Efficient Parallel Algorithms for Covering Binary Images

Dipen Moitra
Ph.D Thesis

TR 89-1013
May 1989

Department of Computer Science
Cornell University
Ithaca, NY 14853-7501
EFFICIENT PARALLEL ALGORITHMS
FOR COVERING BINARY IMAGES

A Dissertation
Presented to the Faculty of the Graduate School
of Cornell University
in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy

by
Dipen Moitra
May 1989
Efficient Parallel Algorithms for Covering Binary Images

Dipen Moitra, Ph. D.
Cornell University 1989

Given a black and white image, represented by an array of $\sqrt{n} \times \sqrt{n}$ binary valued pixels, we wish to cover the black pixels with a minimal set of (possibly overlapping) maximal squares. It was recently shown that obtaining a minimum cover with squares for a polygonal binary image having holes is NP-hard. We derive a processor-time-optimal parallel algorithm for the minimal square cover problem, which for any desired computation time $T$ in $[\log n, n]$ runs on an EREW-PRAM with $(n / T)$ processors. We also outline an implementation on a mesh architecture which runs in $O(\sqrt{n})$ time, and is P-T-optimal. Finally, we also show how to obtain a speedup in the running time of our algorithm when polymorphic communication primitives are available on the mesh. The cornerstone of our algorithm is a novel data structure, the cover graph, which compactly represents the covering relationships between the maximal squares of the image. The size of the cover graph is linear in the number of pixels. This algorithm has applications to problems in VLSI mask generation, incremental update of raster displays, and image compression.
Biographical Sketch

The author was born on November 15th 1955 in Ajmer, a small town nestled between hills and surrounded by many lakes: an oasis in the desert part of Rajasthan, in western India. He then went on (many, many years after birth), to study at the Birla Institute of Technology and Science, in Pilani, and earned a Master's degree in Electrical Engineering in 1978. There he also met Abha, whom he married in 1977. From 1977 through 1983, the author worked at the Tata Institute of Fundamental Research in Bombay, and did many interesting things with digital circuits and with software.

He has found the years at Cornell very enjoyable - with the department being small and close knit, and Ithaca being so incredibly beautiful. The author and his wife got a son Ankur (not by mail order) in 1985, and they have had great fun ever since.
To Abha, who made it all possible; and to Ankur, who almost convinced me that other things were far better.
Acknowledgements

I would like to thank my advisor Gianfranco Bilardi for teaching me about Parallel Computation, for teaching me how to go about solving problems, and more importantly for showing me how to explain the solutions. I would like to thank Joe Mitchell for stimulating my interest in Computational Geometry, and for our many fruitful discussions. I would also like to thank Lov Grover and Tony Reeves for all their encouragement and advice.

I am greatly indebted to Kevin Karplus, Tim Teitelbaum, Charlie Van Loan and Ken Wilson for their encouragement and support during the early years. I gratefully acknowledge the support of NSF under grant DCM-86-02256, ONR through contract N00014-83-K-0640, and IBM under agreement 12060043.

This rite of passage was made so much smoother by the friendship and company of Amitabh, Anne, Brad, Carolyn, Chris, Jacob, Jin Yi, Kieran, Peter, Scott, Shyam, Srikanth, Tommy, Turhan, Vinod and others. Also David, Dexter, Dina, Estie, Fred, Gianfranco, Kay, Keith, Ozalp, Peter and Sam. Any such list is necessarily far from complete, so please accept my thanks all those of you who helped.

Last but not least, they also helped who only stood and stared ...
# Table of Contents

1 Introduction

1.1 Compact Representations of Binary Images .................. 1
1.2 Related Work ........................................... 3
1.3 Variations of the Cover Problem ........................... 5
1.4 Overview of the Minimal Cover Algorithm ................... 7
  1.4.1 Need for a Cover Graph ............................... 8
  1.4.2 A Naive Cover Graph ................................ 8
  1.4.3 A Smaller Cover Graph ............................... 10
  1.4.4 Getting a Minimal Cover from the Cover Graph ...... 13
  1.4.5 The Existence of a Reduced Cover Graph .............. 13
1.5 Algorithm Minimal_Square_Cover ........................... 15

2 Finding Maximal Squares and Dividing Uniform Strips

2.1 Finding Maximal Squares using Prefix Computations ........ 17
  2.1.1 Largest Black Squares which are Maximal .......... 22
2.2 Dividing Uniform Strips .................................. 23

3 Conditions for the Redundancy of Maximal Squares ............. 26

3.1 Basic Cover Relations ...................................... 27
3.2 Finding the Basic Cover .................................... 30
3.3 Normalizing the Basic Cover ................................ 37
3.4 Extended Cover Relations .................................. 38
3.5 Finding the Extended Cover ......................... 41
3.6 Bounded Size of the Extended Cover ............... 48

4 Deriving the Cover Graph ............................. 52
  4.1 Factoring the Cover Relations in Normal Form ...... 53
  4.2 Representing the Cover Function by a Cover Graph ... 55
  4.3 Rectangular Decomposition of Squares ............... 60
  4.4 Constructing the Cover Graph ....................... 65

5 Determining a Minimal Square Cover .................. 69
  5.1 Labelling the Cover Graph ......................... 70
  5.2 Time Bounds for Labelling ......................... 72
  5.3 Correctness of the Labelling Algorithm .......... 73

6 Implementation on Bounded Degree Networks .......... 81
  6.1 Practical Architectures for Parallel Computing .... 83
  6.2 Data Structures for Constructing Square Covers ........ 85
  6.3 Implementation Outline for a Mesh Architecture ... 88
  6.4 Speedup Obtainable on a Polymorphic Mesh .......... 91

7 Conclusions ........................................ 95
  7.1 Future Directions ................................ 97

Appendix 1 Definitions ................................ 99
Appendix 2 Theorems .................................. 104
References .......................................... 107
# List of Figures

1.1 Minimal Cover of Squares  
1.2 Horizontal and Diagonal Sequences of Squares  
1.3 Given collection of squares  
1.4 Reduced Cover Graph for Fig. 1.3  
1.5 Condensing the Naive Cover Graph  

2.1 Algorithm for finding Largest Black Squares  
2.2 Associative combination of Hammerhead regions  
2.3 Associative combination of Arrowhead regions  
2.4 Largest Black Square contained in a larger one  
2.5 Combining tails of adjacent Uniform Strips  
2.6 Dividing Uniform Strips  

3.1 Squares $T_1 \ldots T_k$ overlap with m without being useful covers  
3.2 A basic cover which is not minimal  
3.3 Bounding the size of a Basic Cover  
3.4 T contains no corners of n  
3.5 T contains one corner of n, and pixel $(i-1, j-1)$ is white  
3.6 T contains one corner of n, and pixel $(i-1, j-1)$ is black  
3.7 Size of T equals the size of m  
3.8 Size of T is greater than the size of m  
3.9 Verifying a cover  
3.10 Finding an un-necessarily fine cover for m  
3.11 $\Xi_m$ contained in $\beta_{\Omega(\sqrt{n})}$
3.12  P covers no corners of m
3.13  P covers no corners of m, & Q covers two
3.14  P covers one corner of m
3.15  P covers two corners of m
3.16  Two squares in $\Xi_m$ cut same edge of m

4.1  Need for factoring the covering relationships
4.2  Geometric interpretation of the cover function
4.3  Cover graph for the image in Fig. 4.2(a)
4.4  Coarse decomposition of squares
4.5  An Improper Cover Graph for Fig. 4.4
4.6  Partial order on $\rho$-covers of a square
4.7  Partition of m (residual regions are shaded)
4.8  Induced Cover for $(m, \Xi_m)$ shown in Fig. 4.7
4.9  Global Implication Rule
4.10 Example image
4.11 Induced covers for m and P
4.12 Reduced Cover-graph for Fig. 4.10

5.1  The cases for 2-labelling chains
5.2  Example image for the labelling algorithm
5.3  Squares retained after division of uniform strips
5.4  Residual regions which determine the cover graph
5.5  Reduced cover graph for the image in Figure 5.2
5.6  Color key for Figure 5.7
5.7  Steps of the labelling algorithm
5.8  Irreducible cover obtained by the labelling algorithm
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>Given image</td>
</tr>
<tr>
<td>m</td>
<td>Selected maximal square of discourse</td>
</tr>
<tr>
<td>M_I</td>
<td>All maximal squares in the image</td>
</tr>
<tr>
<td>M_R</td>
<td>Retained maximal squares</td>
</tr>
<tr>
<td>M_D</td>
<td>Discarded maximal squares</td>
</tr>
<tr>
<td>M_A</td>
<td>Available maximal squares (undecided state)</td>
</tr>
<tr>
<td>M_U</td>
<td>Squares not discarded during subdivision of uniform strips</td>
</tr>
<tr>
<td>M_C</td>
<td>Minimal square cover for the image</td>
</tr>
<tr>
<td>G_P</td>
<td>Pixel level cover graph = ((M_I, P_I), E_P)</td>
</tr>
<tr>
<td>P_I</td>
<td>All pixels in the image</td>
</tr>
<tr>
<td>E_P</td>
<td>Edge set for (G_P = {(m, p)</td>
</tr>
<tr>
<td>G_R</td>
<td>Reduced cover graph = ((M_U, R_M), E_R)</td>
</tr>
<tr>
<td>R_M</td>
<td>Rectangular decomposition of (M_U)</td>
</tr>
<tr>
<td>E_R</td>
<td>Edge set for (G_R \subseteq (M_U \times R_M) \cup (R_M \times M_U))</td>
</tr>
<tr>
<td>(\beta_m)</td>
<td>Basic cover for (m) (synonymous with (\beta(m)))</td>
</tr>
<tr>
<td>(\xi_m)</td>
<td>Irreducible cover for (m)</td>
</tr>
<tr>
<td>(\Xi_m)</td>
<td>Extended cover for (m)</td>
</tr>
<tr>
<td>(\eta_m)</td>
<td>Induced cover for (m)</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 Compact Representations of Binary Images

In many applications it is important to encode digitized images compactly, in order to reduce storage and transmission costs, even at the expense of time and hardware for encoding and decoding the raw image. A compact encoding of digitized images can also provide the basis for efficient image analysis and manipulation algorithms, for applications such as remote sensing, VLSI mask generation, incremental update of raster displays, and object representation for sequential frames of a dynamic polygonal scene.

A popular technique for image representation is the Quad-tree [TaP 75], which provides a recursive decomposition of a binary image into quadrants, each quadrant being decomposed further only if it is neither all white nor all black. This yields a 4-ary tree with at most $\frac{1}{2}\log n$ levels, for a digitized image consisting of $\sqrt{n} \times \sqrt{n}$ pixels. Although the Quad-tree supports efficient algorithms for image manipulation, the size of the resulting tree is extremely sensitive to the placement of the origin for the decomposition at the root. For example consider the decomposition of a single square of side $\sqrt{n}$, starting at $(1, 1)$; it contains $\Omega(\sqrt{n})$ nodes, instead
of the single node which suffices for a minimal cover. There have been many attempts to find an optimal placement for the origin, but all the techniques are computationally expensive, and do not yield guaranteed size reductions [Sam-84]. Scott and Iyengar [Sci-86] proposed that instead of finding an optimal origin for the Quad-tree, we could represent a binary image more compactly by means of a suitable small subset of the maximal subsquares of the image.

![Image](image.png)

(a. Given black and white image  b. Cover with maximal squares)

Figure 1.1 Minimal Cover of Squares

The problem we consider in this thesis is as follows. Given a black and white image represented by an array of $\sqrt{n} \times \sqrt{n}$ binary valued pixels, we wish to cover the image with a minimal set of black subsquares of the image, and permit overlaps amongst the squares in the cover (see Figure 1.1). Clearly we lose nothing by insisting that the chosen squares be maximal; in fact the efficiency of our minimal cover algorithm depends crucially upon the maximality of the squares. In the rest of this thesis we will discuss parallel algorithms for obtaining a minimal cover consisting of maximal squares, for digitized black and white images. Our algorithm
[Moi-88] is optimal with respect to both its running time, as well as the number of operations performed. It has a linear processor-time product, and for all desired computation times $T$ in $[\log n, n]$ runs on an EREW-PRAM with $(n/T)$ processors.

**Definition 1.1 (maximal square, minimal square cover):** A maximal square is a black subsquare of the image, not contained in any larger black square. A square cover for a given image is a set $C$ of (possibly overlapping) maximal squares, whose union equals the image. If no subset of $C$ is a square cover, $C$ is called a minimal square cover.

### 1.2 Related Work

Given a digitized binary image, Scott and Iyengar [ScI-86] present an algorithm to find the maximal squares, and then divide the rectangular regions of the image, to cover the rectangles with a minimum number of squares. Their algorithm does not yield a minimal cover over the entire image, is sequential, and requires $O(n \log n)$ time.

It was recently shown by Aupperle et al. [ACK-88] that covering a polygonal binary image having holes with a minimum number of (maximal) squares is NP-hard. They also present a sequential algorithm for obtaining a minimum square cover for simply connected images (without holes), as follows. Given a digitized binary image $I$, construct a cover graph $G_A(I)$ with one vertex for every pixel of the image, and add an edge between every pair of pixels which belong to some common maximal square. Note that the maximal cliques in $G_A(I)$ correspond to the maximal
squares in $I$. It is shown that if the image $I$ has no holes, the resulting cover graph $G_A(I)$ is chordal. Hence the algorithm of Gavril [Gav-72] can be used to find a minimum clique cover for $G_A(I)$, which yields a minimum square cover for $I$.

The above construction yields a graph with $\Omega(n^2)$ edges in the worst case (consider an image consisting of a single square of side $\sqrt{n}$). The required number of operations is reduced to $O(n \sqrt{n})$ by avoiding the explicit construction of $G_A(I)$, as given in [Aup-88]. First find the maximal squares of $I$; then construct a reduced cover graph $G_{AR}(I)$, which corresponds to $G_A(I)$ with every maximal clique condensed to a single vertex. Although the algorithm in [ACK-88, Aup-88] produces a minimum cover for an image without holes, it fails if the image has any holes, since the resulting cover graph $G_{AR}(I)$ is not chordal. (Recall that the problem of finding a minimum cover for general graphs is NP-hard, and remains so even if the graph is planar.) No technique is given for finding even a minimal cover if the image has holes. In contrast, our algorithm works for general images (with or without holes), runs in $O(\log n)$ time on an EREW-PRAM with $n/\log n$ processors, and always yields a minimal cover (which is not necessarily of minimum size).

Recently Bar Yehuda and Ben Chanoch [YeC-89] studied an interesting variant of the problem of covering isothetic polygonal regions (without holes) with a minimum number of squares. They assume that the polygon is specified by a (cyclically) ordered sequence of its vertices, instead of being digitized on a regular grid. Given such a polygon with $p$ vertices, a
sequential algorithm to find a minimum square cover is derived, with a running time of $O(p \log p + opt)$, where $opt$ is the number of squares in a minimum cover. Further, if the image is available as a digitized array of $\sqrt{n} \times \sqrt{n}$ binary valued pixels, it is claimed that this algorithm runs in $O(n)$ time. This algorithm outperforms that of [ACK-88, Aup-88] in all cases, and even more dramatically whenever the polygon needs to be digitized at a very high resolution as compared to the number of its vertices. Such is indeed the case with the specification of VLSI masks. For an image with $h$ holes, Bar Yehuda et al. present an approximation algorithm, which generates a cover of size $\leq opt + h$ in time $O(p \log p + opt + h)$, where is the number of holes in the image.

1.3 Variations of the Cover Problem

Our main motivations for solving the minimal square cover problem are: (i) to obtain a compact encoding of digitized binary images, and (ii) to derive a data structure which is suitable for efficient image representation and manipulation algorithms. In this thesis we will concentrate on the first task, and provide some indications of how this work can provide a starting point for a solution of the second problem.

We are given a digitized black and white image represented by an array of $\sqrt{n} \times \sqrt{n}$ binary valued pixels. If the image can be represented by a collection of $k$ squares, we will need $3k \log n$ bits for specifying the image, since we need $\log n$ bits each for representing the abscissa, ordinate and size of each square. Whenever $k$ is much less than $n / (3 \log n)$, we achieve
good compression. It is clear that the worst case arises when the given image is a checkerboard, and in that case \( k = n / 2 \). It remains an open problem to estimate the expected number of squares in a minimal cover for a random image, with a suitable definition of random images.

It is also clear that in most cases, an isothetic polygonal image can be represented by a much smaller number of maximal rectangles than the number of required maximal squares. Although each rectangle requires \( 4 \times \log n \) bits for specifying it, we would still expect to come out ahead on the average. Two reasons dissuaded us from trying to solve the minimal rectangular cover problem. First, the worst case number of distinct maximal rectangles in a binary image of \( \sqrt{n} \times \sqrt{n} \) pixels is \( O(n \sqrt{n}) \), whereas the worst case number of maximal squares is \( O(n) \), so we can only hope to determine a minimal square cover with a linear number of operations. Second, our eventual goal is to embed the derived minimal cover into a suitable search tree, to permit efficient image manipulation algorithms for such problems as determining the unions and intersections of images. We believe that it is easier to embed a minimal collection of maximal squares into a search tree than it is to embed a minimal collection of maximal rectangles, since the latter approach would require an additional degree of freedom in the search tree.

Many variations of the minimum rectangular cover problem are also known to be NP-hard. It was shown by Conn and O'Rourke [CoR-87] that it is NP-hard to find a minimum rectangular cover for polygonal binary images having holes, even if we are required to cover just the perimeter of
the polygon, or cover only its reflex vertices. Recently Culberson and Reckhow [CuR-88] showed that it is NP-hard to find a minimum rectangular cover for a polygonal binary image even if it has no holes.

Another variation of this problem is that of covering a given isothetic polygon with a minimum number of star shaped polygons, which is known to be NP-hard for polygons with / without holes [OR 87]. (A polygon P is said to be star shaped if there is at least one point in its interior which can see every point on the perimeter of P.) Recently Motwani et al. [MRS-88] gave an \(O(n^{10})\) time algorithm for solving the related problem of covering an orthogonal polygon with a minimum number of polygons which are star-shaped under the more general definition of visibility along staircase paths.

1.4 Overview of the Minimal Cover Algorithm

In this section we provide an overview of our minimal square cover algorithm. First we find the set \(M_I\) of all maximal squares in the given image \(I\). We determine a minimal square cover for \(I\) by a sequence of phases, each modifying a partition \((M_R, M_D, M_A)\) of \(M_I\), where \(M_R\) is the set of retained squares, \(M_D\) is the set of discarded squares, and \(M_A=(M_I-(M_R\cup M_D))\) is the set of available squares whose state is yet undecided. Initially \(M_D\) is empty, and \(M_R\) contains only the essential squares of \(I\) (i.e. those maximal squares which contain at least one pixel not covered by any other maximal square). At the end of the algorithm \(M_R\cup M_D\) equals \(M_I\), and the squares in \(M_R\) constitute the desired minimal cover for \(I\).
1.4.1 Need for a Cover Graph

During each phase of the algorithm, we select a subset $S$ of $M_A$ whose elements can be discarded simultaneously, without violating the safety condition that the image is covered by $M_A \cup M_R$. Consequently, for every square $m$ in $S$, we need to answer efficiently the question: "does $(M_A \cup M_R) - S$ cover $m$?" For this purpose, we construct a cover graph $G(I)$, which provides a compact representation of the covering relationships between the maximal squares in the image $I$. This will let us efficiently determine the relative redundancy of an arbitrary set $S$ of squares with respect to the set of residual squares $(M_A \cup M_R) - S$. Clearly the number of phases of the algorithm (and hence its maximum degree of parallelism) depends crucially on the number of squares that can be discarded simultaneously during each phase. The cover graph helps us to determine a large (independent) set of squares, which can be discarded simultaneously without violating the safety of the cover.

1.4.2 A Naive Cover Graph

Given that we wish to represent the covering relationships between the maximal squares and the pixels of the image, the following construction seems intuitively obvious. Define a bipartite graph $G_P = (M_I, P_I, E_P)$, whose vertex sets $M_I$ and $P_I$ denote respectively the maximal squares and the pixels in the image. An edge $(m, p)$ in $E_P$ denotes that square $m$ contains pixel $p$. Now we can restrict our attention to just the cover graph $G_P$, and determine a minimal subset $M_C$ of $M_I$, such that every pixel is
contained in at least one square in $M_C$, and every square in $M_C$ covers at least one pixel uniquely.

Figure 1.2(a) Horizontal Sequence of Squares

Figure 1.2(b) Diagonal Sequence of Squares

Consider the images shown in Figure 1.2, which show respectively a horizontal and a diagonal sequence of identically sized maximal squares of side $\frac{1}{3} \sqrt{n}$, spaced one pixel apart. One can easily verify that if we construct the cover graph $G_P$ (naively) defined above, it will have $\Omega(n \sqrt{n})$ edges in the worst case, and we will not be able to meet our overall linear resource bounds.
1.4.3 A Smaller Cover Graph

Clearly the problem with the definition of $G_P$ is that each pixel of the image can belong to a large number of different maximal squares. Hence we should not attempt to represent all possible ways of covering each distinct pixel. Although the naive cover graph $G_P$ is too large for our needs, it captures the covering relationships between the maximal squares and the pixels of the image in a very natural manner. We can define a smaller cover graph $G_R$ by analogy with $G_P$, such that $G_R$ embodies the useful properties of $G_P$ as follows.

Let $G_R$ be a directed bipartite graph $((M_G, R_M), E_R)$. One vertex set $M_G$ denotes maximal squares, and the other vertex set $R_M$ denotes (rectangular) subregions of the squares in $M_G$. The edge set $E_R$ denotes containment between the squares and the regions. Associated with each vertex $v$ in $M_G$ we store the location of the top left corner of the square $S_v$ which is represented by $v$. This enables us to obtain a square cover for the image from a vertex cover for $M_G$.

We can reduce the size of the cover graph by discarding some of the obviously redundant maximal squares of the image before constructing $G_R$. Hence $M_G$ is a subset of $M_I$ which satisfies the following condition.

(G1) The union of the squares in $M_G$ equals the image.

We will show a construction for $M_G$ in Section 2.2, where we find minimum covers for some rectangular regions of the image. We can obtain further reductions in the size of $G_R$ by (a) combining those pixels which are
covered by the same set of squares into a single region, and (b) not representing those cover relationships which are guaranteed to be satisfied whenever some other regions of the image are covered. We can even change our viewpoint from trying to cover the pixels of the image with maximal squares, to that of trying to cover the maximal squares of the image with a (minimal) subset of those maximal squares.

Now we need to establish a correspondence between vertex covers in $G_R$ and square covers in the image. We say that a subset $S_m$ of $M_G$ $g$-covers a square $m$ in $G_R$ if every region $r$ preceding $m$ has at least one predecessor $S_r$ such that $S_r$ belongs to $S_m$. We say that $S_m$ $i$-covers $m$ in the image if $m$ is contained in the union of the squares in $S_m$. We can ensure that $S_m$ $i$-covers $m$ in the image whenever $S_m$ $g$-covers $m$ in $G_R$ by requiring that $G_R$ also satisfy the following two conditions.

(G2) A square $m$ in $M_G$ is contained in the union of the regions preceding it in $G_R$.

(G3) A region $r$ in $R_M$ is contained in every one of the squares preceding it in $G_R$.

Finally, we must guarantee that minimality of a $g$-cover of $M_G$ in $G_R$ implies minimality of the corresponding square cover in the image. To this end we require that $G_R$ also satisfies the following condition.

(G4) If a subset $S_m$ of $M_G$ $i$-covers $m$ in the image, then $S_m$ also $g$ covers $m$ in $G_R$.

In Theorem 8 we will prove that a subset $M_C$ of $M_I$ is a minimal square cover of the image $I$ if and only if $M_C$ is a minimal $g$-cover of $M_G$ in the
cover graph $G_R$ for I. In Chapters 3 and 4 we will show how to construct the cover graph for the image. We show in Figures 1.3 and 1.4 a small collection of squares and part of its cover graph, with the rectangular regions denoted by circles. Notice that \{X, Y, B, Z\} covers m, and m and W (or m and B) can both be discarded to produce a minimal cover for the image.

![Figure 1.3 Given collection of squares](image)

![Figure 1.4 Reduced Cover Graph for Fig. 1.3](image)
1.4.4 Getting a Minimal Square Cover from the Cover Graph

Given any cover graph $G_R(I)$ which satisfies the four conditions G1 through G4 listed in the previous section, a minimal square cover for the image I can be constructed by finding a minimal $g$-cover for $G_R(I)$. We obtain a minimal $g$-cover for $G_R(I)$ by labelling it in phases as described in Chapter 5. During each phase we extract non-intersecting sets of maximal paths, consisting of chains of undiscarded vertices and edges of $G_R(I)$. Every alternate vertex in each such chain is retained; and of the remaining vertices, the ones that are covered by the retained vertices are discarded. We show in Chapter 4 that $G_R(I)$ has a constant degree. Each such phase reduces the degree of undiscarded squares by at least two, hence a constant number of phases suffices to label all of $G_R(I)$.

1.4.5 The Existence of a Reduced Cover Graph

The naive cover graph $G_P$ defined earlier clearly satisfies the correctness conditions for a cover graph of the image. What is not obvious is whether there exists any graph appreciably smaller than $G_P$ which also satisfies those conditions. Given below are three transformations, each of which preserves the correctness of any cover graph to which it is applied, and also reduces the number of vertices and edges in the cover graph. In order to represent the cover relationships of the image succintly, we define a new cover graph $G_R = \langle (M_U, R_M), E_R \rangle$, which can be obtained from the definition of $G_P$ as follows.
(i) **Rectangular regions**: Replace every rectangular region which is spanned by a (maximal) sequence of identically sized maximal squares which are vertically (horizontally) contiguous, by its minimum square cover. Call the set of remaining maximal squares $M_\cup$. (Clearly the squares in $M_\cup$ span the entire image.) Delete the vertices associated with the discarded squares, and the edges incident on them.

(ii) **Composite regions**: Replace all the pixels in $G_p$ which are covered by exactly the same set $S$ of squares, with a single region $R_S$ which equals the union of those pixels. Replace the edges incident on the deleted pixels, with edges between $R_S$ and each square in $S$.

(iii) **Prime implicants**: Given two regions $R_p$ and $R_q$ in some square $m$, let $S_p$ and $S_q$ be the sets of all squares which cover $R_p$ and $R_q$ respectively. If $S_p$ is a strict subset of $S_q$, then add the region $R_q$ to the region $R_p$. (The resulting region $R_p \cup R_q$ clearly equals the intersection of all the squares in $S_p$, and is rectangular.) Delete the vertex for $R_q$, and delete its incident edges. (See Figure 1.5.)

![Figure 1.5 Condensing the Naive Cover Graph](image)

Note that we do not actually construct the naive cover graph $G_p$, and transform it to the graph $G_R$. Rather, the above only serves to motivate the existence of the reduced cover graph, which will in fact be constructed as
described in Chapters 3 and 4. There we will show that we can construct a reduced cover graph, whose size is linear in the number of pixels in the image, in $O(\log n)$ time with $n/\log n$ processors.

1.5 Algorithm Minimal_Square_Cover

We now give a high-level description of the Minimal Square Cover Algorithm. The details of individual steps, and proofs of correctness will be developed in subsequent chapters.

1. Determine the maximal squares, as described in Section 2.1. We thus obtain the set $M_1$.

2. Identify rectangular regions, each of which is covered by a contiguous vertical / horizontal sequence of identically sized maximal squares. Divide the longer side of each such rectangle by its width, and retain a minimum square cover for it as described in Section 2.2. Thus we obtain one vertex set $M_1$ for the desired cover graph.

3. For every retained maximal square in $M_1$, determine if it is covered by the union of its neighboring maximal squares, as in Sections 3.1–3.3. If no such cover exists, then mark the square as being essential.

4. Construct the cover graph $G_R(I)$ by constructing the partial cover graphs $G_m$ for each non-essential square $m$ in $M_1$, and unioning the subgraphs componentwise. Each $G_m$ represents the manner in which the pixels of $m$ are covered by the squares neighboring $m$. 
4.1 In order to determine the partial cover graph for square $m$, first derive its *extended cover* $\Xi_m$, which consists of all squares in $M_U$ which are "useful" for covering $m$, as in Chapter 3 (where we show that *not every square which overlaps with a given square $m$ is useful for covering it*).

4.2 Decompose each non-essential square $m$ in $M_U$ into a minimum set of rectangles, each of which can be independently covered by some single square in the extended cover $\Xi_m$ of $m$, as described in Chapter 4. Then obtain the partial cover graph $G_m$ for $m$. At this step we simultaneously determine the remaining vertex set $R_M$, and the edge set $E_R$ for $G_R(I)$.

5. Determine a minimal cover for the image, by coloring $G_R(I)$ as described in Chapter 5.
Chapter 2
Finding Maximal Squares and Dividing Uniform Strips

In this chapter we will determine the maximal squares in the image by using parallel prefix computations. Then we will determine rectangular regions of the image which are spanned by maximal (contiguous) sequences of identically sized maximal squares. We will perform this step also by using parallel prefix computations. For each such rectangular region, we will find a minimum square cover by dividing the longer side of the rectangle by its width.

2.1 Finding Maximal Squares using Prefix Computations

The first step of our algorithm is to find the maximal squares in the image. We begin by computing largest black squares, which are the lower right (diagonal) square suffixes of maximal squares. This initial step not only helps us to find the maximal squares efficiently, but also generates information which will be useful later during the elimination of redundant maximal squares.

Definition 2.1 (largest black square): A largest black square $SE_{\text{L}ij}$ is the largest square of black pixels that has its top left corner (origin) at $(i, j)$, and its diagonally opposite corner to the SouthEast of $(i, j)$. 

17
We also define (and compute) $SW_{ij}$, $NW_{ij}$, and $SE_{ij}$ in a similar manner. We find the largest black squares $SE_{ij}$ in three steps. (i) For every black pixel $(i, j)$ find $R_{ij}$ the length of the longest horizontal black strip containing pixel $(i, j)$ and extending to its right, and also find $D_{ij}$ the length of the longest vertical black strip containing pixel $(i, j)$ and extending downwards. (ii) Hence determine the size of the largest L-shaped region which could potentially form the upper-left border of a black square to the right and below, as the minimum of the distances to the two ends. (iii) Finally, for every pixel, determine the largest black square contained in
the diagonal sequence of L-shaped black borders to the right and below, by combining the L-shaped regions. (See Figure 2.1.) (In the sequel \(L_{ij}\) will mean \(\text{ST}_{ij}\).)

We can cast each of these steps as a parallel prefix computation [LaF-80, KRS-85, CoV-88a]. Given a semigroup \((S, \otimes)\) (where \(\otimes\) is a binary associative operation on \(S\)), a prefix computation over a sequence \(s_1 \ldots s_n\) of elements of \(S\), consists of computing the \(n\) partial products \(s_1 \otimes \ldots \otimes s_k\) for all \(k \leq n\). Similarly, a suffix computation over the sequence \(s_1 \ldots s_n\) of elements of \(S\), consists of computing the \(n\) partial products \(s_k \otimes \ldots \otimes s_n\) for all \(k \leq n\).

A prefix (suffix) computation over \(n\) elements can be performed optimally in parallel, (i) in time \(T, \log n \leq T \leq n\), using an \(n/T\) processor EREW-PRAM, or (ii) in time \(T, \log n / \log \log n \leq T < \log n\), using an \(n/T\) processor CRCW-PRAM [CoV-88a]. The prefix (suffix) operation is also available as a primitive operation in several practical parallel machines such as the Connection Machine [Hil-85, Ble-87], and the Fluent Machine [Ran-89].

As a prelude to determining the largest black squares as described above, we define two geometric objects, a hammerhead and an arrowhead, which can each be combined pairwise in an associative manner. These two intermediate computations allow us to exploit the power of parallel prefix computations for finding largest black squares.
Definition 2.2 (hammerhead): The hammerhead $H_i$ of a sequence of pixels $x_i \ldots x_i+l-1$ is $H_i = (d, l)$; where the depth $d$ of the hammerhead is the maximum number of ones which prefix the sequence $x_i \ldots x_i+l-1$, and $l$ is the length of the sequence. Given two adjacent sequences of pixels $x_i \ldots x_{i+l_1-1}$ and $x_{i+l_1} \ldots x_{i+l_1+l_2-1}$, with respective hammerheads $(d_1, l_1)$ and $(d_2, l_2)$, the combined hammerhead $\langle d_1, l_1 \rangle \otimes (d_2, l_2)$ equals $(d, l_1+l_2)$; where $d = d_1$ if $d_1$ is less than $l_1$, and $d = d_1 + d_2$ otherwise. (See Figure 2.2.)

![Figure 2.2 Associative combination of Hammerhead regions](image)

Definition 2.3 (arrowhead): An arrowhead $A_{ij} = (d, w)$ with origin at $(i, j)$ is an L-shaped region containing only black pixels, whose two arms have equal depth $d$ and equal width $w$. (An arrowhead $A_{ij}$ denotes that pixel $(i, j)$ can potentially be the origin of a black square of size at most $d$, and we have verified that the top $w$ rows and left $w$ columns contain only black pixels.) Given two diagonally adjacent arrowheads $(d_1, w_1)$ and $(d_2, w_2)$ which are similarly oriented, the combined arrowhead $(d_1, w_1) \otimes (d_2, w_2)$ equals $(\min(d_1, d_2 + w_1), (w_1 + w_2))$. (See Figure 2.3.)
First we address the problem of finding the longest horizontal / vertical black strips in the image. Given the \( i \)th row of pixels, we wish to compute \( R_{ij} \), the length of the largest contiguous sequence of black pixels starting at each pixel \( (i, j) \) and extending to its right. We initialize a row of hammerheads \( H_{ij} \), where \( H_{ij} \) is set to \( \langle 1, 1 \rangle \) if the pixel at \( (i, j) \) is black, and \( \langle 0, 1 \rangle \) otherwise. We compute the suffixes of each row of hammerheads by using the associative combining operator defined above. The depth of the resulting hammerhead \( H_{ij} \otimes \ldots \otimes H_{i, n} \) equals \( R_{ij} \). A similar computation is performed along the columns of the image and yields the value of \( D_{ij} \), the length of the largest contiguous sequence of black pixels starting at each pixel \( (i, j) \) and extending downwards.

Next we initialize an arrowhead of unit width at each pixel \( (i, j) \) as \( A_{ij} = \langle \min(R_{ij}, D_{ij}), 1 \rangle \). We can combine arrowheads associatively along the diagonals of the image, using a parallel suffix computation as described earlier. When the width of an arrowhead equals its depth, we have
determined the largest black square \( L_{ij} \) at \((i, j)\). The size of \( L_{ij} \) equals the 
*depth* of \( A_{ij} \otimes \ldots \otimes A_{i+k, j+k} \), where \( k \) is such that \( \max(i+k, j+k) \) equals \( n \).

#### 2.1.1 Largest Black Squares which are Maximal

We observe that *largest black squares* are the lower right square 
suffixes of maximal squares, and are bounded by white pixels to the right 
and/or below. Clearly a largest black square \( L \) is maximal *iff* it is not 
contained in a larger black square \( M \), and the *origin* (top left corner) of 
such an \( M \) can only be above and/or to the left of the origin of \( L \). Further, 
by the suffix property of largest black squares, the existence of a larger 
black square above (to the left of) the origin of \( L \), implies the existence of 
one *immediately above* (immediately to the left of) the origin of \( L \). We 
associate a boolean variable \( M_{ij} \) with every pixel \((i, j)\), which indicates 
whether \((i, j)\) is the *origin* (top left corner) of a maximal square; if so, its size 
is given by \( L_{ij} \). Once we have determined the sizes of the largest black 
squares as described in the previous section, we can identify the maximal 
squares with only a constant number of computations per pixel, as shown 
in Lemma 1 below.

**Lemma 1** Let \( P \) be the largest black square with its *origin* at \((i, j)\), and 
\( l_{ij} \geq 1 \) be its size. Then \( P \) is not *maximal* *iff* \( l_{i+\delta i, j+\delta j} > l_{ij} \), for some \((\delta i, \delta j)\) 
in \{(0, 1), (1, 0), (1, 1)\}.

**Proof** (Refer to Figure 2.4.)

Let \( B(r) \) be the predicate "*every pixel in the region \( r \) is black". Let \( V, H, 
D, X \) be the four squares of size \((l_{ij}+1)\), with their top-left corners
respectively at \((i - 1, j), (i, j - 1), (i - 1, j - 1),\) and \((i, j)\). Clearly \(P\) is contained in each of \(V, H\) and \(D\). Hence if any of \(V, H\) or \(D\) is all black then \(P\) is not maximal. Further \(H, V, D\) and \(X\) are the only 4 squares of size \(l_{ij} + 1\) that contain \(P\). By hypothesis \(B(X)\) is false. Hence \(P\) is not maximal implies \(B(V) \lor B(H) \lor B(D)\); which is equivalent to saying that \(l_{i \delta i, j \delta j} > l_{i j}\), for some \((\delta i, \delta j)\) in \(\{(0, 1), (1, 0), (1, 1)\}\).

\[\square\]

### 2.2 Dividing Uniform Strips

Having obtained the maximal squares, we identify rectangular regions, each of which is covered by a contiguous vertical / horizontal sequence of identically sized maximal squares. We divide the longer side of each such rectangle by its width, and retain a minimum square cover for it as described below. In the sequel we will only be concerned with identifying all possible ways of covering the non-essential squares amongst those which were retained during the division of uniform strips. Without this step of dividing uniform strips, the cover graph can have a maximum degree of \(\Omega(\sqrt{n})\) in the worst case (consider a horizontal sequence of identical maximal squares of side \(\frac{1}{3}\sqrt{n}\), spaced one pixel apart). If we do
not subdivide uniform strips, we will not be able to construct the cover graph or process it efficiently.

**Definition 2.4 (uniform strip, uniform_tail):** A horizontal (vertical) uniform strip is a maximal sequence of identically sized maximal squares, whose origins are horizontally (vertically) contiguous. A (horizontal) uniform_tail $U_{ij} = (s, d, l)$ is the right end of a (horizontal) uniform strip which has its origin at $(i, j - d + 1)$; where $w$ is the width of the rightmost maximal square in the tail, the depth $d$ is the length of the uniform strip at the end of the tail, and $l$ is the length of the tail. We can associatively combine two adjacent uniform_tails into a single uniform_tail, as $(w_2, d_2, l_2) \otimes (w_1, d_1, l_1) = (w_1, d, (l_1 + l_2))$; where $d = d_1$ if ($w_1 \leq w_2$) or ($d_1 < l_1$), and $d = d_1 + d_2$ otherwise. (See Figure 2.5.)

![Diagram of combining tails of adjacent Uniform Strips](image)

**Figure 2.5 Combining tails of adjacent Uniform Strips**

The definition of a uniform strip does not imply that all the pixels surrounding a uniform strip are white. We retain a minimum cover for a uniform strip, by dividing its longer side by its width in the following
manner. For every maximal square \( m \), determine the leftmost (uppermost) corner (origin) of the uniform strip \( U \) which contains it. If the origin of the uniform strip is inside a maximal square \( h \) which is larger than the width of the strip, redefine the origin to be shifted just past the right (bottom) edge of \( h \). Retain the leftmost (uppermost) maximal square in \( U \), and all those maximal squares in \( U \) whose origin is displaced from the origin of \( U \) by an integral multiple of the width \( \omega \) of the strip. In addition to the squares retained by the previous condition, retain the rightmost (lowermost) square in \( U \), unless its last \((\omega - 1)\) columns (rows) are covered by a larger maximal square. Discard all the others. (See Figure 2.6.)

![Uniform strips, with surrounding black regions](image1)

![Squares retained after division](image2)

Figure 2.6 Dividing Uniform Strips

Let \( M_{ij} \) be the maximal square with its origin at \( (i, j) \) and let \( w_{ij} \) be its width. In order to find the left corner of (say) a horizontal uniform strip, we must compute the length of the uniform strip to the left of pixel \( (i, j) \), as \( HS_{ij} = \min \{ k | 0 < k < j \text{ such that } w_{i,j-k} = w_{ij} \} \). This can be transformed to a prefix computation by a technique similar to that of Section 2.1. The required computation is defined recursively as \( U_{ij} \leftarrow U_{i,j-1} \Box (w_{ij}, w_{ij}, w_{ij}) \). Observe that this computation proceeds from right to left, starting at the origin of each maximal square. \( HS_{ij} \) equals the depth of \( U_{i1} \Box \ldots \Box U_{ij} \).
Chapter 3

Conditions for the Redundancy of Maximal Squares

In this chapter we will determine whether a given maximal square $m$ is essential. If $m$ is not essential, we will characterize concisely the distinct sets of maximal squares which cover $m$. A naive approach might be to first determine all the squares which overlap with $m$. In Figure 3.1, $\Omega(\sqrt{n})$ maximal squares $T_1 \ldots T_k$ overlap with $m$. However for every set $S_i$ of maximal squares containing $T_i$ such that $S_i$ covers $m$, $(S_i - T_i)$ also covers $m$, and none of the squares $T_1 \ldots T_k$ are "useful" for deciding if $m$ can be discarded. If a square $T$ is "useful" for covering $m$, there exists a set $S$ of squares containing $T$, such that $S$ covers $m$ but $(S - T)$ does not. This observation motivates the following definitions.

Definition 3.1 (essential square, irreducible cover, useful square): An essential square is a maximal square which contains at least one pixel which is not covered by any other maximal square. Let $m$ be a non-essential maximal square. An irreducible cover $\xi$ of $m$ is a set of maximal squares excluding $m$ which covers $m$, but no proper subset of which covers $m$. A useful square of $m$ is a maximal square which belongs to some irreducible cover of $m$. 

26
3.1 Basic Cover Relations

We consider the set $U_m$ of the useful squares of $m$, and the partial order $<_m$ over $U_m$ defined by $A <_m B$ iff $A \cap m \subseteq B \cap m$. This partial order separates the squares in $U_m$ into chains, each of which starts with a square that overlaps maximally with $m$. We show in Section 3.5 that we can trace each such chain in decreasing order of overlap with $m$, starting with the maximal element of a chain, by a constant sized sequence of cover refinements. Further, as we show in this section and the next, we can efficiently construct the basic cover of $m$ consisting of those useful squares of $m$ which overlap maximally with it.

**Definition 3.2 (basic cover):** The basic cover $\beta_m$ for a maximal square $m$ is the set of all those useful squares of $m$ which overlap maximally with it. A square $S$ overlaps maximally with $m$ if $S \cap m$ is not strictly contained in $T \cap m$, for any square $T$ other than $m$ and $S$. 
We observe that if a square $m$ is not essential, the set $U_m$ of all of its useful squares is non-empty and covers $m$. Given two squares $A$ and $B$ in $U_m$, if $A <_m B$, then $U_m - \{A\}$ also covers $m$. If we discard from $U_m$ all those squares which do not overlap maximally with $m$, we get the basic cover of $m$. Hence a square $m$ is not essential (i.e. is potentially redundant) iff its basic cover is non-empty and covers $m$.

![Figure 3.2 A basic cover which is not minimal](image)

By the above definitions a maximal square $m$ is neither a useful square for itself, nor does it belong to its own basic cover. In Figure 3.2 the basic cover for $m$ is \{T, L, D, E\}, whereas $m$ can be covered (minimally) by either \{T, D, E\} or \{L, D, E\}. Thus we see that a basic cover is not necessarily minimal. We will now show that the basic cover contains at most four (maximal) squares, by showing that any chosen edge of a non-essential square $m$ is cut by at most one square in its basic cover.

**Definition 3.3** We say that a square $S$ cuts an edge $e$ of $m$ if a part of $e$ lies in the interior of $S$. 
Theorem 1 Any chosen edge of a non-essential square $m$ is cut by at most one square in the basic cover $\beta_m$ of $m$.

![Diagram](image)

(a. Abutting / overlapping  b. Separated squares)

Figure 3.3 Bounding the size of a Basic Cover

Proof (Refer to the cases in Figure 3.3.)

Let $e$ be some edge of $m$ which is cut by more than one square in $\beta_m$. Without loss of generality, let $e$ be the top edge of $m$. Let $A$, $B$, $C$ ... etc. be the sequence of squares which cut $e$, in ascending order of the $X$-coordinate of their left edges. Let $A$, $B$ be the first two squares in the sequence.

(a) If $A$ and $B$ abut or overlap along $e$, then we can find another square $S$ in $\beta_m$ such that $S \cap m$ strictly contains $A \cap m$ and $B \cap m$. Then neither $A$ nor $B$ belongs to $\beta_m$.

(b) Else $A$ and $B$ are separated by at least one white pixel $p$. Consider pixel $q$ of $m$, which lies along $e$ and is vertically below $p$. Let square $S$ in $\beta_m$ cover $q$. $S$ does not cut $e$, else its left edge is to the left of the left edge of $B$. Hence $S$ cuts the right edge or the left edge of $m$. Then $S \cap m$ strictly contains either $A \cap m$ or $B \cap m$. Hence either $A$ or $B$ does not belong to $\beta_m$.

In either of the above two cases we have a contradiction. $\Box$
**Theorem 2** The *basic cover* $\beta_m$ for a non-essential square $m$ contains at most four (maximal) squares.

This theorem follows trivially from Theorem 1 above.

### 3.2 Finding the Basic Cover

We now show that we can determine if a given maximal square is non-essential, and if so, find its basic cover, all with only a constant number of operations. Our algorithm for finding the basic cover relies on examining the lengths of maximal black strips which border the square $m$ on each side, starting at each corner. This yields the sizes and locations of maximal squares which overlap maximally with $m$ and which are potentially *useful squares* of $m$. A little case analysis helps to eliminate those squares which either do not have a cover completion, or whose overlap with $m$ is subsumed by another square.

**Theorem 3** Given a maximal square $m$, in $O(1)$ operations we can decide if $m$ is *essential*. If $m$ is not essential, we can construct its *basic cover* $\beta_m$ in $O(1)$ operations.

**Proof**

We prove the above by constructing a set $B_m$ such that $B_m$ is a superset of $\beta_m$, and $B_m$ contains at most 12 squares. We complete the proof by verifying the following assertions in Lemmas 2, 3 and 4.

(i) We can construct a constant sized superset $B_m$ of $\beta_m$ in $O(1)$ operations.
(ii) We can determine in $O(1)$ operations whether $B_m$ covers $m$, and thus determine whether $m$ is essential.

(iii) If $m$ is not essential, we can construct $\beta_m$ from $B_m$ in $O(1)$ operations.

Lemma 2 Given a maximal square $m$, in $O(1)$ operations we can construct $B_m$, a constant sized superset of its basic cover $\beta_m$.

Proof (See Figures 3.4 through 3.8.)

In this proof we will assume that $m$ is not essential, and try to determine the squares in its basic cover. In this process we will carefully avoid examining too many of the squares which overlap with $m$, else we will not be able to meet our stated time bounds. The criteria we will use for not adding a square $S$ to $B_m$ are: (i) $S$ does not overlap maximally with $m$, or (ii) $S$ cannot possibly be a useful square of $m$. This way we can guarantee that the only squares we are not adding to $B_m$ cannot possibly belong to $\beta_m$, and $B_m$ is indeed a superset of $\beta_m$. If $m$ is in fact essential, the constructed set $B_m$ will either be empty, or it will not cover $m$.

Let us first find a square $T$ in $\beta_m$ such that $T$ cuts the North edge $n$ of $m$. (We can do likewise for the remaining edges $w$, $s$ and $e$, respectively the West, South and East edges of $m$.) The three cases that arise are: (1) $T$ contains no corner of $n$, (2) $T$ contains one corner of $n$, and (3) $T$ contains both corners of $n$.
Let the size of \( m \) be \( s \). In order to distinguish between the three cases discussed above, we first examine the sizes of the largest black squares which have one corner respectively at pixels \((i, j-1), (i-1, j), \) and \((i, j+s), \) and which extend into the interior of \( m \). For notational convenience, in this proof we let \( L[i, j, SE] \) stand for \( seL_{i,j} \). Let \( t, l, r \) respectively be the sizes of \( T = L[i-1, j, SE], W = L[i, j-1, SE] \), and \( E = L[i, j+s, SW] \).

![Diagram of squares and notations]

**Figure 3.4 T contains no corners of \( n \)**

**Case 1** (\( T \) contains no corner of \( n \). See Figure 3.4.) If \( m \) is not essential, then surely there are squares \( W \) and \( E \) in \( \beta_m \) which cover the left and right corners of \( n \) respectively. By Theorem 1, neither \( W \) nor \( E \) cuts \( n \). Let \( p \) be the pixel at \((i-1, j+l-1)\), which is in the row just above \( n \) and the column to the right of the right edge of \( W \). A little thought will convince the reader that if \( T \) is a useful square of \( m \), \( T \) must contain \( p \). Hence we can determine the left edge of \( T \) by knowing the size of \( L[i-1, j+l-1, SW] \). Similarly, we can determine the right edge of \( T \) by knowing the size of \( L[i-1, j+l-1, SE] \). If \( T \) overlaps/abuts \( E \), then \( T \) is a useful square of \( m \) and we can add \( T \) to \( B_m \).
Case 2 (T contains one corner of n.) Without loss of generality, let T contain the left corner of n. The two subcases that arise are: (2.1) the pixel at \((i-1, j-1)\) is white, and (2.2) the pixel at \((i-1, j-1)\) is black.

Case 2.1 (The pixel at \((i-1, j-1)\) is white. See Figure 3.5.) We get another two subcases. Subcase (2.1.1) arises when \(l\) does not equal \(t\), and (2.1.2) arises when \(l\) equals \(t\).

Case 2.1.1 \((l \neq t)\) Without loss of generality, let \(t < l\); in particular \(t: l-1\). We see that W covers \(l-1\) columns and \(l\) rows along the top left corner of m. Further, T covers at most \(l-1\) columns and \(l-2\) rows along the top left corner of m. Thus \(T \cap m\) is strictly contained in \(W \cap m\). and we do not add T to \(B_m\).

Case 2.1.2 \((l = t)\) Clearly \((i-1, j)\) is the origin of maximal square T. Let p be the pixel at \((i, j+t)\). If T is a useful square of m, some other square E in \(\beta_m\) contains p. By Theorem 1, E does not cut n. Also, E does not cut edge w of m, otherwise \(E \cap m\) would strictly contain
T ∩ m. Hence E cuts the East edge e of m. Hence if \( r - 1 + t \leq s \), then E contains p, and T is a useful square of m, and we can add T to \( B_m \).

![Diagram](image)

Figure 3.6 T contains one corner of n, and pixel \((i-1, j-1)\) is black

**Case 2.2** (The pixel at \((i-1, j-1)\) is black. See Figure 3.6.) Clearly neither of T and W are maximal squares. A little thought will show that \((i + t - 1, j + t - 1)\) is the lower right corner of a maximal square U, such that \( U \cap m \) is a strict superset of \( T \cap m \). We can determine the origin of U by knowing the size of \( L[i + t - 1, j + t - 1] \), and add U to \( B_m \).

![Diagram](image)

\[ a \max(ll, rr) = s - 2 \]

\[ b \quad ll = rr = s \]

Figure 3.7 Size of T equals the size of m

**Case 3** (T contains both corners of n.) The two subcases which arise are:

1. **(3.1)** size of T equals s, and **(3.2)** size of T is greater than s. Let \( ll, rr \) respectively be the sizes of the longest vertical black strips extending downwards from the pixels at \((i, j-1)\) and \((i, j+s)\).
Case 3.1 (size of $T$ equals $s$. See Figure 3.7.) This case holds whenever $\max(l, r) \leq s - 2$. If $l = r = s$, then the pixels at $(i - 1, j - 1)$ and $(i - 1, j + s)$ are white (since $m$ is maximal), and in this case also the size of $T$ equals $s$. In either case, $(i - 1, j - 1)$ is the origin of $T$, and we can add $T$ to $B_m$.

![Diagram a: T cuts left edge of m](image)

![Diagram b: T cuts left & right edges of m](image)

Figure 3.8 Size of $T$ is greater than the size of $m$

Case 3.2 (size of $T$ is greater than $s$. See Figure 3.8.) This case is true whenever $\max(l, r) \leq s - 1$. The two subcases which arise are: (3.2.1) $T$ cuts only one of the edges $w$ or $e$ of $m$, and (3.2.2) $T$ cuts both $w$ and $e$.

Case 3.2.1 ($T$ cuts only one of $w$ or $e$.) Without loss of generality, let $T$ cut $w$. Since $m$ is maximal, the pixel at $(i + s - 1, j - 1)$ is white, and the bottom edge of $T$ is along row $i + s - 1$. We can verify whether the right edge of $T$ coincides with the right edge of $m$ by knowing the size of $L[i + s - 1, j, NE]$. We can determine the origin of $T$ by examining the size of $L[i + s - 1, j + s - 1, NW]$, and we can add $T$ to $B_m$.

Case 3.2.2 ($T$ cuts both of $w$ and $e$.) Since $m$ is maximal, the pixels at $(i + s - 1, j - 1)$ and $(i + s - 1, j + s)$ are white, and the bottom edge of $T$ is along row $i + s - 1$. We can determine the right edge of $T$ by knowing
the size of $L[i + s - 1, j, NE]$. Hence we can determine the lower right corner of $T$, and thence the origin of $T$, and we can add $T$ to $B_m$.

This completes our proof.

\[\square\]

**Lemma 3** Given a maximal square $m$, and a set $B_m$ which is a constant sized superset of its basic cover $\beta_m$, in $O(1)$ operations we can determine whether $m$ is essential.

![Figure 3.9. Verifying a cover](image)

**Proof** (Refer to Figure 3.9.)

Since $B_m$ is a superset of $\beta_m$, $B_m$ covers $m$ if and only if $m$ is not essential. We can verify whether $B_m$ covers $m$ in the following manner. Sort the squares in $B_m$ cyclically about the perimeter of $m$. Initialize the covered region $R$ of $m$ to the empty region. For each square $S$ in $B_m$, add $S \cap m$ to $R$, to get the augmented region of $m$ covered by $S$. We can keep track of $R$ by keeping track of its perimeter, which consists of a constant number of line segments. While adding $S \cap m$ to $R$, delete the part of the perimeter of $R$ contained in $S$, and add the part of the perimeter of $S \cap m$ not contained inside $R$. Once we have accounted for each square in $B_m$, it is
easy to check if \( R \) equals \( m \). Since the size of \( B_m \) is a constant, the above steps take \( O(1) \) operations.

\[ \square \]

**Lemma 4** Given a non-essential square \( m \), and a set \( B_m \) which is a constant sized superset of its *basic cover* \( \beta_m \), in \( O(1) \) operations we can construct \( \beta_m \).

**Proof**

Since the size of \( B_m \) is a constant, in \( O(1) \) operations we can compare every pair of squares \( X \) and \( Y \) in \( B_m \), and check if \( X <_m Y \). Thus we can determine all the maximal elements of \( B_m \) (with respect to the \( <_m \) relation), which yields the desired *basic cover* \( \beta_m \).

\[ \square \]

### 3.3 Normalizing the Basic Cover

Given a non-essential square \( m \), every square \( S \) in its basic cover is guaranteed to be a *useful square* of \( m \), and to overlap maximally with it. However, \( S \) may already have been discarded during the division of *uniform strips*. In that case we need to locate the predecessor / successor \( P \) of \( S \) in the uniform strip containing \( S \) and \( P \). If \( P \) overlaps with \( m \), we add \( P \) to the *normalized basic cover* of \( m \).
3.4 Extended Cover Relations

As we saw in Chapter 1, our minimal cover algorithm executes in a sequence of phases. During each phase \( i \), we start with a set \( C_i \) of maximal squares such that \( C_i \) covers the image. We then select a set \( D_i \) of squares which can be simultaneously discarded from \( C_i \) to produce another set \( C_{i+1} \) which is also guaranteed to cover the image. (The set \( C_0 \) equals \( M_I \), and the set \( C_1 \) equals \( M_U \)). A square \( m \) in \( C_i \) can be discarded only if \( C_i - \{ m \} \) covers \( m \). A naive way to verify this condition would be to obtain the union of all the squares in \( C_i - \{ m \} \) and check if it covers \( m \).

However, as we saw earlier in this Chapter, a square \( S \) is "useful" for covering \( m \) only if \( S \) belongs to some irreducible cover of \( m \). Consider the set \( U_m \) consisting of all the useful squares of \( m \). We know from our earlier discussions that if \( C_i - \{ m \} \) covers \( m \), then \( (C_i \cap U_m) - \{ m \} \) also covers \( m \). Hence we can check whether \( m \) can be discarded, by taking the union of a smaller set of squares. As we show in Figure 3.10, and explain in the accompanying discussion, the set \( U_m \) can contain \( \Omega(\sqrt{n}) \) maximal squares in the worst case.

Consider the collection of maximal squares shown in Figure 3.10. The square \( m \) has the following irreducible covers: \( \{ T, J, R \}, \{ T, K, R \}, \{ T, K, Q \}, \{ T, L, R \}, \{ T, L, Q \}, \{ T, I, P \} \). In fact, we can extend this example so that \( \Omega(\sqrt{n}) \) squares overlap along the bottom left, and the bottom right corners of \( m \), in the manner of J-L and P-R. Thus \( m \) has \( \Omega(\sqrt{n}) \) useful squares, and any processing based upon the local neighborhood of each square, may not be possible within our stated resource bounds.
Notice however, that the intersection of L and R contains at least one pixel which is not covered by any other square; hence one of L and R is always present in every cover of the image. In which case, any square whose intersection with m is a strict subset of \( L \cap R \), cannot in some sense be \textit{globally useful} for covering m. A similar situation is also shown in Figure 3.11, where m can be covered by \( \{L, R, T\} \). However, A and B are pairwise essential, and in every cover of the image, m - T is guaranteed to be covered by one of A or B. In this example also, neither of L and R is globally useful for covering m. These observations motivate us to define the \textit{extended cover} \( \Xi_m \) of a non-essential square m, and we will show in Theorem 4 that \( \Xi_m \) is in fact the set of squares which are \textit{globally useful} for covering m.
Definition 3.4 (pairwise essential squares, extended cover): Two maximal squares $P$ and $Q$ are pairwise essential if $P \cap Q$ contains a pixel $p$ which does not belong to any other maximal square in $M_I$ (other than $P$ or $Q$). The extended cover $\Xi_m$ of an inessential square $m$ is a subset of $M_U$, such that for every square $S$ in $\Xi_m$: (i) $S$ is a useful square of $m$, and (ii) $(S \cap m)$ is not a strict subset of $(P \cap Q)$ for any squares $P$ and $Q$ which are pairwise essential.

We will show in Sections 3.5 and 3.6 that $\Xi_m$ contains $O(1)$ maximal squares, and we can construct it with $O(1)$ operations. However, if knowing
the set $\Xi_m$ is to be adequate for us to determine whether $m$ can be safely
discarded from an arbitrary cover $C_i$ of the image, we must assure
ourselves that $\Xi_m$ contains all the squares which are in some sense
globally useful for covering $m$. This will be made more precise in the
following theorem.

**Theorem 4** If a subset $M_C$ of $M_U$ covers the image, then every square $m$
in $M_U - M_C$ is covered by the squares in $M_C \cap \Xi_m$.

**Proof**

Clearly $M_C$ covers $m$. We also know that there is a set $U_m$ of all the
useful squares of $m$, such that whenever $m$ is covered by any set $C$ of
squares (not containing $m$), $C \cap U_m$ covers $m$. Hence $M_C \cap U_m$ also covers
$m$. Let $S$ be a square in $U_m$ but not in $\Xi_m$. By definition, we have a pair of
squares $P$ and $Q$ such that $P$ and $Q$ are pairwise essential, and $S \cap m$ is
strictly contained in $P \cap Q$. Also, since $P$ and $Q$ are pairwise essential, at
least one of them belongs to $M_C$, and $S \cap m$ is covered by $M_C \cap \Xi_m$, and the
result follows. $\square$

### 3.5 Finding the Extended Cover

In this section we show that we can determine the extended cover $\Xi_m$ of
a square $m$, starting with its basic cover $\beta_m$, and adding to it the basic
cover $\beta_S$ for every square $S$ in $\beta_m$. Given two squares $A, B$ in $\Xi_m$, we say
that $A <_m B$ iff $A \cap m$ is strictly contained in $B \cap m$. This partial order
separates the squares in $\Xi_m$ into chains, each of which starts with a square
that overlaps maximally with \( m \), i.e. is in its basic cover. We show in the following theorem that each such chain can be traced in decreasing order of overlap with \( m \), by a constant sized sequence of basic cover refinements.

**Theorem 5** Let \( P \) be a maximal square in the extended cover \( \Xi_m \) of a non-essential square \( m \), but not in its normalized basic cover \( \beta_m \). Then \( P \) belongs to the basic cover \( \beta_Q \) of some \( Q \) in \( \beta_m \).

**Proof**

Let top(\( P \)), bot(\( P \)), lt(\( P \)), rt(\( P \)) respectively denote the four edges of a square \( P \). Let the X-axis increase to the right, and Y-axis increase downwards, i.e. the image is in the fourth quadrant. Let usd(\( A \), \( B \)) denote the division of the uniform strip containing squares \( A \) and \( B \), and the associated discarding of squares. Let \( P \in (\Xi_m - \beta_m) \). Clearly there exists \( Q \) in \( \Xi_m \) st \( (Q \cap m) \supseteq (P \cap m) \). Assume wlog that \( P \) extends above the top edge of \( m \). Consider the position of \( P \) wrt \( m \).

\[ \text{\includegraphics{diagram.png}} \]

**Figure 3.12** \( P \) covers no corners of \( m \)

**Proof**

Let top(\( P \)), bot(\( P \)), lt(\( P \)), rt(\( P \)) respectively denote the four edges of a square \( P \). Let the X-axis increase to the right, and Y-axis increase downwards, i.e. the image is in the fourth quadrant. Let usd(\( A \), \( B \)) denote the division of the uniform strip containing squares \( A \) and \( B \), and the associated discarding of squares. Let \( P \in (\Xi_m - \beta_m) \). Clearly there exists \( Q \) in \( \Xi_m \) st \( (Q \cap m) \supseteq (P \cap m) \). Assume wlog that \( P \) extends above the top edge of \( m \). Consider the position of \( P \) wrt \( m \).
Case 1  \( P \) covers no corners of \( m \).

Since \((Q \cap m) \supseteq (P \cap m)\), and \( P \) covers no corners of \( m \): \( \text{lt}(Q) \leq \text{lt}(P) \), \( \text{rt}(Q) \geq \text{rt}(P) \), \( \text{top}(Q) \leq \text{top}(m) \), and \( |Q| \geq |P| \). Since \( P \) is unbounded below, it is bounded to its left and right by white pixels, and \( \text{top}(Q) > \text{top}(P) \). Since \( \text{top}(Q) > \text{top}(P) \) and \( |Q| \geq |P| \), \( \text{bot}(Q) \geq \text{bot}(P) \). If \( |Q| = |P| \), then \( Q \) was discarded during \( \text{usd}(P, Q) \), and \( P \in \beta_m \), a contradiction. Else \( |Q| > |P| \). Let \( Q \) cover no corner of \( m \), as in Figure 3.12(a). If any square \( R \) other than \( Q \) covers the NE corner of \( Q \); then \( R \) either forms a uniform strip with \( Q \), or \( R \) does not abut / overlap the top edge of \( m \). In the latter case \( Q \) is essential, and \( P \notin \Xi_m \). In the former case, the topmost square \( T \) in the uniform strip containing \( \{Q, R\} \) is essential, \((T \cap m) \supseteq (P \cap m)\), and again \( P \notin \Xi_m \).

Else let \( Q \) cover one corner of \( m \), and \( \text{wlog} \) let \( \text{rt}(Q) > \text{rt}(m) \), as in Figure 3.12(b). Either the marked pixels of \( Q \) are covered by the same square \( R \); or \( Q \) is pairwise essential with \( m \), and \( P \notin \Xi_m \). In the former case \( \text{rt}(R) > \text{rt}(Q) \), and \( |R| \geq |Q| \). If \( |R| > |Q| \), \( Q \) is part of a uniform strip all of which is covered by two larger squares \( R \) and \( m \), and \( Q \) was discarded during \( \text{usd}(Q) \). Else \( Q \) and \( R \) form a uniform strip, and \( Q \) was discarded during \( \text{usd}(Q, R) \), a contradiction.

The last subcase is \( Q \) covers two corners of \( m \), as in Figure 3.13. Consider \( R \subseteq Q \) \( \text{st} \) \( R \) covers the SE corner \( q \) of \( Q \). Clearly \( R \) has its SE corner to the right of and / or below \( q \). It is clear that if \( Q \) is not essential, and \( (Q, m) \) are not pairwise essential, then \( R \) covers pixel \( t \) or pixel \( b \). In either case, \( R \) and \( Q \) are of the same size, and must be pairwise essential \( \text{wrt} \) the
Case 2 \(P\) covers only one corner of \(m\).

Since \((Q \cap m) \supsetneq (P \cap m)\): we have (i) top\((Q) \leq\) top\((m)\) and lt\((Q) \leq\) lt\((m)\), (at least one inequality is strict), and (ii) \(P\) is maximal implies top\((Q) >\) top\((P)\) or lt\((Q) >\) lt\((P)\). Having constrained the origin of \(Q\) as above, we see that bot\((Q) <\) bot\((m)\) and rt\((Q) <\) rt\((m)\), since \(m\) is maximal (see Figure 3.14(a)). By examining the marked pixels in Figure 3.14(b), one can see that if top\((P) <\) top\((Q) <\) top\((m)\) and lt\((P) <\) lt\((Q) <\) lt\((m)\), then \(Q\) is essential, hence
$P \in \Xi_m$. Hence $(\text{top}(Q) \leq \text{top}(P) \text{ or } \text{top}(Q) = \text{top}(m))$ or $(\text{lt}(Q) \leq \text{lt}(P) \text{ or } \text{lt}(Q) = \text{lt}(m))$.

\[ \begin{array}{cc}
\text{covered by same square} & \\
Q & \text{c Paired pixels of } Q \\
\text{m} & \\
\end{array} \]

\[ \begin{array}{cc}
Y & \text{d Configuration of } P, Q, m \\
\text{m} & \\
\end{array} \]

Figure 3.14 (contd.) $P$ covers one corner of $m$

Without loss of generality, let $\text{top}(Q) \leq \text{top}(P)$. One can easily see that the marked pixels in Figure 3.14(c) are covered by the same square (say $T$) in $\Xi_Q$, unless $Q$ is essential. In the latter case $P \in \Xi_m$. In the former case, $Q$ is maximal only if $T$ and $Q$ are of the same size; and $T$ covers both marked pixels implies that it abuts / overlaps $m$: whence we get that $Q$ must have been discarded during $\text{usd}(T, Q)$, and $Q \not\in \Xi_m$. Thus we get $\text{top}(Q) = \text{top}(P)$. In this case also, if $\text{lt}(P) < \text{lt}(Q) < \text{lt}(m)$, then $Q$ is essential as in Figure 3.14(b). Putting all the above arguments together, we get $(\text{top}(Q) = \text{top}(P) \text{ and } \text{lt}(Q) = \text{lt}(m))$ or $(\text{top}(Q) = \text{top}(m) \text{ and } \text{lt}(Q) = \text{lt}(P))$.

Let us fix the relative placements of $P$, $Q$ and $m$ as in Figure 3.14(d) above. If $Q \in \beta_m$, $(\exists X \in \beta_m) st (X \cap m) \supseteq (Q \cap m)$. Since $X$ covers all of $Q \cap m$, it cannot extend past $\text{lt}(m)$, else $Q$ is not maximal. Since $X$ covers the NW corner of $m$, it cannot extend past either $\text{rt}(m)$ or $\text{bot}(m)$, else $m$ is not
maximal. If $X$ extends past $\text{top}(m)$, then $X$ and $Q$ are of the same size, $\text{top}(Q) < \text{top}(X) < \text{top}(m)$, and $X$ must have been discarded during $usd(Q, X)$: thus $Q$ is in the normalized basic cover of $m$.

Since $P \notin \Xi_m$ and $(P \cap m) \supseteq (Q \cap m)$, we know that $Q$ is not essential. If $P \in \beta Q$, $(\exists \ Y \in \beta Q)$ s.t. $(Y \cap Q) \supseteq (P \cap Q)$. We note that since $P$ and $Q$ are maximal, $P$ is bounded both above and below by white pixels, and $Q$ is bounded to its left and right by white pixels. If $\text{lt}(Y) < \text{lt}(m)$ & $\text{bot}(Y) > \text{bot}(P)$, then $Y$ is essential, and $P \notin \Xi_m$. There are only three possibilities for $Y$: either (i) $\text{top}(Y) = \text{top}(P)$, $\text{bot}(Y) = \text{bot}(P)$ and $\text{lt}(P) < \text{lt}(Y) < \text{lt}(Q)$; or (ii) $\text{lt}(Y) = \text{lt}(Q)$, $\text{rt}(Y) = \text{rt}(Q)$, $\text{top}(Y) < \text{top}(Q)$ and $\text{bot}(P) \leq \text{bot}(Y) < \text{bot}(Q)$; or (iii) $\text{rt}(Y) = \text{rt}(Q)$, $\text{bot}(Y) = \text{bot}(P)$ and $\text{lt}(P) < \text{lt}(Y) < \text{lt}(Q)$. In case (i) $Y$ is discarded during $usd(P, Y)$; in case (ii) $Q$ is discarded during $usd(Y, Q)$; and in case (iii) $Q$ is discarded during $usd(Q)$. In all cases there is a contradiction, whence we have $(P \in \beta Q)$ and $(Q \in \beta_m)$ as desired.

![Diagram](image)

**Figure 3.15** $P$ covers two corners of $m$
Case 3 P covers two corners of \(m\).

Since \(P\) covers two corners of \(m\), \(|P| \geq |m|\). Without loss of generality, let \(\text{top}(P) < \text{top}(m)\). Then \((Q \cap m) \supsetneq (P \cap m)\) implies that \(Q\) covers the top row of \(m\), and at least one row of \(m\) below \(\text{bot}(P)\). Either \(|P| = |Q| = |m|\), and hence \(\text{top}(P) < \text{top}(Q) < \text{top}(m)\); in this case \(Q\) was discarded during \(\text{usd}(P, Q, R)\), and \(P \in \beta_m\) (see Figure 3.15(a)). Else \(|P| > |m| = |Q|\), in which case (see Figure 3.15(b)) \(Q\) was discarded during \(\text{usd}(Q, m)\), and again \(P \in \beta_m\).

Else \(Q\) is larger than \(m\). Clearly \(m\) is unbounded above, and is bounded along both its left and right edges by white pixels. Consider \(\xi_{mp}\), an irreducible cover of \(m\) s.t. \(P \in \xi_{mp}\). By examining Figures 3.15(c) and 3.15(d), we can see that there is exactly one square, say \(B\) in \(\xi_{mp}\), s.t. \(B \supset (m - P)\). Clearly \(|B| \geq |m|\), and \(\text{bot}(B) > \text{bot}(m)\). If \(|B| = |m|\), \(m\) was discarded during \(\text{usd}(m, B)\), a contradiction. Else if \(|B| > |m|\), \(m\) is part of a uniform strip: all of which is covered by two larger squares, whence \(m\) was discarded during \(\text{usd}(m)\), and again we have a contradiction. \(\square\)
3.6 Bounded Size of the Extended Cover

In Theorem 2 we showed that the basic cover $\beta_m$ of a non-essential square $m$ contains at most four maximal squares. In Theorem 5 we showed that the extended cover $\Xi_m$ is contained in $\beta(\beta(m))$. It follows that the extended cover $\Xi_m$ of any non-essential square $m$ contains at most 16 maximal squares, and we have Theorem 6 as stated below. Actually, we can prove a much stronger result about extended covers, in the same vein as Theorem 1 about basic covers. In Theorem 7 we prove a restriction on the number of squares in $\Xi_m$ which can cut a single edge of $m$.

**Theorem 6** The extended cover $\Xi_m$ of a non-essential square $m$ contains $O(1)$ maximal squares.

**Theorem 7** Let $\Xi_m$ be the extended cover of a non-essential square $m$. If some edge $e$ of $m$ is cut by two squares $T$ and $W$ in $\Xi_m$, then at least one of $T$ and $W$ cuts $l$ or $r$, the two edges of $m$ which are respectively incident on the left and right corners of $e$.

**Proof** (See Figure 3.16.)

Assume to the contrary that neither of $T$ and $W$ covers any corner of $e$. Without loss of generality let $e$ be the top edge of $m$, and the origin of $T$ be to the left of the origin of $W$.

**Case 1**: $T$ and $W$ do not abut / overlap along $e$. Consider pixels a, b, c in $m$ such that all three are adjacent to $e$, and respectively to the left of $T$, between $T$ and $W$, and to the right of $W$, as shown above. Let $\xi_r \subseteq \Xi_m$ be an
irreducible cover of \( m \), st \( T \in \xi_T \). Surely we must have K in \( \xi_T \) st K covers a but not b or c: else \( \xi_T \supseteq m \), or \( T \notin \xi_T \). Similarly, we have R in \( \xi_T \) st R covers b but not a. Neither of K and R can extend above the top edge of T (else T would not be maximal); hence K extends past the left edge of m, and R extends past the right edge of m. Thus R covers b and c but not a. Similarly let \( \xi_w \subseteq \Xi_m \) be an irreducible cover of \( m \), st \( W \in \xi_w \). We have L in \( \xi_w \) st L covers a and b but not c, and S in \( \xi_w \) st S covers c but not a or b. This situation is depicted in Figure 3.16(a). Now R and S both overlap with m, and extend past its right edge: both are not bounded on the left, hence they are bounded above and below by white pixels. Also S extends to the right of R, so it cannot extend past the bottom edge of R.

Case 1a: Either R and S both have the same size (Figure 3.16(a)): in which case R would have been discarded during the division of the uniform strip containing R and S. Then T does not belong to any irreducible cover of m in \( M_U \), and T \( \notin \Xi_m \).
Case 1b: The bottom edge of S is above the bottom edge of R (Figure 3.16(b)). Consider any square X in $\Xi_m$ st X covers $R - (m \cup S)$. X must abut / overlap S along the bottom edge of S. The right edge of X is to the left of the right edge of S, else S is not maximal. Hence the left edge of X is to the right of the right edge of m, else X is not maximal. Hence X does not cover all of $R - (m \cup S)$. By induction on the number of squares needed to cover all of $R - (m \cup S)$, it follows that R is essential. Since $(R \cap m) \supset (W \cap m)$, by the definition of extended covers $W \notin \Xi_m$.

![Diagram showing Case 1b](image)

**c.** Finding a cover for R, $|S| > |R|$

**d.** T and W overlapping

Figure 3.16(contd.) Two squares in $\Xi_m$ cut same edge of m

Case 1c: R and S have their bottom edges aligned, and S extends above T (Figure 3.16(c)). Then all of R is covered by m and S together: both of which are wider than R. In which case also, R is discarded during the division of the uniform strip containing R, and $T \notin \Xi_m$.

Case 2: T and W abut / overlap along e. Then one of them (say T) extends farther above e than the other (Figure 3.16(d)). Because T and W are unbounded below, they are both bounded by white pixels along their left and right edges. By the same argument as applied wrt S and R above
(cases 1a, 1b): either (i) $T$ and $W$ are of the same size, and $W$ (the lower one) was discarded during the division of the uniform strip containing $T$ and $W$; or (ii) $W$ (the lower and wider one) is essential, and $(W \cap m) \supseteq (T \cap m)$. In the latter case $W \notin \Xi_m$. \hfill \Box
Chapter 4

Deriving the Cover Graph

In this chapter we will construct a cover graph for the image, which provides a compact representation of the geometric covering relationships amongst the maximal squares. Recall from Section 1.4 that we find a minimal cover for the image in phases. During each phase we select a subset $S$ of $M_A$ ($M_A$ is the set of available undiscarded squares) whose elements can be discarded simultaneously without violating the safety condition that the image is covered by $M_A \cup M_R$ ($M_R$ is the set of retained squares). Consequently, for every square $m$ in $S$ we need to answer efficiently the question: "does $(M_A \cup M_R) - S$ cover $m$?". Clearly, the number of phases of the algorithm (and hence its maximum degree of parallelism) depends crucially on the number of squares which can be simultaneously discarded during each phase. The cover graph helps us to determine a large (independent) set of squares, which can be discarded simultaneously without violating the safety of the cover.

In the previous chapter we constructed the extended cover $\Xi_m$ for every (non-essential) square $m$ in the image. We also showed that (i) $\Xi_m$ contains all those squares which are "useful" for covering $m$, (ii) $\Xi_m$ can be constructed with a constant number of operations, and (iii) the size of $\Xi_m$ is
bounded from above by a small constant. Given $\Xi_m$, we can list all of its subsets. Some of these subsets will cover $m$, some will not, and a few of them will be irreducible covers for $m$ with respect to $\Xi_m$. In Figure 4.1 we see that \{X, Y, Q, Z\} and \{P, Q, Z\} are the irreducible covers for $m$ with respect to $\Xi_m$. We also see that every cover of $m$ includes both $Q$ and $Z$, and either $P$ or both of $X$ and $Y$. This leads us to suspect that there is a normal form for characterizing the different ways of covering any given maximal square, which we now derive in the following section.

![Diagram](image)

Figure 4.1 Need for factoring the covering relationships

4.1 Factoring the Cover Relations in Normal Form

Our goal is to be able to determine efficiently if an arbitrary set $S$ of maximal squares covers a given maximal square $m$. If $S$ covers $m$, $S$ contains at least one subset $T$ such that no subset of $T$ covers $m$. As we saw before, $T$ is then called an irreducible cover of $m$. Since $T$ is an irreducible cover, each square in $T$ covers some region of $m$ uniquely, and we can subdivide $m$ into distinct regions, each of which can be covered
independently by one of the squares in T. As we saw in Figure 4.1, \( \Xi_m \) can contain more than one irreducible cover for \( m \). We can characterize these distinct sets formally by defining a \textit{cover function} for \( m \) as follows.

\textbf{Definition 4.1 (cover function):} Let \( S = \{m_1, \ldots, m_k\} \) be a subset of the maximal squares in the image, and \( R \) be an arbitrary region covered by \( S \). The \textit{cover function} for \( R \) is the boolean function \( F(R)(s_1, \ldots, s_k) \) which is true if and only if the set of squares \( \{m_i \mid s_i \text{ is true}\} \) covers \( R \).

The cover function for a given region \( R \) can be expressed as a conjunction of the cover functions for each pixel \( p \) in \( R \), where the cover function for a pixel \( p \) is a disjunction over the squares in \( S \) which contain \( p \). Formally, \( F(R) = \bigwedge_{p \in R} \left( \bigvee_i : m_i \supset p(s_i) \right) \). In this CNF representation, the sum of the formula sizes over all the maximal squares in the image can be \( \Omega(n \sqrt{n}) \), as we saw in Section 1.4.2. Hence we are motivated to define the reductions described below.

(i) If pixels \( p \) and \( q \) in \( R \) are covered by the same set of squares, their cover functions will also be the same. Hence the product of \( F(p) \) and \( F(q) \) in \( F(R) \) can be replaced by either one of its factors.

(ii) Given two subregions \( R_p \) and \( R_q \) of \( R \), let \( S_p \) and \( S_q \) be the sets of all squares which cover \( R_p \) and \( R_q \) respectively. If \( S_q \) is a strict superset of \( S_p \), then \( F(R_q) \) equals \( F(R_p) \lor F' \), where \( F' \) is a disjunction over the squares in \( S_q - S_p \). Hence \( F(R_p) \) implies \( F(R_q) \), and the product of the two in \( F(R) \) can be replaced by \( F(R_p) \).
In the above discussion we defined cover functions for arbitrary regions of the image, and the argument for each function ranged over arbitrary sets of maximal squares. However, the only cover functions which need to be known for determining a minimal cover for the image, are the ones for the squares retained after subdividing uniform strips. We also know from Theorem 4 that for a given square \( m \) in \( M_U \), only the squares in its extended cover \( \Xi_m \) are needed in order to represent all of its "usefully distinct" covers. Hence we can restrict the argument for the cover function of \( m \) to range only over the squares in \( \Xi_m \).

### 4.2 Representing the Cover Function by a Cover Graph

The cover function \( F(m) \) has a very nice geometric interpretation. *Each conjunct in \( F(m) \) is a rectangle which equals the intersection of the squares in that conjunct, and \( m \) equals the union of all such rectangles defined by the conjuncts in \( F(m) \).* For the example image in Figure 4.2, \( F(m) = a \cap c \cap (b \lor x) \cap (d \lor y) \cap (b \lor d) \) is the cover function for \( m \), the set of all irreducible covers for \( m \) is \( I_m = \{\{A, C, B, D\}, \{A, C, B, Y\}, \{A, C, X, D\}\} \), and \( m \) equals \( A \cup C \cup (B \cap X) \cup (D \cap Y) \cup (B \cap D) \). The mapping between the various representations is obvious.

We can represent the minimum CNF form of the cover function \( F(m) \) for a square \( m \) by means of a bipartite cover graph \( G_m \) derived from the geometric interpretation of \( F(m) \) given above. Of course, we will never actually construct the required cover functions and minimize them. Instead, in Section 4.3 we will provide a direct geometric construction for
the cover graph $G_m$. In Figure 4.3 we show the cover graph for the image in Figure 4.2(a). We can construct the cover graph $G_R(I)$ for the entire image $I$ by performing a componentwise union of the (partial) cover graphs $G_m$ for each square $m$ in $M_I$. We can characterize $G_R$ as follows.

\begin{figure}[h]
\centering
\includegraphics[width=0.7\textwidth]{figure4.2}
\caption{Geometric interpretation of the cover function}
\end{figure}

Let $G_R$ be a directed bipartite graph $\langle (M_G, R_M), E_R \rangle$. One vertex set $M_G$ denotes a suitable subset of the maximal squares in the image. The other vertex set $R_M$ denotes a set of rectangular regions obtained from a decomposition of the squares in $M_G$, which will be shown in Section 4.3. The edge set $E_R$ denotes containment between the squares and the regions. Associated with each vertex $v$ in $M_G$ we store the location of the top left corner of the square $S_v$, which is represented by $v$. This enables us to obtain a square cover for the image from a vertex cover for $M_G$.

We can reduce the size of the cover graph by discarding some of the obviously redundant maximal squares of the image before constructing $G_R$. However we must guarantee that a vertex cover for the squares in $M_G$
covers the entire image. Hence $M_G$ is a subset of $M_I$ which is required to satisfy the following condition.

\[(G1)\] The union of the squares in $M_G$ equals the image $I$.

In fact we can use the set $M_U$ of squares retained after subdividing uniform strips in place of the set $M_G$. We can obtain further reductions in the size of $G_R$ by (a) combining those pixels which are covered by the same set of squares into a single region, and (b) not representing those cover relationships which are guaranteed to be satisfied whenever some other regions of the image are covered. Now we establish a correspondence between vertex covers in $G_R$ and square covers in the image by means of the following definitions and conditions.

**Definition 4.2** (*g-cover, i-cover, G-cover, minimal G-cover*): We say that a subset $S_m$ of $M_G$ *g-covers* a square $m$ in $G_R$ if every region $r$ preceding $m$ has at least one predecessor $S_r$ such that $S_r$ belongs to $S_m$. We say that $S_m$ *i-covers* $m$ in the image if $m$ is contained in the union of the squares in $S_m$. We say that a subset $S_G$ of $M_G$ *G-covers* $G_R$ if every square $m$ in $M_G$ is *g-covered* by $S_G$. Further, if no subset of $S_G$ *G-covers* $G_R$ then we say that $S_G$ is a *minimal G-cover* of $G_R$.

We ensure that $S_m$ *i-covers* $m$ in the image whenever $S_m$ *g-covers* $m$ in $G_R$, by requiring that $G_R$ also satisfy the following two conditions.

\[(G2)\] A square $m$ in $M_G$ is contained in the union of the regions preceeding it in $G_R$. 
(G3) A region \( r \) in \( R_M \) is contained in every one of the squares preceding it in \( G_R \).

Finally, we must guarantee that minimality of a \( G \)-cover of \( M_G \) in \( G_R \) implies minimality of the corresponding square cover in the image. To this end we require that \( G_R \) also satisfies the following condition.

(G4) If a subset \( S_m \) of \( M_G \) \( i \)-covers \( m \) in the image, then \( S_m \) also \( g \)-covers \( m \) in \( G_R \).

Recall from Theorem 4 that if a subset \( M_C \) of \( M_I \) covers the image, then every square \( m \) in \( M_I - M_C \) is covered by the squares in \( M_C \cap \Xi m \). Hence we do not require \( G_R \) to represent all possible subsets of \( M_U \) which cover a chosen square \( m \), but rather only the subsets of \( \Xi m \) which cover \( m \). We now collate the above observations into a single definition of the cover graph as follows.

**Definition 4.3** (cover graph): A cover graph for an image \( I \) is a directed bipartite graph \( G_R(I) = (M_U, R_M, E_R) \), with vertex sets \( M_U \) (maximal squares) and \( R_M \) (rectangles forming a decomposition of the squares in \( M_U \)), that satisfies the following properties.

(Gr1) The union of the squares in \( M_U \) equals the image.

(Gr2) A square \( m \) in \( M_U \) is contained in the union of the rectangles preceding it in \( G_R \).

(Gr3) A rectangle \( r \) in \( R_M \) is contained in every one of the squares preceding it in \( G_R \).
(Gr4) If a subset $S_m$ of $\Xi_m$ i-covers $m$ in the image, then $S_m$ also $g$ covers $m$ in $G_R$.

![Figure 4.3 Cover graph for the image in Fig. 4.2(a)](image)

**Theorem 8** Given an image $I$ and its cover graph $G_R(I) = (M_U, R_M, E_R)$, if a subset $M_C$ of $M_U$ is a minimal $G$-cover for $G_R(I)$ then $M_C$ is also a minimal square cover for $I$.

**Proof** (i) (safety): Since $M_C$ is a $G$-cover for $G_R(I)$, every square $m$ in $M_U$ is $g$-covered by $M_C$. Hence each rectangle $r$ preceding $m$ in $G_R(I)$ has a predecessor $S_r$ in $G_R(I)$ such that $S_r$ belongs to $M_C$. Conditions Gr2 and Gr3 in the definition of a cover graph together with the above, implies that $M_C$ i-covers $m$. Since $M_C$ i-covers every square $m$ in $M_U$, condition Gr1 implies that the union of the squares in $M_C$ equals the image.

(ii) (minimality): If $M_C$ is not minimal, then some square $S_R$ in $M_C$ is i-covered by $M_C - \{S_R\}$. By Theorem 4 and condition Gr4 in the definition of a cover graph, there is a set of squares $C_R = (M_C - \{S_R\}) \cap \Xi(S_R)$, such that
Consider the example shown in Figure 4.1. In Figure 4.4 below we show a decomposition of the squares into rectangles, which is in some sense "coarser" than the one shown in Figure 4.8, which is in fact the correct one. In Figure 4.5 we show the cover graph corresponding to this decomposition. Notice that squares \{X, Y, R, B\} are essential. In addition to these, any one square of \{P, Q, m\} is enough to provide a minimal cover for the image. However an examination of Figure 4.5 would imply that \{X, Y, R, B, P, Q\} is also a minimal cover for the image, which it is not. In what follows, we will see that amongst all possible correct decompositions of the maximal squares into rectangles, there is a unique decomposition which is the "coarsest", and which is in some sense also the most natural one.

Given a maximal square \(m\) and its extended cover \(\Xi_m\), we can directly construct the cover graph associated with the minimized cover function for \(m\), by exploiting the geometric properties of \(\Xi_m\) as shown below. Perform a plane sweep of the collection of squares in \(\Xi_m \cup \{m\}\), and subdivide \(m\) into a collection of regions each of which is covered by the same subset of \(\Xi_m\).
Label each subregion \( r \) of \( m \) with the set \( S_r \) of the squares in \( \Xi_m \) which contain \( r \). If two regions \( p \) and \( q \) of \( m \) are labelled with \( S_p \) and \( S_q \) respectively such that \( S_p \subsetneq S_q \), then replace \( S_p \) by \( S_p \cup S_q \) and mark \( S_q \) as being non-residual. After comparing every pair of regions in this manner, discard all the non-residual regions.

Figure 4.5 An Improper Cover Graph for Fig. 4.4
The algorithm described above performs the minimizations described in Section 4.1 in a direct geometric manner, and we claim that it yields the rectangular decomposition of \( m \) required for constructing its partial cover graph. The cover graph for the entire image can be constructed by unioning the partial cover graphs for each square in \( M_U \), as described in Section 4.4. In what follows, we show that the rectangular decomposition of \( m \) obtained in this manner is unique and helps us to satisfy condition Gr4 while constructing the cover graph.

Consider the set \( C_m \) of all \( \rho \)-covers (i.e. set of covering regions) of a maximal square \( m \), given by \( C_m = \{ x \mid x \text{ is a set of subregions of } m \text{ none of which contains another, and whose union covers } m \} \). There is a natural complete partial order \( \preceq \) on \( C_m \), defined by

\[
x_1 \preceq x_2 \text{ iff } (\forall p \in x_1)(\exists q \subseteq x_2)(p \subseteq q)
\]

The minimum element \( \bot \) of the lattice \( (C_m, \preceq) \) is the pixel level partition of \( m \), and the maximum element \( \top \) is \( \{m\} \).

Figure 4.6 Partial order on \( \rho \)-covers of a square
In Figure 4.6 we show four different \( \rho \)-covers \( x_0 \) through \( x_3 \) for a single square \( m \). We observe that \( x_0 \) is less than both \( x_1 \) and \( x_2 \), \( x_3 \) is greater than both \( x_1 \) and \( x_2 \), and \( x_1 \) and \( x_2 \) are incomparable. We have also shown two degenerate decompositions \( x_x \) and \( x_y \) of \( m \) consisting of three regions each, with one region being inside another in both cases. If we did not define \( C_m \) as above, our definition of the \( \preceq \) relation would let us claim that \( x_x \preceq x_y \) and \( x_y \preceq x_x \), although \( x_x \) does not equal \( x_y \).

**Definition 4.4 (compatible cover, induced cover):** Let \( S \) be a subset of \( M_U - \{m\} \) which covers maximal square \( m \), and \( I_S \) be the set of all subsets of \( S \) which are irreducible covers of \( m \). Then a \( \rho \)-cover \( x \in C_m \) is compatible with \( S \), if every region \( r \) in \( x \) can be covered by some single square in any \( T \in I_S \). The induced cover \( \eta_m(S) \) is the least upper bound of the \( \rho \)-covers \( x \in C_m \) compatible with \( S \).

![Diagram](image-url)

Figure 4.7 Partition of \( m \) (residual regions are shaded)
Theorem 9  Let $S$ be a set of maximal squares whose union covers a given maximal square $m$, and $m \notin S$. Then the induced cover $\eta_m(S)$ is compatible with $S$.

Proof  (Refer to the construction shown in Figures 4.7 and 4.8.)

Partition $m$ into maximal connected regions, such that all the pixels in a given region are contained in exactly the same subset of squares in $S$. Label each subregion $r$ of $m$ with the largest subset $S_r$ of $S$ all of whose squares contain $r$. If two regions $p$ and $q$ of $m$ are labelled with $S_p$ and $S_q$ respectively such that $S_p \subseteq S_q$, then replace $S_p$ by $S_p \cup S_q$ and mark $S_q$ as being non-residual. After comparing every pair of regions in this manner, discard the non-residual regions. (Observe that the residual regions correspond to the conjuncts in the minimum CNF representation of the cover function for $m$ restricted to $S$.) Call the remaining set of (augmented residual) regions as $\eta$. It is obvious that $\eta$ as constructed is compatible with $S$.

![Diagram](image)

Figure 4.8 Induced Cover for $(m, \Xi_m)$ shown in Fig. 4.7
Now we show that \( \eta \) is also the least upper bound of the \( \rho \)-covers \( x \) of \( m \) compatible with \( S \). Let \( a, \ b \) be a pair of overlapping regions in \( \eta \). Let \( y = a \cap b \), and \( \delta \subseteq y \). Then \( x = (\eta - \{a, b\}) \cup \{a \cup \delta, b \cup (y - \delta)\} \) is a \( \rho \)-cover of \( m \) which is compatible with \( S \). Also note that \( \eta \supseteq x \) for every \( x \) constructed as above. We can obtain all the compatible \( \rho \)-covers of \( m \) by apportioning the shared regions between overlapping pairs of regions \( (a, b) \) in \( \eta \) in varying amounts between \( a \) and \( b \), and the desired conclusion follows. \( \square \)

4.4 Constructing the Cover Graph

Given a maximal square \( m \) and its extended cover \( \Xi_m \), we can construct the induced cover \( \eta_m(\Xi_m) \). From the above theorem we see that \( \eta_m \) provides a decomposition of \( m \) into a collection of rectangles, such that each rectangle can be covered independently by a single square in \( \Xi_m \). Since \( \eta_m \) is the least upper bound of the \( \rho \)-covers of \( m \) which are compatible with \( \Xi_m \), it is unique. We define the partial cover graph \( G_m \) of \( m \) as \( G_m = (\Xi_m \cup \{m\}, \eta_m, E_\eta) \). The edge set \( E_\eta \) of \( G_m \) is defined as \( E_\eta = \langle(s, r), (r, s) \rangle | \text{square } s \text{ in } \Xi_m \cup \{m\} \text{ contains rectangle } r \text{ in } \eta_m \rangle \).

The partial cover graph \( G_m \) for \( m \) as defined above satisfies conditions Gr2 and Gr3 for a cover graph by construction. Since \( \eta_m \) is compatible with \( \Xi_m \), we are assured that \( G_m \) satisfies condition Gr4 in the definition of a cover graph. Once we have constructed the partial cover graphs for each maximal square in \( M_U \), we can obtain the cover graph for the entire image by unioning the partial cover graphs componentwise. We are assured by
our construction of $M_U$ in Chapter 2 that the union of the squares in $M_U$ equals the image, which is condition Gr1.

![Diagram](image)

**Figure 4.9 Global Implication Rule**

While unioning the partial cover graphs, we apply the graph transformation shown in Figure 4.9 to eliminate useless conjuncts from the global cover function for the image. It is easy to verify that this transformation when applied to a correct cover graph, does not cause it to become incorrect.

From Theorem 6 we know that the extended cover $\Xi_m$ of a square $m$ contains $O(1)$ maximal squares. By the construction of $\eta_m(\Xi_m)$ we decompose $m$ into $O(1)$ rectangles, each of which can be contained in at most every square in $\Xi_m$. Hence we see that the partial cover graph $G_m$ for $m$ has a constant degree. These observations prove the following theorem.

**Theorem 10** The maximum degree of any vertex in the cover graph of an image is bounded by a constant.

In Figures 4.11 and 4.12 below we show the undiscarded maximal squares for the image in Figure 4.10, the induced covers for two of the
maximal squares, and the resulting cover graph. In these figures a pair of oppositely directed edges is represented by a solid undirected line, and a single directed edge by a broken directed line.

Figure 4.10 Example image

In this example $\beta_m = \{P, Q, R\}$, $\beta_p = \{X, Y, m\}$, $\beta_Q = \{K, L, m\}$, and

$\Xi_m = \beta_m \cup \beta_p \cup \beta_Q$ which equals $\{P, Q, R, X, Y, K, L\}$. We also observe that when we union the partial cover graphs of $P$ and $m$, we are able to apply the graph transformation defined by Figure 4.9, and the rectangle $P \cap X \cap Y$ in the induced cover of $m$ gets replaced by the rectangle $X \cap Y$. This is the step during which one of the edges in some of the pairs of oppositely directed edges gets deleted, and the cover relation embodied by the cover graph becomes asymmetric.
Figure 4.11 Induced covers for m and P

Figure 4.12 Reduced Cover-graph for Fig. 4.10
Chapter 5

Determining a Minimal Square Cover

In this chapter we will find a minimal square cover for the image $I$ by determining a minimal G-cover for its cover graph $G_R(I)$. We determine a minimal G-cover for $G_R(I)$ by labelling its vertices in a sequence of phases. Each phase modifies a tri-partition $(M_R, M_D, M_A)$ of $M_I$, where $M_R$ is the set of retained squares, $M_D$ is the set of discarded squares, and $M_A = (M_I - (M_R \cup M_D))$ is the set of available squares whose final state is yet undecided. Each square is labelled with the name of the set which contains it. Initially $M_D$ is set to $M_I - M_U$, and $M_R$ contains only the essential squares of $I$. At the end of the algorithm $M_R \cup M_D$ equals $M_I$, and the squares in $M_R$ constitute the desired minimal cover for $I$.

Each rectangle in $G_R(I)$ is labelled as: (i) uniquely-covered if it has a single predecessor in $M_R$ and all others in $M_D$, (ii) multiply-covered if it has at least two predecessors in $M_R$, and (iii) available if all its predecessors are in $M_A$.

During each phase of the algorithm, we select a subset $S$ of $M_A$ whose elements can be discarded simultaneously, without violating the safety condition that the image is covered by $M_A \cup M_R$. For this purpose we
extract a set of vertex-disjoint paths from the subgraph of $G_R(I)$ induced by its available vertices. Every alternate square in each such path is retained. Of the remaining squares, the ones that are covered by the retained squares are discarded. We showed in Chapter 4 that $G_R(I)$ has a constant degree. Each such phase reduces the (residual) degree of the available squares by at least two, hence a constant number of phases suffices to label all of $G_R(I)$. The labelling algorithm is discussed in detail in the following sections.

### 5.1 Labelling the Cover Graph

We initialize the labelling algorithm by deleting all the unidirectional edges from the cover graph $G_R$, and get an undirected graph $G_U$ in the following manner. We delete edge $(r, m)$ if edge $(m, r)$ is not in $G_R$. This deletion has no effect on the minimal cover obtained, which may be seen as follows. Consider a minimal $G$-cover $M_C$ of $G_R$. If $m$ belongs to $M_C$, it has a successor $t$ (different from $r$) which is uniquely covered. If $m$ does not belong to $M_C$, then we require that $r$ be covered by some square $S$ (different from $m$) in $M_C$, which is always true irrespective of whether $m$ belongs to $M_C$.

In the undirected graph $G_U$, we initialize the essential squares as retained, and the remaining squares as available. We place a token on every successor of a retained square, signifying that such a rectangle is covered by at least one retained square. We obtain a minimal cover for $G_U$ by labelling its vertices in phases as described below. During each phase
we examine the residual subgraph $G_{res}$ of $G_U$ induced by the available squares and available rectangles.

**Definition 5.1 (chain-set, chain):** A chain-set $C_{GR}$ for the residual cover graph $G_{res}$ is a set of vertex disjoint simple paths in $G_{res}$ such that (i) $C_{GR}$ spans all the vertices of $G_{res}$, and (ii) no two paths in $C_{GR}$ can be joined together by selecting an edge in $G_{res}$ which goes from an end point of one path to that of another. A chain is any path in the set $C_{GR}$.

We identify a chain-set $C_{GR}$ for $G_{res}$ by suppressing a suitable set of edges of $G_U$ as described later. We obtain a 2-labelling of the squares in each chain in $C_{GR}$, so that alternating squares in a chain are labelled as retained or available. Then we stitch the chains back into $G_{res}$ by reinstating the edges in $G_{res}$ not present in $C_{GR}$. For each square which is retained during the current phase, we place tokens on all of its successor rectangles, signifying that such a rectangle is covered by at least one of the retained squares. Next we examine each available square in $G_{res}$, and discard those squares all of whose predecessor rectangles have acquired tokens. Finally we relabel the rectangles in $G_{res}$ to restore consistency of the labelling state. (See Figure 5.1.)

![Figure 5.1 Cases for 2-labelling chains](See color key in Fig. 5.6.)
We can identify a chain-set for the residual graph \( G_{\text{res}} \) as follows. Starting at each available square \( m \), examine all the paths of length two which end at another available square. Notice that each such path from \( m \) leads to some square \( S_i \) in the extended cover \( \Xi_m \) of \( m \), and there are only a constant number of such paths. We can consider each such path \((m, r, S_i)\) to be really a path from the origin of \( m \) to the origin of \( S_i \). Now consider the cyclic order (say clockwise) in which the origins of each \( S_i \) (reachable from \( m \) via two consecutive edges in \( G_{\text{res}} \)) appear with respect to the origin of \( m \). We can select the one which appears closest to the North-going ray from the origin of \( m \). Thus we obtain a directed spanning forest for \( G_{\text{res}} \). If a square \( S_i \) has multiple paths directed into it, we can select the one which originates at a square whose origin is closest to the South-going ray from the origin of \( S_i \) (in anticlockwise order about the origin of \( S_i \)). By thus insisting that the extracted chains be oriented monotonically along both axes, we can guarantee that we choose only simple acyclic maximal paths.

### 5.2 Time Bounds for Labelling

Finally, let us examine the time bounds for labelling in the manner described above. During each phase we need only \( O(1) \) operations per maximal square for obtaining the residual graph, extracting the chain-set, discarding covered squares, and reestablishing the labelling invariants. We can 2-label all the chains in \( O(\log n) \) time, on an EREW-PRAM with \( (n / \log n) \) processors, using the optimal parallel list ranking technique of Cole and Vishkin [CoV-88b].
Each phase of labelling reduces the residual degree (in $G_{\text{res}}$) of the available squares which participate in chains, by at least two. Note that an available square which does not participate in any chain during a given phase, will always be discarded at the end of that phase, which can be seen as follows. If an available square $S$ does not participate in any chain, all of its neighboring rectangles have a predecessor which participates in some chain; hence each of those rectangles acquires a cover after the chains have been 2-labelled, and $S$ is discarded at the end of that phase. Thus we see that a constant number of phases suffices to label all of $G_R$, and we can meet our overall resource bounds for labelling. A complete example is shown in Figures 5.2 through 5.8.

5.3 Correctness of the Labelling Algorithm

We now define two correctness predicates which are satisfied at the termination of the labelling of $G_R(I)$ if and only if the set $M_C$ of squares retained at termination defines a minimal square cover for the image.

(i) **Safety**: Every rectangle is preceded by at least one square which is retained.

(ii) **Minimality**: A retained square precedes at least one rectangle which is uniquely-covered.

It is easy to see that the above two predicates are satisfied (when the labelling algorithm terminates) if and only if the set $M_C$ of retained squares defines a minimal square cover for the image. By conditions Gr1
and Gr2 in the definition of the cover graph, the union of the squares in
$G_R(I)$ equals the image, and each square is contained in the union of its
preceeding rectangles. Thus the union of the rectangles in $G_R(I)$, and the
union of the retained squares in $M_C$ both equal the image. Should $M_C$ not
be minimal, some square, say $m$, in $M_C$ would be covered by $M_C - \{m\}$;
whence by condition Gr4, each rectangle preceding $m$ would be covered by
some square in $M_C - \{m\}$. Then no rectangle succeeding $m$ would be
uniquely covered, a contradiction.

During each step of the graph labelling algorithm we maintain
consistency of the labelling, and also the following two invariants.

(i) Safety _invariant: Every rectangle is preceded by at least one
square which is retained or available.

(ii) Minimality _invariant: A retained square precedes at least one
rectangle which is uniquely-covered or available.

The algorithm terminates when every square becomes retained or
discarded, and every rectangle becomes multiply-covered or uniquely-
covered. The conjunction of the termination predicate with the invariants
implies that the correctness predicates are satisfied, and that a _minimal
cover_ has been found.
Figure 5.2 Example image for the labelling algorithm

Figure 5.3 Squares retained after division of uniform strips
Figure 5.4 *Residual regions* which determine the cover graph
Figure 5.5 Reduced cover graph for the image in Figure 5.2

- Available square
- Discarded square
- Retained square
- Rectangle with unknown cover
- Rectangle with at least one cover
- Singly covered rectangle
- Multiply covered rectangle

Figure 5.6 Color key for Figure 5.7
a. Labelling state after finding \textit{essential squares}

b. After 2-labelling chains of available squares

Figure 5.7 Steps of the labelling algorithm
c. After stitching extracted chains back into the graph

d. Coloring state after applying discard rule

Figure 5.7 (contd.) Steps of the labelling algorithm
Figure 5.8 Irreducible cover obtained by the labelling algorithm
Chapter 6

Implementation on Bounded Degree Networks

In the previous chapters we discussed our algorithms with respect to the Parallel Random Access Machine (PRAM) model of computation. A PRAM [FoW-78, Vis-83, Coo-85, KaR-88] is a set of synchronously executing processors, which have identical instruction sets, the same stored program, and access to a global shared memory. Additionally, each processor has a unique identifier (called its PID), and it can use this information in order to select from a set of alternative instructions during a single execution step. In all variants of the PRAM model, the processors can simultaneously reference distinct cells of the global memory during a single instruction step. Simultaneous access to the same memory cell are permitted in varying degrees by the variants of the PRAM model such as EREW, CREW and CRCW.

In practice, the PRAM model fails to capture two important constraints of feasible parallel computation [Sny-86]: (i) real computer memory is constructed only in modules of several hundred to several thousand words, and one such module can respond to a read/write request for only a constant number of words per instruction cycle, and (ii) a single
processor or memory component can only be connected to a constant number of other components. Hence, parallel computers are constructed by interconnecting a set of processors and memory modules to each other by means of a low degree interconnection network.

By using the PRAM model, we were able to focus exclusively on the algorithmic and combinatorial aspects of the problem, without concern for the twin issues of: (i) allocating processors to tasks, and memory cells to variables, and (ii) routing read/write requests between processors and non-local memory modules through the interconnection network. Because of these simplifications, performance estimates obtained for the PRAM model are often not accurate with respect to the performance obtainable on realistic multiprocessors.

There have been several studies [HeB-88, Ran-89] which show that we can mechanically simulate arbitrary PRAM algorithms on particular bounded degree multiprocessor networks, by slowing down the performance of the algorithm by a multiplicative factor of at most $O(\log^2 n / \log \log n)$. In many cases we can obtain better resource utilization on real machines by choosing problem/algorithm specific multiprocessor networks, and by tailoring the resource allocation and the communication steps of the algorithm to the chosen network topology. In this chapter we will study techniques for mapping our minimal square cover algorithm onto two specific networks: (i) the Mesh, and (ii) the Polymorphic Mesh.
6.1 Practical Architectures for Parallel Computing

In this section we will outline the architectural features of two closely related multiprocessor networks. The first is the Mesh, which consists of a two dimensional array of identical synchronous processors, with each one connected to its neighbors to the left, to the right, above and below. Additionally, there may also be wrap-around connections between the processors along the left-end and right-end columns, and between those along the top and bottom rows. If the wrap-around connections are present, the network is called a Torus. Each processor has its own local memory module, and access to shared variables is accomplished by exchanging read/write messages through the interconnection network.

The mesh architecture has been widely studied [Ull-84], and a large body of algorithms has been developed to exploit its features. It is naturally suited for problems involving numerical matrix computations, as well as for operations on digitized images. A mesh of $\sqrt{n} \times \sqrt{n}$ processors can be laid out very efficiently, but it has a bisection bandwidth of only $O(\sqrt{n})$, and a diameter of $\Omega(\sqrt{n})$. The low bisection bandwidth (minimum number of links crossing any cut which divides the network into two equal parts) limits the maximum rate of communication across any cut, and acts as the limiting resource for problems such as sorting.

The high diameter of the mesh (separation between farthest pair of processors) limits the speed with which we can compute results which depend upon variables which are distributed amongst widely separated
processors. In our problem, a maximal square could potentially have a size of $\Omega(\sqrt{n})$, and hence we have a lower bound of $\Omega(\sqrt{n})$ on the running time of any algorithm which computes minimal square covers on a mesh connected multiprocessor. We will show in Section 6.3 that we can actually achieve this running time.

A variant of the mesh architecture called Polymorphic Mesh was introduced by Li and Maresca in [LiM-87]. Its major difference with the mesh is the manner in which data can be transmitted between any chosen pair of processors. In a conventional mesh, communication between a pair of processors in the same row (column) requires the data to travel through the intermediate nodes by hopping across links connecting adjacent pairs of processors. If the source and destination are separated by $k$ intermediate processors, the communication requires $k + 1$ machine cycles. Communication between a pair of processors in different rows and columns is achieved by having the data trace any rectilinear path through the mesh, (usually entirely along a single axis before commencing on the other.)

In the Polymorphic Mesh, long communication paths can be effectively shortened under program control, by setting switches which can short circuit any selected pair of communication ports adjacent to a given processor. If the processors have $p$ communication ports each, this short circuit capability can in fact be generalized to include arbitrary subgraphs of the complete graph on $p$ vertices. The pattern of short circuits across the
entire mesh can be data dependent, since it is under the control of the program, and can vary between instruction cycles.

Thus, a group of processors within an arbitrary connected subgraph of the mesh can share a single data value within the time period of a single (suitably elongated) machine cycle. If a group of $k$ processors is short circuited in this manner, their switching capacitances add up, and because of the resultant switching delay, the machine cycle needs to be elongated by a suitable multiple of $k$ time constants. However, given typical circuit technologies, even if $k$ approaches the size of the mesh, the resultant total delay may still compare favourably with the execution cycle time of a typical instruction. In Section 6.3 we will outline techniques for implementing our minimal square cover algorithm on a mesh architecture, and in Section 6.4 we will examine speedups which can be obtained by exploiting polymorphic interconnections.

6.2 Data Structures for Constructing Square Covers

In this section we will describe the data structures needed for the construction of minimal square covers using the algorithms developed in Chapters 2 through 5. The input image is given to us as an array of $\sqrt{n} \times \sqrt{n}$ pixels. We naturally associate one processor with each pixel of the image. The required intermediate data structure will consist of a single array of records whose components are described below.

1. In order to compute the largest black squares (Section 2.1) we need:
   a) The horizontal hammerhead $H_{ij} = (d, l)$ with depth $d$ and length $l$,
which defines the length of the longest horizontal strip of black pixels extending from and to the right of pixel \((i, j)\).

b) The similarly defined vertical hammerhead \(V_{ij}\) at pixel \((i, j)\).

c) The *arrowhead* region \(A_{ij} = (d, w)\), with depth \(d\) and width \(w\), extending to the right of and below pixel \((i, j)\).

d) The size \(\text{SEL}_{ij}\) of the largest black square extending to the right of and below pixel \((i, j)\). Also similar values in the remaining three diagonal directions.

2. In order to determine the *maximal squares* (Section 2.1.1) we need:

   A boolean value \(M_{ij}\) which specifies whether pixel \((i, j)\) is the *origin* of a maximal square.

3. In order to determine and subdivide *uniform strips* (Section 2.2) we need:

   The horizontal *uniform_tail* \(U_{ij} = (w, d, l)\) with width \(w\), depth \(d\) and lookahead \(l\). Also a similar triple in the vertical direction.

   The following items of information are needed for each maximal square \(m\). These can be naturally associated with the *origins* of the corresponding maximal squares, which are just specially qualified pixels as described above.

4. In order to represent *basic covers* (Section 3.2) we need:

   A set of four maximal squares, each identified by its size, and the \(x\) and \(y\) coordinates of its origin.
5. In order to represent *essential squares* (Section 3.3) we need:
   A boolean value \( E_{ij} \) which specifies whether the (possible) maximal square with origin at \((i, j)\) is an essential square.

6. In order to represent *extended covers* (Section 3.6) we need:
   A constant sized set of maximal squares (in practice at most 8).

7. In order to represent the rectangular decomposition of each maximal square, and also its partial cover graph (Section 4.3-4.4) we need:
   a) A constant sized set of rectangular regions (in practice at most 8). Each rectangle \( r \) can be specified by a bit vector identifying the subset of squares in the *extended cover* of \( m \) which contains \( r \). The size and location of each such rectangle can be determined by intersecting the maximal squares which contain it.
   b) For every square \( s \) such that \( m \) is in the extended cover of \( s \), we need a pointer from \( m \) to rectangle \( r_{sm} \) which is contained in \( m \) and is in the rectangular decomposition of \( s \).

8. Finally, to construct the minimal G-cover of the cover graph for the image (Chapter 5) we need:
   a) A scalar label with values from the set \{discard, retain, available\} associated with each maximal square \( m \).
   b) A scalar label with values from the set \{available, covered, uniquely-covered, multiply-covered\} associated with each rectangle in the decomposition of \( m \).
   c) Two edges from \( m \) to its predecessor and successor rectangles which define the participation of \( m \) in some *chain* for 2-labelling.
6.3 Implementation Outline for a Mesh Architecture

Now we will outline techniques for implementing the minimal square cover algorithm on a $\sqrt{n} \times \sqrt{n}$ mesh of processors. Amongst the various steps of our algorithm, we can identify three distinct types of computations which have interesting communication requirements. The first type consists of prefix computations, which play a central role in: (i) combining hammerheads along rows and columns, (ii) combining arrowheads along diagonals, and (iii) combining uniform tails along rows and columns. The second type is that of near neighbor queries, and consists of: (i) queries between the origin of a maximal square and pixels adjacent to its perimeter, which arise while computing basic covers, and (ii) queries between the origins of a pair of maximal squares, which arise while computing extended covers, and are also required for extracting chains and stitching them back while labelling the cover graph. The last type is list ranking, which is required for 2-labelling extracted chains.

Prefix products can be computed along rows / columns / diagonals of the mesh in one pass from the beginning to the end of the associated vector. (Suffix computations require a pass in the opposite direction.) This requires $O(\sqrt{n})$ steps of communication between adjacent processors on the mesh. Since the diameter of the mesh is $\Omega(\sqrt{n})$, we have the same lower bound on the time for computing prefix products, because of the time it takes to communicate information along a diametrical path.
Recall from Section 3.2 that the near neighbor queries required for computing the basic cover of a square \( m \) involve communication between the origin of \( m \) and a constant number of distinct pixels adjacent to its perimeter. If we route all of these queries along the perimeter of \( m \), we are guaranteed that such messages will not give rise to more than \( O(1) \) congestion at any node, since only a constant number of squares can intersect / overlap each other at a pixel. Such queries can clearly be satisfied in \( O(\sqrt{n}) \) time.

The remaining near neighbor queries involve the origin of \( m \), and the origins of the squares in the basic and extended covers of \( m \). A query starting at the origin of \( m \) and destined for a square \( B \) in the basic cover \( \beta \) of \( m \), can be routed along the perimeter of \( m \), and then along the perimeter of \( B \). However, \( B \) may belong to the basic covers of many distinct squares, and the corresponding queries may give rise to congestion at nodes along the perimeter of \( B \).

Since the computational steps of the algorithm proceed in synchrony, \( B \) can anticipate the query which will be directed to it by various squares. For example, in a particular step, all the squares which have \( B \) in their respective basic covers would want to know the basic cover of \( B \). We can break up the query into two phases. In the first, \( B \) propagates the required result (in this case \( \beta(B) \)) along all the nodes in its perimeter. In the next phase, each square such as \( m \) which needs this information can read it from the node at the intersection of the perimeters of \( m \) and \( B \). We will not
have more than $O(1)$ congestion at any node, since only a constant number of squares can intersect/overlap each other at a pixel.

During the graph labelling phases of the minimal cover algorithm, each square $m$ will need information about the label (covered state) of the squares in its extended cover $\Xi$. Recall from Theorem 5 that if a square $P$ belongs to $\Xi(m) - \beta(m)$, then there exists a square $Q$ in $\beta(m)$ such that $P$ belongs to $\beta(Q)$. Hence a query directed from $m$ to $P$ can be resolved in two steps using the perimeter of $Q$ as the intermediate part of the route. First $P$ can replicate the required result along its perimeter. Then the squares such as $Q$ which have $P$ in their basic cover can also replicate the result along their respective perimeters. Finally, $m$ can read the required values from the nodes at the intersection of its perimeter and those of the squares in its basic cover.

Thus we see that all queries involving near neighbors can be satisfied in $O(\sqrt{n})$ time each. Since the entire algorithm requires the execution of only a constant number of near neighbor queries, $O(\sqrt{n})$ time suffices overall.

All that remains to be seen is how to perform list ranking on the chains extracted from the cover graph. Recall that ranking a linked list $L$ consists of determining the number of elements which precede each element of $L$, starting at either end of $L$. Also recall from Section 5.1 that we construct the chain set of a cover graph by tracing monotonic paths in the image. Hence, a sequential scan down each such path is enough to determine the
ranks for the elements in each chain. We are again guaranteed $O(1)$ congestion at each node along a chain, and there are only a constant number of phases of graph labelling, so $O(\sqrt{n})$ time suffices overall.

### 6.4 Speedup Obtainable on a Polymorphic Mesh

In this section we will discuss ways in which the implementation of our algorithms on a mesh as outlined above, can be speeded up by exploiting polymorphic communication capabilities. First we need to examine the currently defined communication modes of the polymorphic mesh, as well as those which could be reasonably realized by means of suitable architectural modifications. These communication modes are described below with respect to the transmission of a single bit along a path $P_0, P_1, \ldots, P_k$ consisting of electrically adjacent processors. The selection of the path is done under program control, by evaluating a suitable predicate involving the IDs of the relevant processors. The first two modes have been implemented, but the third has only been proposed.

1. **Store-and-Forward**: Each processor along the specified path reads the bit from its predecessor, possibly loads it into a local register, and then forwards it to its successor. If the instruction cycle time is $T_I$, this communication takes $k \cdot T_I$ time units for completion. This mode is the only one available on a conventional mesh architecture.

2. **Circuit-Switched**: This mode sets up an electrically continuous circuit along the entire path, by connecting suitable input and output ports of the intervening processors using switches. If the transmission time
through a switch is $T_S$, this communication takes $kT_S$ time units for completion. In most technologies, $T_S$ is much less than $T_1$, and the ratio between the two could be as much as $1:100$.

3. *Pipelined:* In this mode the processors along the path are set up as a two phase shift register. The shift time $T_p$ through each stage of the shift register could be as little as 2 to 4 times $T_S$ in some technologies. If $b$ bits need to be communicated along the path $P_0, P_1, \ldots, P_k$, the total communication time is $(b+k)T_p$. The same $b$ bits would require $b \times k \times T_S$ time units for communication using the circuit-switched mode. We see that whenever $b > (T_p/T_S)$ and $k > bT_p/(bT_S - 1)$, the pipelined mode always wins out over the circuit-switched mode.

Now we will see techniques by which computational steps involving prefix computations, near neighbor queries, and list ranking can be speeded up by using the circuit switched and the pipelined modes of polymorphic communication.

Consider a prefix computation over a sequence $x_0, x_1, \ldots, x_k$ of values distributed in order amongst a sequence of processors $P_0, P_1, \ldots, P_k$, each of which can communicate with its predecessor and successor in the sequence. For example, the values could respectively be hammerheads / arrowheads / uniform tails, and the paths could be rows / columns / diagonals of the image array. The value of $k$ in all of these cases is $\sqrt{n}$, and each value $x_i$ consists of $b = C_v \log n$ bits. We see that if this computation is performed using the store-and-forward mode of communication, it will
take \((T_{op} + b T_1)\sqrt{n}\) time units for completion, where \(T_{op}\) is the time for computing the combining operation of the appropriate semigroup. We note that \(T_{op}\) is \(\Theta(\log n)\).

If we use the circuit-switched mode of communication, we can perform the prefix computation by using a tree paradigm in \(\log n\) ascending and descending phases as follows. During ascending phase number \(i\), each active processor combines its partial result with that of the processor which is a distance \(2^{i-1}\) away along the desired path, and the intervening processors are inactive. During the descending phases, the processors communicate with progressively closer neighbors. The total computation and communication time is \(2(T_{op} + b T_S)\sqrt{n}\). If we use the pipelined mode of communication, we require \(2(T_{op} + (b/\sqrt{n} + 1)T_p)\sqrt{n}\) time. A little thought will convince the reader that for the ratios of constants mentioned earlier, the store-and-forward mode is slowest, the pipelined mode is fastest, and the circuit switched mode is intermediate.

We can perform the required list ranking operations by using the tree paradigm outlined above for prefix computations, and we will get the same relationship between the total computation times for the three communication modes.

Recall from the previous section that all of the required near neighbor queries can be computed on the mesh architecture by communicating along the perimeters of the relevant maximal squares. In this case also, we can take advantage of the store-and-forward and pipelined modes of the
polymorphic mesh in the manner described at the beginning of this section for communication along simple paths. The same tradeoffs also apply. In this case, the number of bits required to be transmitted during each communication step is significantly higher, and the pipelined mode emerges as the winner by an even larger margin.
Chapter 7

Conclusions

The algorithm of Scott and Iyengar [ScI-86] does not yield a minimal cover, is sequential, and requires $O(n \log n)$ time. It was recently shown by Aupperle et al [ACK-88] that obtaining a minimum cover for a multiply connected polygonal binary image (with holes), using (maximal) squares is NP-hard. In the same paper a sequential algorithm for obtaining a minimum cover for simply connected images (without holes) was also presented. Given an image $I$, they construct a cover graph $G(I)$ with one vertex for every pixel of the image, and add an edge between every pair of pixels which belong to some common maximal square. We note that the maximal cliques in $G(I)$ correspond to the maximal squares in $I$. They show that if the image has no holes, the resulting cover graph $G(I)$ is chordal, and hence one can use the well known algorithm of Gavril [Gav-72] to find a minimum clique cover for $G(I)$. The above construction yields a graph with $\Omega(n^2)$ edges in the worst case (consider an image consisting of a single square of side $\sqrt{n}$). They reduce the required number of operations to $O(n^{1.5} \log n)$ by avoiding the explicit construction of $G(I)$, and rely instead on processing the maximal cliques of $G(I)$ [Aup-88].
Although their algorithm produces a minimum cover for an image without holes, if the image has any holes, then no technique is presented for finding any cover (even irreducible). In fact, the problem of finding a minimum cover for general graphs is NP-hard, even if the graph is planar. Not only does our algorithm run in \( O(\log n) \) time on an EREW-PRAM with \( (n / \log n) \) processors, but it also works on non-simply connected images. For comparison, the best known algorithm [Lub-85] for finding a minimal cover for arbitrary graphs runs in \( O(\log^2 n) \) time, using \( O(|V|^2 \times |E|) \) processors, which is \( \Omega(n^3) \). Of course our algorithm always yields an irreducible cover, and not necessarily the smallest cover. We suspect that we can produce covers that are no worse than 4/3 times the minimum in the worst case, and are considering heuristics which could bound this even better.

Our algorithm presently runs in \( O(\log n) \) time on an EREW-PRAM with \( (n / \log n) \) processors. It follows from the lower bound on the time for computing the boolean OR / AND of \( n \) bits [CDR-86], that minimal covers with maximal squares cannot be found in less than \( \Omega(\log n) \) time on exclusive write PRAMs (EREW / CREW) with a polynomial number of processors. (Given a vector of \( n \) bits, we can transform it in \( O(1) \) time into a \( \sqrt{n} \times \sqrt{n} \) binary array, and ask if \((0, 0)\) is the origin of a maximal square of size \( \sqrt{n} \).) However, except for the single step of obtaining the ranks of elements from the head of each chain (linked list) of undiscarded squares, all the other steps can be performed in \( O(\log n / \log \log n) \) time on a CRCW-PRAM with \( (n \log \log n / \log n) \) processors [CoV-88a, CoV-88b]. If the list ranking problem gets solved for \( O(\log n / \log \log n) \) time, so will our
algorithm. Alternatively, one could consider representing the chains in arrays, by exploiting the bounded degree of the cover graph, as well as the inherent directional ordering amongst its edges. Then a prefix computation would yield the desired result in sub-logarithmic time.

7.1 Future Directions

A problem worth exploring is to cover a given polygonal image with maximal rectangles, instead of squares. Clearly a cover with rectangles will always provide better image compression than one with squares. It is known that finding a minimum rectangular cover (with overlaps allowed), for an arbitrary polygonal region (with or without holes) is NP-hard [Mas-79, CoR-87, CuR-88]. Franzblau and Kleitman [FrK-84] find a minimum rectangular cover for polygons which are convex along one axis, using \( O(n^2) \) time. An open problem is to extend our present techniques to derive an efficient parallel algorithm for covering a polygonal image with a minimal set of maximal rectangles.

Another possibility is to partition the image into rectangles or squares. Ferrari et. al. [FSS-84] give a sequential algorithm for finding the minimum rectangular partition of a binary image, but their algorithm relies on finding a minimum vertex cover for a bipartite graph. There does not seem to be any simple way of deriving a deterministic parallel (NC) algorithm utilizing their results, since for an image with holes, their cover graph is not convex. (Although finding the minimum vertex cover for an arbitrary graph is NP-hard, Dekel and Sahni [DeS-84] give an NC algorithm for finding the minimum vertex cover for convex bipartite
graphs.) Also, partitioning a polygon into squares or rectangles will always provide worse compression than would a cover. For example, there exist polygons for which \( k \) rectangles (squares) suffice for a cover, but \( \Omega(k^2) \) rectangles (squares) are required for a partitioning. However, in applications in which it is important for the cover to be of disjoint elements, as in the convolution application suggested by Ferrari et al., we would need to obtain a partition of the image.

We are also exploring the possibility of embedding the cover we obtain into a search tree, and thus obtain a greatly reduced image representation which could compete with Quad-trees [Sam-84] for such operations as finding the union and intersection of images. It is clear that the Quad-tree provides a nested partition of the bounded image plane into squares. If we remove the condition that the sizes and placements of the squares be related to each other by multiples of two, (multiplicatively for size, and additively for placement) a compact representation of the image is provided by the smallest subset of maximal subsquares of the image which spans the entire image. If these squares are then embedded in a search tree, the complete functionality of the Quad-tree can be obtained with significantly less storage. We suspect that for this problem maximal squares will be more useful than maximal rectangles.
Appendix 1

Definitions

**Definition 1.1** (maximal square, minimal square cover): A maximal square is a black subsquare of the image, not contained in any larger black square. A square cover for a given image is a set $C$ of (possibly overlapping) maximal squares, whose union equals the image. If no subset of $C$ is a square cover, $C$ is called a minimal square cover.

**Definition 2.1** (largest black square): A largest black square $S_{Eij}$ is the largest square of black pixels that has its top left corner (origin) at $(i, j)$, and its diagonally opposite corner to the SouthEast of $(i, j)$.

**Definition 2.2** (hammerhead): The hammerhead $H_i$ of a sequence of pixels $x_i \ldots x_{i+l-1}$ is $H_i = \langle d, l \rangle$; where the depth $d$ of the hammerhead is the maximum number of ones which prefix the sequence $x_i \ldots x_{i+l-1}$, and $l$ is the length of the sequence. Given two adjacent sequences of pixels $x_i \ldots x_{i+l_1-1}$ and $x_{i+l_1} \ldots x_{i+l_1+l_2-1}$, with respective hammerheads $\langle d_1, l_1 \rangle$ and $\langle d_2, l_2 \rangle$, the combined hammerhead $\langle d_1, l_1 \rangle \otimes \langle d_2, l_2 \rangle$ equals $\langle d, (l_1 + l_2) \rangle$; where $d = d_1$ if $d_1$ is less than $l_1$, and $d = d_1 + d_2$ otherwise.
**Definition 2.3** (arrowhead): An arrowhead $A_{ij} = \langle d, w \rangle$ with origin at $\langle i, j \rangle$ is an L-shaped region containing only black pixels, whose two arms have equal depth $d$ and equal width $w$. (An arrowhead $A_{ij}$ denotes that pixel $\langle i, j \rangle$ can potentially be the origin of a black square of size at most $d$, and we have verified that the top $w$ rows and left $w$ columns contain only black pixels.) Given two diagonally adjacent arrowheads $\langle d_1, w_1 \rangle$ and $\langle d_2, w_2 \rangle$ which are similarly oriented, the combined arrowhead $\langle d_1, w_1 \rangle \otimes \langle d_2, w_2 \rangle$ equals $\langle \min(d_1, d_2 + w_1), (w_1 + w_2) \rangle$.

**Definition 2.4** (uniform strip, uniform tail): A horizontal (vertical) uniform strip is a maximal sequence of identically sized maximal squares, whose origins are horizontally (vertically) contiguous. A (horizontal) uniform tail $U_{ij} = \langle s, d, l \rangle$ is the right end of a (horizontal) uniform strip which has its origin at $\langle i, j - d + 1 \rangle$; where $s$ is the size of the rightmost maximal square in the tail, the depth $d$ is the length of the uniform strip at the end of the tail, and $l$ is the length of the tail. We can associatively combine two adjacent uniform tails into a single uniform tail, as $\langle s_2, d_2, l_2 \rangle \otimes \langle s_1, d_1, l_1 \rangle = \langle s_1, d, (l_1 + l_2) \rangle$; where $d = d_1$ if $(s_1 = s_2)$ or $(d_1 < l_1)$, and $d = d_1 + d_2$ otherwise.

**Definition 3.1** (essential square, irreducible cover, useful square): An essential square is a maximal square which contains at least one pixel which is not covered by any other maximal square. Let $m$ be a non-essential maximal square. An irreducible cover $\xi$ of $m$ is a set of maximal squares excluding $m$ which covers $m$, but no proper subset of which covers
m. A useful square of m is a maximal square which belongs to some irreducible cover of m.

**Definition 3.2 (basic cover):** The basic cover $\beta_m$ for a maximal square m is the set of all those useful squares of m which overlap maximally with it. A square S overlaps maximally with m if $S \cap m$ is not strictly contained in $T \cap m$, for any square T other than m and S.

**Definition 3.3** We say that a square S cuts an edge e of m if a part of e lies in the interior of S.

**Definition 3.4 (pairwise essential squares, extended cover):** Two maximal squares P and Q are pairwise essential if $P \cap Q$ contains a pixel p which does not belong to any other maximal square in $M_1$ (other than P or Q). The extended cover $\Xi_m$ of an inessential square m is a subset of $M_U$, such that for every square S in $\Xi_m$: (i) S is a useful square of m, and (ii) $(S \cap m)$ is not a strict subset of $(P \cap Q)$ for any squares P and Q which are pairwise essential.

**Definition 4.1 (cover function):** Let $S = \{m_1, \ldots, m_k\}$ be a subset of the maximal squares in the image, and $\rho$ be an arbitrary region covered by $S$. The cover function for $\rho$ is the boolean function $F(\rho|s_1, \ldots, s_k)$ which is true if and only if the set of squares $\{m_i | s_i \text{ is true}\}$ covers $\rho$.

**Definition 4.2 (g-cover, i-cover, G-cover, minimal G-cover):** We say that a subset $S_m$ of $M_G$ g-covers a square m in $G_R$ if every region r preceding m has at least one predecessor $S_r$ such that $S_r$ belongs to $S_m$. We say that $S_m$
*i-covers* \( m \) in the image if \( m \) is contained in the union of the squares in \( S_m \). We say that a subset \( S_G \) of \( M_G \) *G-covers* \( G_R \) if every square \( m \) in \( M_G \) is *\( g \)-covered* by \( S_G \). Further, if no subset of \( S_G \) *G-covers* \( G_R \) then we say that \( S_G \) is a *minimal G-cover* of \( G_R \).

**Definition 4.3** (*cover graph*): A *cover graph* for an image \( I \) is a directed bipartite graph \( G_R(I) = (M_U, R_M, E_R) \), with vertex sets \( M_U \) (maximal squares) and \( R_M \) (rectangles forming a decomposition of the squares in \( M_U \)), that satisfies the following properties.

1. **(Gr1)** The union of the squares in \( M_U \) equals the image.
2. **(Gr2)** A square \( m \) in \( M_U \) is contained in the union of the rectangles preceeding it in \( G_R \).
3. **(Gr3)** A rectangle \( r \) in \( R_M \) is contained in every one of the squares preceeding it in \( G_R \).
4. **(Gr4)** If a subset \( S_m \) of \( \Xi_m \) *i-covers* \( m \) in the image, then \( S_m \) also *\( g \)-covers* \( m \) in \( G_R \).

**Definition 4.4** (*compatible cover, induced cover*): Let \( S \) be a subset of \( M_U - \{m\} \) which covers maximal square \( m \), and \( I_S \) be the set of all subsets of \( S \) which are irreducible covers of \( m \). Then a \( \rho \)-cover \( x \in C_m \) is *compatible* with \( S \), if every region \( r \) in \( x \) can be covered by some single square in any \( T \in I_S \). The *induced cover* \( \eta_m(S) \) is the *least upper bound* of the \( \rho \)-covers \( x \in C_m \) compatible with \( S \).
Definition 5.1 (chain-set): A \emph{chain-set} $C_{GR}$ for the residual cover graph $G_{res}$ is a set of vertex disjoint simple paths in $G_{res}$ such that (i) $C_{GR}$ spans all the vertices of $G_{res}$, and (ii) no two paths in $C_{GR}$ can be joined together by selecting an edge in $G_{res}$ which goes from an end point of one path to that of another.
Appendix 2

Theorems

Lemma 1  Let P be the largest black square with its origin at \((i, j)\), and
   \(l_{ij} \geq 1\) be its size. Then P is not maximal iff
   \(l_{i-\delta i, j-\delta j} > l_{ij}\), for some \((\delta i, \delta j)\)
in \(\{(0, 1), (1, 0), (1, 1)\}\).

Theorem 1  Any chosen edge of a non-essential square m cuts at most one
   square in the basic cover \(\beta_m\) of m.

Theorem 2  The basic cover \(\beta_m\) for a non-essential square m contains at
   most four (maximal) squares.

Lemma 2  Given a maximal square m, in O(1) operations we can construct
   \(B_m\), a constant sized superset of its basic cover \(\beta_m\).

Lemma 3  Given a maximal square m, and a set \(B_m\) which is a constant
   sized superset of its basic cover \(\beta_m\), in O(1) operations we can determine
   whether m is essential.

Lemma 4  Given a non-essential square m, and a set \(B_m\) which is a
   constant sized superset of its basic cover \(\beta_m\), in O(1) operations we can
   construct \(\beta_m\).
Theorem 3 Given a maximal square m, in $O(1)$ operations we can decide if m is essential. If m is not essential, we can construct its basic cover $\beta_m$ in $O(1)$ operations.

Theorem 4 If a subset $M_C$ of $M_U$ covers the image, then every square m in $M_U - M_C$ is covered by the squares in $M_C \cap \Xi_m$.

Theorem 5 Let P be a maximal square in the extended cover $\Xi_m$ of a non-essential square m, but not in its normalized basic cover $\beta_m$. Then P belongs to the basic cover $\beta_Q$ of some Q in $\beta_m$.

Theorem 6 The extended cover $\Xi_m$ of a non-essential square m contains $O(1)$ maximal squares.

Theorem 7 Let $\Xi_m$ be the extended cover of a non-essential square m. If some edge e of m is cut by two squares T and W in $\Xi_m$, then at least one of T and W cuts l or r, the two edges of m which are respectively incident on the left and right corners of e.

Theorem 8 Given an image I and its cover graph $G_R(I) = (M_U, R_M, E_R)$, if a subset $M_C$ of $M_U$ is a minimal G-cover for $G_R(I)$ then $M_C$ is also a minimal square cover for I.

Theorem 9 Let S be a set of maximal squares whose union covers a given maximal square m, and m $\notin$ S. Then the induced cover $\eta_m(S)$ is compatible with S.
Theorem 10 The maximum degree of any vertex in the cover graph of an image is bounded by a constant.
References

[ACK-88] L.J. Aupperle, H.E. Conn, J.M. Keil and J.O'Rourke,
"Covering Orthogonal Polygons with Squares", 26th Ann. Allerton
Conf. on Communications, Control and Computing, Urbana IL, 28-30
September 1988. (also Johns Hopkins Univ., JHU-88/16.)

[Aup-88] L.J. Aupperle,
"An Algorithm for Covering Polygons with Squares", Draft, Dept. of

[Ble-87] G. Blelloch,
"Scans as Primitive Parallel Operations", Proc. of Intl. Conf. on

[CDR-86] S. Cook, C. Dwork and R. Reischuk,
"Upper and Lower Time Bounds for Parallel Random Access Machines
1, pp. 87-97, Feb. 1986.

[CIK-88] Y. Cheng, S.S. Iyengar and R.L. Kashyap,
"A New Method of Image Compression using Irreducible Covers of
Maximal Rectangles", IEEE Trans. on Software Engineering, vol. 14,
[Coo-85] S. A. Cook,
"A Taxonomy of Problems with Fast Parallel Algorithms", 

[CoR-87] H.E.Conn and J.O'Rourke,
"Some Restricted Rectangle Covering Problems", *Dept. of Comp. Sc.,

[CoV-88a] R.Cole and U.Vishkin,
"Faster Optimal Parallel Prefix Sums and List Ranking", submitted to 
*Information and Computation*. (also Courant Institute CS-TR-277)

[CoV-88b] R.Cole and U.Vishkin,
"Approximate Parallel Scheduling, Part I: the basic technique, with 
applications to Optimal Parallel List Ranking in Logarithmic Time", 

[CuR-88] J. C. Culberson and R. A. Reckhow,
"Covering Polygons is Hard", *Proc. 29th Symp. on Foundations of 

[DeS-84] E.Dekel and S.Sahni,
"A Parallel Matching Algorithm for Convex Bipartite Graphs and 
Applications to Scheduling", *Jour. of Parallel and Distributed 

[FoW-78] S. Fortune and J. Wyllie,
"Parallelism in Random Access Machines", *Proc. 10th Ann. ACM 

[FrK-84] D.S.Franzblau and D.J.Kleitman,
"An Algorithm for Constructing Regions with Rectangles: 
Independence and Minimum Generating Sets for Collections of

[FSS-84] L. Ferrari, P.V. Sankar and J. Sklansky,

[Gav-72] F. Gavril,

[GPS-87] A.V. Goldberg, S.A. Plotkin and G.E. Shannon,

[HeB-88] K. T. Herley and G. Bilardi,

[Hil-85] W. D. Hillis,

[KaR-88] R. M. Karp and V. Ramachandran,

[Kle-88] P. N. Klein,
[KRS-85] C.P. Kruskal, L. Rudolph and M. Snir,

[LaF-80] R. E. Ladner and M. J. Fisher,

[LiM-87] H. Li and M. Maresca,

[Lub-85] M. Luby,

[Mas-79] W. J. Masek,

[Moi-88] D. Moitra,

[MRS-88] R. Motwani, A. Raghunathan and H. Saran,

[OR-87] J. O'Rourke,
[Ran-89] A. G. Ranade,
"Fluent Parallel Computation", Dept. of Comp. Sci., Yale Univ., Ph. D.

[Sam-84] H. Samet,
"The Quadtree and Related Hierarchical Data Structures", ACM

[ScI-86] D.S.Scott and S.S.Iyengar,
"TID: a Translation Invariant Data Structure for Storing Images",

[Sny-86] L. Snyder,
"Type Architectures, Shared Memory and the Corollary of Modest
Potential", Dept. of Comp. Sci., Univ. of Washington, TR-86-03-04,

[TaP-75] S. L. Tanimoto and T. Pavlidis,
"A Hierarchical Data Structure for Picture Processing", Computer

[Ull-84] J. D. Ullman,

[Vis-83] U. Vishkin,
"Synchronous Parallel Computation - a Survey", Dept. of Comp. Sci.,

[YeC-89] R. Bar Yehuda and E. Ben Chanoch,