Optimal Rectangle Covers for Convex Rectilinear Polygons

by

Paul Franklin

B.Sc., Simon Fraser University, 1984

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
in the School
of
Computing Science

SIMON FRASER UNIVERSITY
March 1986

Approval

Name: Paul Franklin

Degree: Master of Science

Title of Thesis: Optimal Rectangle Covers for Convex Rectilinear Polygons

Dr. Binay Bhattacharya
Senior Supervisor

Dr. Arthur Liestman

Dr Joseph Peters

(in absentia)

Dr. Hossam El-Gindy
External Examiner

6 March 1986

Date Approved

PARTIAL COPYRIGHT LICENSE

I hereby grant to Simon Fraser University the right to lend my thesis, project or extended essay (the title of which is shown below) to users of the Simon Fraser University Library, and to make partial or single copies only for such users or in response to a request from the library of any other university, or other educational institution, on its own behalf or for one of its users. I further agree that permission for multiple copying of this work for scholarly purposes may be granted by me or the Dean of Graduate Studies. It is understood that copying or publication of this work for financial gain shall not be allowed without my written permission.

Title of Thesis/Project/Extended Essay

Optimal Rectangle Covers for Convex Rectilinear Polygons

Author:

(signature)

Paul Wickham Franklin

(name)

16 April 1986

(date)

Abstract

Optimal Rectangle Covers for Convex Rectilinear Polygons

Rectilinear Polygons have all their edges parallel to the **x** and **y** axes. The problem of finding a minimum cardinality set of rectilinearly oriented rectangles that cover the area of a rectilinear polygon has applications in image processing, pattern recognition and VLSI, and is of use in solving other problems in computational geometry. Such a set is called an optimal rectangle area cover (ORAC). Associated problems are those of finding an optimal rectangle corner cover (ORCC) or edge cover (OREC) of a rectilinear polygon.

A rectilinear polygon is convex in the **x** (**y**) direction if any horizontal (vertical) line drawn through the polygon intersects it no more than twice. A **semi-convex** polygon is convex in either the **x** or **y** direction. A **convex** rectilinear polygon is convex in both **x** and **y** directions.

Five linear algorithms are presented. The first algorithm finds an ORCC for a convex rectilinear polygon. The remaining algorithms are adaptions of this first method. Algorithm E1 finds an OREC for a class of convex rectilinear polygons called **irreducible**. Algorithm E2 finds an edge cover within one rectangle of optimal for general polygons. Algorithms A1 and A2 find area covers within one rectangle of optimal for irreducible and general polygons, respectively.

A method used in the ORCC algorithm finds a maximum matching in a bipartite graph with a rectilinearly convex adjacency matrix in O(|V|) time, providing the ranges of adjacency or the adjacency matrix is given. A similar method can be used to find maximum matchings in bipartite graphs whose adjacency matrices have the consecutive 1's property in O(|V|log|V|) time providing the ranges of adjacency or the adjacency matrix is given.

Acknowledgements

I would like to thank my senior supervisor, Dr. Binay Bhattacharya, for his constant enthusiasm and encouragement. At times, the problems to be solved appeared intractable, and his support was always there when I needed it.

Dr. Hossam El-Gindy from the University of Pennsylvania was my external examiner. He provided a most thoughtful critique of the work. In particular, his suggestions on simplifying the unobscured algorithm have been adopted. I am extremely grateful to him for his time and effort.

Thanks are also due to the rest of my thesis committee, Dr. Arthur Liestman and Dr. Joseph Peters, for their interest in this research, and their helpful comments and criticisms.

This research was supported in part by a Simon Fraser University Graduate Research Fellowship.

I am actually indebted to nearly everyone, especially to my bank.

Dedication

To my wife Gail, who has supported and encouraged me in these endeavors from the beginning, and to our first child, Anne, who has patiently waited to be born until this thesis was finished.

Table of Contents

List of Tables

List of Figures

Chapter 1

An Introduction to the Problem

1.1. Introduction

We present five linear time algorithms for convex rectilinear polygons. The first finds a minimum cardinality set of rectangles that covers the corners of a convex rectilinear polygon. This set is termed an optimal rectangle corner cover (ORCC). The second algorithm, E1, finds an optimal rectangle edge cover (OREC) for irreducible convex rectilinear polygons. Algorithm E2 finds an edge cover within one rectangle of optimal for general convex rectilinear polygons.

Area cover algorithms A1 and A2 use the algorithm E1 which returns an OREC for irreducible convex rectilinear polygons. This OREC is sometimes also an optimal rectangle area cover (ORAC). When it is not, an area cover may be obtained by adding one more rectangle. As the holes in an OREC may not be "necessary", the area cover obtained by adding a rectangle may not be optimal. Algorithms A1 and A2 find area covers within one rectangle of optimal for irreducible and general convex rectilinear polygons, respectively.

A method used in the ORCC algorithm finds a maximum matching in a bipartite graph with a rectilinearly convex adjacency matrix in O(|V|) time.

1.2. Motivation

The problem of finding a minimum cardinality collection of rectangles that covers the corners, edges or area of a rectilinear region has applications in several areas. Photolithographic masking in the manufacture of integrated circuits may be accomplished by using a union of rectangles to create a rectilinear shape [CKSS81]. The construction of letters and shapes on video displays may require similar techniques. [CKSS81] [FrK84] There are also applications in pattern recognition and in decomposition techniques for polygon computations. [ORS83] A similar problem, that of partitioning the interior of a polygonal region into basic shapes, has applications in image processing, architectural database and manipulation of VLSI artwork data [AsA83].

1.3. Basic Definitions

A polygon is described by its vertices, listed in anticlockwise order. We do not allow three consecutive vertices to be collinear. The vertices of a polygon are described by their x and y coordinates. All polygons are simple, that is two non-consecutive edges do not intersect.

A polygon is **rectilinear** if all its edges are parallel to the x and y axes. In this paper, unless otherwise specified, all rectangles and polygons will be assumed to be rectilinear.

A rectilinear polygon is **convex** in the x (y) direction if any horizontal (vertical) line drawn through the polygon intersects it no more than twice. (Note: This precludes the possibility of the polygon having "holes" in it.)

A **semi-convex rectilinear polygon** is convex in one of the two axes only, i.e. either the x or y direction. A semi-convex polygon may also be called **vertically** or **horizontally** convex. A **convex rectilinear polygon** is convex in both the x and y directions. In this paper, we consider only the problem of finding covers for convex rectilinear polygons. Unless otherwise stated, the term **polygon** will be taken to mean a convex rectilinear polygon.

Non-convex rectilinear polygons may be **simply connected** (no holes) or **multiply connected** (with holes)[1]. The various types of rectilinear polygons are illustrated in Figure 1-1.

There are several other terms in common use that are used to describe rectilinearity. Derick Wood prefers the term **isothetic** [Woo85], and the expression **iso-oriented** also occurs in [LiP80]. **Orthogonal** is also used to describe geometry where all edges are aligned to two orthogonal axes.

1.4. Covers

We will consider only the problem of finding a set of rectilinear rectangles that constitute a cover of a convex rectilinear polygon in this thesis. It has been shown that, for a non-convex rectilinear polygon, non-rectilinear rectangles may occur in a ORAC [ORo82] (in fact non-rectangular shapes may also). Non-rectilinear rectangles will not appear in an optimal edge or corner cover.

---

[1] Terminology from O'Rourke [ORo82]

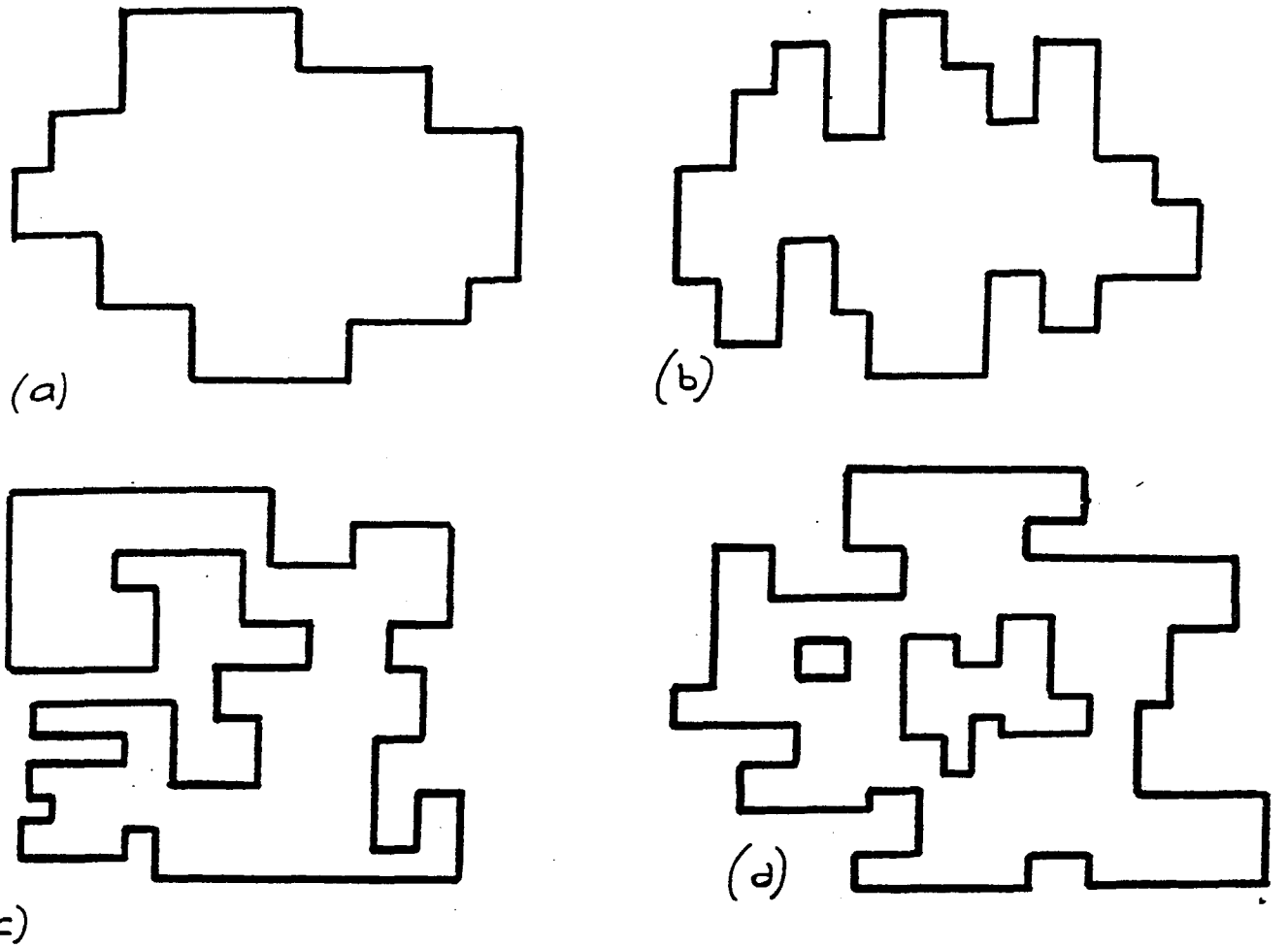


Figure 1-1: Rectilinear Polygons

(a) convex (b) semi-convex (c) non-convex simply-connected (d) non-convex multiply connected

Rectangles used in a cover are not allowed to extend outside the polygon. Thus any set of rectangles we consider as a cover must be contained in the polygon they cover and be rectilinear.

A rectangle contained in a polygon may be expanded until its sides meet edges of the polygon. Such an expanded rectangle we call **maximal**. We will assume that all rectangles used in a cover are maximal.

1.4.1. Corner Covers

A polygon's **corner** is a vertex of the polygon whose two adjacent edges form an interior angle of 90 degrees. We say a polygon's **corner** vertex is covered if there exists a rectangle in the cover with one of its corner vertices coincident with the polygon's corner vertex.

To distinguish a non-corner vertex from a corner, we identify a vertex whose two adjacent edges form an interior angle of 270 degrees as a **reflex vertex**.

A set of rectangles interior to the polygon is a **corner cover** of a polygon if each corner of the polygon coincides with a corner of some interior rectangle. The sides constituting the rectangle's corner may be of shorter length than the edges making up the polygon's corner.

1.4.2. Edge Covers

A polygon's **edge** is covered by a set of interior rectangles if every point on that edge is a point on at least one rectangle's side.

A set of interior rectangles is an **edge cover** of a polygon if each edge of the polygon is covered by the set of rectangles. Clearly, an edge cover of a polygon is also a corner cover. However, a corner cover is not necessarily an edge cover (see Figure 1-2).

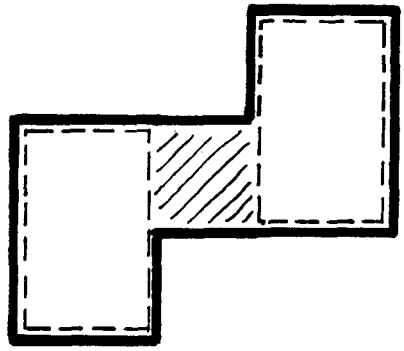


Figure 1-2: A Corner Cover Not Covering All the Edges

The shaded region is not covered by the two rectangles that cover the corners.

1.4.3. Area Covers

A polygon's interior region is covered by a set of interior rectangles if every point of the polygon is contained in at least one of the rectangles. In other words, the union of the rectangles is the polygon itself. Area covers are also known as **rectangle covers**.

Clearly, an area cover of a polygon is also a corner and edge cover. However, an edge cover may not be an area cover (see Figure 1-3).

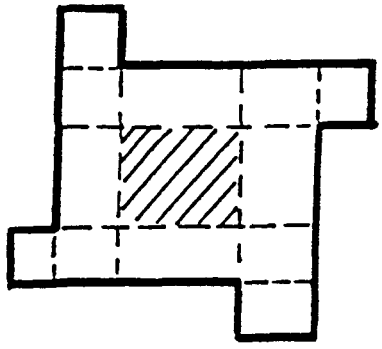


Figure 1-3: A OREC That Is Not an Area Cover

The shaded region is not covered by the four rectangles covering all the edges.

1.5. The Problem Definition

The problems to be considered for sets of rectilinearly oriented rectangles are then:

1. **ORCC (Optimal Rectangle Corner Cover):** Find a minimum cardinality set of rectangles that is a corner cover of a given rectilinear polygon.

2. **OREC (Optimal Rectangle Edge Cover):** Find a minimum cardinality set of rectangles that is an edge cover of a given rectilinear polygon.

3. **ORAC (Optimal Rectangle Area Cover):** Find a minimum cardinality set of rectangles that is an area cover of a given rectilinear polygon.

These problems may be posed for convex, semi-convex or non-convex rectilinear polygons. They will often be referred to as simply ORCC, OREC and ORAC, as will a solution to one of these problems. This thesis will only consider the problems for convex rectilinear polygons.

1.6. Reiteration of Basic Assumptions

- All polygons are simple.
- All polygons are rectilinear.
- All polygons are convex, unless otherwise stated.
- All rectangles are rectilinear.
- All rectangles used in a cover are completely contained by the polygon.
- All rectangles used in a cover are maximal in each direction.

1.7. A Survey of the Literature

A general approach to the field of computational geometry is given in [Sha78], and in [PrS85]. A recent article [LeP84] provides a survey of current results for most problems in the field of computational geometry. Results in rectilinear computational geometry are given in [Sac84] and are surveyed in [Woo85]. Some rectilinear computational geometry problems are also considered in [Ull84].

1.7.1. The Unit Square Model

Several papers make use of the unit square model in describing rectilinear polygons. In this model, the plane is seen as an infinite grid of unit squares and a rectilinear polygon is a set of adjacent squares.

There is a direct correspondence between this model and the rectilinearly oriented cartesian coordinate system we use. Any cartesian rectilinear polygon can be converted to a polygon of the unit square model as follows. Scan the polygon from left to right. Each time a new x coordinate appears, assign it the next available unit square coordinate in the x direction. Scan the polygon from bottom to top. Each time a new y coordinate appears, assign it the next available unit square coordinate in the y direction (Figure 1-4).

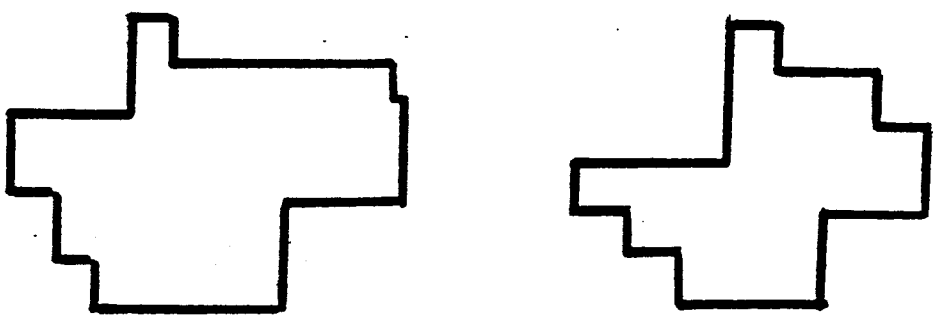


Figure 1-4: Transformation from Cartesian to Unit Square Model

It is easy to see that the optimal covers for both models will be the same, i.e. the rectangles will be constrained by the same sets of corners and edges in each case. (This might not be so if the cover set of rectangles were not rectilinearly oriented.)

Because of this direct correspondence, we will make no distinction between models.

1.7.2. Covering Polygons by Rectangles [CKSS81]

This paper by Chaiken, Kleitman, Saks and Shearer took a primarily mathematical approach. An **antirectangle** in a polygon is defined to be a set of unit squares (we can use points in our model quite satisfactorily), no two of which are contained in any possible internal rectangle.

They proved that if a rectilinear polygon is convex in both the x and y directions, then the minimum cardinality of a rectangle area cover equals the maximum cardinality of an antirectangle. Chvatal had originally conjectured that this was true in the general non-convex case. It has been shown that this conjecture is not true.

The polygon used in the unit square model is referred to as a **board**. The approach used in [CKSS81] involves **reducing** the board by removing areas covered by the rectangles that cover "tabs". Tabs are the extremal parts of a polygon (see section 2.1 for a more rigorous definition). It is established that the maximal rectangles that cover tabs must be in any optimal cover. Once such allowable reductions have been done, the resulting polygon is called an **irreducible** board. One useful result they found is that if every optimal edge cover (OREC) of an irreducible board is not an area cover, then an ORAC can be obtained by adding one rectangle to some optimal edge cover.

Their method of proof can be used to obtain a polynomial time algorithms for finding an optimal corner cover (ORCC), or area cover (ORAC) of a convex rectilinear

polygon, or an optimal edge cover (OREC) of an irreducible convex polygon. The details were omitted in their paper.

1.7.3. Minimum Convex Polygon Covers for Non-Rectilinear Polygons [ORo82]

O'Rourke showed the more general problem of finding a minimum covering set of convex polygons for multiply connected non-rectilinear polygons to be decidable in this 1982 paper. Although Masek had shown that the problem was NP-hard, thus establishing a lower bound, until this time no upper bound was known.

Brute-force searching algorithms for minimum convex covers establish that the problem is decidable. O'Rourke considered several situations.

Steiner points are points that are not vertices of a polygon, but that may be required as vertices of the convex pieces of a minimum cover. If Steiner points are not allowed as vertices of the covering set, then clearly a finite number of convex polygons can be generated and a searching algorithm is possible: of all such convex polygons, find the minimum cardinality subset that is a cover. If Steiner points are allowed, is it reasonable to assume that they must occur on intersections of extensions of edges? If so, there are again a finite number of convex polygons that can be generated.

In a previous paper, O'Rourke established that under the edge extension restriction, searching for minimum covers can be accomplished by minimizing a Boolean expression, which is a known NP-complete problem. (This did not establish the problem as being NP-complete.) [ORo82a]

The author quotes Masek as having shown that finding a minimum rectangle cover of a multiply connected rectilinear polygon is NP-complete [ORo82]. This reference to Masek is from an unpublished manuscript [Mas79].

It is shown that the edge extension restriction for Steiner points is indeed a restriction of the problem, as there exist polygons whose minimum covers require convex polygons that have vertices not in this category. This implies that no naive searching algorithm for minimum covers can exist, since the Steiner points cannot be restricted to a finite class.

Decidability is shown using Tarski's decision procedure for geometry.

1.7.4. Some NP-hard Polygon Decomposition Problems [ORS83]

In this 1983 paper, O'Rourke and Supowit proved that the problems of decomposing a non-rectilinear polygon into convex, star-shaped or spiral subsets are NP-hard. They allow overlap (covers as opposed to partitioning) and the inclusion of holes (multiply connected).

NP-hardness is shown by polynomial transformation from 3SAT [GaJ79]. As it is not seen how to check a solution in polynomial time, this process shows that the problems are NP-hard, but not necessarily NP-complete.

1.7.5. Minimum Generating Sets [FrK84]

This 1984 paper by Franzblau and Kleitman also uses the unit square model in approaching the rectilinear problem. They show an algorithm with $O(n^2)$ behaviour for finding a minimum rectangle area cover (ORAC) for vertically convex polygons.

The authors use a result attributed to Frank: for a vertically convex polygon, the only information one needs to construct a rectangle cover is the set of distinct horizontal intervals determined by the vertical sides of the rectangles. The rectangle cover is then the set of maximal rectangles in the polygon generated by the intervals [FrK84].

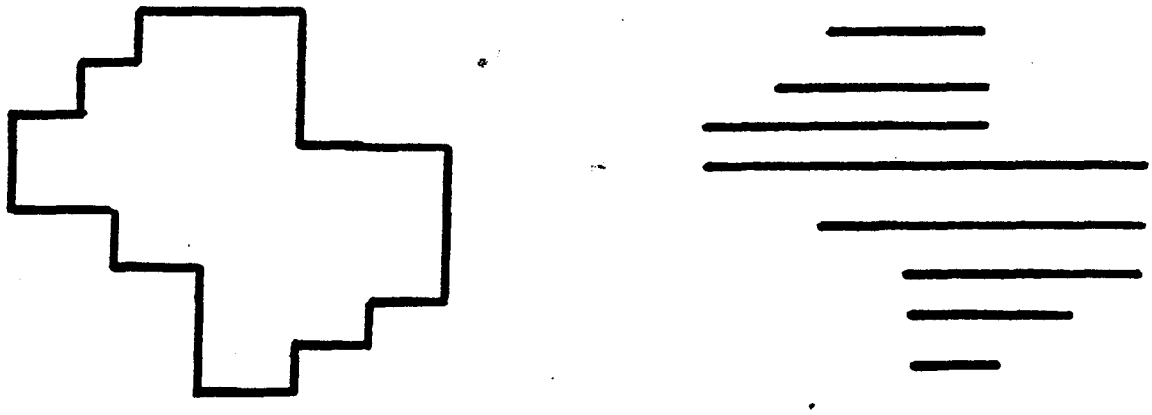


Figure 1-5: A Convex Polygon and Its Intervals

The method generates a set S of all possible horizontal intervals in the polygon (Figure 1-5).

Then a minimum cardinality set of intervals (the minimum generating set) which

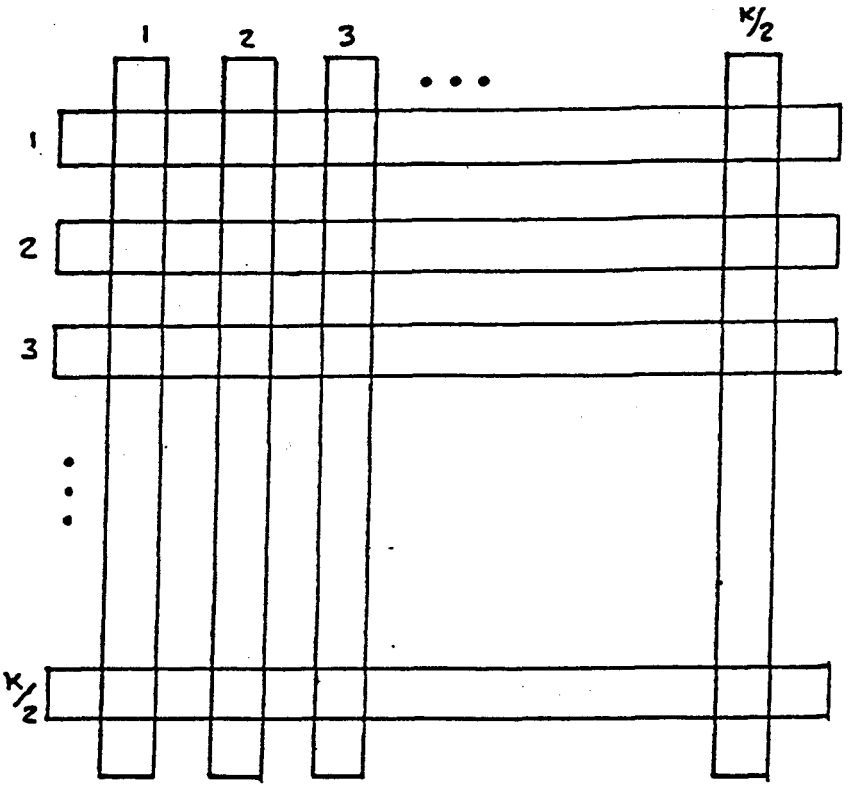


Figure 1-6: An $O(\sqrt{n})$ Sized Cover

If there are k/2 horizontal and k/2 vertical rectangles, their union has n edges, where n is $O(k^2)$. Thus a covering is obtained with k rectangles, where k is $O(\sqrt{n})$. A partition of this figure would require O(n) rectangles.

generates S is found. Here **generation** means that each interval in S is the union of some generating intervals. We can see that a minimum generating set of intervals corresponds to a minimum rectangle cover in the polygon. This allows the problem to be reduced from a planar covering problem to a one dimensional interval covering problem.

The algorithm to determine the minimum generating set has $O(n^2)$ worst-case

complexity, where n is the number of intervals. The number of intervals a rectilinear polygon can generate is O(n) (actually less than or equal to n/2-1 in the convex or semi-convex cases - see Chapter 3). An implementation of this algorithm is described in the paper.

Franzblau and Kleitman also mention the relationship of rectangle covering to partitioning. For rectilinear polygons, partitioning into rectangles can be done in $O(n^{5/2})$ time [Oht82]. However, examples are known for which partitioning requires $O(k^2)$ rectangles, where covering can be done with k rectangles. (See Figure 1-6.)

1.8. Summary of Known Results

Table 1-1 of known problem complexities is taken from [ORS83], with the exception of the convex and semi-convex rectilinear cases, where [FrK84] have shown an $O(n^2)$ algorithm. The figures all refer to area covers.

Class of Polygons	Shape of Components	Complexity
multiply connected	convex	NP-hard
	star-shaped	NP-hard
	spiral	NP-hard
simply connected	all	?
multiply connected rectilinear	rectangular	NP-complete
simply connected rectilinear	rectangular	?
semi-convex rectilinear	rectangular	$O(n^2)$
convex rectilinear	rectangular	$O(n^2)$

Table 1-1: Complexity of Area Covering Problems

Chapter 2

Basic Terminology and Covering Properties

Before presenting the algorithms, we define a number of necessary terms. This thesis has an index which should aid in finding an individual definition quickly. In order to provide a unified view of the problem, we have tried to adopt or create terms which are intuitively obvious.

Following an explanation of terminology, we show which rectangles must be in any optimal corner, edge and area cover for convex rectilinear polygons.

2.1. Basic Terminology

2.1.1. Extremal Edges, Tabs and Chains

An **extremal edge** of a rectilinear polygon is one with both its endpoints extreme in either the x or y direction (see Figure 2-1). In a convex rectilinear polygon we must then have exactly four extremal edges, two parallel to the x axis and two parallel to the y axis.

An extremal edge of a polygon must have both its endpoints being **corners**. If this were not so, the edge would not be extremal. Following the conventions of [CKSS81] we term this extremal edge with its two adjacent edges a **tab**.

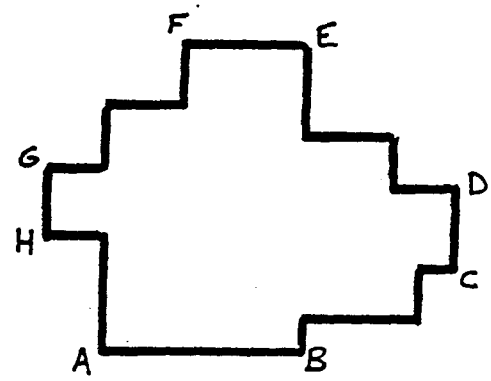


Figure 2-1: Extremal Edges

Edges AB, CD, EF and GH are extremal edges.

A convex (rectilinear) polygon can be seen as being composed of four or fewer distinct **chains** of corners and edges, bounded by the extremal edges of the polygon. The chain of edges from the (vertical) minimum x edge to the (horizontal) maximum y edge is the **top left chain**, denoted TL. Similarly, we may describe the remaining chains as **top right, bottom left** and **bottom right**, denoted TR, BL and BR respectively, as seen in Figure 2-2. Each chain starts and ends at an extremal edge. If two extremal edges are adjacent in the polygon, the chain between them is composed of just one corner.

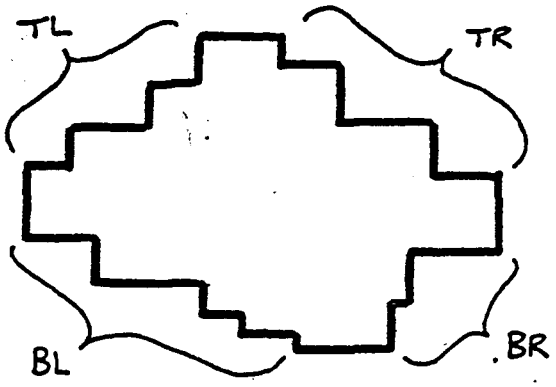


Figure 2-2: Chains

2.1.2. Corner Types

Corners with the same orientation are of **common** type. A pair of corners with 90 degree difference in orientation we call of **adjacent** type, and a pair with 180 degree difference in orientation we term of **opposite type**, as in [CKSS81]. Clearly, in a convex polygon, corners of common type must be in the same chain, corners of adjacent type must be in adjoining chains, while corners of opposite type will be in opposite chains. The various types of corner pairs are illustrated in Figure 2-3.

Two corners of adjacent type we term **collinear adjacent corners** if they have the same x or y coordinates (Figure 2-4).

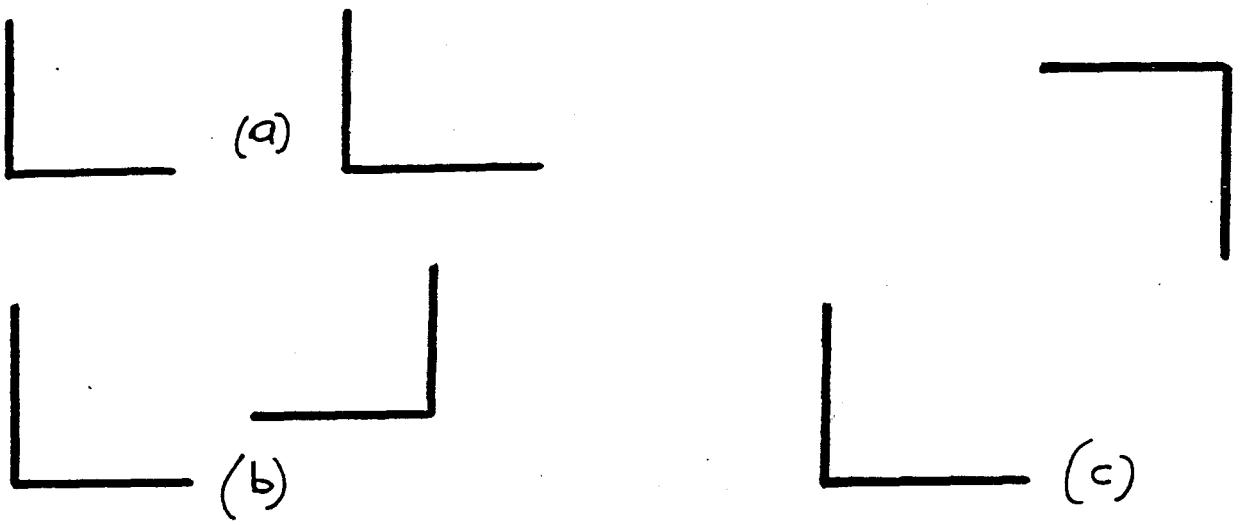


Figure 2-3: Corner Types

(a) Common, (b) Adjacent and (c) Opposite corner pairs.

Figure 2-4: Collinear Adjacent Corners

2.1.3. n-somes and Obscuring

A **4-some** is a group of four corners aligned so that they form the four corners of some rectangle. A set of three corners aligned so that they form three corners of some rectangle, and are not part of a 4-some, we call a **3-some**. If the rectangle defined by a 4-some or 3-some has some other chain of edges passing through it, we call that 4-some or 3-some **obscured**. In a convex polygon, a 4-some cannot be obscured.

We call a pair of collinear adjacent corners both of which are not part of a 4-some or an unobscured 3-some a **CA2-some**.

A CA2-some's rectangle may be obscured in the sense that its maximal rectangle cannot cover the full extent of the edges of both corners. This is only of concern for edge and area covers.

A pair of opposite corners that may define a legitimate (unobscured) rectangle in a cover we term an **OPP2-some**.

A corner not being used with some other corners to define a rectangle in a cover is called a **singleton**.

We will also refer to the rectangles defined by n-somes as 4-somes, 3-somes, CA2-somes, OPP2-somes and singletons when speaking of the rectangles to take for a particular cover.

2.1.4. Quadrant, Range, Shadow & Scope

The **quadrant** of a corner is defined simply as the quadrant made by the infinite extension of its edges. The boundaries (edge extensions) are included in this region.

The **range** of a corner is defined as the region created by the rectangle defined by its two edges. We do not include the boundary of this rectangle in the region defined when speaking of range intersections.

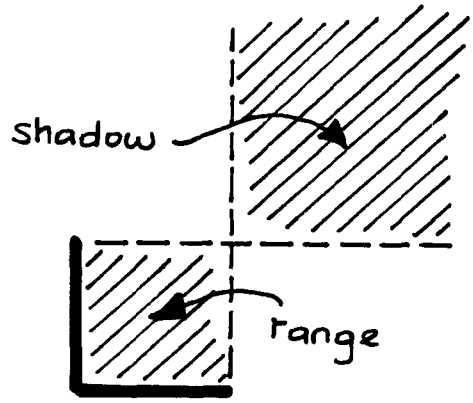


Figure 2-5: The Range and Shadow of a Corner

The **shadow** of a corner is defined as its quadrant translated to the opposite corner of the corner's range. The shadow includes its own boundaries. (See figure 2-5.)

The **scope** of a corner is the list of corners it may legitimately pair with to create a rectangle as part of an OPP2-some in a particular kind of cover.

In a corner cover, we are only concerned with covering corners, and thus the scope of a corner A is the list of all corners in the opposite chain whose vertices are in the quadrant of A, and whose rectangles made with A are not blocked by the other two chains. Figure 2-6 shows the scope of a corner for a corner cover.

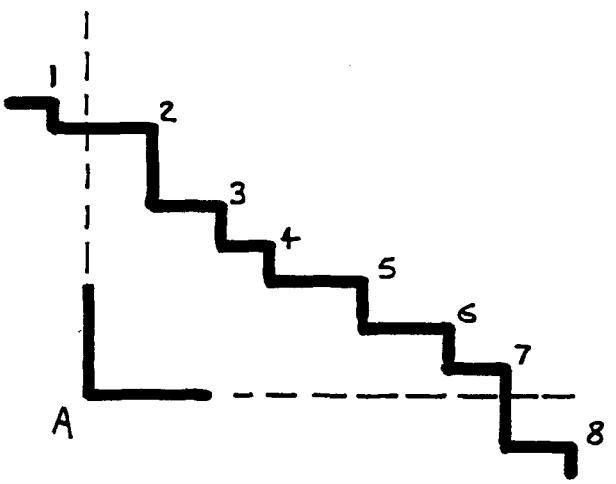


Figure 2-6: The Scope of a Corner in a Corner Cover

The scope of corner A for a corner cover is all opposite corner vertices in the quadrant of A, and whose rectangles are unobscured. Here, A's scope is opposite corners 2 to 7.

In an edge cover, the **scope** of a corner A is defined as being the list of corners of the opposite chain that A may form an OPP2-some with, such that the edges of both corners are covered by the rectangle. This will be the corners in the opposite chain whose vertices lie in the shadow of A, whose edges lie fully in the quadrant of A,

and whose rectangles with A are not blocked. A corner whose range is intersected by the opposite chain cannot have any OPP2-somes, as no rectangle can cover both its edges. Figure 2-7 shows the scope of a corner in an edge cover.

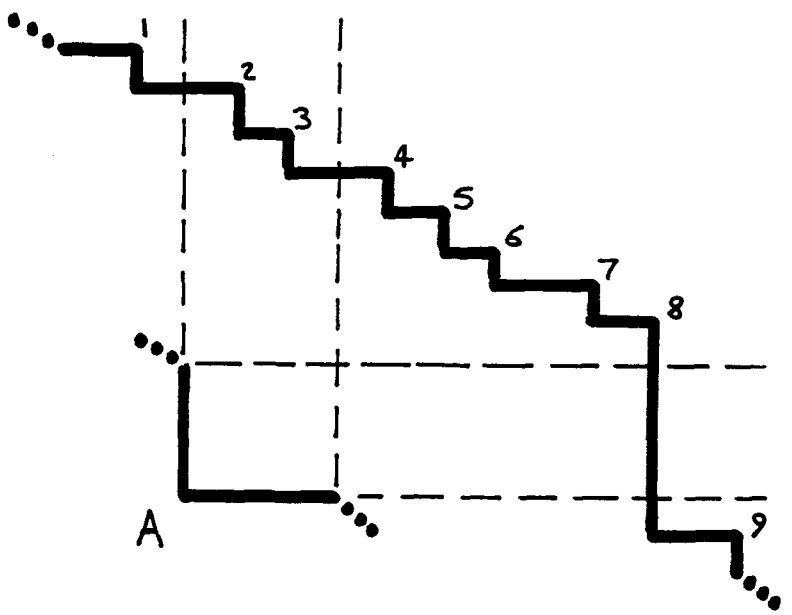


Figure 2-7: The Scope of a Corner in an Edge Cover

Corner A's scope in an edge cover is opposite corners 4 to 7.

A **range intersection** or **RI** occurs when a corner's range is intersected by the opposite chain. We shall use the term **range intersection** and its abbreviation **RI** interchangeably. We note that a corner whose range is intersected cannot have both its edges covered by the same rectangle, so its edges will have to be covered by more than one rectangle. We call this a **multiple covering** of a corner. This term is defined more exactly in Chapter 6.

2.2. How Rectangles Cover Corners, Edges and Area

2.2.1. Rectangle Expansion

We assume that all rectangles in a cover are rectilinear and maximal in each direction. Thus, given convexity of the polygon being covered, a rectangle may be constrained from further growth by several possible combinations of edges and corners:

1. Four edges.

2. One corner and two or more edges

3. Two collinear adjacent corners and an edge or two.

4. Two opposite corners, and possibly one or two edges.

5. Three corners, and possibly an edge.

6. Four corners.

Examples of these are illustrated in figure 2-8.

A pair of corners of adjacent type must be a CA2-some if they are to "share" the same rectangle. (One example of this would be the rectangle expanding to fill a tab.)

Similarly, a rectangle may be defined by two corners of opposite type, provided this

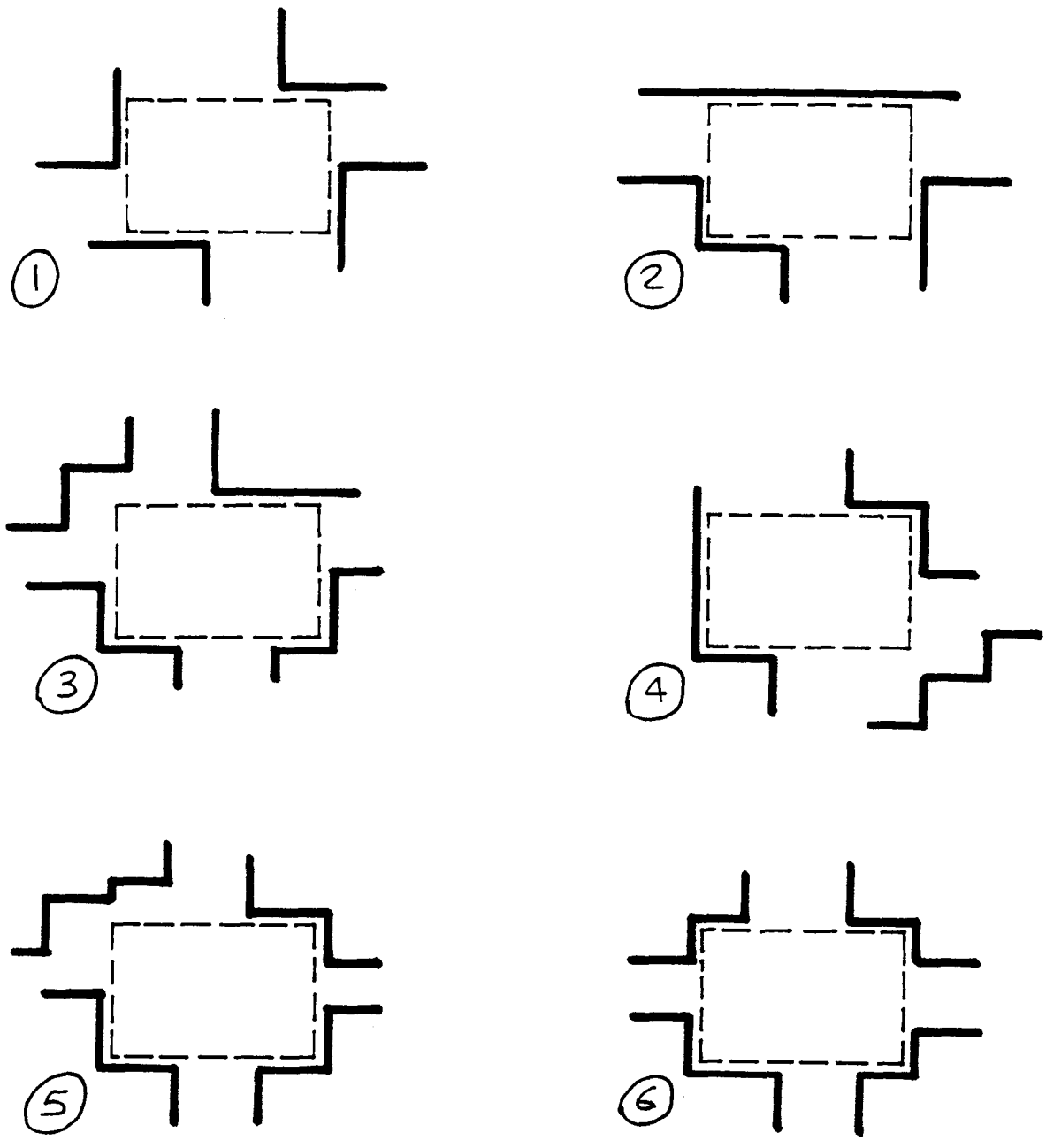


Figure 2-8: Examples of Convex Polygon Rectangle Constraint Types

expanded rectangle is completely contained in the polygon. It is easy to see that if this expansion is blocked, it must be blocked by edges of the other two chains.

We note that a rectangle constrained by an edge or a corner may not completely cover that edge or corner's edges.

It is necessary to investigate the ways in which rectangles can cover corners, edges and area. We assume that a CA2-some or a 3-some is not part of a 4-some, and that a CA2-some is not part of an unobscured 3-some.

2.2.2. 4-somes

A 4-some (a rectangle that can cover four corners) must be in any optimal corner, edge or area cover of a convex rectilinear polygon. A maximal rectangle for any of the four corners will expand automatically to cover the other three corners [CKSS81]. All vertices and edges of the four corners are covered by this rectangle.

2.2.3. 3-somes

Any unobscured 3-some's rectangle must be in any minimum corner, edge or area cover, as the middle corner's maximal rectangle will expand to cover the other two corners anyway [CKSS81]. This rectangle will cover the middle corner, and the edges of at least one of the outer corners (there may be an edge extending past the rectangle on one side). In a corner cover, we consider all three corners covered. In an edge cover, an extending edge is not fully covered by the maximal rectangle.

If the 3-some is obscured, its rectangle cannot be used, and it is ignored, which

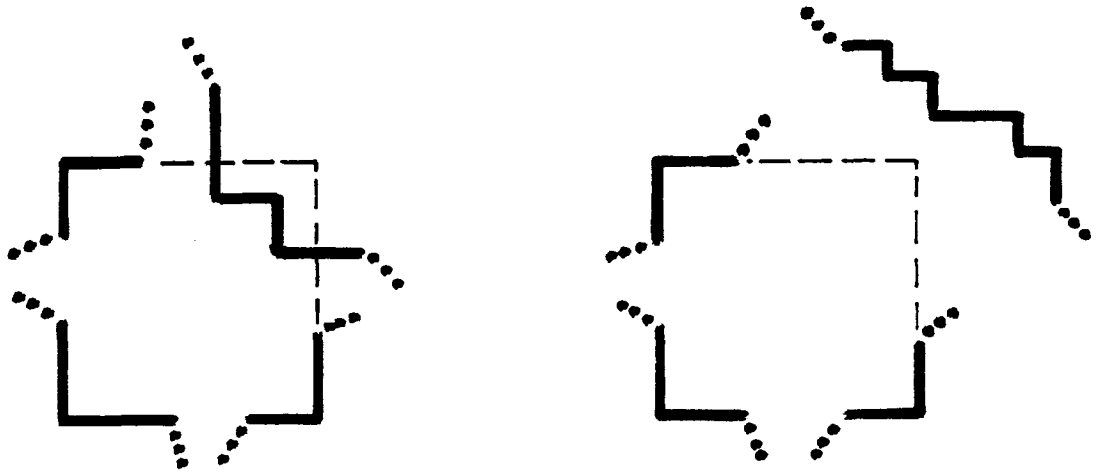


Figure 2-9: Obscured and Unobscured 3-somes

results in its being recognized as two adjoining CA2-somes. Their two rectangles cover the corners and edges of the obscured 3-some. The 3-some may be obscured because it adjoins one or two other 3-somes.

Where three 3-somes adjoin, as in figure 2-10, the outer two 3-some's rectangles are unobscured. These two rectangles cover all three corners of the obscured middle 3-some.

Where two 3-somes only adjoin, only one of them can be unobscured. The unobscured 3-some's rectangle is taken for any cover, which covers two of the

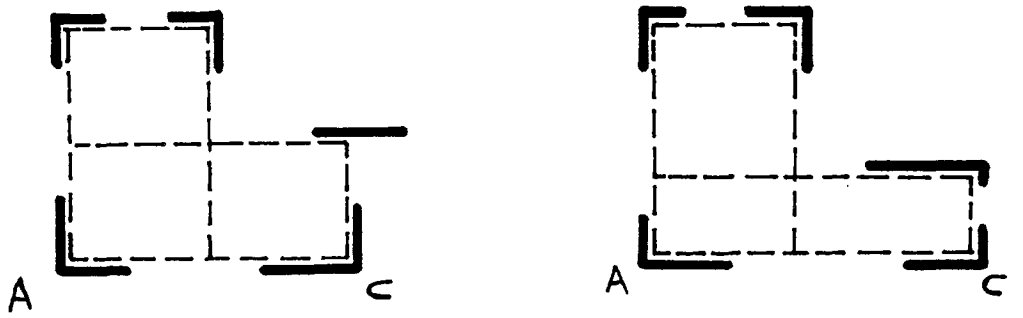


Figure 2-10: Adjoining 3-somes

obscured 3-some's corners. The remaining uncovered corner C is collinearly adjacent to a covered corner A, and can only expand to that corner A, as their rectangle is blocked opposite C. If C is left unrecognized as part of a CA2-some, it can only be expanded to A as a singleton at the end of the algorithm. This also covers any part of A's edge that extended past the unobscured 3-some's rectangle (see figure 2-10). Again, all corners and edges are covered optimally by taking unobscured 3-some's rectangles, and ignoring obscured 3-some's corners not covered by these rectangles.

2.2.4. CA2-somes

Finally, all CA2-somes (a pair of collinear adjacent corners, both uncovered by 4-somes or unobscured 3-somes) must define a rectangle in any minimum corner, edge or area cover. (see Figure 2-11). The maximal rectangle defined by corners A and B is blocked at some level, say above A. Corner A's maximal rectangle can only expand

to the blocking edge and corner B. Thus the rectangle defined by A and B must be in any ORCC, OREC or ORAC.

However, in a corner cover, we consider both corners covered by such a rectangle, where in the edge cover algorithms presented we are concerned about the maximal rectangle fully covering the edges of both corners.

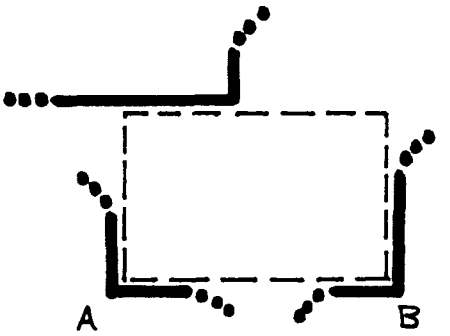


Figure 2-11: Rectangle Covering a CA2-some

The maximal rectangle defined by corners A and B is blocked at some level, say above A. Corner A's maximal rectangle can only expand to the blocking edge and corner B. Thus the rectangle defined by A and B must be in any ORCC, OREC or ORAC.

2.2.5. Remaining Corners

Now all remaining rectangles in any cover can only be defined by OPP2-somes or by singletons and some edges.

A **maximum independent set** or **MIS** from all possible OPP2-somes is defined as a largest cardinality set of pairs of corners forming OPP2-somes such that no corner is in more than one pair. There may be more than one such set for a given list of possible OPP2-somes.

All that remains to complete an ORCC is find a MIS out of all possible OPP2-somes. Lastly we take the remaining singletons and allow their rectangles to expand to an available opposite corner or to some edges.

To complete an edge cover, we need a more detailed analysis. This is presented in Chapter 5.

Chapter 3

Bounds on Cardinalities of Covers

It is of interest to know the range of cardinalities of covering sets possible for various problems. The following results have been determined.

3.1. Five Lemmas

Let n=|V|=|E|.

Lemma 1: In a rectilinear polygon, n must be even.

Proof: If we remove all the horizontal edges, the disconnected vertical edges and all the vertices remain. The vertices must be in pairs, one pair for each vertical edge. Thus |V| must be even. □

Lemma 2: In a rectilinear polygon, the number of horizontal edges is equal to the number of vertical edges.

Proof: In a tour of the polygon (or its multiply connected components), one must encounter alternately horizontal and vertical edges, and n is even, from Lemma 1. An alternate proof can be seen from Lemma 1. If we rotate the polygon by 90 degrees and remove the horizontal (formerly vertical) edges we have exactly one disconnected edge for each pair of vertices. Thus the number of vertical and horizontal edges must be the same. □

Lemma 3: In a simply-connected rectilinear polygon, the number of distinct horizontal or vertical intervals that can be generated is less than or equal to (n/2)-1.

Proof: In a vertical (horizontal) scan, the first interval encountered is at a tab, and has two adjacent vertical (horizontal) edges. Each successive vertical (horizontal) edge encountered adds at most one new interval (if two edges enter at the same coordinate, only one new interval is created for the two of them). In the semi-convex or non-convex case, there may be more than one tab on a side, each requiring two vertical edges to start an interval, but possibly creating an interval when non-convex indentations end. As we have from lemma 2 that the number of horizontal or vertical edges = n/2, the result follows. □

Lemma 4: In a multiply-connected rectilinear polygon, the number of distinct horizontal or vertical intervals that can be generated is less than or equal to n/2+m-1, where m is the number of holes.

Proof: For multiply connected polygons in a vertical (horizontal) scan, each interior component (hole) requires two vertical (horizontal) edges to begin with at its tab. This component creates at most two new intervals when it enters the scan, and one new interval for each successive vertical (horizontal) edge that enters the scan, as before. When this interior component leaves the scan however, one last interval is created, if the edges of the polygon that the interior component makes intervals with have changed during this scan. Thus, for each interior component with k vertical (horizontal) edges, we have at most k+1 intervals created (again, less per edge accounting-wise if other edges of the polygon enter the scan at the same coordinate).

We note that if the hole itself is non-convex, the number of intervals that can be generated per edge is reduced. This gives an upper bound on the number of intervals in a multiply-connected non-convex polygon of n/2+m-1, where m is the number of interior components. □

Lemma 5: The number of **corners** in a simply-connected rectilinear polygon is always 4+((n-4)/2).

Proof: If we tour the edges of the polygon anti-clockwise with the interior on our left-hand side, a left turn is a **corner**, and a right turn is a **reflex vertex**. As we tour the polygon and return to the starting point, there must be exactly four more corners than there are reflex vertices. □

3.2. Upper Bounds on Cardinalities of Minimum Covers

As no more than (n/2)-1 horizontal or vertical intervals can be generated in a simply-connected rectilinear polygon, this number of rectangles must always be sufficient to cover the corners, edges or area of such a polygon. This is a tight bound for all three kinds of covers, as the example of a square or rectangle shows.

Lemma 4 shows that, for multiply-connected non-convex polygons, n/2+m-1 (where m is the number of holes) rectangles will always be sufficient. This too is a tight upper bound, as Fig. 3-1 shows.

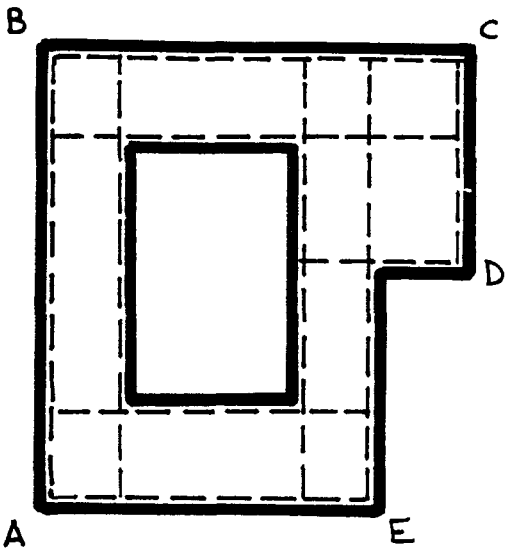


Figure 3-1: A Tight Upper Bound for the Multiply Connected Case

Here, n=10, and n/2+m-1=5, the minimum number of rectangles necessary to cover the area or edges of the polygon. A corner cover can be obtained with 3 rectangles.

3.3. Lower Bounds on Cardinalities of Minimum Covers

Analysis is based on the number of corners, denoted c. We know from Lemma 5 that c=4+((n-4)/2) in a convex polygon.

For convex polygons, if every rectangle covers four corners (the best possible performance for any cover), then we require at least $\lceil c/4 \rceil$ or $1+\lfloor n-4/8 \rfloor$ rectangles. This is a tight lower bound, as examples can be shown that give this performance, e.g. figure 3-2.

Figure 3-2 shows a tight lower bound for all three kinds of covers.

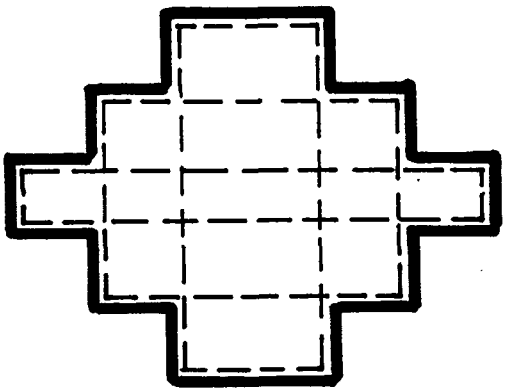


Figure 3-2: A Tight Lower Bound for All Three Kinds of Cover

Here, n=20, c=12, and $\lfloor c/4 \rfloor$=3, the size of a minimum corner, edge or area cover.

In the case of a multiply connected non-convex polygon, $O(\sqrt{n})$ edges may be covered by each rectangle. This would give an $O(\sqrt{n})$ cardinality cover. Figure 1-6 shows a multiply-connected polygon with O(n) vertices with an edge or area cover of $O(\sqrt{n})$ rectangles.

Chapter 4

A Linear ORCC Algorithm

4.1. Finding an ORCC for a Convex Rectilinear Polygon

Chapter 2 lists the rectangles that must be in any ORCC for convex rectilinear polygons. We only need to cover the remaining corners as economically as possible to obtain an ORCC. This leads to the following algorithm.

4.1.1. Algorithm Steps

1. Find all 4-somes. List their rectangles, and mark their corners covered.

2. Find all 3-somes not part of 4-somes. List the rectangles of all unobscured 3-somes, and mark their corners covered.

3. Find all CA2-somes not part of 4-somes or unobscured 3-somes. List their rectangles.

4. Find the set of all possible OPP2-somes (permissible opposite corner defined rectangles) among the corners left uncovered.

5. Find a maximum independent set (MIS) of OPP2-somes. List their rectangles and mark their corners covered.

6. Find rectangle expansions for the remaining uncovered corners (singletons) and list them.

4.1.2. Proof of Correctness

We have shown that rectangles defined by 4-somes, remaining unobscured 3-somes and then by remaining CA2-somes must be in any ORCC. Of the corners remaining, we must show that our method gives a minimum cover.

Finding a maximum cardinality independent set of OPP2-somes (pairings representing permissible rectangle expansions between opposite corners) does this. We find as many OPP2-somes as possible of the remaining uncovered corners that each cover two unique corners. The singletons then remaining we give their own maximal rectangle. Although it may expand to an opposite corner, that opposite corner has already been paired, or else we could have added one more pair to our MIS.

4.2. A Linear Implementation of the ORCC Algorithm

We assume that the convex rectilinear polygon P is described by the coordinates of its vertices, in anticlockwise order. We can find extremal edges and all corners (as opposed to reflex vertices) in linear time by a simple tour. This gives us the starting and ending points of the four chains. Let the four chains be identified A1 to A4 and numbered as in figure 4-1.

We represent the x-coordinate of a corner i in the chain Aj by Aj[i].x, and its y-coordinate by Aj[i].y.

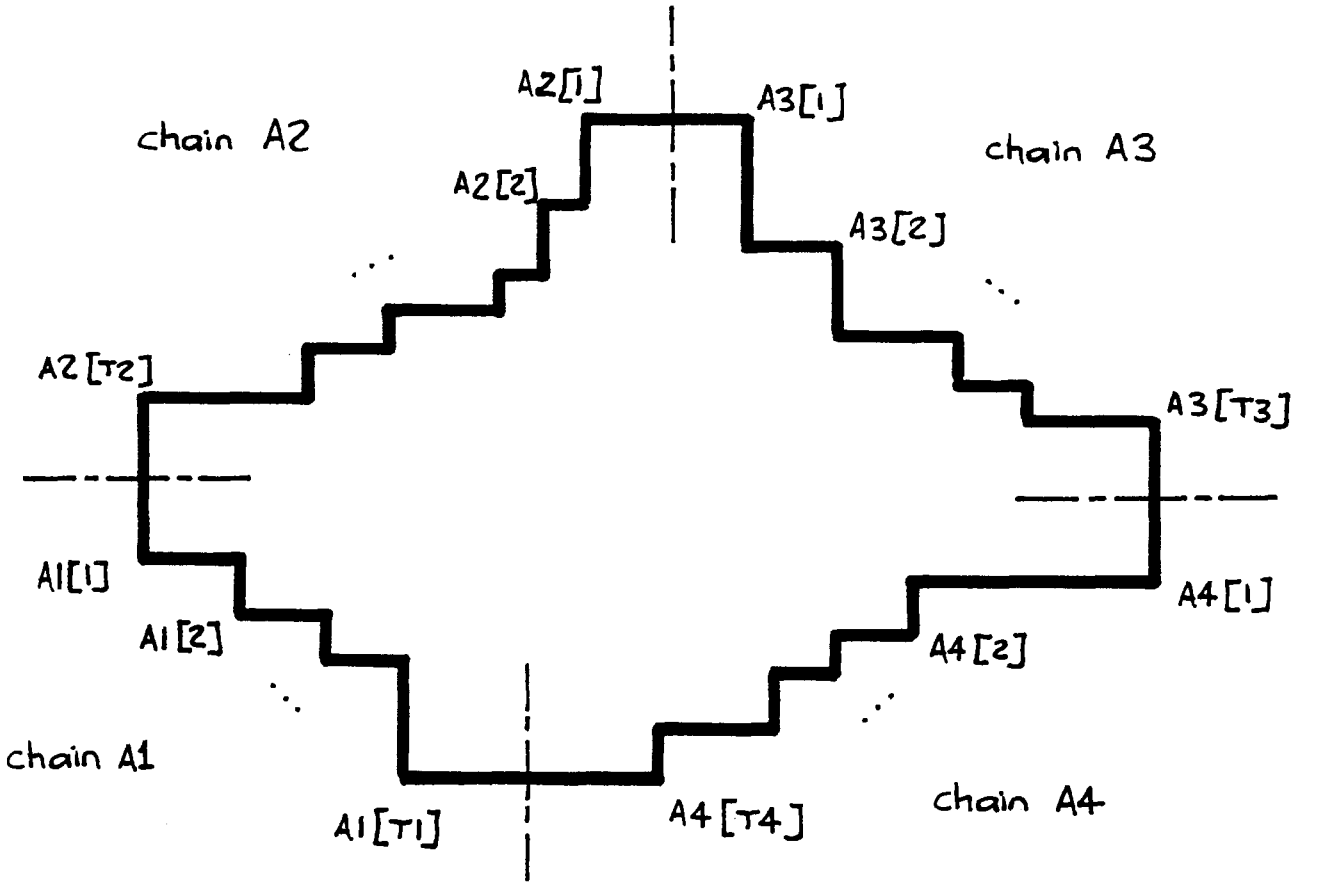


Figure 4-1: Chain and Corner Numbering for Algorithms

The four chains are numbered A1 to A4, clockwise from the bottom left chain. We assume there are Ti corners in each chain Ai. Corners in a chain Ai are then numbered from 1 at a tab to Ti at the other tab, as shown. Corners in the chains are numbered from top to bottom for convenience in algorithm notation.

4.2.1. Finding n-somes in Linear Time

We can find all 4-somes, unobscured 3-somes, CA2-somes and the ranges of possible OPP2-somes in linear time. The first step is to find all collinear adjacent pairs. We use the abbreviations CW and CCW for clockwise and counter-clockwise, respectively.

We need to check the four pairings of adjoining chains, marking each corner in the CW-most chain with the number of any collinear adjacent neighbour to its CCW side.

The corners are numbered as in figure 4-1. The following algorithm finds all collinear adjacent pairs between chains A1 and A2 in linear time. The collinear adjacent pairs are reported once each, with their corners in CCW order.

To simplify the indexing of the corners in this algorithm, we assume that a simple linear tour has already found the four collinear adjacent pairs that are **tabs**. Thus the algorithm works between corners 2 and Ti-1 in each chain.

```
begin CApairs

/* example for chains A1 and A2 */
/* initialize */
i=2
j=T2-1

while (i ≠ T1 & j ≠ 1) do

  if A1[i].x = A2[j].x
    then do
      list collinear adjacent pair j,i
                    /* listed in CCW order */
      /* store the blocking edge and increment */
      CW blocking for A1[i] is A2[j].y
      i=i+1
      j=j-1
    end

    else if A1[i].x < A2[j].x
    /* store the blocking edge and increment */
      then do
        CW blocking for A1[i] is A2[j+1].y
        i=i+1
      enddo
      else j=j-1

endwhile

end CApairs
```

When all collinear adjacent pairs have been found, the 4-somes, 3-somes and CA2-somes can be obtained. This can be done in one CCW sweep of the polygon, but the algorithms are written here separately for each set of n-somes (for clarity). Care must be taken to ensure that the CA2-somes and 3-somes contained in a 4-some or the CA2-somes in an unobscured 3-some are not listed. For this reason, a corner is tagged 4 or 3 when it has been found part of a 4-some or 3-some.

For the notation of this algorithm, it is convenient to assume the corners are numbered in CCW sequence in an array C (perhaps starting at corner A1[1]).

As the corners of collinear adjacent pairs are listed in CCW order, finding a

collinear adjacent neighbour for a particular corner will produce a match to the CCW side of it only. A 3-some is then found by finding a corner that has a collinear adjacent neighbor (to its right CCW) which also has a collinear adjacent neighbor (to its right CCW again). Thus each CA2-some or 3-some is only listed once, with their corners in CCW order.

One sweep of a particular chain looking for 4-somes will produce all foursomes, as a 4-some must have a corner in each chain. We know that all 4-somes in a convex polygon must be unobscured. Once these are found, and their corners tagged with a 4, all 3-somes can be found.

It is necessary to ensure that 3-somes are unobscured, otherwise they are ignored, and their corners are treated as two adjoining CA2-somes (except for edge covers in the adjoining 3-somes cases - see Chapter 2). Unobscured 3-somes have their rectangles placed in the cover set, and have their corners tagged with a 3. Finally, any collinear adjacent pairs both not tagged by the previous operations are listed as CA2-somes.

As n-somes are found, their corners are marked covered, and the rectangle they define is put in the cover set. The following algorithms find the n-somes.

First the 4-somes are found and reported. The corners involved are tagged so that CA2-somes or 3-somes contained in 4-somes will not be reported. We use CA-neighbor as an abbreviation for collinear adjacent neighbor. We assume the collinear adjacent pairs are in a list, ordered by the first corner in the pair. The array C

keeps track of the n-somes corners are assigned to. Initially, it is set to zero for all corners.

```
begin find4-somes

/* initialize array */
C=0  /* no corners assigned n-somes */

for w = 1 to number_of_corners_in_first_chain
     /* all 4-somes are found by end of first chain */

  if w has CA-neighbor x
    then if x has CA-neighbor y
      then if y has CA-neighbor z
        then if z has CA-neighbor w
          then do
            C[w],C[x],C[y],C[z]=4
            list 4-some w,x,y,z
          end

endfor

end find4-somes
```

Now 3-somes not contained in 4-somes are found. If they are unobscured, they are reported and their corners tagged.

```
begin find3-somes

for w = 1 to number_of_corners

  if C[w] ≠ 4  /* if not part of 4-some */
    then if w has CA-neighbor x
      then if x has CA-neighbor y
        then if unobscured[w,x,y]
          do
            list 3-some w,x,y
            C[w],C[x],C[y]=3
          enddo

endfor

end find3-somes
```

Finally collinear adjacent pairs of uncovered corners that are not contained in 4-somes or unobscured 3-somes are found and reported as CA2-somes.

```
begin findCA2somes

for w = 1 to number_of_corners

if C[w] = 0  /* if corner not yet tagged */
  then if w has CA-neighbor x
    if C[x] = 0 then list CA2-some w,x

endfor

end findCA2-somes
```

This is the function for checking whether a 3-some is obscured. It is written for checking 3-somes starting in chain A1.

```
begin unobscured[a,b,c]

/* assuming corner a in A1, b in A4 and c in A3. clearly  */
/* this implementation is chain dependent.                */
/* x & y coordinates represented by A1[a].x & A1[a].y     */

if A1[a].x > A2[1].x
  then return[YES]   /* obscuring cannot occur */

/* check blocking to see if rectangle unobscured */
else if CW blocking for A1[a] < A3[c].y return[NO]
    else return[YES]

end unobscured
```

4.2.2. Complexity of n-some Algorithms

The algorithm to find all collinear adjacent pairs of corners has been shown to have a linear implementation. The algorithms to find 4-somes, 3-somes and CA2-somes are also linear, given the list of collinear adjacencies. Can we find whether any of the 3-somes are obscured in O(n) total time?

The answer is yes, as the CW blocking edges have been stored during the CApairs algorithm. If the 3-some is obscured, the CW blocking edge for its CW-most corner will intersect the 3-some's rectangle. A check of this blocking edge with the CCW-

most corner's coordinates will show whether the 3-some is unobscured or not. As the information is already stored, total time for all 3-somes is linear.

4.2.3. Ranges of Possible OPP2-somes

It is necessary to check both pairs of opposing chains against each other to list all possible OPP2-somes (opposite pairings of corners), as pairings can occur in both opposing sets of chains. Only corners uncovered by n-somes are of interest, but if the range of possible pairings is calculated, covered corners can be deleted easily later on. A single pair of opposite chains can be checked and all possible pairings between them can be found in linear time, even though $O(n^2)$ pairings may exist.

In a convex rectilinear polygon, if a corner may define an unobscured rectangle with two opposite corners, it must also be able to define unobscured rectangles with all opposite corners between them. Thus we only need to record the range of possible pairings of each corner of one chain against its opposing chain. This gives all possible OPP2-somes between the two opposing chains.

Chains and corners are numbered as in figure 4-1.

Starting at the top of chain A1, we observe that if blocking occurs from the adjacent chain, A2, its result diminishes as we move farther away from it. Similarly, if blocking occurs from the chain A4, its result increases as we move towards it. Figure 4-2 illustrates this situation.

Thus the beginning of the scopes for consecutive corners of A1 will be non-

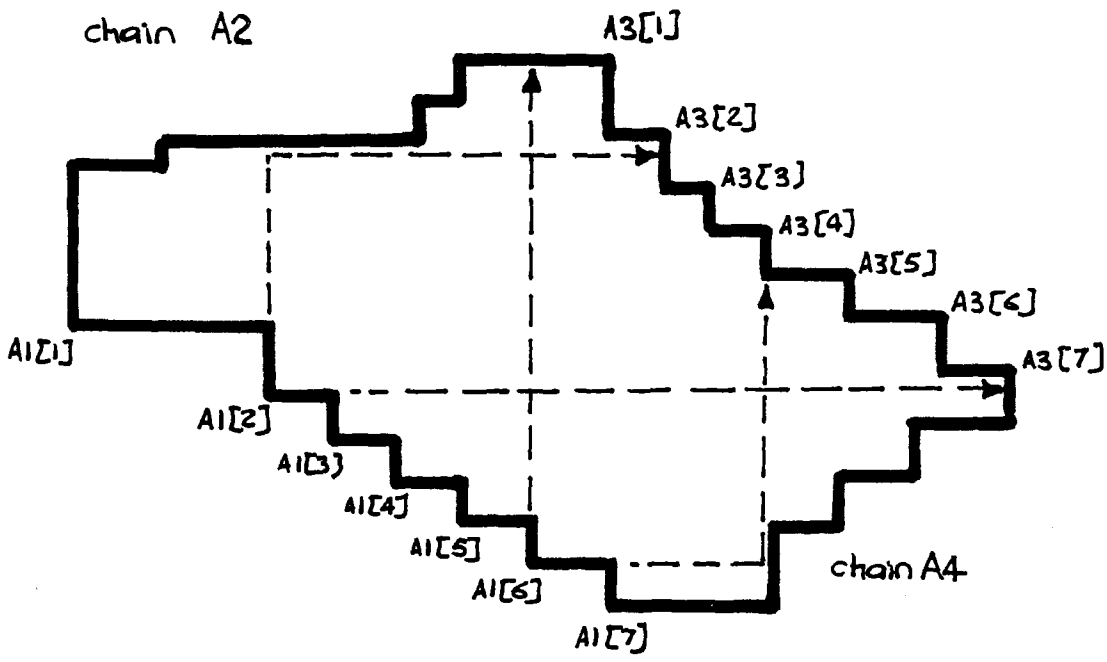


Figure 4-2: Blocking Effects on Scopes for Chain A1

Corner A1[2] can define legitimate rectangles with opposite corners A3[3] to A3[6], while the scope of corner A1[6] is from A3[1] to A3[4], in a corner cover.

increasing in terms of the vertex numbers of A3 while there is blocking from A2, then be non-decreasing, i.e. it is a unimodal function. Also, the ends of the scopes for consecutive corners of A1 may be unblocked at first, and be non-decreasing, then become blocked by A4, becoming non-increasing. Thus the ends of scopes too are unimodal.

We can find the starting point for an initial corner's range of acceptable pairings on the opposing chain in a simple linear sequential search. Successive corner's scopes

depend on blocking. For chain A1, if blocking by chain A2 occurs, scope starting points for successive corners will be non-increasing until the blocking ends for some corner in A1. Then the starting points of scope for successive corners in A1 will be non-decreasing. The behaviour is unimodal. The behaviour of the ending points of the scopes behaves similarly.

By keeping track of the previous corner's scope, we are taking at most two tours of the vertices on the opposing chain while finding all possible OPP2-somes for the corners on the first chain. This gives a linear time algorithm.

The algorithm for finding the ranges of OPP2-some pairing between two opposing chains is as follows. For illustration, we write the algorithm for chains A1 and A3.

```
begin OPP2algorithm

for k = 2 to T1-1                    /* for all non-tab corners */

  if A1[k].x < A2[1].x               /* if blocking occurs */
    then do
      find an m such that
      A2[m-1].x > A1[k].x > A2[m].x   /* find blocking corner */
                           /*  A2[m].y is blocking altitude  */
      if A2[m].y > A3[T3].y      /* if A3 exists at this level */
      then do
        /* find where the blocking altitude hits A3 */
        find an i such that
        A3[i-1].y > A2[m].y > A3[i].y
        range-start[k] = A3[i]
      end

      else range-start[k] = NIL
      /* all A3 above blocking altitude */
    end

    else do /* if no blocking */
      find an i such that
      A3[i-1].x < A1[k].x < A3[i].x
      range-start[k] = A3[i]
    end

  /* now find the end of the range for A1[k]  */

  if A1[k].y < A4[1].y          /* horizontal blocking occurs */
```

```
    then do
      find an m such that
      A4[m].y > A1[k].y > A4[m+1].y  /* find blocking corner */

        /* A4[m].x is now  vertical blocking line */

      if A4[m].x > A3[1].x          /* if A3 exists here */
        then do
          find an i such that
          A3[i].x < A4[m].x < A3[i+1].x
          range-end[k] = A3[i]
        end
      else range-end[k] = NIL
    end

    else do  /* if no horizontal blocking */
      find an i such that
      A3[i].y > A1[k].y > A3[i+1].y
      range-end[k] = A3[i]
    end
  end
        /* Now eliminate impossible ranges */

  if (range-start[k] = NIL
       |range-end[k] = NIL
       |range-start[k] > range-end[k])
    then do
      range-start[k] = NIL
      range-end[k] = NIL
    end

endfor

end OPP2algorithm
```

A mirror-image version of this algorithm will compute any OPP2-somes between chains A2 and A4. A linear implementation of this algorithm simply makes use of the fact the the changes in values of i and m are unimodal. During finding the beginnings of ranges, the values of m must be non-increasing as vertical blocking decreases. The values of i must be non-increasing as vertical blocking decreases, then must be non-decreasing. If we save the previous values of i and m the subsequent values are easily found in linear total time. In effect, no corner is visited more than twice.

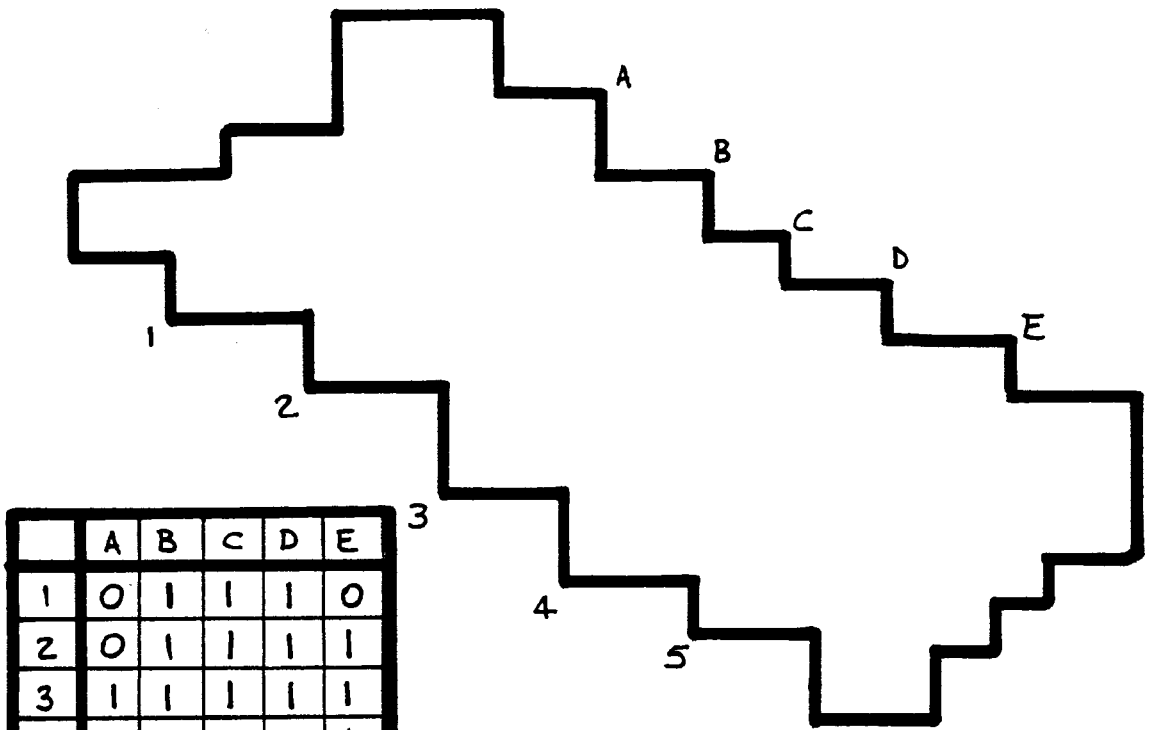
The singletons that remain can each be given a maximal rectangle in total linear time, by expanding them in CCW order. They may either be expanded to an unused OPP2-some, if any exist, or to the CW blocking edges found in the CApairs algorithm.

4.3. Finding the Maximum Cardinality Independent Set

A maximum cardinality independent set (MIS) out of all possible OPP2-somes can be found in linear time. As there may be OPP2-somes between both pairs of opposite chains, this procedure may have to be carried out for each pair.

4.3.1. A 0,1 Matrix Model of the Problem

. This problem of finding a MIS of OPP2-somes (where none of the OPP2-somes share a corner) can be modelled in the following way. Assume n uncovered corners remain, k in one chain, and n-k in the opposite chain. We number the uncovered corners of the first chain from 1 to k. The uncovered corners of the opposite chain are given numbers from k+1 to n. All possible OPP2-somes that could be formed between the two chains may now be represented in a k by n-k 0,1 **OPP2-some matrix**. A 1 in the i'th row, j'th column signifies that corner i may form an OPP2-some with corner j. All other entries are zeros. Figure 4-3 shows a polygon and its OPP2-some matrix after tabs have been taken.



	A	B	C	D	E
1	0	1	1	1	0
2	0	1	1	1	1
3	1	1	1	1	1
4	0	1	1	1	1
5	0	0	1	1	0

Figure 4-3: A Polygon and Its OPP2-some Matrix

The 0,1 matrix shows possible OPP2-some pairings between chains A1 and A3. For clarity, the corners are numbered simply and the tabs are not included.

4.3.2. Properties of the Matrix

We note some useful properties of the matrix. First, all 1's in a row or column must be contiguous (when contiguity is in one direction only, this is also known as the consecutive 1's property). This is because, in a convex polygon, if a corner may make an OPP2-some with two separate corners A and B in the opposite chain, it must also be able to form OPP2-somes with all corners between A and B. Secondly, this contiguity property means that the 1's in the matrix form a rectilinearly convex shape.

We call such a 0,1 matrix a **convex** matrix. This provides the key to our linear algorithm.

A 0,1 matrix where the 1's are contiguous in the horizontal direction only we term a **horizontally convex matrix**, or a matrix with the **consecutive 1's property**.

4.3.3. Redefining the Problem

The OPP2-some MIS problem may now be redefined as: given a convex 0,1 matrix M, find a maximum cardinality set of 1's such that no two 1's share any row or column. Each 1 selected then represents an OPP2-some that covers two unique corners.

4.3.4. The Cautious Method

The Cautious Method described here will find a maximum cardinality independent set S of 1's (where no two of the 1's in S share the same row or column) in a 0,1 matrix M with the consecutive 1's property. We call this a **maximum independent set** or **MIS** in M.

This algorithm works for any matrix with the consecutive 1's property, however, a linear implementation is possible for the convex OPP2-some matrix described, which has 1's contiguous both horizontally and vertically.

Given the k by n-k 0,1 OPP2-some matrix M, and an initially empty set S, do the following:

```
begin cautious-method

for j = 1 to n-k  /* for each column */

  find the set of 1's in column j

  find the first row i among these
    with the least number of 1's to
    the right of M_ij

  S = S ∪ {i,j}

  remove row i from the matrix

end for

end cautious-method
```

Now the set S is a MIS of OPP2-somes in M.

In effect, at each step we take the possible match with the least future choice. As the range of choices for all corners is always contiguous, we always leave ourselves the most future choice for matches.

A linear implementation of the Cautious Method for convex OPP2-some matrices is described later in this chapter.

4.3.5. Optimality of the Cautious Method

Theorem 1: The Cautious Method is optimal for horizontally convex matrices.

Proof: By contradiction. Given such a matrix M and a maximum independent set A obtained by the Cautious Method, we show that there cannot exist a larger cardinality independent set B.

Let us define a matrix X to be **contained** in a matrix Y in regard to a column j iff for every line of 1's to the right of column j in X, we can assign a line of 1's of equal or larger length in Y.

Given sets A and B, we proceed across the matrix M column by column. At some column, the entry taken for A must differ from that taken for B. Let this column be r, the row taken for A be p and the row taken for B be q.

As we take a row for A that has the least number of 1's to the right of M_{ij} for any row j where $M_{ij}=1$, we know that M_{qr} cannot have less 1's to the right of it than M_{pr}. Two possibilities remain: M_{qr} has the same number of ones to its right, or it has more. If it has the same number, the matrix M' obtained by removing the row q from M is contained in the matrix M" obtained by removing row p from M. The resulting matrices after

the appropriate rows have been removed have rows with exactly the same size, only with different row ordering.

If M_{qr} has more 1's to its right than M_{pr}, then the matrix M' that remains after row q is removed differs from the matrix M" with row p removed, in one respect. M' has the row p but not q, and M" has the row q but not p. The row q we removed from the matrix to form M' had more 1's to the right of column r than did the row p we removed to form M". As row p had less 1's to its right, M' has rows with the same number of 1's to the right of column r as does M", except for one row, where M" has more. The matrix M' representing choices left for pairing is always contained in the matrix M" representing choices made by the Cautious Method.

At each column where set A differs from set B, we find the same result. The successive matrices representing choices remaining for choosing set members for B are always contained in the matrices representing choices still available for A. Thus being able to choose more set members for B after set A has been chosen is a contradiction.

This shows there can never be a larger independent set member choosable by any other method of selection. □

4.3.6. A Linear Implementation of the Cautious Method

A linear implementation of the cautious method is possible for convex OPP2-some matrices, where the 1's in the matrix are contiguous in both horizontal and vertical directions. We require the ranges of the 1's in each row of the matrix, or the matrix itself.

Given the adjacency matrix, the ranges can be obtained in linear time by a tour of the rectilinearly convex set of outer 1's in the matrix.

Any operations on a matrix are not going to result in a linear algorithm, so we store only the ranges of possible pairings for a particular chain. Each row of the matrix is represented in a doubly linked list by an entry containing the numbers of the first and last columns containing 1's in that row. We call these numbers the range start and the range end.

We use the following data structures.

- A doubly linked list **L** containing the range start and range end of 1's for each row of M. This can easily be built in linear time, as we find the ranges of OPP2-somes possible between two chains.

- An array **A** of pointers to nodes in L, numbered such that the i'th pointer points to the node representing the i'th row of M. This can be added as the list is built.

- Pointers **TOP** and **BOTTOM** pointing to the topmost and bottom-most rows containing 1's in column j. Initially, the pointers TOP and BOTTOM are set to the topmost and bottom-most rows containing 1's in column 1.

The algorithm makes use of the rectilinear convexity property of the matrix. Given the topmost and bottom-most 1's in a column, there must be contiguous 1's between them, and a minimum number of 1's to the right of that column for any of the rows must be in the topmost or bottom-most row. (Otherwise, the 1's in the matrix would not be rectilinearly convex.) The minimum range to the right of the column may be common to several rows, but one of those must be the row pointed to by TOP or BOTTOM.

Let RE[TOP] and RE[BOTTOM] represent the range ends of the rows pointed to by TOP and BOTTOM respectively. Entries to the MIS will be a row-column pair, where the row number is stored in TOP or BOTTOM, and the column number in the current value of j.

For the convex OPP2-some matrix M, with m rows and n columns:

```
begin MISfind

for j = 1 to n do

  update TOP and BOTTOM for column j

  if TOP ≠ NULL then do

    if RE[TOP] > RE[BOTTOM]
      then do
        add BOTTOM,j to MIS
        remove BOTTOM -> row from linked list
      end do
      else do
        add TOP,j to MIS
        remove TOP -> row from linked list
      end do

  enddo  /* for then do */

endfor

end MISfind
```

It is easy to see that this implementation is linear in the number of rows and columns in the matrix. Removing a row can be done in constant time through the array A of pointers to each row's representation. TOP and BOTTOM are updated in total linear time over the whole algorithm by simple linear search of neighbours.

Clearly the cardinality of the MIS cannot be greater than the smaller of m and n. If m is greater (more rows than columns), the algorithm terminates when j = n. If n is greater, after the m'th row is removed from the linked list L, TOP will return NULL as the linked list of rows will be empty, and no further independent pairs will be reported.

4.3.7. Applications of the Cautious Method to Bipartite Matching

This maximum independent set problem in a convex 0,1 matrix is the same as finding a maximum matching in the bipartite graph represented by the matrix. Maximum matchings in bipartite graphs can be found in $O(\sqrt{|V||E|})$ time as shown in [HoK73]. In the convex OPP2-some graph, $|E|$ may be $O(|V|^2)$ giving $O(n^{5/2})$ performance.

However, given the contiguity of the non-zero entries (1's) in a convex or horizontally convex matrix, the bipartite graph obtained from such a matrix is not an arbitrary bipartite graph. When the 1's are contiguous in the horizontal direction only, we have a bipartite graph where all edges from the vertex set V1 (representing the rows) go to a set of consecutive nodes in the other vertex set, V2 (representing the columns). In order to preserve notation, we term such a bipartite graph **horizontally convex**, where its adjacency matrix forms a horizontally convex matrix.

A rectilinearly convex matrix yields an even more restricted form of bipartite graph. We call the resulting bipartite graph **convex**. Let V1 and V2 be the two sets of vertices in the convex bipartite graph G. Such a graph has edges from V1 to consecutive vertices in V2, and vice versa. Further, the beginnings and endings of the ranges of adjacency for the vertices in V1 and V2 are unimodal as we move from vertex to vertex along a vertex set.

This leads to the following theorem.

Theorem 2: The linear implementation of the Cautious Method finds a

maximum matching in a convex bipartite graph in time linear in the number of vertices in the graph, given the ranges of adjacency for the rows or columns, or the adjacency matrix.

For the horizontally convex matrix and bipartite graph, a simple brute-force algorithm finds a maximum matching in $O(|V|^2)$ time. For each column, a sequential search finds a row containing a 1 with the least number of 1's remaining to the right of that column. However, using sorting of the range ends and beginnings, an $O(|V| \log |V|)$ implementation is possible.

Theorem 3: An $O(|V| \log |V|)$ algorithm exists to find a maximum matching in a bipartite graph whose adjacency matrix has the consecutive 1's property, given the ranges of adjacency in the consecutive 1's rows (or columns), or the adjacency matrix.

Chapter 5

Edge Covering Algorithms

5.1. A Look at Edge Covering Problems

A corner cover may not cover all of each edge, and so the ORCC algorithm for corner covers will not necessarily produce an edge cover. In the ORCC algorithm, a rectangle only has to cover the vertex of a corner.

An edge cover algorithm may be constructed using the essential ideas of the ORCC algorithm, if it is possible to think in terms of covering corners rather than edges. In this method, we think of covering both edges of a corner simultaneously by some rectangle, and can then deal with n-somes and singletons as before. Some terms have to be redefined, and n-somes are taken for a cover under slightly different rules. We remove n-somes that must be in any OREC, and find the most economical cover of the remaining edges by pairing as many uncovered corners as possible in OPP2-somes. Here, in an OPP2-some both edges of each corner must be fully covered by the ensuing maximal rectangle.

This would provide an optimal edge cover only if we may disregard the covering of one edge of a corner by one rectangle, and the other edge by a different rectangle, where no one rectangle fully covers both edges of the corner (multiple coverings). We may do this in an irreducible polygon [CKSS81], but not in the general case, as there are polygons for whom every OREC has such a configuration.

Two linear algorithms are presented. **Algorithm E1** provides an OREC for a class of polygons called irreducible. This algorithm is used later in finding an area cover. **Algorithm E2** finds an OREC for general polygons whose necessary multiple covers are always part of an obscured 3-some or of a CA2-some with an extending edge.

Some more terminology must be introduced before dealing with edge covers.

5.1.1. Partial and Multiple Covering

We say that a corner is **multiply covered** if its two edges are covered by a combination of rectangles, where no one rectangle in the cover fully covers both edges by itself.

We say that a rectangle **partially covers** a corner if it does not cover all of both edges of the corner. We may also speak of a rectangle partially covering a particular edge.

5.1.2. Necessary Multiple Coverings

We will call a multiple covering of a corner A **necessary** when no OREC for the polygon exists that contains a rectangle fully covering both edges of A. Range intersections always result in necessary multiple coverings. A RI and a non-RI necessary multiple cover of a corner are shown in figure 5-1, where every OREC for the polygons must have corner A multiply covered as shown.

Figure 5-2 shows an example of an unnecessary non-RI multiple covering of a corner, with an alternate solution. Both are OREC's.

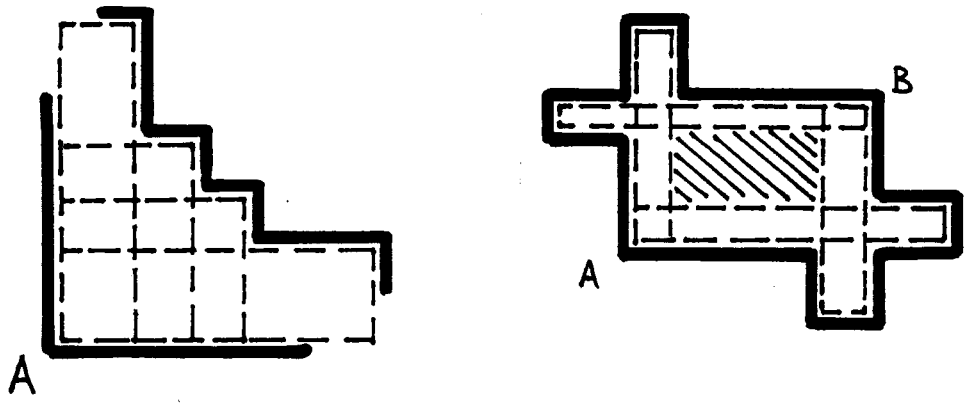


Figure 5-1: A RI and a Non-RI Necessary Multiple Covering

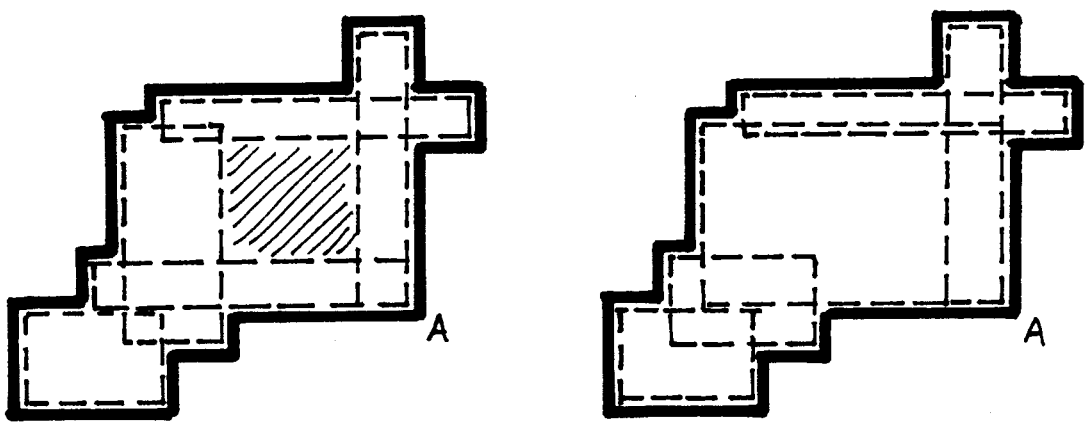


Figure 5-2: An Unnecessary Multiple Covering

The OREC on the left shows corner A multiply covered, which leaves a hole. The cover on the right is also an OREC, and has no hole.

5.1.3. Necessary Multiple Covers in an OREC

Figure 5-1 shows that some polygons may require multiple covering of a corner in every possible OREC. There is currently no known algorithm for finding and identifying all such corners. If such an algorithm existed, we could adapt the ORCC algorithm to find an OREC by first dealing with such corners before or during taking n-somes, and finally using the Cautious Method for pairing OPP2-somes, as in the ORCC algorithm.

There are two kinds of multiple coverings in OREC's, those caused by RI's (described in Chapter 2), and non-RI ones. However, Algorithm E1 (for irreducible polygons) does not have to consider non-RI multiple covers, and it will be shown that no range intersections can occur in an irreducible polygon.

Necessary multiple coverings are always either the result of RI's or they create a hole in the OREC. Any multiple covering of a corner in a RI-free polygon must leave a hole, as shown in figure 5-3. If a corner A's range is not intersected, and no one rectangle covers both edges, then there must exist some region in the range of A not covered by any rectangle. Thus no non-RI multiple covering will occur in an area cover.

In Algorithm E1, we consider the OREC problem only for a class of polygons called **irreducible** described in [CKSS81] which have been shown not to have non-RI multiple coverings in their OREC's. Work on finding non-RI multiple coverings is shown in more detail in the section on Algorithm E2.

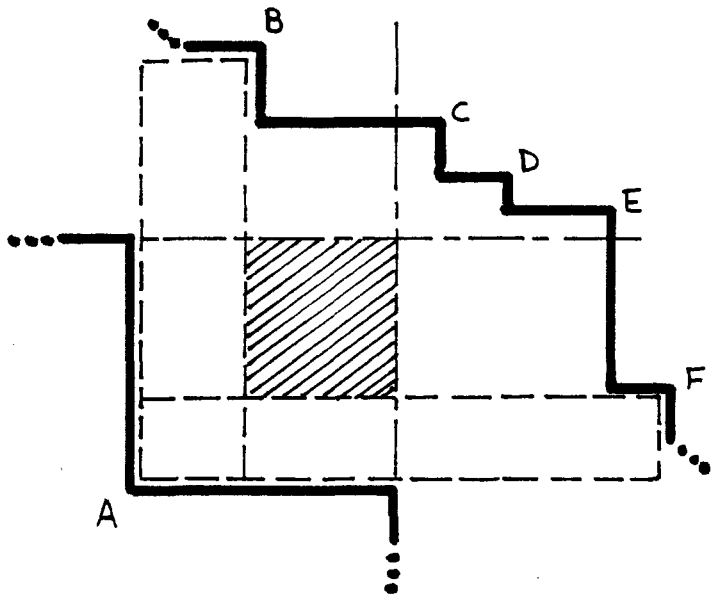


Figure 5-3: Non-RI Multiple Covering Leaves a Hole

We assume corners C to E are paired with corners to either side of corner A. Any of C, D or E could fully cover both edges of A.

5.2. Irreducible Polygons

An irreducible polygon is obtained from a general reducible one by two types of operation involving extremal edges or "tabs", tab reduction and partial tab reduction. When no further such operations are possible, the polygon is irreducible.

5.2.1. Tab Reduction

Given the rectangle R associated with an extremal edge, if its side opposite the extremal edge lies entirely on another edge of the polygon, then tab reduction may be performed.

This is done by removing all the area of the rectangle R from the polygon, as shown in fig 5-4.

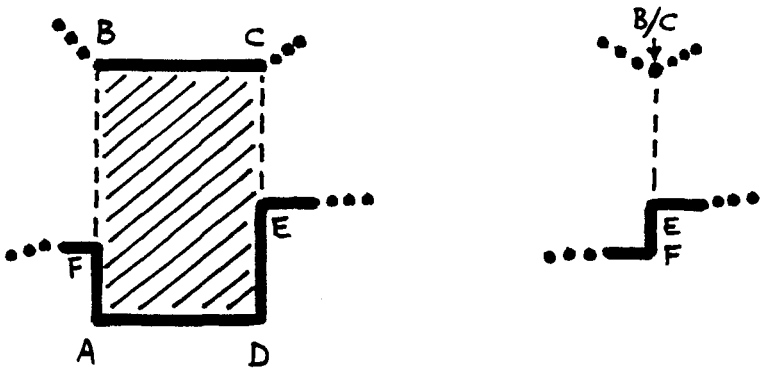


Figure 5-4: Tab Reduction

5.2.2. Partial Tab Reduction

Partial tab reduction may be performed when two conditions hold.

1. The polygon has no tab reductions possible.

2. An extension of the tab rectangle does not intersect any points of the tab rectangle to its right CCW (see fig 5-5).

Partial tab reduction is performed by removing the shaded area as shown in the figure.

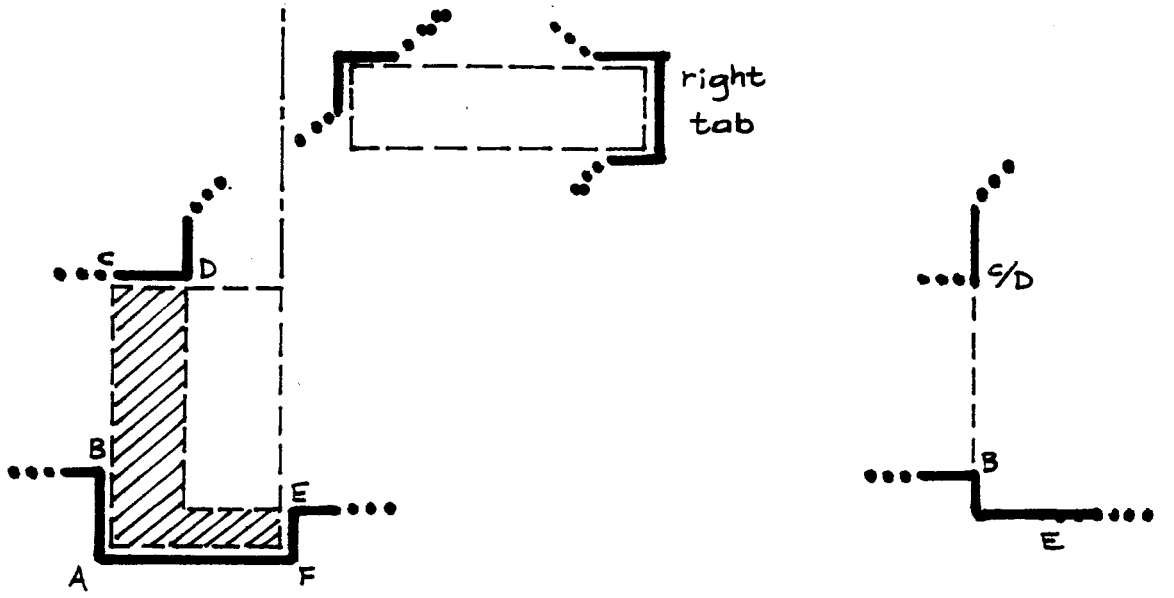


Figure 5-5: Partial Tab Reduction

The right tab must not intersect an extension of the bottom tab. The shaded area is removed in a partial tab reduction.

Now it is simple to prove the following theorem.

Theorem 1: No range intersections can occur in an irreducible polygon.

Proof: By contradiction. We refer to the conditions for irreducibility. If corner A's range is intersected (see figure 5-6) then either the bottom tab cannot see the left tab, and the polygon is reducible, or else edge AB is the bottom tab, and there is a full tab reduction possible for the right tab. Thus, if A's corner has a range intersection, A cannot be in an irreducible polygon. □

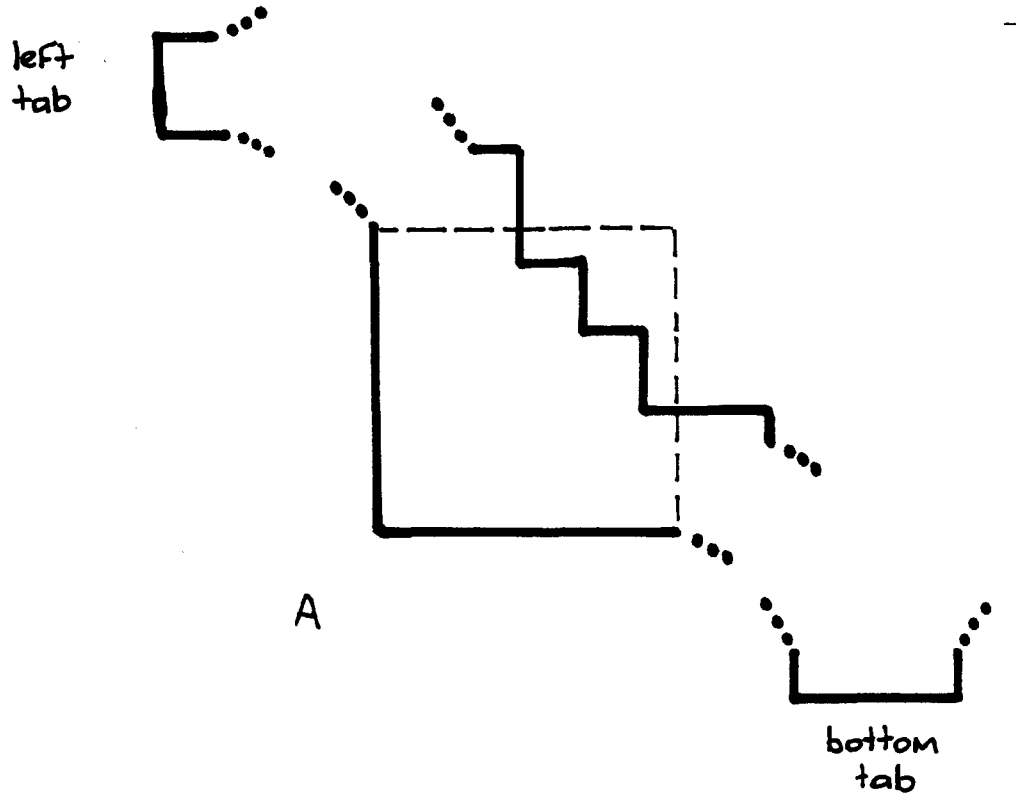


Figure 5-6: If A Has a RI, the Polygon is Reducible

We now present Algorithm E1 for irreducible, RI-free polygons.

5.3. Algorithm E1: an OREC for Irreducible Polygons

In an irreducible polygon, there are no range intersections. We note that in a non-RI multiple cover there is always a full or partial tab reduction possible. Thus there will also be no non-RI multiple covers. We look at the n-somes that must be in an OREC for irreducible polygons.

5.3.1. 4-somes

Due to convexity, a foursome must be unobscured. The rectangle defined by a 4-some must be in any optimal cover, as a maximal rectangle expanded from any of the corner vertices expands to cover all four corners and their edges.

5.3.2. 3-somes

The rectangle defined by a 3-some may be obscured, even in an irreducible polygon. We have two cases, the obscured and the unobscured.

The unobscured 3-some's rectangle must be in the cover, as any maximal rectangle covering the middle corner's edges can only expand to cover the other two corner's vertices, and the portions of their edges that lie along the rectangle.

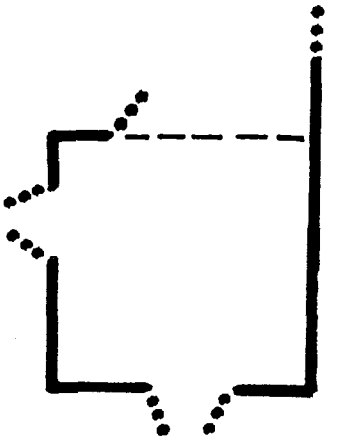


Figure 5-7: An Unobscured 3-some with an Extending Edge

One edge may extend beyond the rectangle in a 3-some, as in Figure 5-7. An

extending edge in this polygon means that the left tab has a full tab reduction against the extending edge. In an irreducible polygon, the extending edge cannot occur.

Obscured 3-somes may be part of connected 3-somes, as shown in figure 2-10. If the 3-some is obscured by the remaining chain, as in Figure 5-8 we note that the obscuring chain must cross the 3-some's rectangle at some points X and Y.

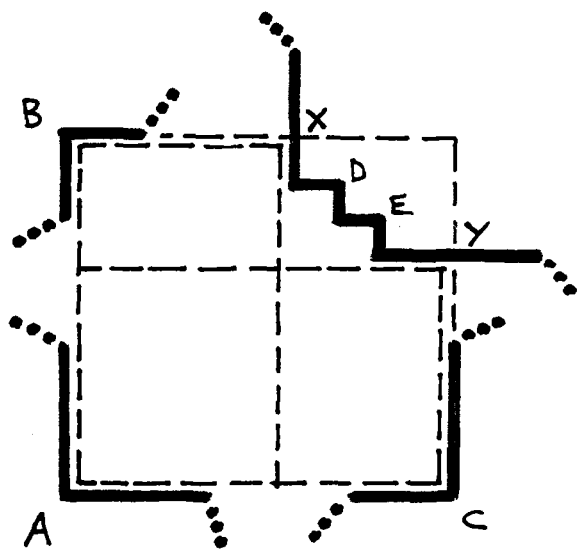


Figure 5-8: Covering an Obscured 3-some

The rectangles covering corners B and C can only expand to A in one direction, and to X and Y in the other, (given convexity) and must be in the minimum edge cover.

Several cases are possible. The 3-some may be part of a connected sequence of two

or three adjoining 3-somes, or it may be isolated. If three connecting 3-somes occur, the middle obscured one is fully covered by the unobscured rectangles of the outer two 3-somes.

If only two 3-somes are connected, one is obscured and one is unobscured (refer to figure 2-10). The unobscured rectangle is taken, and the one remaining uncovered corner of the obscured 3-some is not tagged with a 3. This means it will be left as a singleton (it cannot pair as any OPP2-some) and can only be expanded to the obscured 3-some's central corner. All edges are optimally covered in this way.

If the obscured 3-some is isolated, it must be covered by two maximal rectangles. This is done by ignoring the 3-some which results in its being listed as two (adjoining) CA2-somes. Corner A must be fully covered by each of the CA2-some rectangles in an irreducible polygon, (otherwise a tab may fully reduce against A, and the polygon is reducible).

We see that the ORCC algorithms suffice to find all the necessary rectangles to edge cover 3-somes in irreducible polygons.

5.3.3. Collinear Adjacent 2-somes

We saw in Chapter 2 that every CA2-some defines a rectangle that must be in any ORCC, OREC or ORAC. Now a corner's edge cannot extend past the CA2-some's rectangle in an irreducible polygon, or a tab may fully reduce against the extending edge. Thus the edges of both corners are fully covered. Again, the ORCC algorithms suffice in irreducible polygons.

5.3.4. OPP2-somes

The ORCC algorithm for finding OPP2-somes must be slightly modified for edge covers. The scope of a corner is defined differently, as a rectangle must cover all edges of both corners in an OPP2-some for edge covers (see Chapter 2). The change is slight, and the algorithm can still be performed in linear time.

5.3.5. Algorithm E1

We have now seen all the rectangles that must be in any OREC for a given irreducible convex polygon. 4-somes, 3-somes and CA2-somes are found and covered with variations of the ORCC algorithms. Then all that remains is OPP2-somes and singletons. An application of the Cautious Method from the linear ORCC algorithm will provide the most efficient rectangle cover of the remaining edges.

1. Find and cover 4-somes, unobscured 3-somes and CA2-somes. Expand these rectangles and mark the appropriate corners covered.

2. Find the set of all possible OPP2-somes among the still uncovered corners.

3. Find a maximum independent set (MIS) among the OPP2-somes. List these rectangles and mark the corners covered.

4. Expand the remaining uncovered singletons.

5.3.6. Optimality of Algorithm E1

The 4-somes, unobscured 3-somes and CA2-somes must be covered as described in every OREC for irreducible polygons. The remaining corners can only be covered as part of an OPP2-some, or as singletons. The Cautious Method gives the largest number of independent OPP2-somes possible, and every remaining singleton can be covered with one maximal rectangle as they cannot have range intersections. Thus, as in the ORCC algorithm, we obtain a most economical cover for the remaining corners.

5.3.7. Complexity of Algorithm E1

Finding all 4-somes, unobscured 3-somes and CA2-somes is linear. We use the procedures described in the linear ORCC algorithm. The algorithm for finding OPP2-somes has only to be slightly modified for the OREC problem. For an OREC, the scope of a corner is calculated differently (see Chapter 2). The Cautious Method may be applied as in the ORCC, giving a MIS of OPP2-somes in linear time. Expanding singletons may also be done in one tour of the polygon in linear time. The total time is linear.

5.4. Algorithm E2: A Stronger Result

A stronger result may yet be obtained. In a general, reducible polygon, range intersections may occur, and non-RI necessary multiple covering may occur for certain corners. Range intersection regions must always be covered by a certain set of rectangles. RI's and these rectangles may be found in linear time.

Non-RI multiple coverings come in six configurations. Five of the six possible

configurations occur as part of obscured 3-somes and of CA2-somes with extending edges. They may also be found and handled in linear time during the algorithm. There is at present no known algorithm to find the sixth configuration, which we call an OPP3-some multiple covering. These terms are explained later in the chapter.

First we look at range intersections.

5.4.1. Range Intersections in OREC's

When a corner A's range is intersected by some portion of the opposite chain, then A's two edges cannot be covered by a single rectangle. If A's range is intersected by some portion of the opposite chain Aj, then an optimal edge or area cover of the region in the range of A must be the rectangles defined between A and all opposite corners in Aj whose vertices and edges lie fully within A's range (they can expand nowhere else), plus a rectangle along any edges crossing A's range. The rectangles along edges crossing A's range may or may not meet another corner (see Figure 5-9).

Rectangles expanding to a corner B whose edge crosses the range of corner A may only partially cover B's other edge (see figure 5-10). In this case the range of B is then also intersected by an edge of corner A and some part of the continuing chain. In fact, several RI's may be strung together (see Figure 5-11).

Eventually these linking RI's must end with a corner fully covered by the last rectangle. We can cover the whole succession of linking RI's as we encounter each one, noting that where two range intersections of opposite type join we must be careful not to list the same rectangle twice.

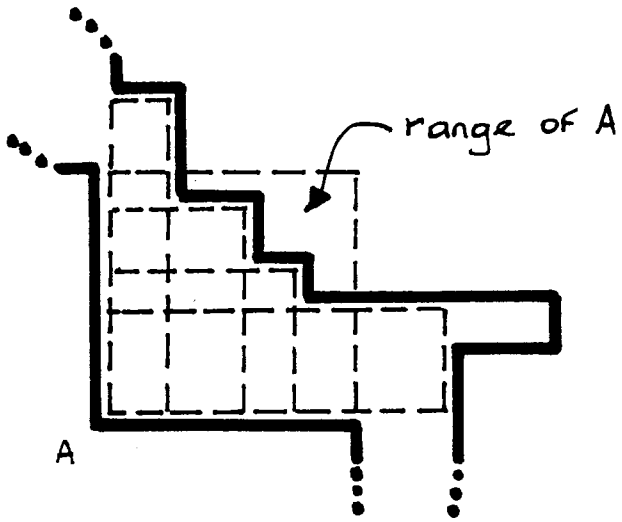


Figure 5-9: Range Intersection Edge and Area Covering

Corners in A's range can only expand to A. Edges crossing A's range also require a rectangle expanded from A in order to cover the outer part of A's edges. These rectangles do not necessarily meet another corner at the end of their expansion.

5.4.2. Decomposing into RI-free Components

Knowing the rectangles that must be in any optimal edge or area cover of a range intersection part of a polygon, we can cover these areas first. The covering of RI's and linking RI's decomposes the polygon into RI-free components left to be covered.

The general edge covering problem is then reduced to finding the rectangles needed

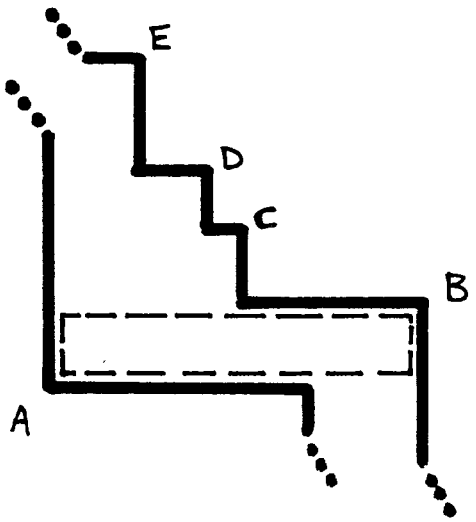


Figure 5-10: Two Adjoining Range Intersections

Corner A is range intersected by corners B,C,D and E. If the corner B's other edge is only partially covered, then B is also range intersected by A.

to cover range-intersection regions, then finding an OREC for the remaining RI-free uncovered components. Figure 5-12 shows the decomposition of a polygon into RI-free components whose OREC's may be combined with the RI's edge cover to form an OREC for the entire polygon.

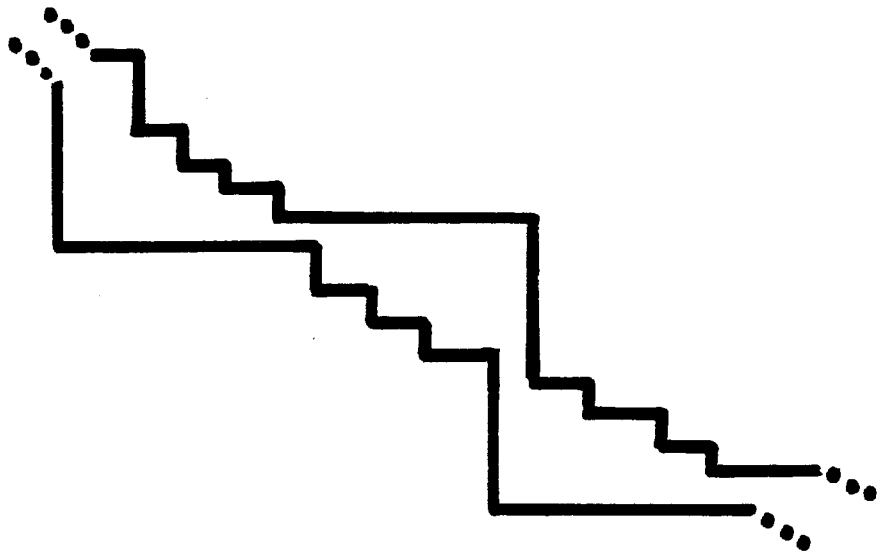


Figure 5-11: Several Connecting Range Intersections

5.4.3. Finding and Covering RI's in Linear Time

All RI's in a convex rectilinear polygon may be found and covered in linear time. We assume the chain and corner numbering is as shown in figure 4-1. A corner's range can only be intersected by a corner of the opposite chain.

Lemma 2: If a range intersection is found between a pair of opposing chains, no range intersection may occur between the other two chains.

Proof: If a range intersection occurs betwen two chains, the other two chains must reside on different sides of the found RI, and thus cannot intersect each other's range. □

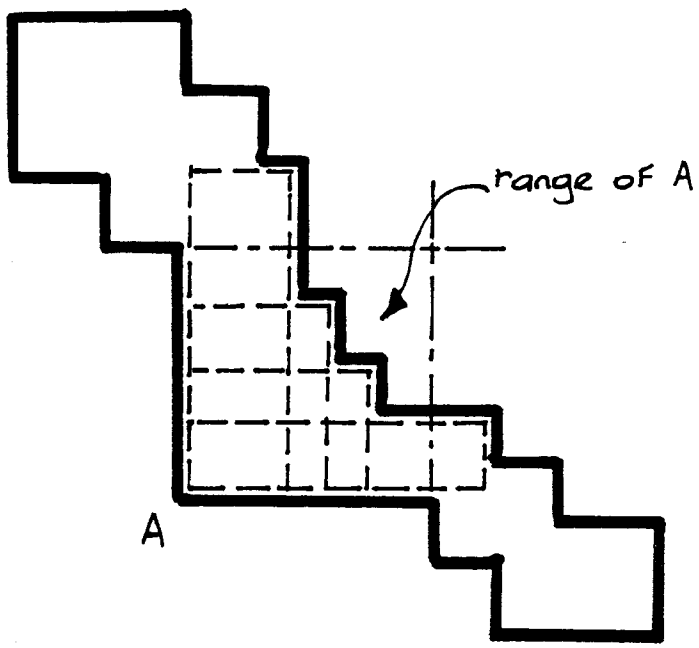


Figure 5-12: RI Decomposition

Lemma 2 shows that it is only necessary to check the second pair of chains for range intersections if none have been found between the first pair.

For a pair of opposing chains, say A1 and A3, sweep along both chains simultaneously from left to right. One sweep will find RI's of A1. A second sweep will find RI's of A3. The second sweep is modified so that adjoining RI's do not result in the same rectangle being listed twice.

It is easy to find range intersections of all four tabs in linear time, so they are taken care of first. Thus the corners looked at will be numbered from 2 to Ti-1. The following algorithm is written in terms of A1 and A3, looking for RI's of A1.

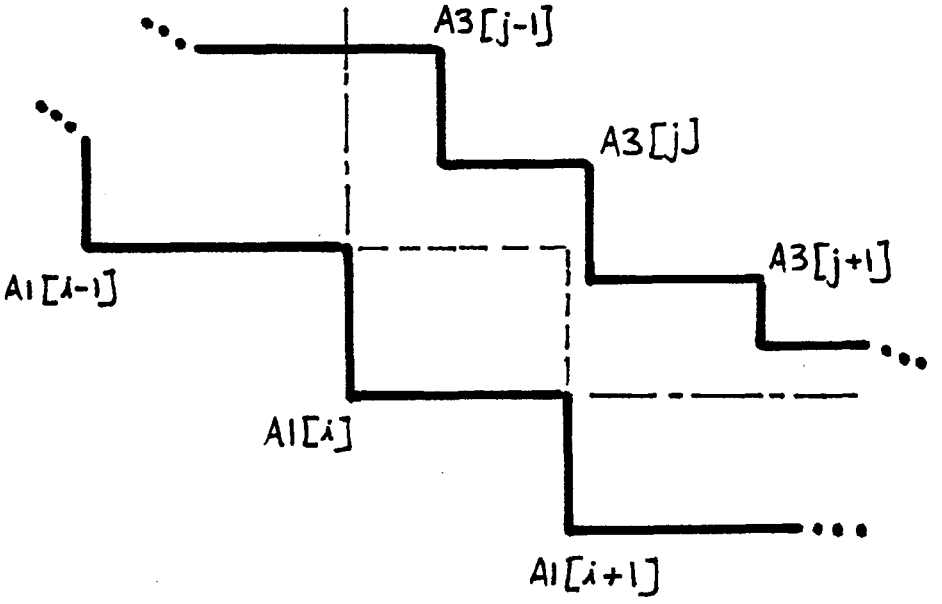


Figure 5-13: Index Numbering in the RIfind Algorithm

Figure 5-13 illustrates the numbering involved in the search.

```
begin RIfind

i=2  /* start at first corner after tab */
j=1  /* start where other chain begins  */

do while(i < T1 & j < T3)  /* for the corners in chain A1 */

  if A3[j].x < A1[i+1].x
    then do
      if A3[j+1].y < A1[i-1].y
        report range intersection
        of corner A1[i] by A3[j]
      j=j+1
    end do
    else i=i+1

endwhile

end RIfind
```

5.4.4. Finding Non-RI Multiple Coverings in an OREC

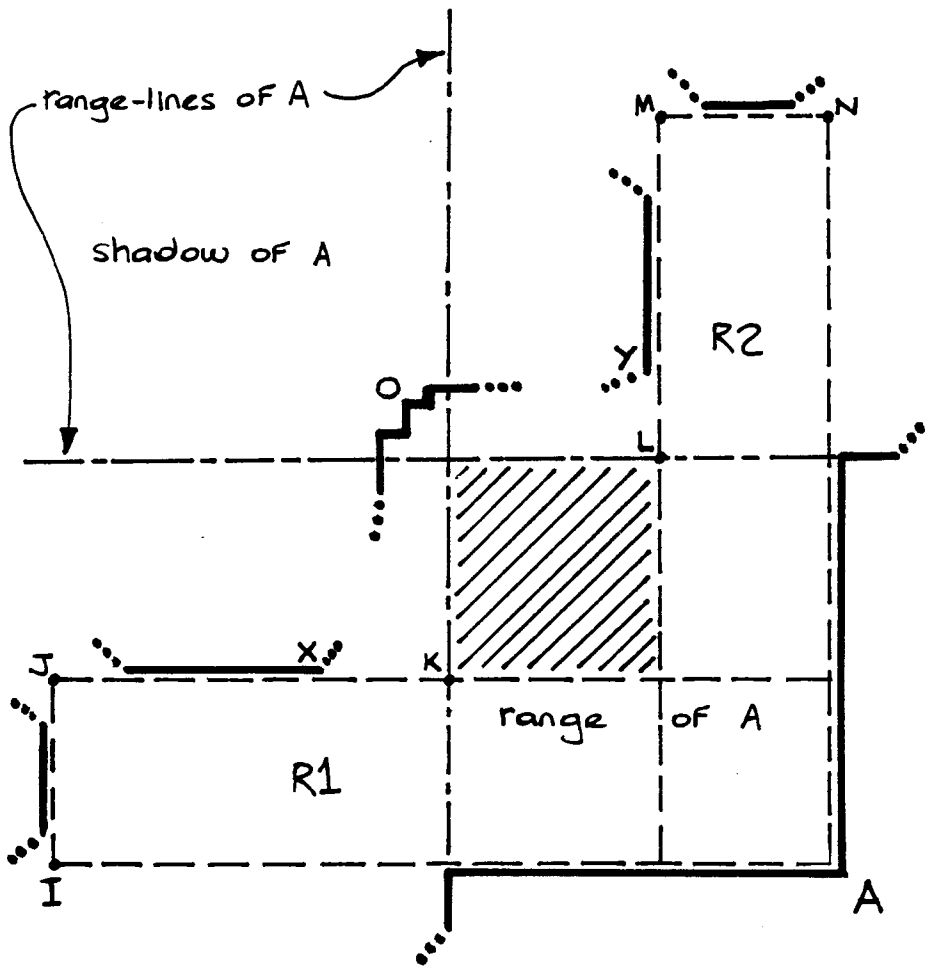


Figure 5-14: Constraints on a Non-RI Multiple Covering

In a given convex rectilinear polygon with a non-RI necessary multiple covering of corner A for any OREC, the following constraints must occur (see Figure 5-14):

- A must have one edge fully covered by some maximal rectangle R1, and the other edge fully covered by some other maximal rectangle R2.

- As neither R1 nor R2 fully covers A by itself, the range of A extends beyond (R1 U R2).

- Both R1 and R2 must be bounded at their extremal ends by some edge, and be bounded on their sides away from A's edges by some edge. As R1 and R2 are maximal, we require an edge of the polygon to occur on each of the lines IJ, JK, LM and MN. (These edges may extend past the lines, but not into A's range.) Let X and Y be the closest points to A of the edges along JK and LM respectively.

- As the polygon is convex, and there is no RI of A, we see that there must always be at least one corner O in the shadow of A that could be paired with A as an OPP2-some to cover fully the edges of both corners.

- If corner A is multiply covered in a particular edge cover, then every such corner O must have been paired instead with some other corner on either side of A, or be multiply covered itself from both sides of A as in figure 5-1, where both A and B are necessary multiple covered. If O was a singleton, it could have been expanded to A, fully covering it in some OREC, and this would not have been a necessary multiple cover of A.

At least one of I and J must be a corner, as must at least one of M and N. There are six basic configurations possible. Figure 5-15 illustrates them.

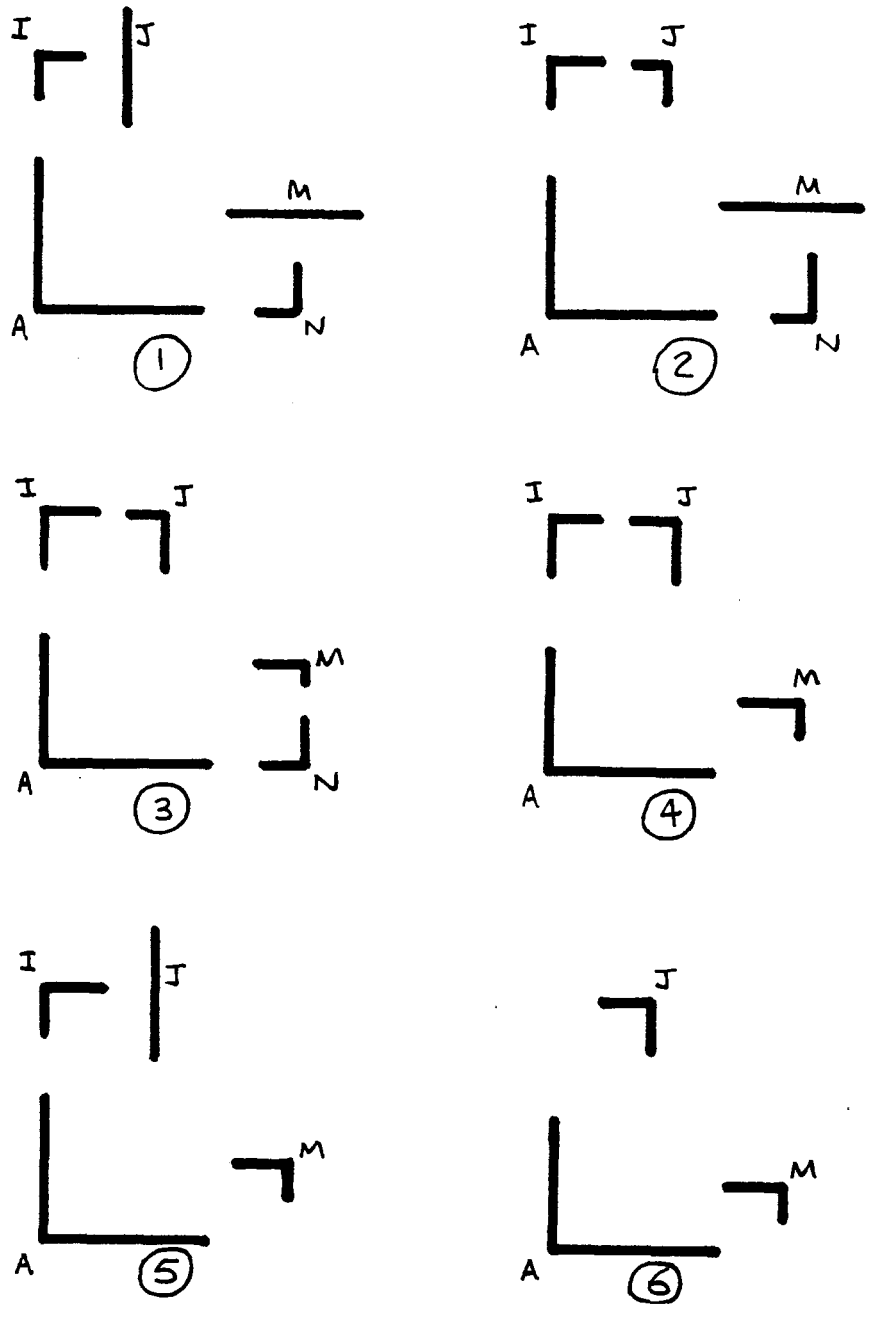


Figure 5-15: The Six Basic Multiple Covering Configurations

5.4.5. Handling the six configurations

Five of the six configurations are recognizable as involving 3-somes and CA2-somes, and may be optimally covered during the n-some finding algorithms in a manner that permits multiple covering of the corner. It is necessary to find if 3-somes adjoin each other before dealing with them. This only takes one extra pass of the algorithm.

If I and N are corners, IAN is an obscured 3-some (configurations 1 to 3). In configurations 2,3 and 4 there are other 3-somes involved. Configuration 5 has a CA2-some with an extending edge. The configurations are dealt with as follows:

Configuration 1: The 3-some is a simple obscured 3-some. The 3-some is dealt with when found to be obscured, as if it were two adjoining CA2-somes. All three corners are marked covered by the two rectangles thus defined. This allows the possible multiple covering of the middle corner.

Configuration 2: Appears as two connected 3-somes. When two or three connected 3-somes are found during the 3-some finding algorithm, all corners in the unobscured 3-somes are marked covered, and the un obscured 3-some corners are labelled with a 3 so that they will not be processed later as CA2-somes. In the case here of two connected 3-somes, the unobscured one (JIA) will have an extended edge. We solve this by taking the CA2-some rectangle defined with the yet uncovered corner N of the obscured 3-some at this time, and labeling this corner with a 3 so that it doesn't get made part of a CA2-some later.

Configuration 3: Is covered under configuration 2. When three connecting 3-somes are found, the two rectangles are listed, and all corners involved are marked covered. This allows the possible multiple covering of A.

Configuration 4: This appears as a single unobscured 3-some with an extending edge. If we perturb the extending edge very slightly, at the point where it would be intersected by the 3-some's rectangle, and create a new corner, the rest of this edge gets covered, and corner A is allowed to be multiply covered. Thus, whenever we find a single 3-some with an extending edge we allow the extending edge to be perhaps multiply covered.

Configuration 5: This is a CA2-some with an extending edge. As in configuration 4, if the extending edge is perturbed slightly at the point where it leaves the rectangle, the edge gets fully covered later, and corner A may be multiply covered. Figure 5-16 illustrates perturbation for both configurations 4 and 5.

Configuration 6: This is the configuration we call an **OPP3-some** multiple covering. No efficient algorithm is known which can find an OREC for a polygon that must have this kind of necessary multiply covered corner in its OREC, aside from the obvious brute-force method.

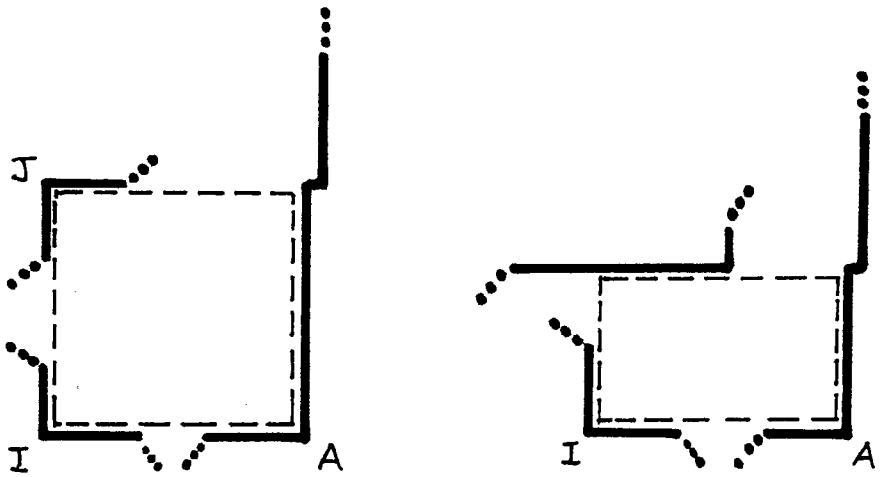


Figure 5-16: Perturbation for Configurations 4 and 5

5.4.6. Algorithm E2: for General Convex Rectilinear Polygons

For general, reducible polygons then, an edge cover may be obtained which is optimal for all polygons except those which have configuration 6 multiple coverings of corners in every OREC.

The algorithm steps are as follows:

1. Find and cover all range intersection regions of the polygon. Mark the appropriate corners covered.

2. Find and cover all 4-somes.

3. Find all 3-somes in the polygon. Find all connected 3-somes among these, and deal with them as described in the list of configurations for necessary multiple coverings.

4. Cover remaining unobscured 3-somes, perturbing extended edges to create new corners, allowing multiple covering to occur.

5. Find all remaining untagged CA2-somes. Among these, find those with extending edges. Perturb these edges to create new corners, allowing multiple covering to occur. List the rectangles and mark the appropriate corners covered.

6. Find the set of all possible OPP2-somes among the uncovered corners remaining.

7. Find a MIS among these with the Cautious Method, as in the ORCC algorithm. List these rectangles and mark the corners covered.

8. Expand the remaining singletons' rectangles and list them.

9. Remove the perturbed corners, allowing their rectangles to expand fully, either fully or partially covering the corners they meet.

5.4.7. Cardinality of Necessary Multiple Covers

Given that necessary multiple covers may occur in an optimal edge cover, it is useful to know how many can occur. In particular, as we cannot recognize the OPP3-some configuration, we need to know how many we may miss in Algorithm E2, and thus how far from optimal the edge cover obtained might be.

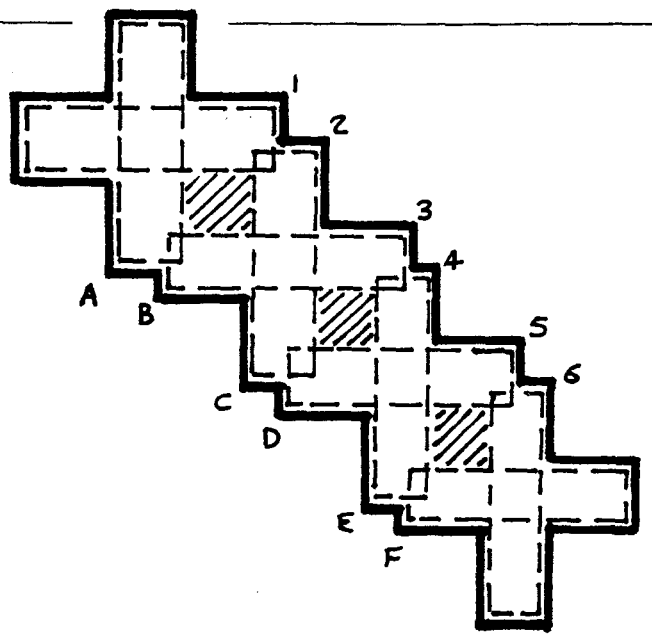


Figure 5-17: An Edge Cover with O(n) Unnecessary Holes

If an OPP3-some configuration of a multiple cover of a corner A occurs in every OREC for a given polygon, certain conditions must be observed. Figure 5-18 shows that, for corner A to be necessary multiple covered in an OREC, blocking of the OPP2-some rectangles defined by J and P and by M and R must occur as shown, or else an alternate OREC exists in which A is not multiply covered.

An unnecessary OPP3-some, where blocking does not occur as shown, is illustrated

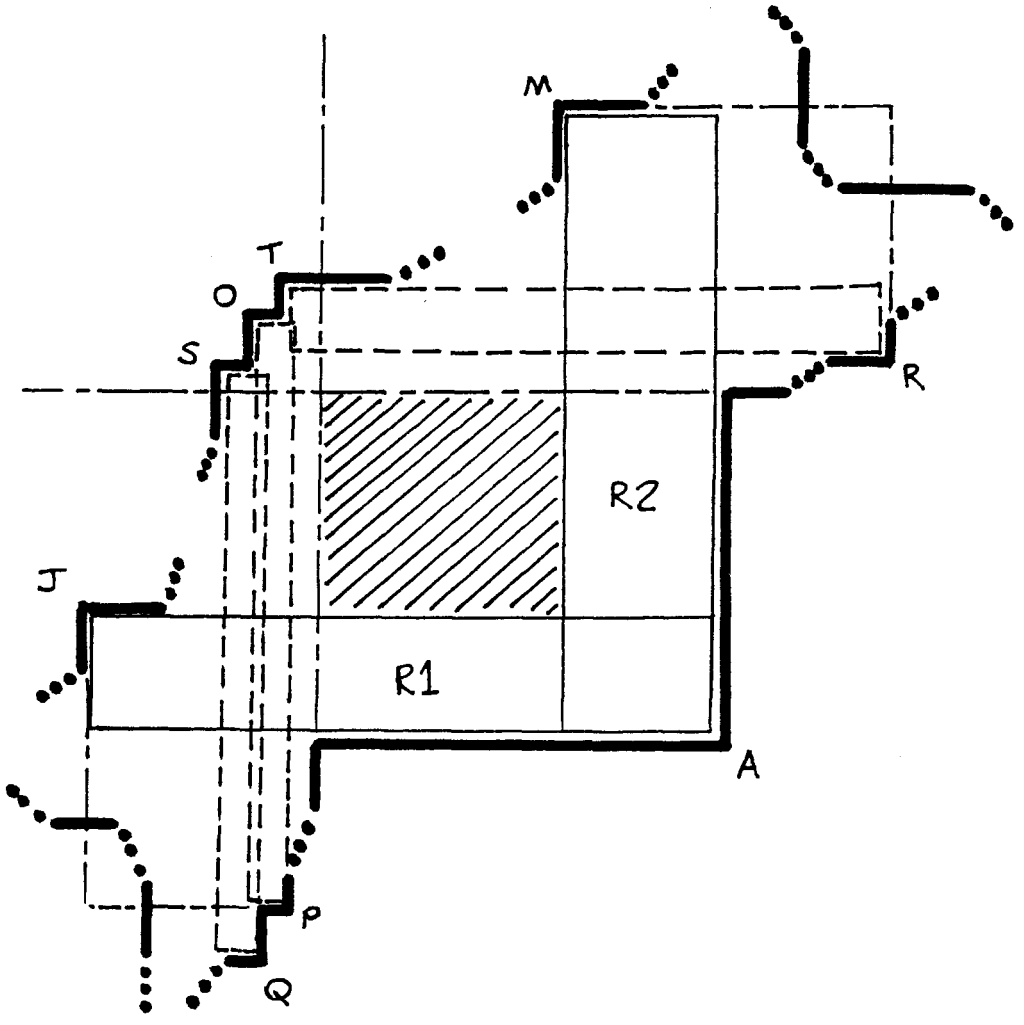


Figure 5-18: Conditions for a Necessary OPP3-some Multiple Cover

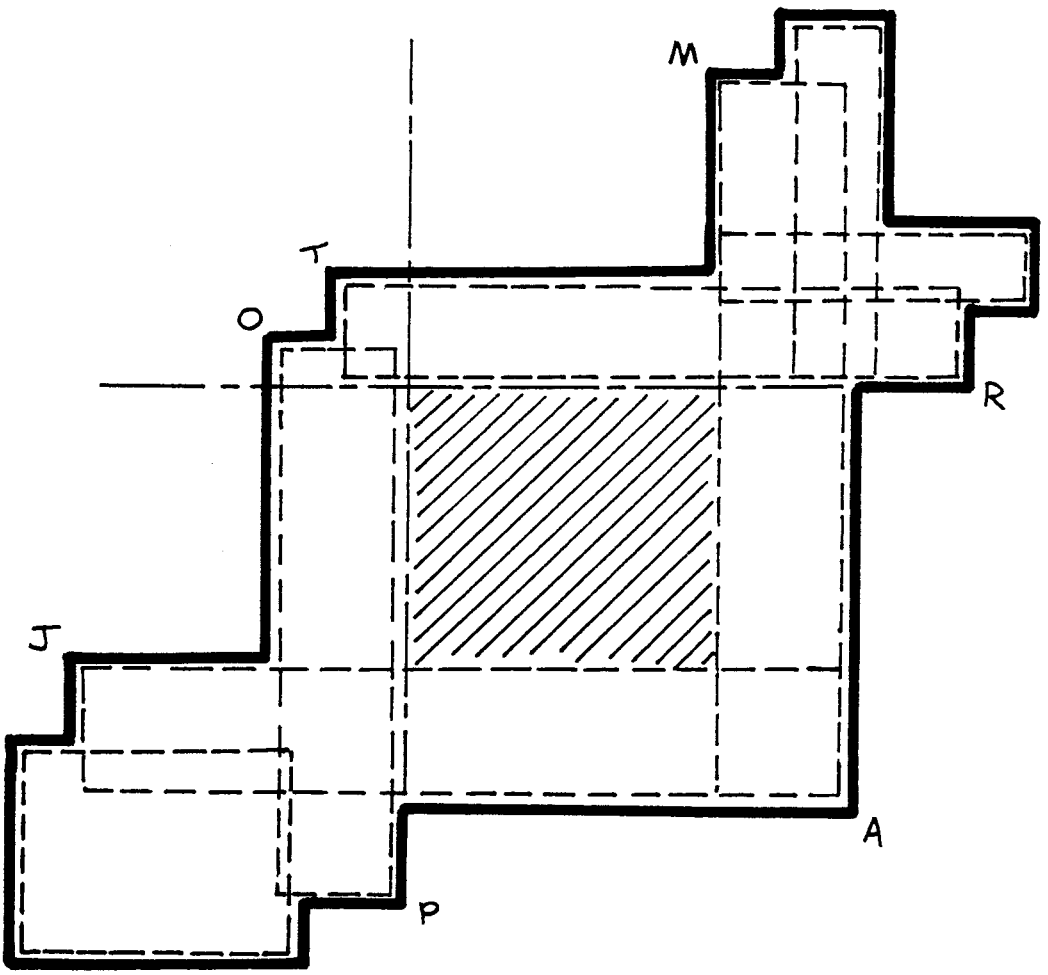


Figure 5-19: An Unnecessary OPP3-some in an OREC

in figure 5-19. If corners J and P are not blocked from forming an OPP2-some, then we may pair J with P, and O with A to obtain an OREC with no holes. Then the multiple cover is not necessary.

Theorem 3: No more than two necessary OPP3-some multiple coverings of corners can occur in an OREC. Further, if two occur, the two multiply covered corners must be opposite each other, and both be fully edge coverable by the legitimate (unobscured) OPP2-some rectangle they define.

Proof: To prove this theorem, we must show the following:

1. Two OPP3-some necessary multiple covered corners may occur in an OREC.

2. Two OPP3-some necessary multiple covered corners cannot occur in the same chain.

3. Two OPP3-some necessary multiple covered corners cannot occur in adjacent chains.

4. Two OPP3-some necessary multiple covered corners cannot occur in the opposite chains unless these corners are coverable by one rectangle.

We prove property 1 by example. Figure 5-20 shows an OREC with two OPP3-some necessary multiple covered corners. These two corners may be both fully covered by one OPP2-some rectangle.

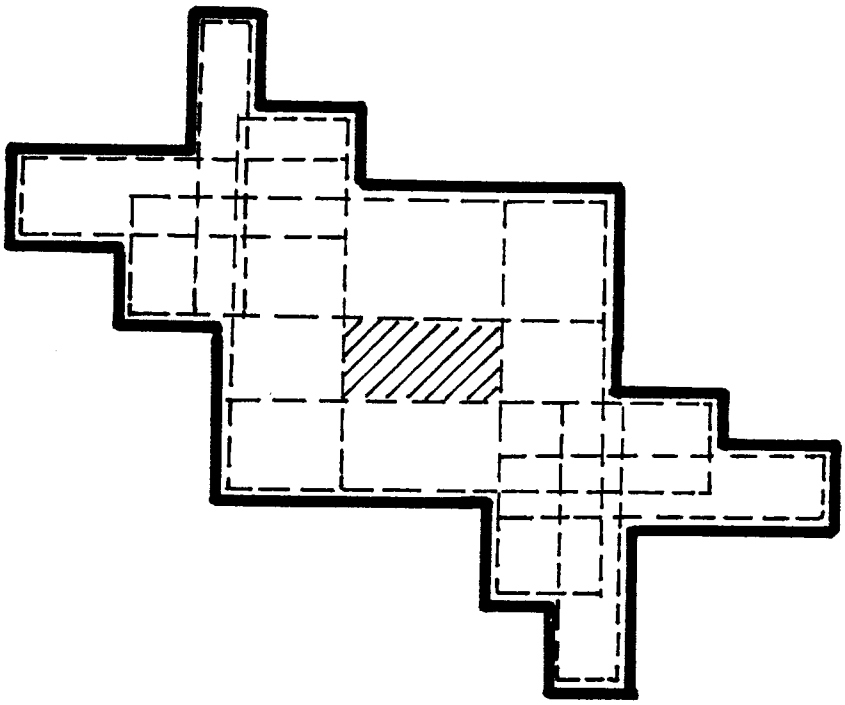


Figure 5-20: An OREC with Two OPP3-some Necessary Multiple Covers

Property 2 is shown by the fact that blocking must occur at both ends of the chain containing the necessary multiple covered corner. If corner A is multiply covered by opposite corners J and M, and corner A' in the same chain as A is also multiply covered by opposite corners J' and M', we note that the blocking required for the covers to be necessary cannot exist.

Property 3 is shown by the fact that a necessary multiply covered corner always has OPP2-somes possible with the opposite chain. It is easy to see that both sets of opposite chains cannot define unobscured OPP2-somes in the configuration required for a necessary multiple cover to exist in an OREC.

Finally, property 4 is shown by the fact that if two opposite necessary multiple covered corners are not coverable by a single OPP2-some rectangle, the the corners' vertices are not in each others' shadows, and the blocking requirements for the multiple covers to be necessary cannot be met. If the two vertices are in each others' shadows, they define a legitimate unobscured OPP2-some rectangle which fully covers the edges of both corners.□

This proof leads immediately to a stronger result, as the arguments used all apply to multiply covered corners of any configuration. The argument for property 2 needs a little reworking as J and M may not be corners in the general case. However, blocking must occur from the other two chains in the region shown, and cannot occur as required for two corners in the same chain to be necessary multiply covered.

Theorem 4: No more than two necessary multiply covered corners can occur in an OREC for a convex rectilinear polygon, and if two occur, their multiply covered corners must be opposite each other and be edge coverable by the unobscured OPP2-some rectangle they define.

Corollary: Any scheme that finds an OREC for a convex rectilinear polygon, ignoring necessary multiply covered corners, is at most one rectangle from optimal.

5.4.8. Optimality of Algorithm E2

Range intersection regions must be covered as described. The 4-somes, 3-somes and CA2-somes must also be covered as described. The Cautious Method finds the most economical cover of the remaining corners, **except** where an OPP3-some necessary multiple covering of some corner was in every OREC. The algorithm is then only optimal for polygons in which this situation does not occur.

We have shown that only one or two OPP3-somes can occur in a polygon, and if two occur, the multiply covered corners must be opposite each other, and must both be coverable by the OPP2-some rectangle they define. Thus our algorithm is guaranteed to be within one rectangle of optimal for general, reducible polygons.

5.4.9. Complexity of Algorithm E2

Range intersection finding and covering has been shown to have a linear implementation. Finding and dealing with n-somes is performed with modified versions of the Algorithm E1 programs. Finding connected 3-somes requires a look-ahead, but is still linear, as is the covering of them required. The number of new corners created by extended edges is linear. Finding OPP2-somes and using the Cautious Method as in Algorithm E1 is linear. Total time is therefore linear.

Chapter 6

Area Cover Algorithms

6.1. A Look at Area Cover Problems

An edge cover is not necessarily an area cover, so Algorithms E1 and E2 will not always produce ORAC's or even non-optimal area covers. Algorithm E2 found edge covers for general reducible polygons that may have necessary multiply covered corners in their OREC's. As non-RI multiply covered corners always create holes in the edge cover, we confine our attentions to Algorithm E1 and irreducible polygons.

If we are to adapt Algorithm E1 to find an area cover, we must be able to find and cover the hole or holes that may occur in an optimal edge cover. If the OREC has no holes, it is also an ORAC.

6.1.1. Holes in OREC's of Irreducible Polygons

We have seen that an OREC for a convex rectilinear polygon may have a hole in it. In fact, an OREC may have O(n) holes (see figure 6-1).

We define **necessary holes** as holes in a region where some holes must occur in any OREC for that particular polygon. We note that variations of the holes may exist. The hole boundaries may be variable among several OREC's, and the number of holes in the region might be different, but for holes to be **necessary**, any OREC for that

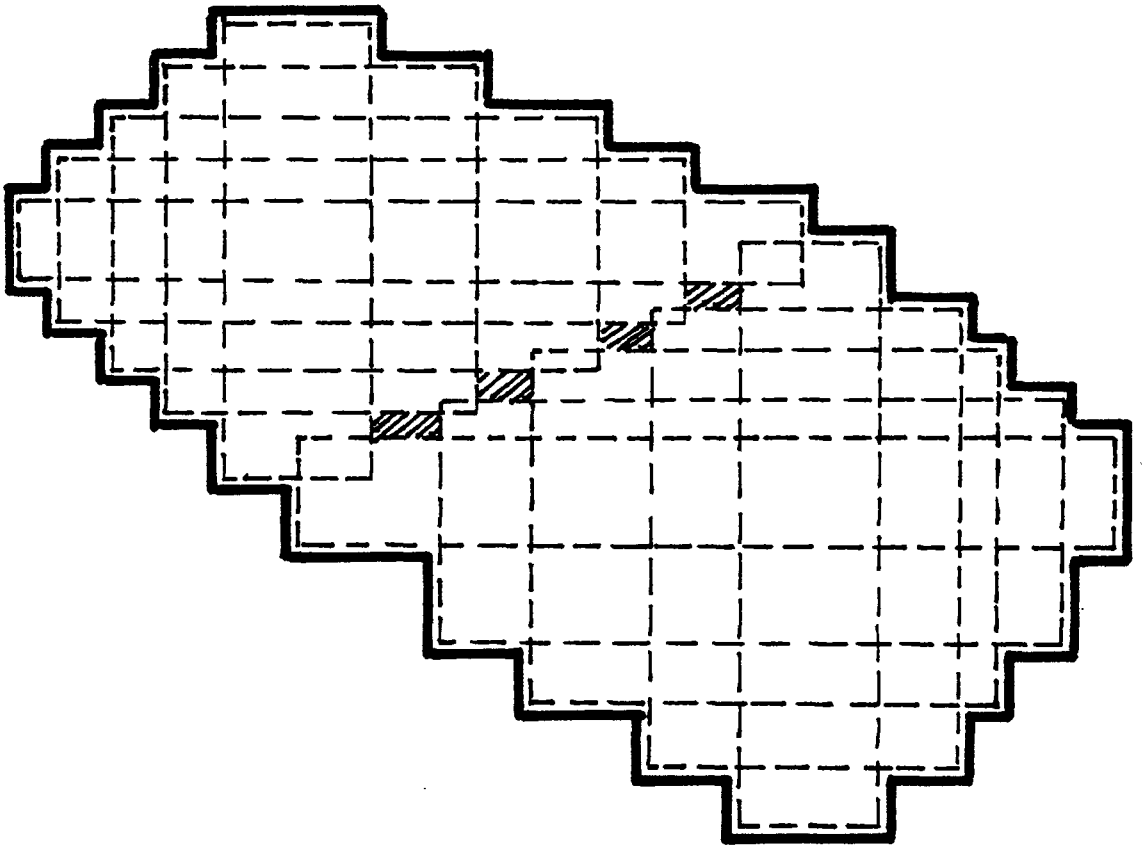


Figure 6-1: An OREC in an Irreducible Polygon with O(n) Holes

polygon will have a hole or holes in a certain region. Figure 6-2 shows two versions of a necessary hole in an irreducible polygon.

To convert an OREC with necessary or unnecessary holes to an area cover, we will need to find and cover these holes with extra rectangles. It is shown in [CKSS81] that, for some optimal edge cover, one rectangle will always cover all the holes.

We use in our area cover algorithms an important result of Chaiken et. al.

Lemma 1: An edge cover by maximal rectangles in an irreducible convex

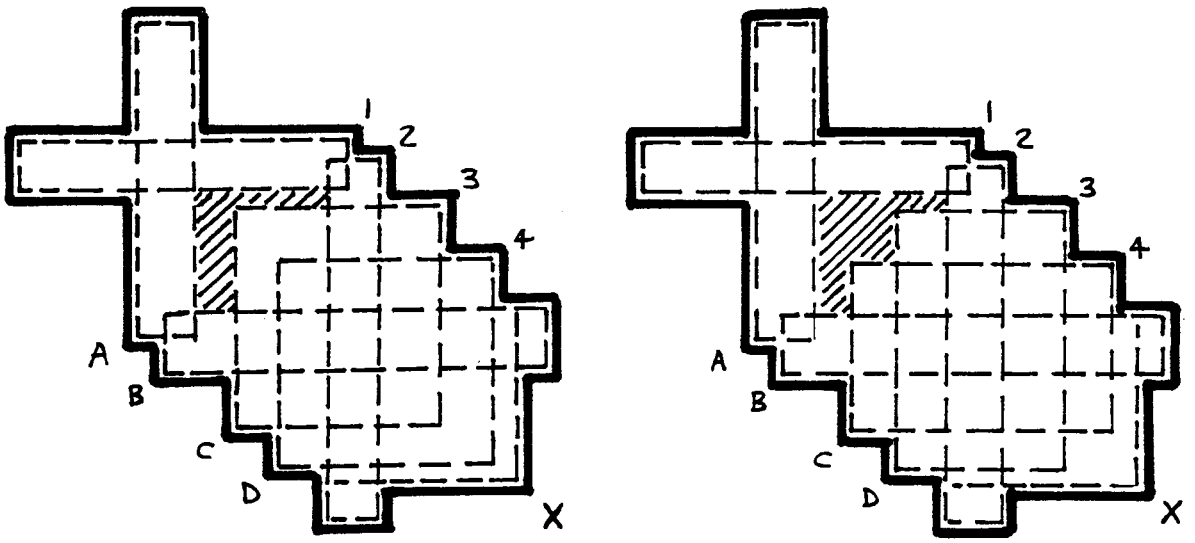


Figure 6-2: 2 Different Holes in OREC's

polygon can only have holes in the area between the four tab rectangles [CKSS81].

We call this region between the four tab rectangles the **hole region**. If this region is non-existent, there can be no holes.

A hole in an OREC may not be rectangular. It must be bounded on all sides by maximal rectangles. Given the convexity of the rectilinear polygon, we can show that only certain configurations of corners and edges of the polygon can exist. Figure 6-3 shows the constraints placed by convexity around a hole region.

These constraints mean that in a convex rectilinear polygon necessary holes must

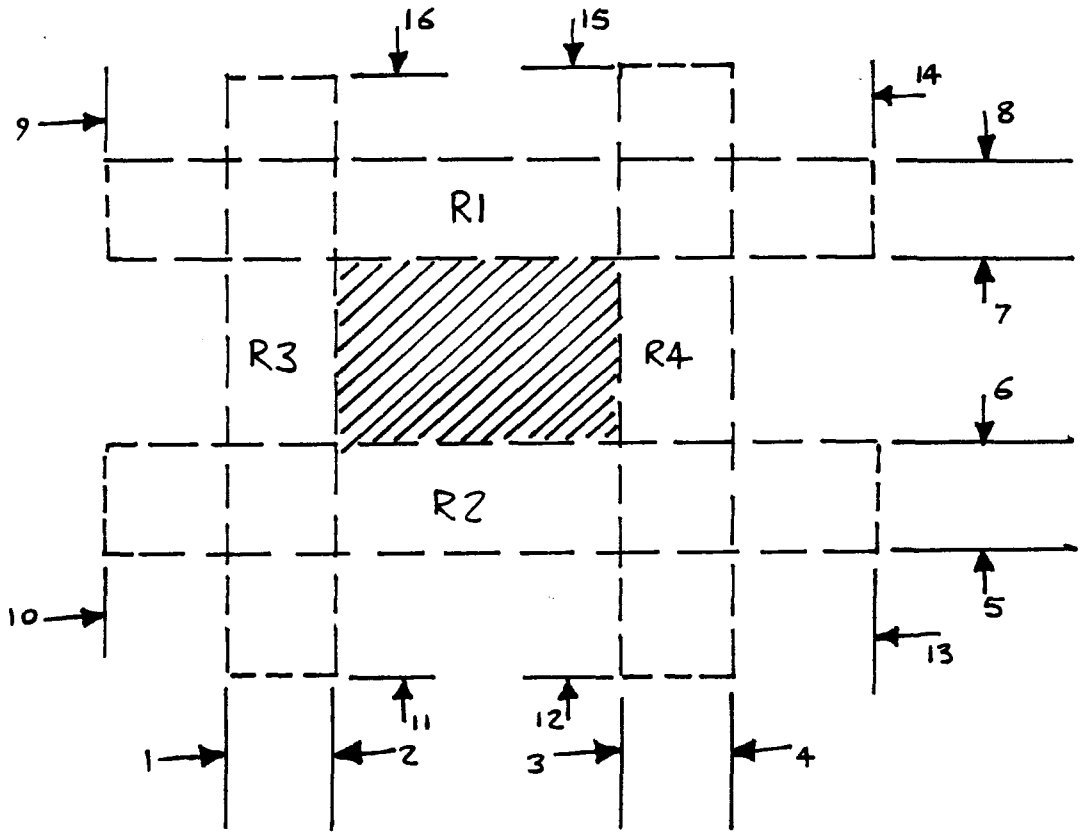


Figure 6-3: Hole Region Constraints in a Convex Irreducible Polygon

The maximal tab rectangles R1,R2,R3 and R4 surround the hole region. As they are maximal, edges of the polygon must appear along some portion of each of their edges. These constraints are shown by numbered lines. The rectangles may not be expanded exactly as shown.

occur in a variation of one of two configurations. These two configurations we call **Type 1** and **Type 2**. They are illustrated in figure 6-4.

Type 1 necessary holes: Type 1 holes are the result of blocking at both ends of

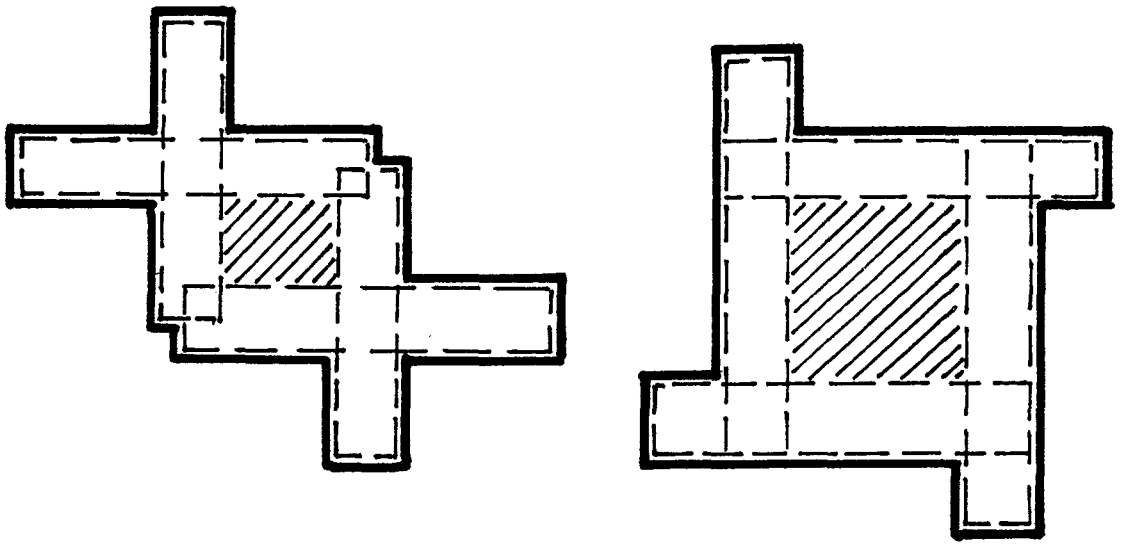


Figure 6-4: Examples of Type 1 and Type 2 Hole Configurations

the polygon, forcing two criss-crossed sets of rectangles to meet in the middle of the polygon forming a hole.

Type 2 necessary holes: In type 2 holes no OPP2-some rectangles can occur. Type 2 holes cannot occur in irreducible polygons, as there are always full tab reductions possible in this configuration.

6.2. Algorithm A1: an Area Cover for Irreducible Polygons

An area cover for an irreducible polygon may be constructed from an OREC (provided by Algorithm E1). If the OREC has no holes in it, it must also be an ORAC.

If it does have holes, we must find this out, and add a maximal rectangle covering the hole region. This area cover may be optimal, or it may be one rectangle more than optimal if the holes in the OREC were not necessary holes.

A simpler algorithm: A simpler version of this algorithm would be to just add a maximal rectangle to cover the hole region, if this region exists. Clearly, we can check this in constant time. This avoids the task of finding holes and is still guaranteed within one rectangle of optimal. However, when the OREC is already an ORAC, an additional unnecessary rectangle is added to the cover. Finding out whether holes exist enables the algorithm to return an optimal area cover in more cases.

We can further minimize the chance of there being holes by expanding the singletons in the OREC at this time, not when the OREC is obtained. They may be expanded in the direction that covers the largest possible part of the hole region.

Figure 6-5 shows the regions that are of interest when expanding singletons and looking for holes.

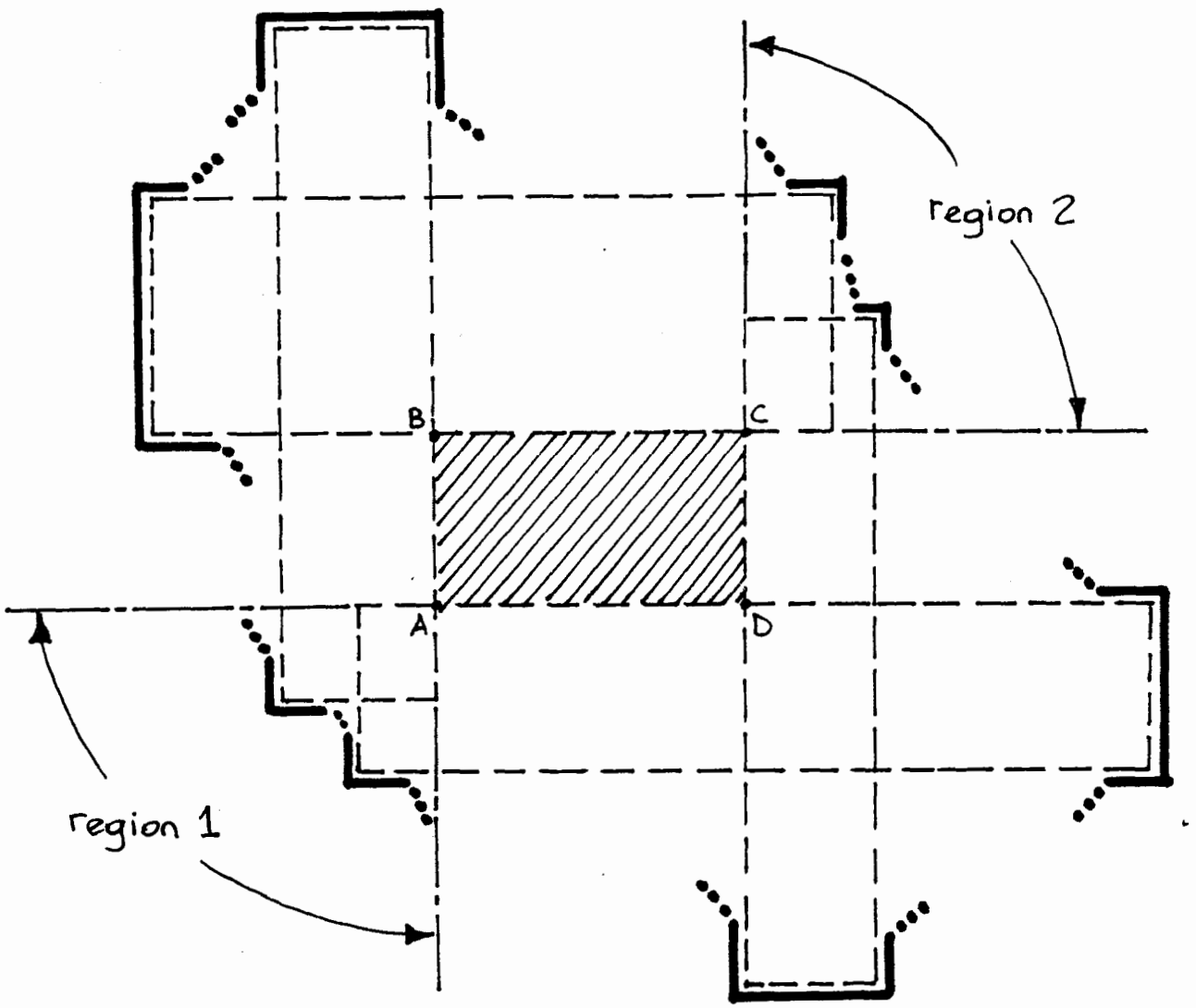


Figure 6-5: Corner Regions of Importance for Type 1 Holes

A mirror-image of this figure is also possible. Algorithms in the text are described in terms of this configuration. Any holes must occur in the hole region ABCD. Any singleton in regions 1 or 2 can cover the entire hole region.

6.2.1. Finding Holes

We assume the configuration illustrated for the purposes of this algorithm description. A mirror image configuration is also possible. Assuming the hole region exists, a search for holes in the OREC must be done. First we can see if the hole region can be reduced by expanding singletons properly. We note that if a singleton can affect the hole region it can be expanded so that its rectangle fully crosses the hole region. If some singleton is in region 1 or 2 (refer to figure 6-5) it can be expanded to cover the entire hole region. In this case, the OREC is now an ORAC, and the hole finding algorithm may terminate.

If the singletons do not fully cover the hole region, we may still have succeeded in reducing it. This reduction can be done in total linear time

We can also find rectangles in the OREC that cross the entire reduced hole region, and do a further reduction in total linear time in one tour of the polygon, knowing the rectangles assigned to each corner. If these reductions have removed the hole, the algorithm terminates, and returns an OREC that is an ORAC.

We now find out how the remaining rectangles cover this possibly reduced hole region.

It is possible for some rectangles to pass through the hole region, splitting it. These we check last.

We first find rectangles that intersect the left or top edges of the reduced hole

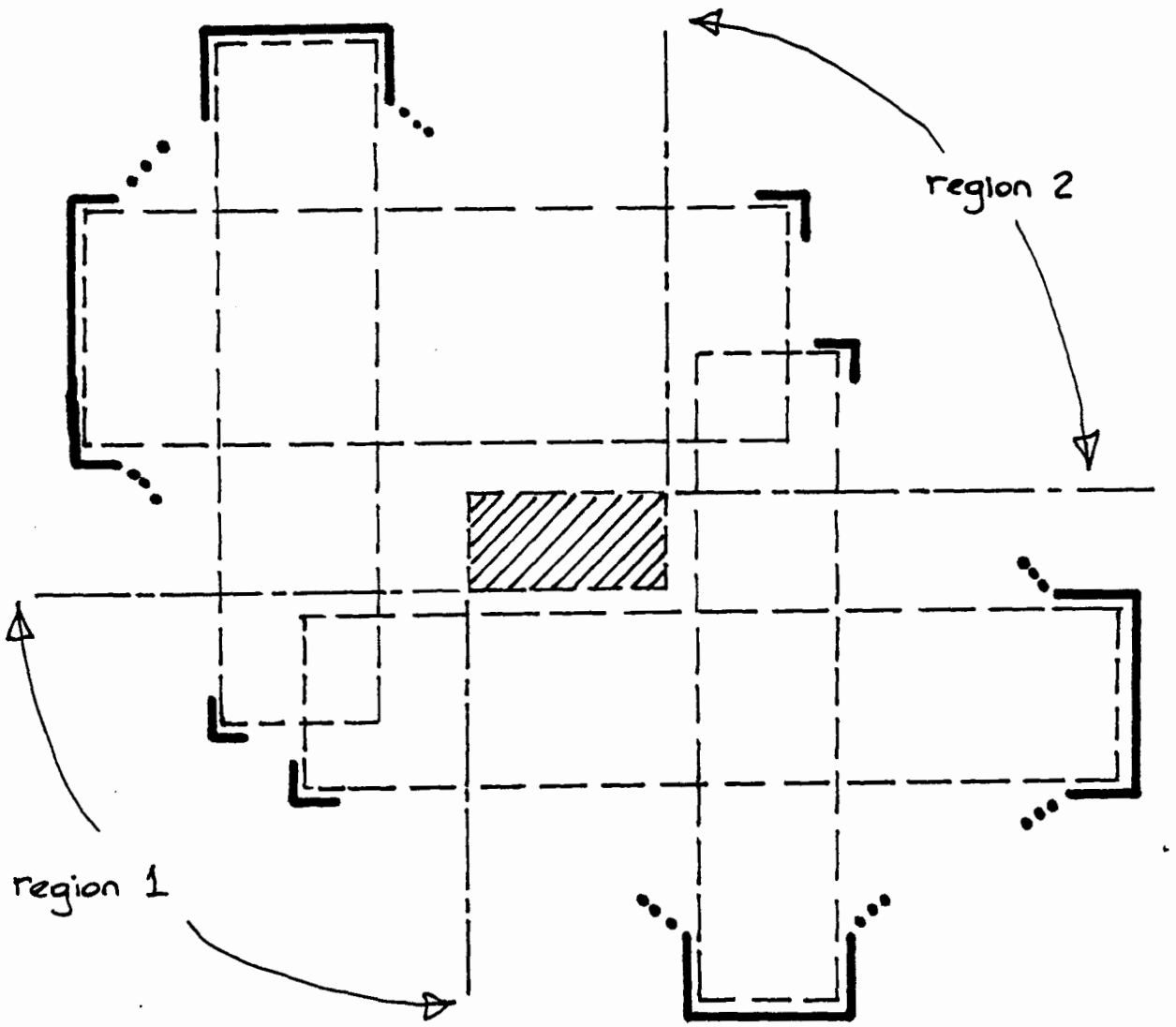


Figure 6-6: The Possibly Reduced Hole Region

A mirror-image of this figure is another possible configuration. Algorithms in the text are described in terms of this configuration.

region, but do not split the region. The reduced hole region is illustrated in figure 6-6. This is done in one sweep of the two opposing chains.

To sort by ascending y-values of the bottom edges of these rectangles, we note that the rectangles defined by a corner in chain A1 are already in this sorted order. The only other non-splitting rectangles that can cross the top or left edges and not be defined by a corner in A1 must be CA2-somes between A2 and A3. These are easily found in the cover, and it is seen that their expansion downwards must end against an edge of A1, as A4 is to the right of all corners in A2.

These rectangles are assigned to the corners in A1 whose edges they expand against. This puts them in sorted order by y-values with the other rectangles. A similar method can be used to sort the rectangles that cross the bottom or right of the hole region and do not split it.

We then find the zig-zag line their union makes across the hole region in linear time. Taking one rectangle at a time, we use a stack to store successive rectangles that increase the hole area covered. We take rectangles in descending order of their bottom edge y-coordinates. Each rectangle will then cross the left edge at the same or a lower point than the previous rectangle. If its right edge is to the right of or the same as the top rectangle on the stack, we remove the top rectangle from the stack. When we find a rectangle on the stack with a right edge to the right of the new rectangle (or an empty stack) we add the new rectangle to the stack. We call this the **top-left path** in this configuration.

We then do a similar procedure for rectangles crossing the bottom or right edges of the reduced hole region, and not splitting it. This is called the **bottom-right path** in this configuration.

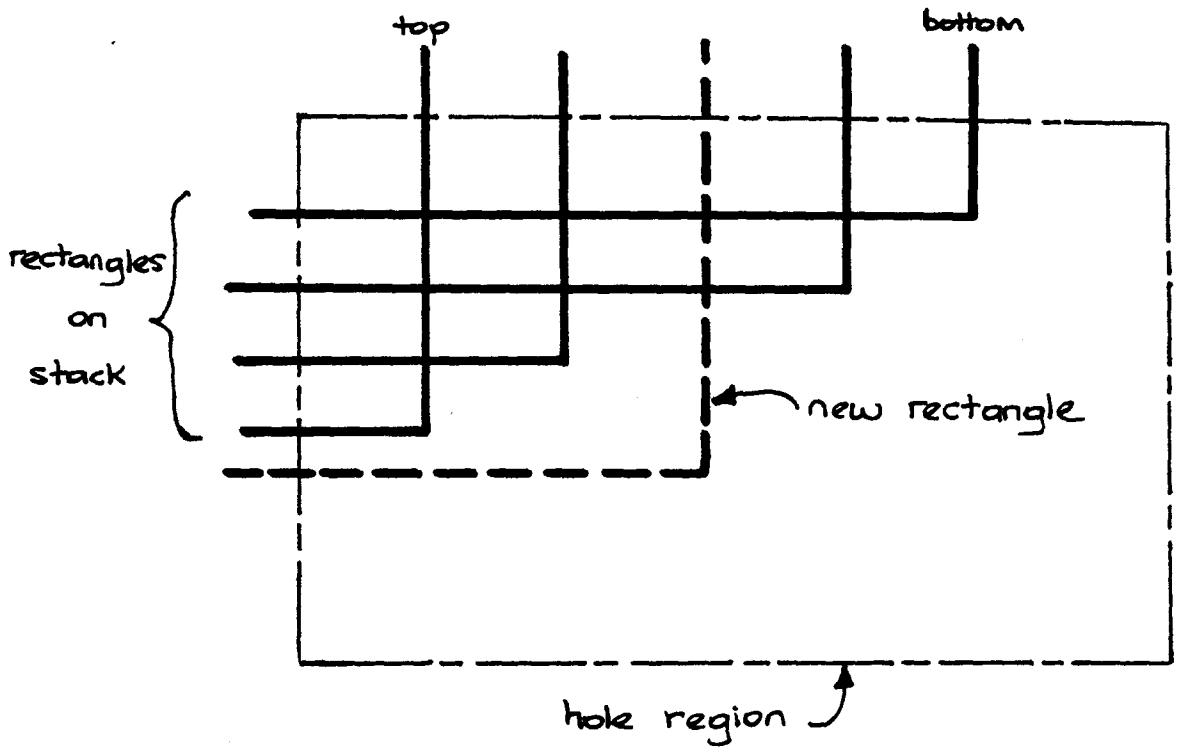


Figure 6-7: Updating the Top-left Zig-zag Path

In this configuration, rectangles in the cover cannot cross both the top and right edges or both the bottom and left edges of the hole region as they would have to be defined by corners from the chains A2 and A4. These chains cannot share a rectangle, as they are completely out of each other's scope, being completely blocked by A1 and A3.

We now have two (possibly empty) zig-zag paths across the hole region. We can run along the two of them in tandem. If the top-left zig-zag path is below the bottom-right path all the way across the hole region, the hole region is covered. If

there are still holes, places where the top-left path is above the bottom-right path, we store them in a linked list as we traverse the hole region. This can be done in linear time.

All that remains to be checked are the splitters, rectangles with two edges crossing the top and bottom or left and right edges of the reduced hole region. As these must be defined by corners from the two opposite chains A1 and A3, they are easily found in sorted order by traversing the chains. These we can easily check against the linked list of remaining holes in linear time, as they are already in sorted order by x-value in the two chains.

The resulting implementation of a hole finding algorithm is then linear.

6.2.2. Algorithm A1

To obtain an area cover for an irreducible polygon the following steps are performed:

1. Find an OREC for the polygon, using algorithm E1.

2. See if the hole region enclosed by the four tab rectangles is empty. If it is, the OREC is also an ORAC for the polygon.

3. If the hole region is non-empty, find out if any holes exist in the hole region, expanding singletons in the most advantageous manner.

4. If holes do not exist, the OREC is an ORAC.

5. If holes do exist, add one maximal rectangle to cover the hole region. The OREC plus this rectangle is an area cover.

6.2.3. Optimality of Algorithm A1

Algorithm E1 provides an OREC. Lemma 1 shows that any holes in an OREC must be in the hole region. If no holes exist, the OREC is an ORAC. If holes do exist, we cannot be certain from Algorithm E1 that they are necessary, i.e. that some other equal cardinality OREC isn't holeless. Thus the area cover obtained by adding a rectangle to the hole region can only be guaranteed to be within one rectangle of optimal.

6.2.4. Complexity of Algorithm A1

The Algorithm E1 used to find an OREC is linear. Finding whether the hole region exists or not can be done in linear time. The hole-finding algorithm has been shown to have a linear implementation. Finally, adding one maximal rectangle, to complete an area cover if required, can be done in constant time. Algorithm A1 is linear.

6.3. Algorithm A2: an Area Cover for General Polygons

Another important result of Chaiken, Kleitman, Saks and Shearer is used here.

Lemma 2: An optimal area cover for a reducible polygon may be obtained by reducing the polygon, using the tab rectangles defined at each reduction, and adding them to an optimal area cover of the irreducible polygon finally obtained [CKSS81].

6.3.1. Full and Partial Tab Reduction Configurations

Tab reduction is described in Chapter 5. The tab reduction algorithms can collapse the polygon completely. For example, a rectangular polygon gets one full tab reduction and then disappears. Some tab reductions may also collapse part of a polygon into a line (as in full tab configuration 6). In this case, there are always other tab reductions that may be performed first, leaving a rectangle for the final reduction.

The key to a linear implementation of full and partial tab reductions is the fact that, after each tab reduction, the new tab is adjacent to where the old one was.

Figure 6-8 shows some possible configurations for full tab reduction, and the resulting reduced polygons. Figure 6-9 does the same for partial tab reductions.

6.3.2. Linear Tab Reduction Algorithms

Chaiken, Kleitman, Saks and Shearer speak of shrinking the polygon each time a tab reduction is done. An straightforward implementation of this would entail changing the coordinates of approximately half of the corners at each reduction step. This would take $O(n^2)$ time.

An inspection of the new polygon boundaries in the region of a tab reduction (see figures 6-8, 6-9 and 6-10) shows that the topological changes are local only. We propose instead, a restructuring of the local area only, which can be done in constant time for each configuration. This is illustrated in figure 6-11 for a full and a partial tab reduction.

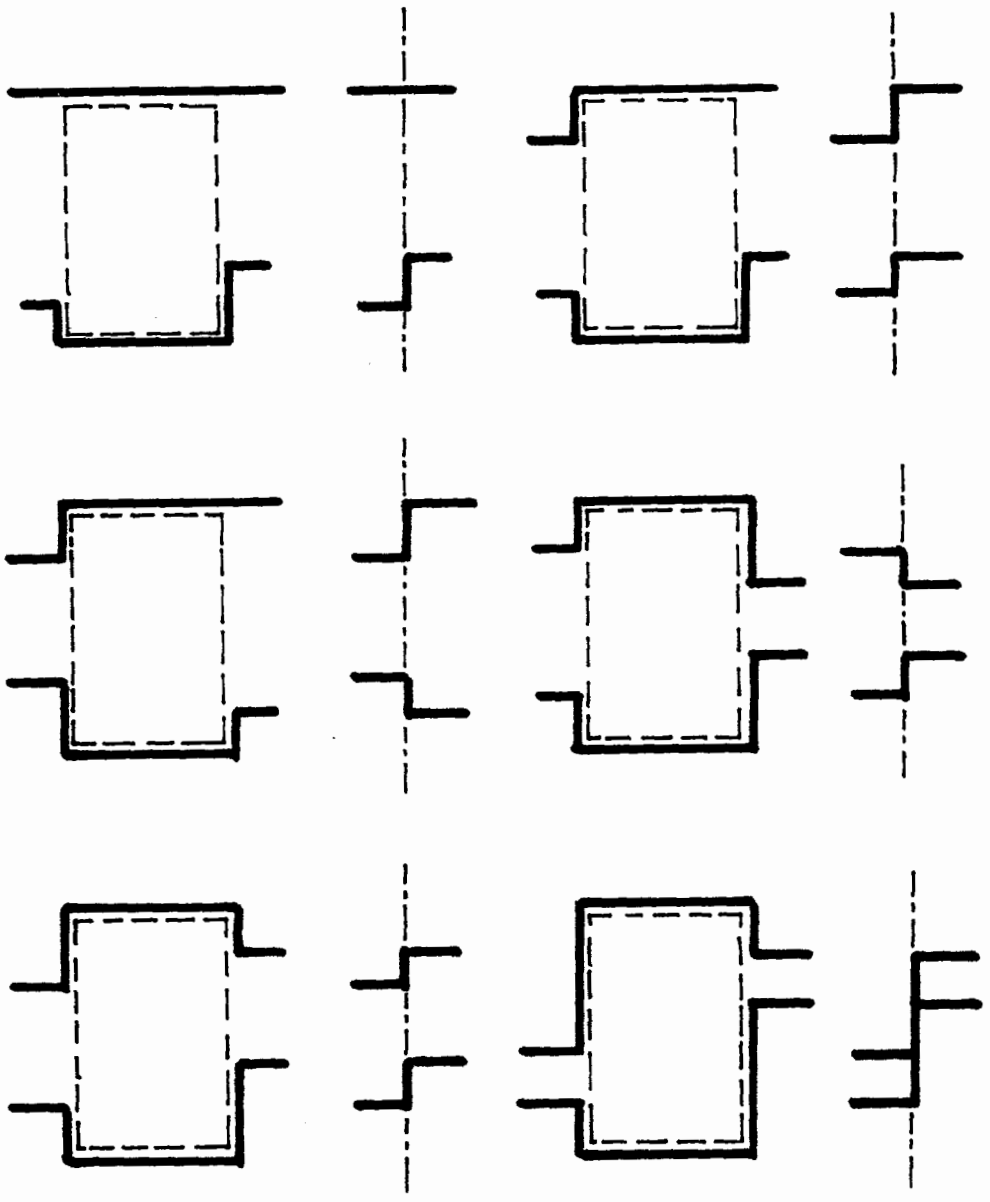


Figure 6-8: Some Full Tab Reduction Configurations

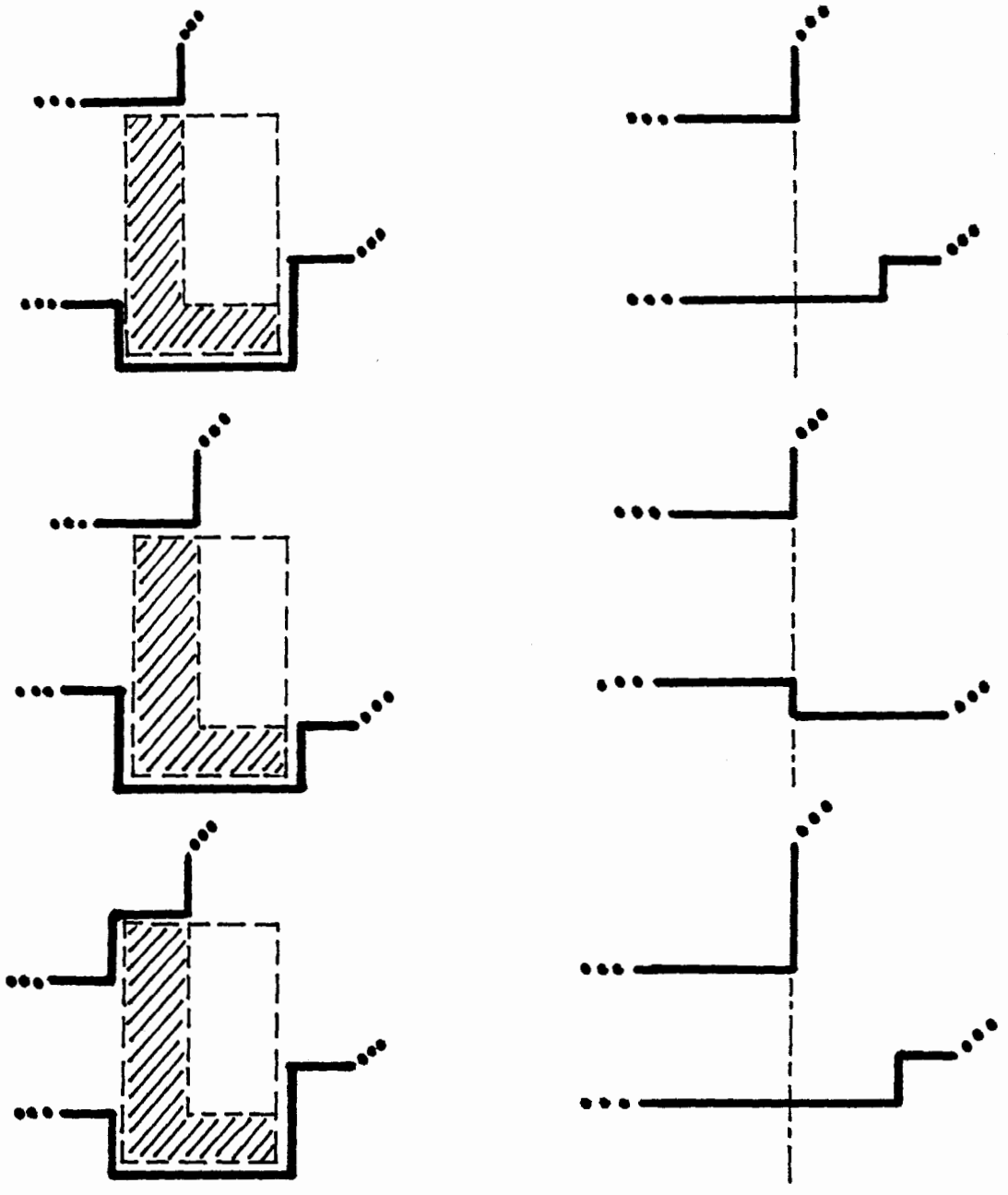


Figure 6-9: Some Partial Tab Reduction Configurations

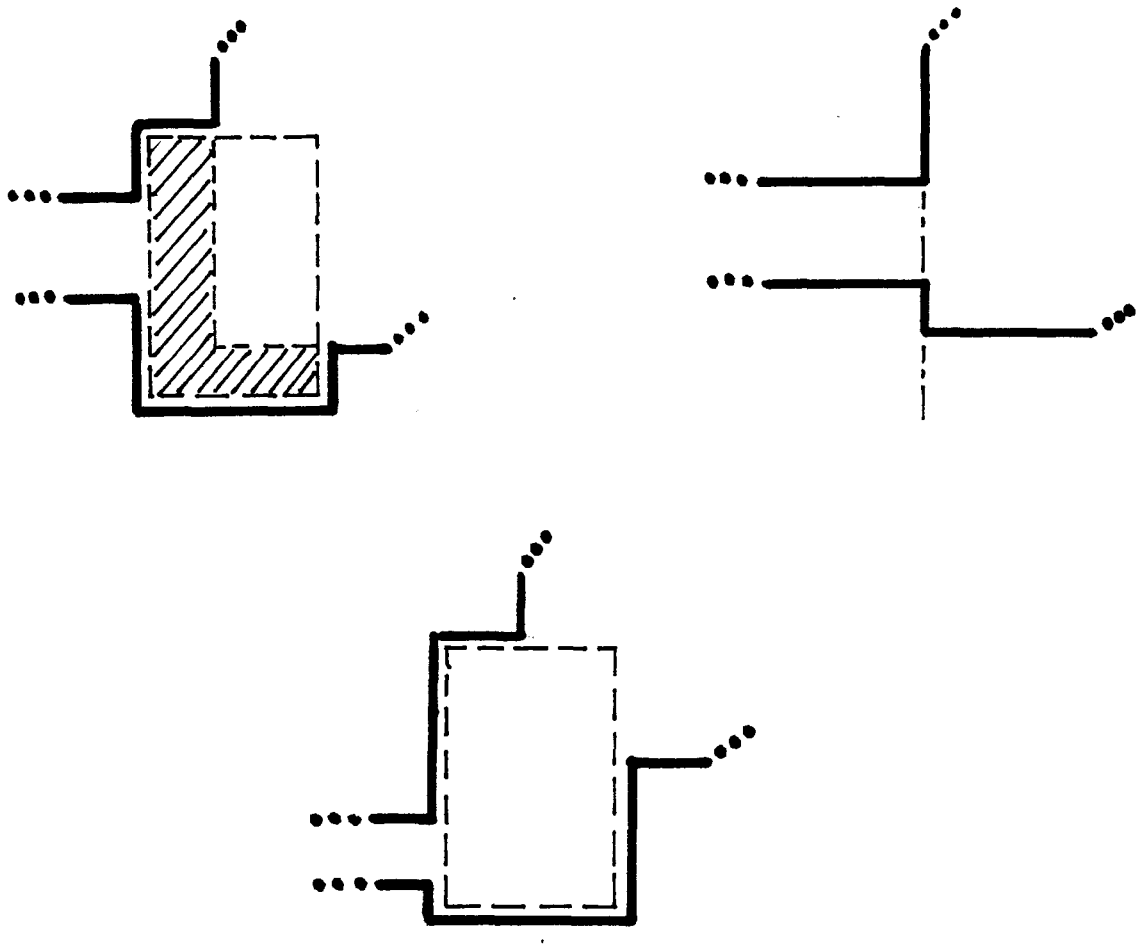


Figure 6-10: Some More Partial Tab Reduction Configurations

The bottom configuration cannot occur, as there would be a full tab reduction possible.

Two problems remain, for the implementation to be linear. Finding the new tabs and checking whether the new tab rectangles intersect each other (necessary to know for partial tab reductions) must be done in total linear time. Secondly, finding where the new tab rectangles intersect the other side of the polygon must be done in total linear time (necessary to determine the kind of reduction possible).

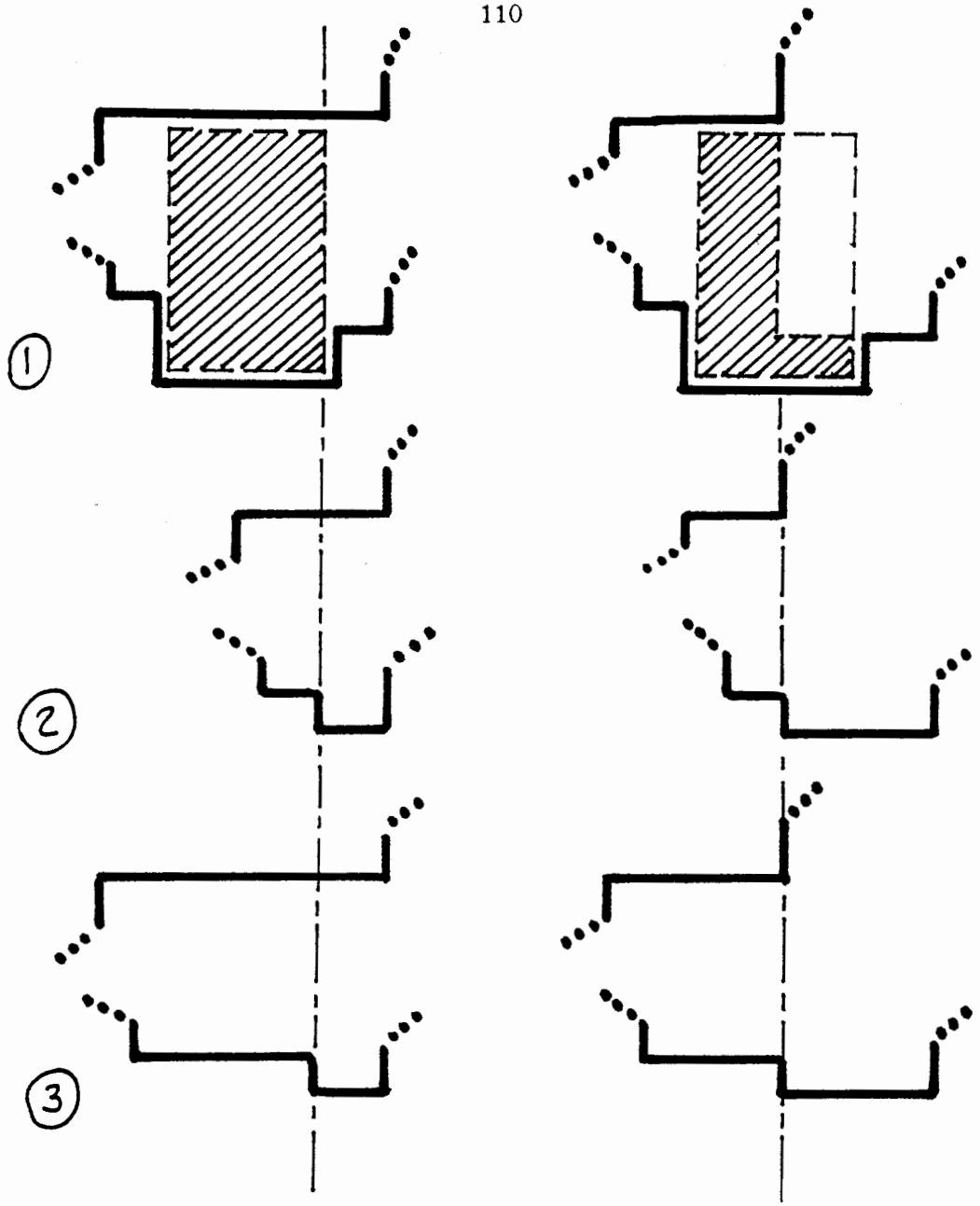


Figure 6-11: Global and Local Tab Reduction Changes

Global (2) and local (3) restructuring are shown for both full (left) and partial (right) tab reductions.

We can find all of these things for the first time in linear time. When a full or partial tab reduction is performed, the new tab is always adjacent to where the old one was. Each new tab will then be easy to find in constant time after a reduction has occurred. The new tab rectangle will intersect the other side of the polygon at an edge either at the same place or left or right of the original intersection.

Each new intersection will then occur to the left of or to the right of previous intersections. Thus, again, we have total linear performance, as the search for intersecting edges broadens out and never turns back inwards. All we need to do is to keep track of previous leftmost and rightmost intersection edges.

Knowing where the new tab rectangle intersects the other side of the polygon, we can update whether the tabs intersect each other in constant time at each reduction.

Thus an irreducible polygon can be obtained from a reducible polygon in linear time. If the polygon disappears during reduction, the rectangles defined during the process constitute an ORAC for the reducible polygon, by Lemma 2.

6.3.3. Algorithm A2

An area cover for general reducible polygons may be obtained by the following steps.

1. Perform full and partial tab reductions on the polygon until either an irreducible polygon is obtained, or a final full tab reduction causes the polygon to disappear. If the polygon disappears, the rectangles defined during tab reduction operations constitute an ORAC.

2. If an irreducible polygon is obtained, find an area cover for this polygon using Algorithm A1.

3. Combine this area cover with the rectangles defined during reduction steps for an area cover of the whole polygon.

6.3.4. Optimality of Algorithm A2

If the polygon disappeared during reduction, then by Lemma 2 the rectangles removed constituted an ORAC for the original polygon.

If an irreducible polygon was obtained after reduction, then optimality depends on the area cover for this irreducible polygon returned by Algorithm A1. If an ORAC was returned, the resulting area cover is also an ORAC. If a not necessarily optimal area cover was returned, the combined cover is also not necessarily optimal. It is however, guaranteed within one rectangle of optimal.

6.3.5. Complexity of Algorithm A2

We have shown that full and partial tab reduction operations can be done in linear time. Algorithm A1 has been shown to be linear. Algorithm A2 is then linear.

Chapter 7

Conclusions

7.1. General Properties of Covers

We have shown that 4-somes, unobscured 3-somes and CA2-somes generate rectangles that must be in any ORCC, OREC or ORAC.

The bounds on cardinalities of optimal covers were analysed in Chapter 2 and several basic properties of rectilinear polygons were shown.

7.2. The Corner Cover Algorithm

A linear algorithm has been presented that obtains an optimal rectangle corner cover for a convex rectilinear polygon.

Part of the algorithm entails finding a maximum independent set in the convex OPP2-some matrix. It is shown that the Cautious Method finds a maximum independent set in a matrix with the consecutive 1's property. A linear implementation of the Cautious Method is shown to find a MIS in the convex OPP2-some matrix (which has the consecutive 1's property in both axes).

It is also shown that this implementation can be used to find a maximum matching in any bipartite graph with a rectilinearly convex adjacency graph, in time linear in

the number of vertices in the graph, given the adjacency matrix or the ranges of adjacency.

The use of a sorting algorithm implies an O(|V| log|V|) algorithmn for bipartite graphs with the consecutive 1's property in their adjacency matrix, given the ranges of adjacency or the matrix itself.

7.3. Edge Cover Algorithms

For edge and area covers, we have shown that range intersection regions have a unique covering. It is shown that no range intersections can occur in an irreducible polygon.

Two linear algorithms are presented for edge covers. Algorithm E1 finds an optimal edge cover of an irreducible convex rectilinear polygon. Algorithm E2 finds an edge cover of a general (reducible) convex rectilinear polygon that is guaranteed within one rectangle of optimality.

7.4. Area Cover Algorithms

Multiple covering of corners is seen to always create a hole in the cover. We show there are six configurations of multiple covers that may be required in an OREC, and that we can allow five of them to occur in our algorithm.

Two linear algorithms are presented. Algorithm A1 finds an area cover of an irreducible convex rectilinear polygon that is within one rectangle of optimal. This algorithm uses a linear time hole-finding algorithm to cover any holes in an OREC for an irreducible polygon returned by Algorithm E1.

Algorithm A2 finds an area cover of a general reducible convex rectilinear polygon that is also guaranteed within one rectangle of optimal. This algorithm uses a linear implementation of tab reductions to obtain an irreducible polygon. It then uses Algorithm A1 to find an area cover of the irreducible polygon, which it combines with the rectangles defined during tab reductions to create an area cover of the whole polygon.

7.5. Open Problems

Several important open problems remain. Solving the first problem would provide the means for finding an OREC for any convex rectilinear polygon in linear time. Solving the second problem would enable the area cover algorithms to obtain optimal rectangle area covers for all convex rectilinear polygons in linear time.

1. Find any non-RI necessary multiple covers in an OREC for a general convex polygon in O(n) time and report them.

2. Find the best possible MIS and singleton expansion in an OREC for an irreducible polygon so that if an OREC that is also an ORAC exists, the algorithm finds it in O(n) time.

3. Extend these results where possible to finding an ORCC, OREC or ORAC for a semi-convex polygon in sub-quadratic time.

Much initial work has been done on the first two problems, but the desired results have not yet been obtained.

7.6. Conjectures

Two properties have begun to appear most likely during this research, but have not yet been proven. These conjectures are presented as questions suitable for further investigation.

1. If there is a single range intersection, no necessary holes will occur in an OREC for convex rectilinear polygons.

2. If there are OPP2-some pairings possible from both sets of opposite chains, there can be no necessary holes in the OREC for a convex rectilinear polygon.

7.7. Summary

The rectangle cover problems considered in this thesis are considerably more difficult than they appeared at first. The methods used here for convex rectilinear polygons rely very much on the convexity of the polygons. They do not appear to be easily generalized to solutions for semi-convex and non-convex polygons.

References

[AsA83] Asano, Tetsuo & Asano, Takao.
Minimum Partition of Polygonal Regions into Trapezoids.
In *24th Symposium on Foundations of Computer Science*, pages 233-241. IEEE Computer Society, November, 1983.

[CKSS81] Chaiken, Seth; Kleitman, Daniel J.; Saks, Michael & Shearer, James.
Covering Regions by Rectangles.
SIAM J. Alg. Disc. Meth. 2(4):394-410, December, 1981.

[FrK84] Franzblau, Deborah S. & Kleitman, Daniel J.
An Algorithm for Constructing Regions with Rectangles.
In *Proceedings of 16th Annual ACM STOC*, pages 167-174. ACM, April, 1984.

[GaJ79] Garey, Michael R., & Johnson, David S.
Computers and Intractability: A Guide to the Theory of NP-Completeness.
W. H. Freeman & Company, New York, 1979.

[HoK73] Hopcroft, J.E., & Karp, R.M.
A $n^{5/2}$ Algorithm for Maximum Matching in Bipartite Graphs.
J. SIAM Comp. 2:225-231, 1973.

[LeP84] Lee, D.T., & Preparata, Franco P.
Computational Geometry - A Survey.
IEEE Transactions on Computers C-33(12):1072-1101, December, 1984.

[LiP80] Lipski, Witold jr., & Preparata, Franco P.
Finding the Contour of a Union of Iso-Oriented Rectangles.
Journal of Algorithms 1:235-246, 1980.

[Mas79] Masek, J.W.
Some NP-complete set covering problems.
Unpublished manuscript, Aug. 1979.

[ORo82] O'Rourke, Joseph.
The Complexity of Computing Minimum Convex Covers for Polygons.
In *Proceedings of 20th Annual Allerton Conference on Communication, Control & Computing*, pages 75-84. Illinois, October 6-8, 1982.

[ORo82a] O'Rourke, Joseph.
Polygon Decomposition and Switching Function Minimization.
Computer Graphics and Image Processing 19:384-391, August, 1982.

[ORS83] O'Rourke, Joseph & Supowit, Kenneth J.
Some NP-Hard Polygon Decomposition Problems.
IEEE Trans. on Information Theory IT-29(2):181-190, March, 1983.

[Oht82] Ohtsuki, T.
Minimum Dissection of Rectilinear Regions.
In *Proc. IEEE Int. Symposium on Circuits and Systems*, pages 1210-1213. IEEE, Rome, 1982.

[PrS85] Preparata, Franco P., & Shamos, Michael Ian.
Texts & Monographs in Computer Science: Computational Geometry - An Introduction.
Springer-Verlag, New York, 1985.

[Sac84] Sack, Jorg-Rudiger.
Rectilinear Computational Geometry.
PhD thesis, Carleton University, June, 1984.

[Sha78] Shamos, M.I.
Computational Geometry.
PhD thesis, Yale University, 1978.

[Ull84] Ullman, Jeffrey.
Computational Aspects of VLSI Design.
Computer Science Press, New York, 1984.

[Woo85] Wood, Derick.
An Isothetic View of Computational Geometry.
In G. T. Toussaint (editor), *Computational Geometry*, North Holland, 1985.

Index

Unit square model 9