
<td>151</td>
</tr>
<tr>
<td>M. Lang and A.J. Walker</td>
<td></td>
</tr>
<tr>
<td>Design and Implementation of a C++ Package for Two-Dimensional Numerical Integration</td>
<td>162</td>
</tr>
<tr>
<td>D.P. Laurie, L Pluym and Ronald Cools</td>
<td></td>
</tr>
<tr>
<td>Algebraic Factorization of Integers Using BDE's</td>
<td>169</td>
</tr>
<tr>
<td>H. Messerschmidt and J. Robertson</td>
<td></td>
</tr>
</tbody>
</table>
Global Optimization of Routes after the Process of Recovery
M. Mphahlele and J. Roos

Using a Lattice to Enhance Adaptation Guided Retrieval in Example Based Machine Translation
G.D. Oosthuizen and S.L. Serutla

Information Systems Development and Multi Criteria Decision Making / Systems Thinking
D. Petkov, O. Petkova

The Development of a Tutoring System to Assist Students to Develop Answering Techniques
N Pillay

Combining Rule-Based Artificial Intelligence with Geographic Information Systems to Plan the Physical Layer of Wireless Networks in Greenfield Areas
K. Prag, P. Premjeeth and K. Sandrasegaran

A Distributed Approach to the Scheduling Problem
V. Ram and P. Warren

More readings than I thought: Quantifier Interaction in Analysing the Temporal Structure of Repeated Eventualities
S. Rock

Ray Guarding Configuration of Adjacent Rectangles
I. Sanders, D. Lubinsky and M. Sears

Developing Soft Skills in Computer Students
C Schroder, T. Thomas

Information Security Awareness, a Must for Every Organization
M. Thomson and R. von Solms

Pla Va: A Lightweight Persistent Java Virtual Machine
S Tjasink and S. Berman

Beliefs on Resource-Bounded Agent
E. Viljoen

Object-Orientated Business Modelling and Re-engineering
M. Watzenboeck
On Indexing in Case Based Reasoning Applied to Pre-Transportation
Decision Making for Hazardous Waste Handling
K.L. Wortmann, D. Petkov and E Senior

Author Index
## List of Contributors

<table>
<thead>
<tr>
<th>Name</th>
<th>Institution</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>S.A. Ajila</td>
<td>Department of Mathematics and Computer Science</td>
<td>Roma, 180, Lesotho</td>
</tr>
<tr>
<td>L. Baart</td>
<td>Department of Mathematics</td>
<td>PO Box 1174</td>
</tr>
<tr>
<td>L. Barnard</td>
<td>Faculty of Computer Studies</td>
<td>Vanderbijpark, 1900</td>
</tr>
<tr>
<td>S. Berman</td>
<td>University of Cape Town</td>
<td>Rondebosch, 7701</td>
</tr>
<tr>
<td>L. Bester</td>
<td>Faculty of Computer Studies</td>
<td>Port Elizabeth, 6000</td>
</tr>
<tr>
<td>J.M. Bishop</td>
<td>Computer Science Department</td>
<td>Pretoria, 0002</td>
</tr>
<tr>
<td>L. Botha</td>
<td>Computer Science Department</td>
<td>Pretoria, 0002</td>
</tr>
<tr>
<td>R.A. Botha</td>
<td>Faculty of Computer Studies</td>
<td>Port Elizabeth, 6000</td>
</tr>
<tr>
<td>B. Braude</td>
<td>Software Engineering Applications Laboratory</td>
<td>University of Wits</td>
</tr>
<tr>
<td>T. Breetzke</td>
<td>Faculty of Computer Studies</td>
<td>Private Bag 3</td>
</tr>
<tr>
<td>C. Brink</td>
<td>University of Cape Town</td>
<td>Rondebosch, 7700</td>
</tr>
<tr>
<td>M. Bruynooghe</td>
<td>Departement Computerwetenschappen</td>
<td>Katholieke Universiteit Leuven Celestijnenlaan 200A B-3001 Heverlee Belgium</td>
</tr>
<tr>
<td>M.A. Coetzee</td>
<td>Department of Mathematics</td>
<td>Private Bag X6001</td>
</tr>
<tr>
<td>R. Cools</td>
<td>Katholieke Universiteit Leuven</td>
<td>Private Bag X6001</td>
</tr>
<tr>
<td>E. de Preez</td>
<td>Faculty of Computer Studies</td>
<td>Private Bag X6001</td>
</tr>
<tr>
<td>D.A. De Waal</td>
<td>Department of Computer Science and Information Systems</td>
<td>Private Bag X6001</td>
</tr>
<tr>
<td>M. Denecker</td>
<td>Departement Computerwetenschappen</td>
<td>Private Bag X6001</td>
</tr>
<tr>
<td>M. Dunley-Owen</td>
<td>Department of Information Systems</td>
<td>Private Bag X6001</td>
</tr>
<tr>
<td>R. Fiqueira</td>
<td>University of Cape Town</td>
<td>Private Bag X6001</td>
</tr>
<tr>
<td>A. Foster</td>
<td>Department of Computer Science</td>
<td>Private Bag X6001</td>
</tr>
<tr>
<td>C. Gee</td>
<td>Software Engineering Applications Laboratory</td>
<td>Private Bag X6001</td>
</tr>
</tbody>
</table>

vii
Ray Guarding Configurations of Adjacent Rectangles

Ian Sanders * David Lubinsky † Michael Sears ‡

October 9, 1997

Abstract

Guarding and covering problems have great importance in Computational Geometry. In this paper the notion of a ray guard, a guard that can only 'see' along a single ray, is introduced. The problem of siting the fewest possible such guards so that they guard all adjacencies in an orthogonal arrangement of adjacent non-overlapping rectangles is discussed. The problem is further restricted by requiring that the direction of sight be parallel to an axis and that the guards cannot 'see' outside the rectangles. The problem is motivated by applications in architecture and urban planning. This paper shows that the problem is NP-Complete because of the locally indeterminate choice which can be introduced in positioning guards. A heuristic algorithm to produce a non-redundant set of guards is then presented. Given n rectangles, the algorithm runs in O(n^2) time and O(n^2) space and proves to be a good approximation in a number of test cases.

1 The Problem

1.1 Background

Guarding and covering problems are common in the field of Computational Geometry (see O'Rourke's monograph [15] for example). This paper introduces a new variation on guarding problems where the guards can only see along a single ray. The problem has its origin in the area of town planning and urban design — Hillier et al.'s idea of Space Syntax Analysis [12]. The idea of Space Syntax Analysis is to give a globalising perspective of the design by determining how easy it is to traverse the town. This analysis is accomplished by the positioning of axial lines on a town plan — the fewest such lines are required.

The problem is similar to the many art-gallery guarding problems [2, 10, 4] since the lines can be thought of as guards whose vision is restricted to a single ray. The situation can be envisaged as an art gallery made up of a number of adjacent rooms where the designers wish to position the most doors between rooms (to allow easy access) in such a fashion that all doorways can be guarded by the minimum number of ray guards.

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.

This paper considers only the problem where the rays are restricted to being parallel to the Euclidean axes and the space cut by the lines is a collection of adjacent orthogonal rectangles. The problem is presented below — Section 1.2.

1.2 Statement of the Problem

Given a number of adjacent orthogonally aligned rectangles find the fewest maximal orthogonal straight line segments contained wholly inside the rectangles which will pass through all of the shared boundaries (adjacencies) between adjacent rectangles.

*Department of Computer Science, University of the Witwatersrand, Johannesburg, E-mail: ian@cs.wits.ac.za
†Department of Computer Science, University of the Witwatersrand, Johannesburg, E-mail: david@cs.wits.ac.za
‡Department of Mathematics, University of the Witwatersrand, Johannesburg, E-mail: sears@gauss.cam.wits.ac.za
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 cut at least once.

2. Any adjacency has exactly one orthogonal line passing through it.

![Figure 1: Solutions to problems 1 and 2 for a simple configuration](image)

Figure 1 shows the difference between these two specifications for a simple configuration. In this paper only problem 1 is addressed.

1.3 Related Work

The problem of covering a polygon with simpler polygons has been the subject of much research [18, 8, 5, 6, 7] and many of the problems in this class have been shown to be NP-Hard. One of these problems is that of finding the minimum number of star polygons needed to cover a given polygon, this problem is equivalent to the placement of the minimum number of point guards so that each point inside the polygon is visible to some guard. This problem is intractable for polygons with holes and remains so for simple polygons. Other guarding problems are discussed by Györi et al [11], Shermer [17], Bjorling-Sachs [1] and Bjorling-Sachs and Souvaine [3, 4] among others. In a related area Gewali and Ntafos [10] present an $O(n^3)$ algorithm for finding periscope guards (able to see around at most one bend) for simple grids and show that this algorithm can be applied to obtain minimum periscope guard covers for simple orthogonal polygons with a constant number of turret corners.

1.4 The layout of the paper

In the next section of the paper the terminology used is given. Section 3 discusses the problem in somewhat more detail and introduces the issue of choice. Section 4 shows that the problem is NP-Complete by means of a transformation from vertex cover. The remaining sections of the paper discuss an algorithm which produces a non-redundant set of lines to cover all the adjacencies between rectangles.

2 Terminology

In the remainder of this paper the common definitions in Computational Geometry are used. The definitions given below have particular relevance to this paper.

- A line is orthogonal, or orthogonally aligned, if it is parallel to one of the Cartesian axes. Line segments are simply referred to as lines.
If all of the edges of a rectangle are orthogonally aligned then this rectangle is referred to as an orthogonal rectangle.

Edges of rectangles (or portions of edges) which are coincident are called adjacencies. Only vertical adjacencies (horizontal lines) are considered in this chapter — the problem for horizontal adjacencies is similar.

A line is defined by the set of adjacencies between rectangles which are cut by an orthogonal line through these rectangles.

A line must be wholly contained in the collection of rectangles.

A line which is maximal in length is an orthogonal line such that if any other orthogonal line includes the same set of adjacencies then it includes no other adjacencies (see Figure 1).

Note that throughout this chapter any mention to rectangles should be taken to mean orthogonal rectangles.

3 Addressing the problem

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

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 which gives $e \leq 3v - 6$ ($e$ the number of edges and $v$ the number of vertices). This implies that the number of adjacencies must be $O(n)$. The question is then: How easy is it to find the minimum number of maximal lines which cover all of the adjacencies?

The problem is interesting and difficult to solve efficiently because of the issue of choice. The simplest case of choice is illustrated in Figure 2. In this case there are seven adjacencies which must be crossed by straight lines. It is easy to see that most of the adjacencies can be cut by the lines marked $a$ and $b$ (0-3-5-6 and 2-3-5-6) but the adjacency between rectangles 1 and 3 can be cut by lines $c$ (1-3-4-6) and $d$ (1-3-5-6). Only one of these “choice” 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 conflict of this type.
4 Proving NP-Completeness of the problem of finding choice lines

The presence of choice lines means that all possible choices will have to be considered to obtain an exact solution to the problem. This section shows that because of this choice the problem is in fact NP-Complete. The proof of this will be accomplished through a transformation from VERTEX COVER for a planar graph [9, 14], to a restricted instance of the problem under consideration, i.e. the problem of choosing the fewest maximal lines to cover all the adjacencies in a collection of rectangles.

The problem to be considered can be stated as below.

ORTHOGONAL LINES

Instance: A collection of orthogonal rectangles \( R_1 \ldots R_m \), 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 lines where each line is maximal in length, each line is contained wholly within the rectangles, each vertical adjacency is crossed at least once by the lines in \( P \) and \( | P | \leq O \)?

The transformation from PLANAR VERTEX COVER [9, 14] will be done by mapping vertices in a planar graph to choice lines in the problem being considered. Edges in the planar graph will be mapped to adjacencies which are covered by the choice lines. In this mapping an edge between two vertices represents an adjacency which is covered by two choice lines.

Before discussing the transformation, it is desirable to prove one other result — vertex cover for a biconnected planar graph\(^1\) is NP-complete. Once it has been shown that this result holds the transformation from BICONNECTED PLANAR VERTEX COVER to ORTHOGONAL LINES can be done more easily. This result is addressed in Lemma 4.1.

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 \( V' \)?

BICONNECTED PLANAR VERTEX COVER

Instance: Planar Biconnected 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 \( V' \)?

Lemma 4.1 BICONNECTED PLANAR VERTEX COVER is NP-Complete

Proof

Clearly BICONNECTED PLANAR VERTEX COVER is in NP.

Now transform PLANAR VERTEX COVER to BICONNECTED PLANAR VERTEX COVER.

Given a planar graph \( G(E, V) \), let \( R_0 \) be the infinite region of the graph and let \( R_1 \ldots R_n \) be the regions of the graph defined by some planar embedding of \( G \). Let \( L_j \) be the set of vertices of degree 1 contained in \( R_j \) and let \( C_j \) be the set of vertices which form the cycle enclosing \( R_j \). Let \( J = \{ j \mid L_j \) is non-empty \}. Define a graph \( S \) which consists of the vertices \( s, r, m_1, m_2 \) and \( m_3 \) and edges from \( s \) to each \( m_i \) and from each \( m_i \) to \( r \), \( i = 1, \ldots, 3 \). An example of one of these graphs \( S \) is shown in Figure 3.

If the graph \( G \) is a tree (the only region in the graph is the infinite region \( R_0 \)) then construct a graph \( G'(E', V') \) by adding a copy of the graph \( S \) to \( G \) and connecting every vertex in \( L_0 \) to the \( s \) vertex of \( S \). The resulting graph \( G'(E', V') \) (\( E' = E \cup L_0 \times \{ s \}, V' = V \cup \{ s, r, m_1, m_2, m_3 \} \)) is biconnected.

If \( G \) is not a tree, construct a graph \( G'(E', V') \) by inserting an instance of the graph \( S \) into each region \( R_j \) where \( j \in J \). Call this instance \( S_j \). Let the \( s \) vertex of \( S_j \) be \( s_j \), the \( r \) vertex of \( S_j \) be \( r_j \) and

\( ^1\)Edge connectivity of a graph is the minimum number of edges whose removal results in a disconnected or trivial graph. A graph \( G \) is \( k \)-connected or \( k \)-edge connected if the edge connectivity of \( G \) is at least \( k \)
Figure 3: An example of the graph $S$

Figure 4: An example of adding an $S$ graph to a region

the $m_i$ vertices of $S_j$ be $m_{ij}$. Add the edges $E_j = L_j \times \{s_j\}$ and the edge $a_j$ connecting $s_j$ to some vertex $c_j \in C_j$ to $G'$. If $L_0$ is non-empty then add a copy $S_0$ of $S$ to the infinite region and connect every vertex of $L_0$ to the $s$ vertex of $S_0$ and then connect the $s$ vertex of $s_0$ to any vertex which is in a cycle which is adjacent to the infinite region. The resulting graph $G' = (E', V') = (E \cup \bigcup_j E_j \cup a_j, V' = V \cup \bigcup_j \{s_j, r_j, m_j, m_{j1}, m_{j2}, \}, j \in J)$ is biconnected. This follows because the graph $s_j$ is biconnected, the cycle enclosing $R_j$ is biconnected and each edge on a path to the vertices in $L_j$ is connected to the vertices in $C_j$ by two paths — one including the edge $a_j$ and one not including $a_j$. Figure 4 shows an example of adding a copy of the graph $S$ to a region. Suppose there were $k$ non-empty sets $L_j$, then $k$ $S$ graphs ($5k$ vertices and at most $2|V| + 6k$ new edges) have been added.

The graph $G$ is a planar graph, each of the graphs $S_j$ is a planar graph and each $S_j$ is added inside a region $R_j$ of the graph $G$ where there are vertices of degree 1. The edges $E_j$ connect the vertices of degree 1 in region $R_j$ to the vertex $s_j$ of $S_j$. The edges in $E_j$ do not leave the region $R_j$, in addition they can be added in such a fashion that no edge in $E_j$ crosses another edge in $E_j$. In addition $a_j$ can be added without crossing any edge in $E_j$. $G'$ is thus also a planar graph.

To determine the vertex cover for $G'$, the original graph $G$ plus the new edges and vertices must be considered. The structure of the special graphs means that each $s_j$ and $r_j$, $j \in J$, must be in the vertex cover of $G'$. This is because this choice of vertices covers all of the edges in $S_j$ using only two vertices. If either of these vertices were not in the vertex cover then all of the three $m_{ij}$ vertices would have to be in the vertex cover to cover all of the 'interior' edges of $S_j$. This choice also covers all of the new edges, $E_n$ and $a_j$, added in the construction of $G'$ from $G$. The rest of the vertex cover of $G'$ is chosen only from vertices which were in the original graph $G$ and it follows that the vertex cover of $G'$ is the vertex cover of $G$ plus the vertices $s_j$ and $r_j$, $j \in J$. The vertex cover of $G'$ is the $K$ vertices of the vertex cover of $G$ plus the $2k$ vertices $s_j$ and $r_j$, $j \in J$. Thus PLANAR VERTEX COVER, $G$, $K$, is true if and only if BICONNECTED PLANAR VERTEX COVER is true for $K' = K + 2k$.

The transformation from $G$ to $G'$ can clearly be accomplished in polynomial time. For each cycle in the original graph at most one graph of the form $S$ is added. Each vertex of degree 1 in the original
BICONNECTED PLANAR VERTEX COVER can now be used to show that ORTHOGONAL LINES is NP-Complete. This will be done in two steps. First, a biconnected planar graph is transformed to a 'stick diagram'. In this 'stick diagram' a horizontal line representing a choice line is added for each vertex in the original graph and each edge in the original graph is mapped to a vertical line which is cut 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 cut all of the vertical lines.

STICK DIAGRAM

**Instance:** A collection $S$ of $H$ horizontal lines and $V$ vertical lines such that each vertical line is cut by exactly two horizontal lines and a positive integer $K \leq H$.

**Question:** Is there a set of horizontal lines of size $K$ or less in $S$ such that every vertical line in $S$ is cut at least once?

Second, the stick diagram is represented as a collection of adjacent rectangles and horizontal lines cutting all of the adjacencies in the collection of rectangles. These horizontal lines will be of two types "essential lines" which are the only lines to cut a particular adjacency and "choice lines" where a number of lines (none of which are essential) cut the adjacency. Not all of the choice lines are necessary to cover all of the adjacencies in the collection of rectangles. If it is possible to determine in polynomial time which of the set of choice lines cover all of the adjacencies in the diagram then it is possible to solve BICONNECTED PLANAR VERTEX COVER in polynomial time — finding the set of choice lines is equivalent to finding vertex cover of the original graph.

The proof is accomplished by means of the following two theorems — 4.1 and 4.2.

**Theorem 4.1** **STICK DIAGRAM** is NP-Complete.

**Proof**

Clearly STICK DIAGRAM is in NP — given a set of horizontal lines of size less than or equal to $K$, it is possible to check in polynomial time that every vertical line in $S$ is cut at least once.

Now transform BICONNECTED PLANAR VERTEX COVER to STICK DIAGRAM. Given a biconnected planar graph $G(U, V)$, let $R_0$ be the infinite region of the graph and $R_1, \ldots, R_n$ be the interior regions of the graph. Choose $C_s$ as being any longest cycle bounding a region $R_s$ of the graph which is adjacent to the infinite region $R_0$ of the graph.

Choose any two vertices $x$ and $y$ joined by an edge $A$ which form part of $C_s$. Figure 5 (a) shows the horizontal lines $x$ and $y$ cutting the vertical line $A$.

Consider the path, $B$, formed by removing edge $A$ from $C_s$. For the moment treat $B$ as if it were a simple edge. This then gives the horizontal lines $x$ and $y$ cutting the vertical line $A$.

Now break the path $B$ (which was treated as a virtual edge) into its component edges. This will mean adding edges (vertical lines) and vertices (choice lines) to the stick diagram. 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, moving along the path $B$ from $x$ to $v_0$ the first edge encountered is $C$. The stick diagram must be altered to include this new edge, this is shown in Figure 5(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 from $x$ to $y$. 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 5 (d). A stick diagram which represents the originally selected closed region $R_s$ of the original graph has now been created. This stick diagram must now be extended for the other closed regions of the graph.

This is accomplished by selecting another closed region $R_m, 1 \leq m \leq n$ which is adjacent to $R_s$ (shares a path with $R_s$). A further restriction is that the cycle $C_m$ enclosing $R_m$ must be made up of two paths. One path of edges which have already been "visited" (included in the stick diagram) and one path $D$ of edges which have not yet been visited — because the graph $G$ is biconnected and planar this is always possible.

Again in the first instance this path $D$ can be treated as a single (simple) edge. This simple edge is added to the stick diagram. The two vertices representing the start and end of the path are on the boundary of the closed region $R_s$. They are thus accessible (can have edges connected to them) from inside and from outside of the closed region. Their representations in the stick diagram must also be accessible from inside and outside of the closed region. The representation of the new edge (adjacency) in the stick diagram is done by adding a new vertical line connecting the two horizontal lines representing the vertices. This is possible because the lines representing the vertices have one end inside the closed region and one end outside the closed region. The way the first phase of the construction was performed, the end of the lines outside of the closed region can both be extended in the same direction (if necessary) and then cut by the new vertical line. See Figure 5 (e) for an example. In this figure, a new path between $v_0$ and $v_k$ is being added.

This path can again be broken up into its constituent edges in the same fashion as the path in the first phase of the construction. Again (by a similar argument to before) any vertices and associated edges which are inside the new closed region in the original graph must be inside the new closed region in the stick diagram.

The process described above can then be repeated for all of the closed regions in the original graph.

This completes the construction of the stick diagram from a biconnected planar graph. A complete example of this is shown in Figure 6. It can be seen that the stick diagram here maintains the property of accessibility of the planar graph — vertices inside the closed region $x - y - w - v$ can only be reached from other vertices inside that closed region.
From the construction it can be seen that if a vertex cover can be found for $G$ then a set of lines can be found for $S$ — each vertex in $G$ is a horizontal line in $S$ and each edge in $G$ is a vertical line in $S$. Conversely if a set of horizontal lines of size $\leq K$ could be found to cut each vertical line in $S$, then a vertex cover for $G$ could be found.

The transformation from **BICONNECTED PLANAR VERTEX COVER** to **STICK DIAGRAM** can be accomplished in polynomial time. Each region in the graph $G$ is considered in turn and once only. As the region 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 lengthened if necessary. Each vertex can only occur in as many regions 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.

Theorem 4.1 shows that a stick diagram can be constructed for any biconnected planar graph. It is now necessary to show that a stick diagram can be represented by a collection of adjacent rectangles. In this case it is not possible just to replace every adjacency in the stick diagram with two adjacent rectangles. Two adjacent rectangles do not produce choice, the arrangement of rectangles which does produce choice is much more complicated. It is also necessary to be able to join the collections which do produce choice in such a fashion as to maintain the choice available and not introduce additional choice. This is considered in Theorem 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 cut by essential lines and choice lines. Not all of the choice lines are necessary and Theorem 4.2 also shows that the problem of choosing the minimum number of such lines in NP-Complete.

**Theorem 4.2** **ORTHOGONAL LINES** is NP-Complete

**Proof**

Clearly **ORTHOGONAL LINES** is in NP. Given a set of lines it is possible to check in polynomial time that each adjacency has been cut by at least one line.

Now transform **STICK DIAGRAM** to **ORTHOGONAL LINES**.

The rectangles which create choice lines can be represented by the canonical choice unit, shown in Figure 7. This canonical choice unit, $ccu$, is based on the concept illustrated in Figure 2 where two lines are necessary to cover the upper and lower left hand adjacencies and one of two lines could be chosen to cover the middle left hand adjacency. In this case, the lines originating in the four small rectangles and ending in the four darker shaded rectangles are enough to cover all of the adjacencies in the $ccu$ except those between the middle rectangle and the two tall rectangles bordering it. These adjacencies can be cut by two possible maximal lines, only one of which is necessary. Scaling of the canonical choice unit does not change the fact that it can/does produce choice lines.

This transformation will proceed by replacing each vertical line in the stick diagram by a $ccu$ of an appropriate size. The horizontal lines which cut through the vertical line are to be represented by the
choice lines of the ccu. It is necessary to show that these canonical choice units can be joined together in a fashion which does not limit the choice available and does not introduce additional choice. There are four ways in which horizontal lines could cut through successive vertical lines or in which choice lines could cut through successive adjacencies. These are

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

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

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

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

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. This is accomplished by making use of the darker shaded "connector" rectangles of each ccu (see Figure 7 and where appropriate making use of "connecting" rectangles). The cases discussed above are now considered in turn.

- **upper to upper**
  In this case no connecting rectangles are required. Joining the two ccus together is accomplished by merging the appropriate darker shaded rectangles. This is shown in Figure 8.

- **lower to lower**
  This is a mirror image of the case above about the x-axis.

- **upper to lower**
  Here the two ccu’s are connected by placing a single rectangle of appropriate size into the positions indicated in Figure 9 and extending the connector rectangles of the ccu. The “connecting” rectangle ensures that the upper choice line through the left ccu and the lower choice line through the
Figure 8: Joining the upper choice lines of two choice units

Figure 9: Joining the upper choice line of one unit to the lower choice line of the next unit

right ccu are the same choice line. After adding the connecting rectangle, the adjacencies in each ccu which were only cut by choice lines are still only cut by choice lines. All other adjacencies are cut by essential lines. Thus choice is maintained.

- lower to upper

Again this case is a mirror image of the case above.

If all of the vertical lines in the stick diagram are replaced by ccus, connecting rectangles are added if appropriate and the appropriate changes are made to the connector rectangles then the choice 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 10.

The transformation from STICK DIAGRAM to ORTHOGONAL LINES 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.

It is now necessary to show that there is a solution to STICK DIAGRAM if and only if there is a solution to ORTHOGONAL LINES. The construction of the collection of adjacent rectangles from the stick diagram changes the horizontal lines in the stick diagram to choice lines in the collection of rectangles. It also introduces 4 essential lines for every ccu added, these essential lines must be in the final solution to ORTHOGONAL LINES. Suppose there is a solution for STICK DIAGRAM, i.e. there exists a set of lines $S$ such that $|S| \leq K$, then there must be a solution $P$ to ORTHOGONAL LINES with $O = |S| + 4K$. This is because the essential lines must be in $P$ and the choice lines which correspond to the horizontal lines in $S$ must also be in $P$. Conversely if there is a solution $P$ to ORTHOGONAL LINES then there must be a solution $S = P - \{e \mid e$ is an essential line in $P\}$ to
STICK DIAGRAM.

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 ORTHOGONAL LINES.

ORTHOGONAL LINES is thus NP-Complete.

This section shows that the general case of covering all the adjacencies of a collection of adjacent rectangles with the minimum number of maximal-length orthogonal lines (ray guarding) is NP-Complete. The next section of this chapter discusses a heuristic approach to solving the problem and a later section discusses special cases in which an exact solution can be found.

5 Heuristic Algorithm

A heuristic algorithm to find a non-redundant set (a set of lines such that none can be removed without leaving an adjacency uncovered) of orthogonal lines to cover all the adjacencies is presented in this section. The algorithm has two phases, first the digraph of adjacencies among rectangles is constructed, and then the lines covering all adjacencies are determined.

5.1 Determining the adjacencies and constructing the directed graph

In determining the adjacencies between rectangles and creating the directed graph, horizontal and vertical adjacencies are treated as separate cases. Only the algorithm for vertical adjacencies (and horizontal lines) will be discussed here (that for horizontal adjacencies is analogous). Suppose that the coordinates of each rectangle $R$ are given as $R_L$, $R_B$, $R_R$, $R_T$ (left, bottom, right and top denoting the bottom left and top right corners of the rectangle). An algorithm to produce the directed graph of adjacencies is outlined in Figure 11.

Left and Right are lists of records made up of the rectangle number and the coordinates of the left and right edges of the rectangles respectively.
Create a list Left of all the rectangles
Sort the list Left in ascending order of the Left[i]L (break ties based on increasing Left[i]B)
Create a similar list Right of all the rectangles
Sort the list Right in ascending order of the Right[j]R (break ties based on increasing Right[j]B)
Set i = j = 1
WHILE i <= N AND j <= N
  IF Left[i]L < Right[j]R
    THEN increment i
  IF Left[i]L > Right[j]R
    THEN increment j
  IF Left[i]L = Right[j]R
    THEN
        Increment i
      IF Left[i]B < Right[j]T
        THEN add edge P_i - P_j to the digraph G
    If Left[i]T <= Right[j]T
      THEN increment i
      ELSE increment j

Figure 11: Determining the adjacency digraph

The sorting phase of this portion of the algorithm is clearly O(n log 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 log n).

5.2 Determining the orthogonal lines

The algorithm generates all the possible straight lines which cross the adjacencies between rectangles, determines which lines are essential (are the only lines which cut a particular adjacency), removes any lines which only cut adjacencies cut by the essential lines (redundant lines) and then resolves the choice conflict. The resolving of the choice is done by repeatedly choosing the choice line which cuts the highest number of previously uncut adjacencies. In this algorithm, lines are represented as a list of rectangles stored in an array. For example, the line 1 - 2 - 3 means a line which cuts the adjacency between rectangles 1 and 2 and the adjacency between rectangles 2 and 3.

An outline of the algorithm is given in Figures 12, 13, 14 and 15. The algorithm uses the directed graph G created above (Figure 11) and visits the rectangles based on the order in the list Left used in Figure 11. The algorithm uses an adjacency list, C, of the adjacencies in the graph to keep track of the number of lines cutting 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.

5.3 The Correctness of the method

The algorithm in Figures 12 and 13 generates all the possible orthogonal lines which cross the adjacencies. It also extends all lines as far as possible to the left and right. Clearly any line which is the only
Set $L$ to be empty ($L$ is the set of all possible lines)

For each rectangle $P_i$ in $Left$
   For each rectangle $P_j$ adjacent to $P_i$ (determined from $G$)
      For each line $l$ in $L$
         IF $l$ can be extended into $P_j$
            THEN Add $l \cup \{P_j\}$ to $L$
         IF no candidate line $l$ can be extended into $P_j$
            THEN
               Set $T$ to be empty
               Set $t$ to be $\{P_i, P_j\}$
               Add $t$ to $T$
               For each line $k$ in $L$ which ends in $P_i$
                  Set $u$ to be equal to $t$
                  Set extending to TRUE
                  Set $r$ to be $|k| - 1$
                  ($r$ points to the second last polygon in line $k$)
                  WHILE extending
                     IF $k[r] \cup u$ is a valid line
                        THEN
                           Replace $u$ with $k[r] \cup u$
                           Decrement $r$
                        ELSE
                           Set extending to FALSE
                           Add $u$ to $T$
                  Add the longest line, $m$ in $T$ to $L$

Figure 12: Determining all possible orthogonal lines — Phase 1

Set $E$ to be empty

For each line $l$ in $L$
   For each pair of adjacent rectangles $P_i$ and $P_j$ in $l$
      Mark in $C$ that $l$ crosses the adjacency $c_{ij}$ between $P_i$ and $P_j$
   For each adjacency $c_{ij}$ in $C$
      IF $c_{ij}$ is only cut by one candidate line $e$
         THEN
            Add $e$ to $E$
            Remove $e$ from $L$

Figure 13: Finding the essential lines — Phase 2
For each line $e$ in $E$

For each pair of adjacent rectangles $P_i$ and $P_j$ in $e$

Mark in $C$ that $e$ crosses the adjacency $c_{ij}$ between $P_i$ and $P_j$

For each line $l$ in $L$

For each pair of adjacent rectangles $P_i$ and $P_j$ in $l$

Check if adjacency $c_{ij}$ is cut by an essential line

IF all adjacencies are cut by lines in $E$

THEN Remove $l$ from $L$

---

**Figure 14:** Removing Redundant lines — Phase 3

---

For each remaining line $q$ in $L$ (These are the choice lines)

Determine how many previously uncut adjacencies it cuts

REPEAT

Choose the line $r$ which crosses the most previously uncut adjacencies

Add $r$ to $E$

For each adjacency $c_{kl}$ crossed by $r$

For each line $t$ which crosses $c_{kl}$

Decrement the number of adjacencies cut by $t$

If the number of adjacencies cut by $t$ is equal to 0

THEN Remove $t$ from $L$

Remove $r$ from $L$

UNTIL all of the adjacencies have been cut.

---

**Figure 15:** Resolving the issue of choice — Phase 4

---

Figure 16: A configuration in which $O(n^2)$ adjacencies are crossed
line to cut a particular adjacency must be in the final set of lines otherwise there would be at least one
adjacency which has not been crossed. Also any line which only cuts adjacencies which are cut 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 which crosses the highest number of adjacencies
which are previously uncut. This means that the selected line can be treated as essential provided no line
selected previously cuts any of those adjacencies. This is clearly the case. In addition, no line selected
later can cut only those adjacencies and adjacencies already cut by the essential lines. A line chosen later
has to cut at least one previously uncut adjacency. Thus each line selected from the set of choice lines
cuts at least one previously uncut adjacency and is thus required.

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.

6 Complexity Argument

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 performance for the algorithm will occur for situations such as that shown in Figure
16. In this case the rectangles on the left edge of the collection will give rise to n/2 lines which must be
extended through another n/2 rectangles. This means that the worst case for the portion of the algorithm
which generates the full collection of lines is \( O(n^2) \) — there are a possible \( O(n^2) \) adjacency crossings.

The adjacencies which are cut by only a single line can be found in \( O(n^2) \) by traversing each line and
marking off in the adjacency list each adjacency as it is crossed. The first time it is crossed it is marked
with the identity of the line which crosses it and subsequent crossings are marked as such. 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, \( O(n) \), to determine which adjacencies are only cut 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 cut by
the essential lines and then determining which lines in the set of candidate lines only cut adjacencies
already cut 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 previously uncut adjacencies
are cut 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 a number of steps are performed — the line which crosses the
highest number of these previously uncut adjacencies is selected \( (O(n)) \); this line is added to \( E \ (O(1)) \); each adjacency of the selected line is considered and other lines cutting this adjacency have their counts
decremented and are removed from \( L \) if they no longer cut uncut adjacencies (this could be as expensive
as \( O(n^2) \) if the line selected has \( O(n) \) adjacencies and each adjacency is cut 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 \( O(n^2) \) but the total amount of work done in this phase of the algorithm is bounded by the number of
adjacency crossings which could occur in the worst case \( (O(n^2)) \). The total work in this phase is then
\( O(n^2) \).

The overall time complexity of the algorithm to produce a non-redundant set of lines is thus \( O(n^2) \).
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 list could require $O(n^2)$ space.

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 $\epsilon$-approximation [16] (for any $\epsilon < 1$ the error ratio grows as $\log n$ and thus no $\epsilon$ 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 [16]. This algorithm has an approximation threshold of at most $1/2$ — a solution which is at most twice the optimum solution. Hochbaum [13] also discusses a heuristic which gives a value which 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.

6.4 Experimental Results
The greedy algorithm (longest line heuristic), an algorithm based on the random selection of lines and an algorithm which produces the exact number of lines required were implemented and tested. The algorithm which produces the exact solution to the problem was made as efficient as possible by partitioning the choice lines into subsets which have common previously uncut adjacencies. 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 uncut adjacencies which are common to all of the choice 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 is then grown from left to right by randomly generating rectangles which are adjacent to the right hand edges of those which have already been placed in the configuration. The advantage of generating the data in this fashion is that a large number of adjacencies between rectangles in the horizontal direction is guaranteed. All rectangles in the configuration have breadth and height randomly chosen in the range from 5 to 15 units. The final configuration of rectangles is 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 cover all the adjacencies. The random algorithm ranged in accuracy from producing the same result as the exact solution to requiring six extra lines to cover all adjacencies. Table 1 shows the results of the testing of the heuristics on configurations of rectangles of this form.

From these results it can be seen that the heuristic of choosing the longest of the available choice lines at any stage results 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 which could occur in the problem being studied.

7 Extensions
As extensions to the work done here there are a number of problems to be considered.
<table>
<thead>
<tr>
<th>Number of rectangles</th>
<th>1000</th>
<th>1500</th>
<th>2000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of tests</td>
<td>50</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>Minimum number of lines</td>
<td>321</td>
<td>477</td>
<td>655</td>
</tr>
<tr>
<td>Maximum number of lines</td>
<td>365</td>
<td>527</td>
<td>703</td>
</tr>
<tr>
<td>Average number of lines</td>
<td>341</td>
<td>510</td>
<td>680</td>
</tr>
<tr>
<td>Standard deviation</td>
<td>10.86</td>
<td>10.81</td>
<td>13.44</td>
</tr>
<tr>
<td>Minimum number of essential lines</td>
<td>277</td>
<td>426</td>
<td>569</td>
</tr>
<tr>
<td>Maximum number of essential lines</td>
<td>322</td>
<td>476</td>
<td>616</td>
</tr>
<tr>
<td>Average number of essential lines</td>
<td>300</td>
<td>450</td>
<td>592</td>
</tr>
<tr>
<td>Minimum number of choice lines</td>
<td>4</td>
<td>4</td>
<td>18</td>
</tr>
<tr>
<td>Maximum number of choice lines</td>
<td>27</td>
<td>36</td>
<td>52</td>
</tr>
<tr>
<td>Average number of choice lines</td>
<td>14</td>
<td>20</td>
<td>32</td>
</tr>
<tr>
<td>Standard deviation</td>
<td>5.83</td>
<td>8.44</td>
<td>8.77</td>
</tr>
<tr>
<td>Minimum error for longest line heuristic</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Maximum error for longest line heuristic</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Average error for longest line heuristic</td>
<td>0.02</td>
<td>0.10</td>
<td>0.05</td>
</tr>
<tr>
<td>Standard deviation</td>
<td>0.14</td>
<td>0.31</td>
<td>0.22</td>
</tr>
<tr>
<td>Minimum error for random choice heuristic</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Maximum error for random choice heuristic</td>
<td>4</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>Average error for random choice heuristic</td>
<td>1.26</td>
<td>1.30</td>
<td>1.90</td>
</tr>
<tr>
<td>Standard deviation</td>
<td>0.94</td>
<td>1.66</td>
<td>1.55</td>
</tr>
</tbody>
</table>

Table 1: Experimental results

- Reducing the amount of work required to calculate the set of lines which covers all of the adjacencies. In particular, attempting to address the issue of redundant calculations which are made for the collection of rectangles shown in Figure 16 or similar configurations.

- Addressing the related problem where each adjacency is only allowed to be cut by a single line.

- Allowing the adjacencies to be covered by non-orthogonal lines

- Attempting to solve the more general problem of adjacent convex polygons.

8 Conclusion

This paper addresses the problem of finding the fewest longest orthogonal straight lines that pass through all of the shared adjacencies between adjacent rectangles. The problem is made NP-Complete by the fact that various instances of choice can arise. An $O(n^2)$ algorithm is presented which finds a non-redundant set of lines to cover 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.

References


