University of Alberta

Fast Adaptive Region Growing for Image Segmentation

by

Yian Leng Chang

A thesis

submitted to the Faculty of Graduate Studies and Research

in partial fulfillment of the requirements for the degree

of Doctor of Philosophy

Department of Computing Science

Edmonton, Alberta

FALL 1993

ISBN 0-315-88119-4

UNIVERSITY OF ALBERTA

RELEASE FORM

NAME OF AUTHOR: Yian Leng Chang

TITLE OF THESIS: Fast Adaptive Region Growing for Image Segmentation

DEGREE: Doctor of Philosophy

YEAR THIS DEGREE GRANTED: 1993

(Signed) [illegible]

Permanent Address:

2E St. Helier's Avenue

SINGAPORE 1955

Date: August 6, 1993

Manifest plainness,
Embrace simplicity,
Reduce selfishness,
Have few desires.

Lao-tze
c.604–531 B.C.

UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research, for acceptance, a thesis entitled **Fast Adaptive Region Growing for Image Segmentation** submitted by **Yian Leng Chang** in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

[illegible]
X. Li (supervisor)

[illegible]
A. Basu

Jack Mowchenko
J. Mowchenko

[illegible]
J. Schaeffer

[illegible]
H. Zhang

[illegible]
L. S. Davis, University of Maryland

Date: August 6, 1993

Abstract

Image segmentation is a process that produces a meaningful partition of an image such that each segment in the partition corresponds to an entity in a scene. In this project, we propose a simple and general region growing framework for image segmentation. The region growing process is guided by regional feature analysis; no parameter tuning or *a priori* knowledge about the image is required. Also, our algorithm is fast and suitable for parallelization.

An undirected graph is used to represent an image in a region growing process. A vertex in the graph denotes an image region, and an arc indicates the adjacency relationship of two regions. Whenever two connected vertices satisfy a certain merge criterion, they are merged into one. Region growing is an agglomerative and iterative process which terminates when no more vertices in the graph can be merged. Since a merge operation affects only a neighborhood of the graph, many pairs of vertices can be merged independently at the same time.

A new merge decision rule, called the Fast-merge Adaptive criterion, is proposed in this study. Instead of comparing the difference of two feature means with a predefined threshold, we assess the homogeneity of two merging regions from their feature distributions. This results in an algorithm that is robust with respect to various image characteristics. When several regions are homogeneous enough to merge with a given region, the Fast-merge policy is applied to select a region for the binary merge. Fast-merge enhances the quality of segmentation. More importantly, it minimizes the number of merge rejections and results in a fast region growing process that is also amenable to parallelization.

Our algorithm is tested on the BBN TC2000 shared memory multiprocessors. Experimental results show that the algorithm produces a balanced load on the processors, and maintains a high degree of parallelism throughout the segmentation process. A distributed implementation on a network of SPARC workstations is also described. Through theoretical analysis and empirical studies, we have shown that our algorithm requires shorter processing times and produces better segmentations than the existing Best-merge Fixed Threshold algorithm.

Acknowledgments

I would like to express my deepest gratitude to my supervisor, Dr. Xiaobo Li, for guiding me through the research and the preparation of this thesis. I am thankful for the unrestricted access to all resources at his disposal, and for his unfailing confidence in my motivation and capability. During his short "vacation" at home from his sabbatical visit in the United States, he sacrificed precious weekend afternoons with his family to go through pages of proofs with me on the whiteboard. At the final stage of this project, despite reading was painful and difficult due to his eye injury, he meticulously read every draft of my thesis and provided timely and constructive feedback. The successful completion of this project owes much to his insights and dedication.

I would also like to thank the members of my examining committee, Dr. Jonathan Schaeffer, Dr. Anup Basu, Dr. Jack Mowchenko, Dr. Hong Zhang, and Dr. Larry Davis, for carefully reading this thesis and making positive criticisms.

In the course of this project, I benefited from the discussions with many individuals: the electronic mail exchanges with Dr. Richard C. Dubes (Michigan State University), Dr. Clark Y. Bie (University of Waterloo), and Dr. Kenneth I. Laws (SRI) helped me to gain better insight on textural measures and the segmentation problem; Nesrine Abbas helped me to look at the region growing problem from the graph-theoretic perspective; Steve Sutphen and Andrew Mullhaupt tirelessly directed me in identifying the performance bottleneck of my SPARC implementation, and suggested many useful ways to alleviate the paging problem; Paul Lu, our local TC2000 expert, had given me many useful pointers on using the machine; Brent Gorda and Linda Woods (Lawrence Livermore National Laboratory) provided excellent technical support on the TC2000, thus enabling me to complete my experiment on schedule.

This thesis is dedicated to Shuang, my husband, who so graciously endured the neglect since I started on this project, and so cheerfully proceeded to marry me despite the project seemed never ending. Without his encouragement and understanding, this thesis would never have been completed. I am also grateful to my parents, brothers and sisters, for their unconditional wholehearted support on whatever I pursue, and for their love and care. To all my friends, I thank them for always being there when I needed them.

Finally, I would like to acknowledge the financial support of the Department of Computing Science, University of Alberta. Special thanks to Lawrence Livermore National Laboratory for the access to their TC2000 system; PAMI Laboratory at the University of Waterloo for supplying the Brodatz texture set; PRIM Laboratory at the Michigan State University for providing me with several intensity images for my experiment.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Image Segmentation: Definition and Goal

In computer vision, a *scene* refers to the physical environment of interest; an *image* is a two-dimensional projection of the scene, and the output of a sensor used to *see* this environment. An image, in image processing terminology, is often a two-dimensional array of pixels. From the computational perspective, a pixel is simply a number or a vector of numbers.

A computer vision system recovers useful information about a scene from images. Before such a system can make some sense from an image, it will usually have to obtain a meaningful partition of the image such that each segment in this partition corresponds to an entity in the scene. This image partitioning process is known as image segmentation. In the algorithmic sense, image segmentation is a process that assigns a label to each image pixel or pixel block according to some criteria. At the end of that process, all spatially connected pixels or blocks with the same label constitute an image region.

Image segmentation is a very important problem that has been the subject of extensive research in computer vision over the past decades. It is a low-level image operation carried out at the initial stage of an image analysis process, where in many cases no *a priori* knowledge about the objects in a scene is available. The primary purpose of segmentation is to provide sufficient information to support higher level reasoning in the vision system. The success of a vision task often relies on good segmentations.

Image segmentation is also a difficult and ill-defined problem [8]. It is the nature of segmentation that its definition and quality cannot be formulated precisely. There is no ground truth for the segmentation of any real image, no objective and scientific measure for the quality of such a segmentation. To illustrate our point, we quote the following definition of a good segmentation given by Haralick *et al.* [28]:

> "Regions of an image segmentation should be uniform and homogeneous with respect to some characteristic such as gray tone or texture. Region interiors should be simple and without many small holes. Adjacent regions of a segmentation should have significant different values with respect to the characteristic on which they are uniform. Boundaries of each segment should be simple, not ragged, and must be partially accurate...."

Despite the decades of effort, we are unable to define an image region anywhere more precise than it being "uniform and homogeneous with respect to some characteristic". In fact, this rather vague condition has served as a working criterion for most segmentation algorithms.

1.2 Motivation and Objectives

Besides being an important and difficult problem, segmentation on any nontrivial image is a tedious and time consuming process. In order to produce useful results, segmentation methods tend to be ad hoc and application specific. It is unrealistic to strive for a general-purpose segmentation algorithm that will decompose arbitrary images into meaningful parts. However, a flexible and effective segmentation model which can be easily tailored for specific applications is a worthwhile endeavor. It is our ultimate goal that through this study on region growing, a better understanding on the segmentation problem can be achieved.

The main objectives of this project are itemized as follows:

- *To provide a simple, flexible and effective segmentation algorithm that can be easily adapted to specific applications*

Our goal is to provide a general segmentation framework that can be easily tailored to work well for specific applications on a wide range of computer systems. In region growing, such adaptations include feature measure and feature window size selections. There are two aspects on the generality of our method. First, our model should be simple, application-independent, and be able to provide reasonable segmentations even without *a priori* knowledge about the images to be segmented. Furthermore, our algorithm should allow efficient implementations on widely accessible computer systems, including conventional single-processor machines, distributed networks and shared memory multiprocessor systems.

- *To improve segmentation results by allowing local image characteristics to guide the segmentation process*
  Current region growing methods use predefined thresholds for the entire image and/or throughout the region growing process. Since image properties vary with locations of the image, an adaptive approach that is guided by these local variations would produce better segmentation results. In our adaptive region growing algorithm, parameters required by the merging decision would be computed automatically based on local property measures. Our method would not require parameter tuning or *a priori* knowledge about an image, although domain-specific knowledge may be incorporated into the algorithm to improve performance.

- *To achieve fast segmentation by harnessing the computational power of multiprocessor systems*
  Image understanding systems often have stringent requirement on process turn around time. Therefore, the speed of a segmentation process is an important measure on the usefulness of the algorithm. To achieve fast segmentation, our region growing algorithm is simple and can be easily and efficiently implemented on multiprocessor systems.

In summary, we want to propose a simple, easy to use and effective region growing framework that can produce fast, satisfactory segmentations on a wide range of images.

1.3 Why Region Growing

Although the multitude of alternatives make it impossible to adjudge that a particular segmentation method is clearly a superior selection for treating the problem at hand, we decide to study the region growing approach for the following reasons:

1. Region growing has proven to be an effective segmentation approach for a variety of applications [12, 28, 50, 62, 70]. For example, it is more robust with respect to noise and image data irregularities compared to edge detection segmentation [23]. Region growing is a general and applicable segmentation method for a wide range of images.
2. We believe that spatial information is indispensable for producing good segmentations, and region growing provides a natural, effective way to incorporate spatial and image feature information in the segmentation process. Segmentation by thresholding and clustering lack this property [28].
3. It is easier to customize a region growing algorithm than other segmentation methods for specific classes of images. In region growing, the domain-specific module, namely the feature extraction from image pixels, is easily replaceable without affecting other parts of the algorithm.
4. Region merges can be performed simultaneously in region growing, making the process well suited for parallelization, and has the potential to provide very fast segmentation.

1.4 Scope of Study

Our goal of this project is to provide a general and effective region growing model. As a result, no *a priori* knowledge, such as the number of regions or the shapes and sizes of regions, are required by our algorithm. Also, no attempt is made to customize our algorithm to suit a particular application or a class of images. The selection of feature measure and feature window size, though crucial to the success of the segmentation process, are not dealt with in great detail due to their domain-dependent nature.

To increase the usefulness of our algorithm, we design it with widely accessible machine architectures in mind. These architectures include the conventional single processor machine, general SIMD, MIMD, and distributed systems. Our design does not attempt to accommodate architectures that are not easily accessible, e.g. the pyramid machine.

Enhancement procedures that are common pre- and post-processings for segmentation, such as noise smoothing, small region elimination, and boundary refinement, are not included in this study. This is because such procedures are often application-dependent. In addition, the robustness of our algorithm allows us to place little emphasis on enhancement issues.

Reports on region growing algorithms in the literature exhibit at least one of the following drawbacks making accurate reproduction of results impossible: Mathematical formulas are not stated explicitly; algorithms are not clearly specified; parameters are not given; details of test images are not provided; timing performance are not presented. In the evaluation of parallel performance, a fair and unbiased comparison is impossible due to differences in machine architectures. Because of these difficulties, instead of implementing specific multiple-threshold merging criteria in the literature, we compare our adaptive criterion with the most general and widely used Fixed Threshold method [10, 28, 63, 67, 70]. On the control of merging order, the Best-merge paradigm which was adopted in several recent studies [48, 63, 67] is used to compare against our Fast-merge paradigm.

1.5 Outline of Presentation

In the next chapter, we give an overview on segmentation techniques, followed by a summary on existing work related to the major focus of this thesis, namely adaptive and parallel region growing.

Chapter 3 presents the two-dimensional random field image model. and provides formal definitions of region growing, the Fixed Threshold and the Adaptive homogeneity tests. An intuitive explanation of our adaptive approach is given, followed by an analytical comparison on the two tests using probability theory.

The Fast Adaptive Segmentation (FAS) algorithm is described in Chapter 4. In this chapter, we define the graph model for region growing, and provide an overview of the region growing process. The merge criterion, which is the main focus of this study, is examined in detail. We also present experimental results on the evaluation and comparison of several merge criteria. Four sets of experiments are presented in the final section. These experiments are designed to assess the FAS algorithm on its sensitivity to small regions, its robustness on noisy images, grayscale image segmentation, and the use of multiple features in segmentation. Comparisons are made on the FAS and the existing Best-merge Fixed Threshold

(BFS) algorithms.

A shared memory SPMD region growing model is presented in Chapter 5. We describe the TC2000 implementations of the FAS and the BFS algorithms. Sequential and parallel timing results from the two algorithms on grayscale and textured images are discussed and compared. Further experiments are performed to investigate the effects of several image characteristics on segmentation timing.

In Chapter 6, we discuss other possible implementations of the region growing algorithm on different computational models. Strengths and weaknesses of the various approaches are examined. A distributed implementation on a network of workstations is given in detail.

We give our concluding remarks in Chapter 7. We also suggest several avenues for the further study of the region growing problem.

Chapter 2

Related Work

This chapter presents an overview on image segmentation by outlining the four common techniques: thresholding, clustering, boundary approach, and region growing. In the latter sections, we survey work done on using varying thresholds for region growing, and parallel region growing.

2.1 Thresholding

The simplest form of thresholding is to divide image pixels into two groups according to a single gray-level threshold. The threshold is determined from the valley between two modes in the image histogram. This method, known as point-dependent global thresholding [54], works well for segmenting simple bimodal images containing objects on uniform backgrounds. Global thresholding refers to the use of a single threshold for the entire image, whereas point-dependent means that the threshold is determined solely from the gray-level of each pixel. If the threshold is determined from some local property, for example the gray-level distribution in the neighborhood of each pixel, then the method is said to be region-dependent.

In local thresholding, an image is partitioned into sub-images and a threshold is determined for each of the sub-images. As in global thresholding, the threshold in a local thresholding method can be point- or region-dependent. A smoothing technique is usually applied to eliminate gray-level discontinuities at the boundaries of sub-images. Local thresholding techniques are useful for images with region feature distributions having a high level of overlap. A large number of methods have been proposed for threshold selection in thresholding techniques [23, 41, 54, 66].

Although thresholding works reasonably well on a wide class of images, it has a number of disadvantages. Since thresholding does not apply any spatial information in the segmentation process, many images can have the same histogram. As a result, *a priori* knowledge is required to segment the images. Also, regions in the final segmentation may not be contiguous. The determination of an accurate threshold is difficult and not always achievable using the existing methods.

2.2 Clustering

Segmentation by clustering [14, 23, 28, 37, 61] can be considered the multidimensional extension of thresholding. This approach is based on the assumption that each region forms a distinct cluster in the hyperspace of characteristic features. Often, the Euclidean distance is used for measuring dissimilarity between feature vectors computed from pixel blocks. A clustering algorithm [65] is performed on the feature vectors. The resulting clusters are then mapped back to the spatial domain. An edge finding process may follow to define the region boundaries.

Clustering is an iterative and highly compute-intensive process. In terms of segmentation performance, the clustering approach has several drawbacks: The technique will fail when the distinct cluster assumption is not satisfied. Also, since the number of image segments is not known in general, popular clustering methods, such as the square-error Forgy's algorithm and any approach that is based on mixture decomposition [34], may not be applied efficiently. Furthermore, as in thresholding, the inherent image spatial information has not been successfully incorporated in the clustering approach.

2.3 Boundary Approach

A boundary approach [23, 27, 32] is a segmentation technique based on the detection of discontinuity in an image. This method can be broken down into two steps: edge element extraction and boundary formation. In the first step, an edge detection operator, such as a high pass filter or a gradient operator, is applied everywhere on an image to extract edge elements. The main purpose of the second step is to eliminate false edges and streaks, and combining streaks into boundaries. Thinning or skeletonizing on edges may be applied before edges are connected. Heuristic search, dynamic programming, relaxation, and curve fitting are possible techniques used for boundary formation.

The boundary approach is suitable for simple scenes containing man-made objects where edges are clear and predominant, such as the "blocks" world of robotics. On natural, noisy, or more complex images (eg. textured images), a boundary method often fails to produce satisfactory segmentations. More specifically, since a boundary approach is based on the detection of discontinuity and not all discontinuities signify region boundaries, extra edges are formed in segmentation. Also, when the transitions between regions are not abrupt enough, edges will not form closed connected curves. Hence region boundaries will have gaps.

Although edge element extraction can be parallelized with high efficiency, boundary formation is inherently sequential. Moreover, due to the variations on boundary length and boundary density on an image, work partitioning is especially difficult in a parallel implementation.

2.4 Region Growing

In the segmentation surveys by Zucker [70] and Haralick and Shapiro [28], region growing encompasses a large number of segmentation methods. Stated loosely, a region growing algorithm forms image regions by using the spatial information and characteristic feature of image data. Contrary to what the name implies, the region formation process can be agglomerative, divisive, or a combination of both. Existing region growing algorithms differ in the criteria used for region merging and splitting, and the way features are extracted.

We illustrate a region growing process by presenting the early work of Muerle and Allen [43]. The process has the following steps:

1. Divide the image into initial regions (say 4×4 pixels).

2. Compute a feature measure from each initial region, e.g. the average gray tone intensity.

3. Beginning with the first region in the upper left-hand corner, compare its feature with each of its neighboring region to determine if they are similar. If the feature difference is less than a predefined value, merge the two regions and update the feature measure.

4. Continue growing the region until no neighbors remain which can be merged. Label the region as a completed region.

5. Move to the next incomplete region, and repeat these steps until all regions are labelled.

Zucker [70] named the above approach the *Regional Neighbor Search*. It is a sequential process that forms regions by agglomeration. In the agglomerative approaches described in Perry [48] and Raafat [50], some seed regions are selected after step 1. Seed regions are chosen as small groups of initial regions having the most uniform or homogeneous feature values. The merging process begins from these regions.

Region growing can also be a top-down process [52]. That is, the process starts with the whole image as a single region and successively divides regions into smaller regions [47]. The process stops when all regions satisfy an appropriate homogeneity criterion. Compared with the previous agglomerative approach, the splitting approach has the advantage of a decreased overall computation. Non-homogeneous areas in the image are isolated by the partitioning process so that additional processing can be devoted to these areas.

To achieve efficient implementation, the split operation has to divide a region into regular, equal sized sub-regions. Often, the quadtree data structure is used in splitting, allowing a region to split into 4 regions. All regions have dimensions of the form 2^k, where k is a positive integer. At the end of the splitting process, the result is a group of square regions of various sizes. To produce the final segmentation, a grouping or merging process is required to examine and join adjacent regions.

The previous paragraph has described the split-and-merge [10, 11, 13, 31] approach – a region growing process that combines both region merging and splitting. We now examine the pyramid split-and-merge model discussed in [10]. The algorithm uses the quadratic picture tree which has $N+1$ levels for an $2^N \times 2^N$ image. The leaf nodes of the tree correspond to single pixels in the image and the root (level 0) represents the entire image. Except for the leaf nodes, each node of the tree has four children. The entire tree structure forms a pyramid, with nodes of a level correspond to a pyramid plane

To save memory and computation time, an intermediate level l in the tree is selected to start the segmentation. The set of all nodes at that level is called the segmentation set and each one of them represents a square sub-image of size $2^{l-1} \times 2^{l-1}$. Each square has its region number, and the segmentation set is stored in the memory in a matrix form. Since a single pixel has no texture information, split-and-merge proceeds down to regions of $k \times k$ pixels (eg. $k = 8$) instead of reaching the leaf nodes.

Beginning at the segmentation level l, the merge operations propagate towards the highest level while the split operations propagate towards level k of the pyramid. A merging criterion for any region R may be defined as in (2.1). Nodes that do not satisfy the merging criterion are said to satisfy the splitting criterion.

$$(max(C_1, C_2, C_3, C_4) - min(C_1, C_2, C_3, C_4)) < T_c \tag{2.1}$$

In (2.1), T_c is a preassigned threshold and C_j is the feature vector of a successor region of R.

The split-and-merge algorithm given in [10] is described as follows. The merge and split operations can occur at the same time:

1. *Initialization* The segmentation matrix is initialized so that all entries correspond to a l level node are assigned the same region number. That is, the image is first partitioned into $2^l \times 2^l$ regions. Subsequently, node values are initialized from level k to l. The feature measure of a parent node is the average feature of its four sons (2.2), with exceptions for the level k nodes which have texture measures as node values.

$$C_{parent} = \frac{1}{4}(C_1 + C_2 + C_3 + C_4) \tag{2.2}$$

2. *Merging: From segmentation level to level N* Each level l node in the segmentation set is traversed to check if any four nodes, sharing the same parent node at level $l + 1$ satisfy the merging criterion. If they do, the four

nodes are considered a single region. The four nodes in the segmentation set are then replaced by their parent node in the sense that the four corresponding blocks in the segmentation set matrix are labelled by a single new region number. Node value of the parent node is recomputed using (2.2). Node traversing in the segmentation set is repeated and propagated to the top of the pyramid until no node replacement occurs or the highest level of the pyramid is reached.

3. *Splitting: From segmentation level to level k* Meanwhile, a node at the initial segmentation set is tested using the splitting criterion. If the splitting criterion is satisfied, a node is replaced by its children in the segmentation set in the sense that the original block in the segmentation set matrix is split into four blocks, each labelled by four different region numbers. The test is recursively applied to the children until level k is reached.

4. *Grouping* The intermediate image at this stage is a collection of square regions of varying sizes. Grouping is necessary to combine adjacent regions of similar features. The data structures required are the region adjacency graph (whose nodes correspond to regions and branches connecting nodes correspond to adjacent regions) and a stack. A depth-first search is performed on the graph, and adjacent regions that satisfy the merging criterion are combined.

5. *Small region elimination* After the grouping operation, small isolated regions within large regions and those along the boundary areas are merged with the most similar regions adjacent to them.

In the above algorithm, Chen and Pavlidis [10] use the co-occurrence matrix as texture measure. The major disadvantages of split-and-merge are the blocky, unnatural boundaries in resulting segmentations, and the high memory requirements in the splitting, merging and grouping procedures.

Recent split-and-merge algorithms [12, 62, 67] do not adopt the pyramid model. In these implementations, split and merge are two distinct processing phases. The main purpose of the split process is to reduce processing by rapidly identifying large homogeneous regions. However, in parallel region growing where key implementation issues are very different from that on the Von Neumann machine, the split process does not necessarily shorten the total processing time. A detailed discussion is given in Section 6.3.

2.5 Other Segmentation Approaches

There exist many other segmentation algorithms that do not fit precisely into the categories given in the previous sections. Some of these techniques are hybrids of the four mentioned approaches, and others are based on paradigms which are widely used for solving problems unrelated to image segmentation. Most of these techniques are designed to work well on specific classes of images, for examples textured, range, biomedical, remote sensing images. We complete our discussion on segmentation approaches by naming a few of these methods.

Markov Random Field model-based algorithms [13, 20, 22, 38] have been proposed to segment textured and other natural images. They are optimization algorithms and can be loosely classified into three categories: simulated annealing, iterated conditional modes, and maximizer of the posterior marginals. These methods have a sound theoretical basis. However, due to difficulties in implementation, assumptions on image data and approximations to actual models often need to be made. The application of these methods are restricted to the types of images which conform to the assumptions of their respective mathematical models.

In range image segmentation, Yokoya and Levine [69] based their hybrid method on the computation of partial derivatives. A region-based and two edge-based segmentations are obtained by using three differential geometric properties. The three segmentations are then combined to produce the final results.

To segment cytoplasm images, Aggawal and Bacus [1], and Cahn *et al.* [7] suggested two hybrid algorithms. In these algorithms, thresholding is first performed on a cytoplasm image to extract the nucleus, a clustering technique is then applied to segment the cytoplasm. In the hybrid algorithm for blood cell neutrophil images given by Mui *et al.* [44], the initially thresholding process provides information on the number of clusters and cluster center locations for the latter clustering process.

Thresholding and region growing approaches have also been unified. Tsuji and Tomita [64] apply thresholding on texture images to identify points that are homogeneous with respect to some feature measure. These points then serve as seeds for the region growing process.

2.6 The Use of Varying Threshold in Region Growing

In region growing, the merging decision for two regions is often based on comparing the difference of their feature measures with a predefined value known as the segmentation threshold. The selection of an appropriate threshold is crucial to the success of a region growing process. Without domain-specific knowledge, segmentation is a tedious process involving repeated executions of a segmentation algorithm using different thresholds until a satisfactory segmentation is obtained. Although single thresholds may be sufficient to segment simple images, multiple thresholds are often required to produce segmentations of more complex images. The trial and error process on single threshold determination is even more impractical for multiple-threshold segmentations. As a result, most multiple-threshold region growing algorithms rely on domain-specific knowledge to determine the thresholds.

In the existing multiple-threshold region growing algorithms, either position-varied or time-varied thresholds are used. In position-varied threshold region growing, position-dependent thresholds, usually determined from *a priori* knowledge about an image, are applied on different parts of the image. Time-varied threshold region growing, less common than the position-varied method, refers to the application of varying thresholds during different stages of the region growing process. Due to the lack of a standard test image set, no detailed comparisons among the various multiple-threshold approaches have been reported in the literature. Unlike the existing methods, our adaptive algorithm does not require *a priori* knowledge and it uses both position- and time-varied thresholds that are computed automatically in the region growing process.

It has long been realized that a single constant threshold is often inadequate for segmenting an image. Harlow and Eisenbeis [30] suggested the use of position-dependent thresholds for the segmentation of radiographic images. Their approach applies position-dependent thresholds on an image using *a priori* knowledge represented as an image semantic tree. For example, the semantic tree of a chest X-ray has the nodes closest to the root representing the left, right lungs, and heart; and the left lung node leads to nodes for the ribs and lung background. Each node of this tree is associated with a threshold suitable for segmenting the intended region.

In a range image segmentation algorithm, Taylor, Savini, and Reeves [62] control the order of merge attempts by gradually relaxing the homogeneity criterion in the process. A strict threshold that gives many small but homogeneous regions is applied on the entire image at the beginning of the merging process, then successively larger thresholds are used to create larger regions in the subsequent iterations. This dynamic merge relaxation technique helps to improve segmentation results significantly. No detail is given on the threshold relaxation process.

As in the above mentioned algorithms, several other adaptive or multiple-threshold region growing algorithms, such as those described in [35, 56, 68], are designed for specific applications and require domain-specific knowledge to determine segmentation thresholds. An initial effort in general adaptive region growing is presented in [12]. In this study, Chen, Lin, and Chen developed an approach to automatically compute position-dependent thresholds using localized characteristic features. The determination of region homogeneity is treated as a sequence of decision problems in terms of predicates in their hypothesis model. Several thresholds are used in the algorithm for decision tests to be applied at different stages under different conditions in the process. When image data does not conform to assumptions made in the mathematical model, heuristic rules are applied. To compute these position-dependent thresholds, histogram analysis on co-occurrence features are performed on overlapped rectangular windows of the image prior to region growing. Segmentations on four natural scene and aerial images, and four X-ray scans of the corpus colosum and pituitary gland were reported.

Our algorithm differs from the existing adaptive methods in the simplicity and generality of our adaptive homogeneity test. We base our feature histogram analysis on regions formed in the merging process; no domain-specific knowledge is necessary. The determination of adaptive thresholds is integrated in the region merging process so that the thresholds are position- as well as time-varied. Unlike the hypothesis model given by Chen *et al.*, our adaptive test can be applied on features having non-normal, arbitrary distributions. In this study, we compare our simple and general purpose adaptive approach with the simple and still widely used fixed threshold method.

2.7 Parallel Region Growing Models and Implementations

Pietikainen and Rosenfeld [49], Burt *et al.* [6] suggested a linked pyramid computational model for parallel split-and-merge. The proposed pyramid structure is a layered arrangement of square arrays in which each array is half as long and wide as the array below it. A father-son relationship which may be updated at each iteration is defined between nodes of adjacent layers, Segmentation is a multiresolution, iterative process. The results from each node of the pyramid are used to adjust and improve performance of some other nodes in the next iteration. No timing results on this algorithm were published.

Tilton and Cox [63] presented two single-instruction multiple-data stream (SIMD) model algorithms for segmenting Landsat images using the pure merge approach. In the first method, an image is subdivided into $N \times M$ regions, with each region being mapped onto an individual processor. In each iteration, a similarity measure between each pair of adjacent regions is computed. The algorithm terminates if the globally best similarity measure is less than a preset minimum. Otherwise, this pair of regions is merged. This algorithm was implemented on a Massively Parallel Processor (MPP) and has a time complexity of $O(N \times M)$. The second SIMD algorithm is designed for a reconfigurable parallel machine such as the ZMOB. It has a time complexity of $N \times M(1 - Q)^{(I-1)}$, where Q is the percentage of merges in each iteration and I is the total number of iterations. The improvement was achieved by using tables to store feature values and adjacency relationships of regions. Since the sizes of these tables decrease with each iteration, the computational requirement also decreases. The pixel-based MPP algorithm was tested on a panchromatic aircraft scanner image but no timing performance of the algorithm was reported.

Another SIMD region growing approach implemented on the MPP was given by Willebeek-LeMair and Reeves [67]. The pixels of an image are spatially mapped onto a mesh-connected SIMD architecture. A recursive split phase at the beginning of the process divides an image into homogeneous, square regions of various dimensions. The subsequent merge phase is an iterative process that uses the Best-merge criterion (that will be discussed in detail in Chapter 4) to merge adjacent region pairs. An embedded tree structure, for linking together pixels belonging to the same region, facilitates the routing of information along irregularly shaped regions and boundaries. If d_i is the longest path between two points in region i, the primitives for creating the embedded trees, gathering and distributing information to all region points have a time complexity of $O(d_{max})$, where d_{max} is the maximum of all d_i's in the image. The value of d_i depends on the size and shape of a region. For example, a simple blob-like region has a smaller d_i than a spiral region of the same size. The number of iterations required to complete the merging process also depends on the shape of the image regions.

Willebeek-LeMair and Reeves [67] also presented a message passing multiple-instruction multiple-data stream (MIMD) model for medium grain parallel region growing. Their MIMD algorithm was implemented on a iPSC/2 hypercube machine. Initially, an image is partitioned into equal sized blocks and mapped onto a two-dimensional array of processors. As in the SIMD algorithm, a top-down split process is invoked to rapidly deal with very large regions. Adjacent processors will then exchange boundary information to update data structures before the iterative merge process is initiated. A merge may be local to a processor (intra-processor merge) or involve regions hosted by different processors (inter-processor merge). An inter-processor merge incurs communication and synchronization costs that are not present in an intra-processor merge. In the message passing implementation, process synchronization is a complicated issue that requires careful handling. Since it is impossible to determine the best workload partition for the processors before run time, processor utilization is also an implementation concern.

On the MPP, the SIMD algorithm given in [67] requires 22 iterations and takes 3.05 seconds to merge a 20×52 pixels rectangular object. The same algorithm requires 24 iterations and 2.89 seconds to merge a 36 pixel diameter circular object. The split process takes about 3.4 milliseconds for both objects. The MIMD algorithm performance on the iPSC/2 depends on the mapping of region data to processors. In a four-processor run with the rectangular object shared by two processors, about 67 and 23 milliseconds are required by the split and merge processes respectively. The circular object, when shared by all four processors, requires 62 and 46 milliseconds to split and merge.

A coarse grain parallel segmentation approach, known as image seaming, was proposed by Chen and Pavlidis [9]. In their method, an image is decomposed into overlapping tiles and then independently processed by individual processors. Seaming refers to the assembling of results from these processors. In the seaming phase, two or more cut regions are merged if enough evidence suggests that they belong to the same region in the whole image. The segmentation results are then relabelled. Segmentations of a 512×512 aerial image are obtained from a sequential split-and-merge algorithm and the parallel seaming algorithm, and the results were compared. For a 64×64 image, a total speed factor of 2.4 is recorded using four processors on the SEQUENT machine.

Chapter 3

The Adaptive Homogeneity Test

The homogeneity test is an important component of a region growing algorithm. The quality of segmentations produced by a region growing algorithm is greatly determined by its homogeneity test. In this chapter, we propose a new test called the Adaptive homogeneity test. Our test is shown, both empirically and analytically, to be superior to the widely used Fixed Threshold test.

3.1 Definitions

We adopt the region growing definition given in [12, 67, 70]. Let an image I be represented by a two-dimensional array of pixels. An image region R is defined as a connected subset of I that is homogeneous with respect to some image properties such as grayscale or texture. Let $\mathcal{H}$ be a logical predicate on a homogeneity measure defined on region R such that

$$\mathcal{H}(R) = \begin{cases} \text{true} & \text{if } R \text{ is homogeneous,} \\ \text{false} & \text{otherwise.} \end{cases} \tag{3.1}$$

A segmentation of an image is a partition of I into regions R_i, $i = 1, ..., m$ satisfying the following conditions:

1. $I = \bigcup_{i=1}^{m} R_i$
2. $R_i \bigcap R_j = \phi$, $\quad 1 \leq i,j \leq m$, and $i \neq j$.
3. $\mathcal{H}(R_i) = \text{true}$ $\quad$ for all i.
4. $\mathcal{H}(R_i \bigcup R_j) = \text{false}$, $\quad 1 \leq i,j \leq m$, $i \neq j$, and R_i, R_j are adjacent.

Conditions 1 and 2 state that every pixel of a segmented image belongs to a region, and the regions are non-overlapping. In region growing segmentation, an image is first divided into many small, usually equal sized, **primitive regions** that satisfy condition 3. These primitive regions are then repeatedly merged to form larger regions until condition 4 is fulfilled. A **region** is a spatially connected set of primitive regions. The logical predicate $\mathcal{H}$ is called the **homogeneity test.**

To further discuss the homogeneity test, we introduce the widely used two-dimensional random field image model [11, 17, 20]. The stochastic nature of this model makes it particularly suitable for modelling textured images and images with high degrees of irregularities. We view an image as a two-dimensional random field, with each random variable describing a property of a primitive region. In region growing, such properties can be the average gray-tone value or some textural feature that are useful in distinguishing regions in the segmentation process. For example, if we choose gray-tone level as the property measure and let G represent this random variable, then the probability of G having a gray-level g can be written as

$$f_G(g) = Pr(G = g). \tag{3.2}$$

We called f_G the probability density function of G.

Using this model, a homogeneous image region is represented by a set of random variables having the same distribution function; different regions are defined by different distribution functions. Consequently, the objective of a region growing process is to produce a spatial partition of the random field in such a way that random variables within each group have the same distribution function.

3.2 Homogeneity Tests

In a binary merge, each of the two regions subjected to a homogeneity test is assumed to be homogeneous. The homogeneity test attempts to answer the question: "If the two regions are merged, will the resulting region be homogeneous?".

The regions are allowed to merge only if the test result is positive. Existing region growing algorithms differ in the definitions of their homogeneity tests.

In most existing region growing algorithms [10, 11, 28, 67], a predefined, fixed threshold ϵ is applied on the entire image, throughout the process, to decide if any two regions can be merged. We refer to these methods as Fixed Threshold region growing.

Let R_i and R_j be two regions that are made up of n and m primitive regions, respectively. That is, $R_i = \{X_{i1}, \ldots, X_{in}\}$ and $R_j = \{X_{j1}, \ldots, X_{jm}\}$. Also, let $\bar{X}_i$ and $\bar{X}_j$ denote the means of the random variables of R_i and R_j, respectively. Then, the **Fixed Threshold homogeneity test** with threshold ϵ on the regions R_i and R_j is defined as

$$\mathcal{H}(R_i \bigcup R_j) = \begin{cases} \text{true} & \text{if } |\bar{X}_i - \bar{X}_j| < \epsilon, \\ \text{false} & \text{otherwise.} \end{cases} \tag{3.3}$$

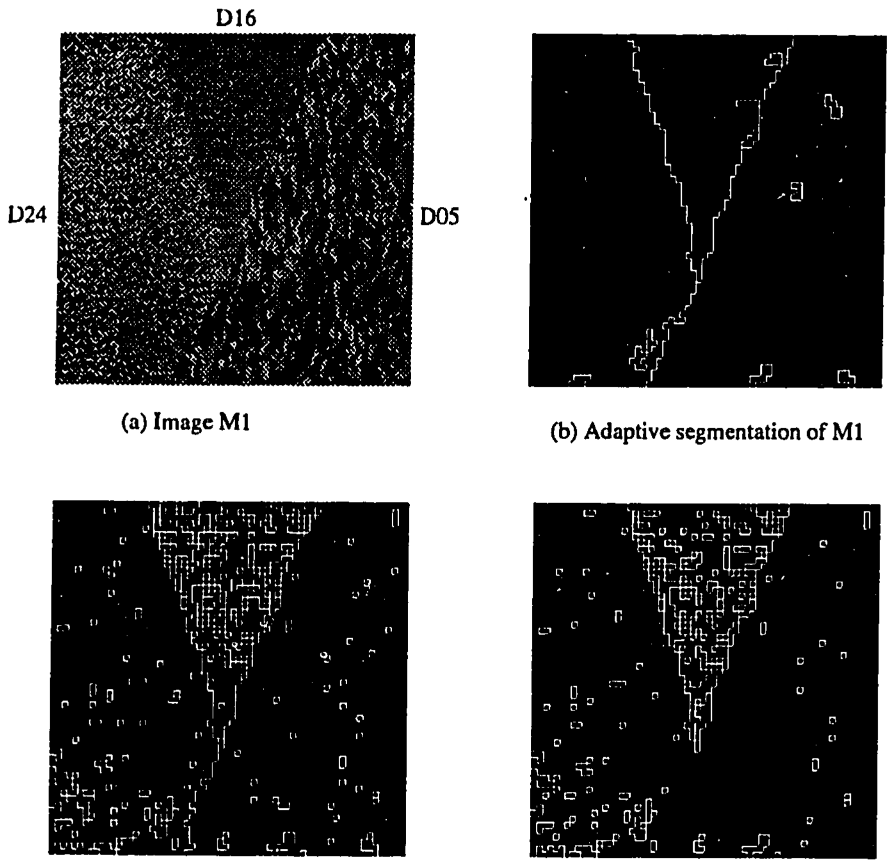
If an appropriate threshold is found, the Fixed Threshold test is simple and efficient to implement. However, in many cases, there may not be a threshold which possesses the ideal separating power for the entire image due to the diversity of regional feature homogeneities. In such a case, a threshold that works well for one region may be too small or too large for another region. Even if a good threshold does exist for an image, there is no reliable method for finding it.

To illustrate the importance of incorporating regional feature variation in the homogeneity test, we give a segmentation example as shown in Figure 3.1. The test image M1 in (a) is a simple three-segment image composed of the Brodatz textures [4] D05, D16, and D24. In the following discussion, these texture codes will be used to identify the segments. We use the co-occurrence contrast feature in the segmentation of M1. The definitions of co-occurrence features used in this study are given in Appendix A.

The contrast feature histograms given in Figure 3.2 are constructed using an equal number of feature values from the three textures. The means and standard deviations of these contrast measures are given in Table 3.1. It is clear that the three segments in M1 have different contrast variations.

Figure 3.1(c) and (d) give the segmentations of the Fixed Threshold method using different thresholds. Despite careful adjustments on the threshold, the Fixed Threshold method fails to produce a satisfactory segmentation. Due to the large contrast variation of the D16 segment, a large threshold is needed to merge this segment into one region. However, if a threshold of greater than 0.25

is applied, the left D24 and right D05 regions would be incorrectly merged into one, while the upper D16 segment still consists of a group of small regions. The Fixed Threshold segmentation on M1 shows that a single fixed threshold on the contrast measure may not work well on differentiating multiple textures.



(a) Image M1

(b) Adaptive segmentation of M1

(c) Fixed Threshold segmentation of M1 (threshold=0.25)

(d) Fixed Threshold segmentation of M1 (threshold=0.26)

Figure 3.1: Fixed Threshold and Adaptive Segmentations

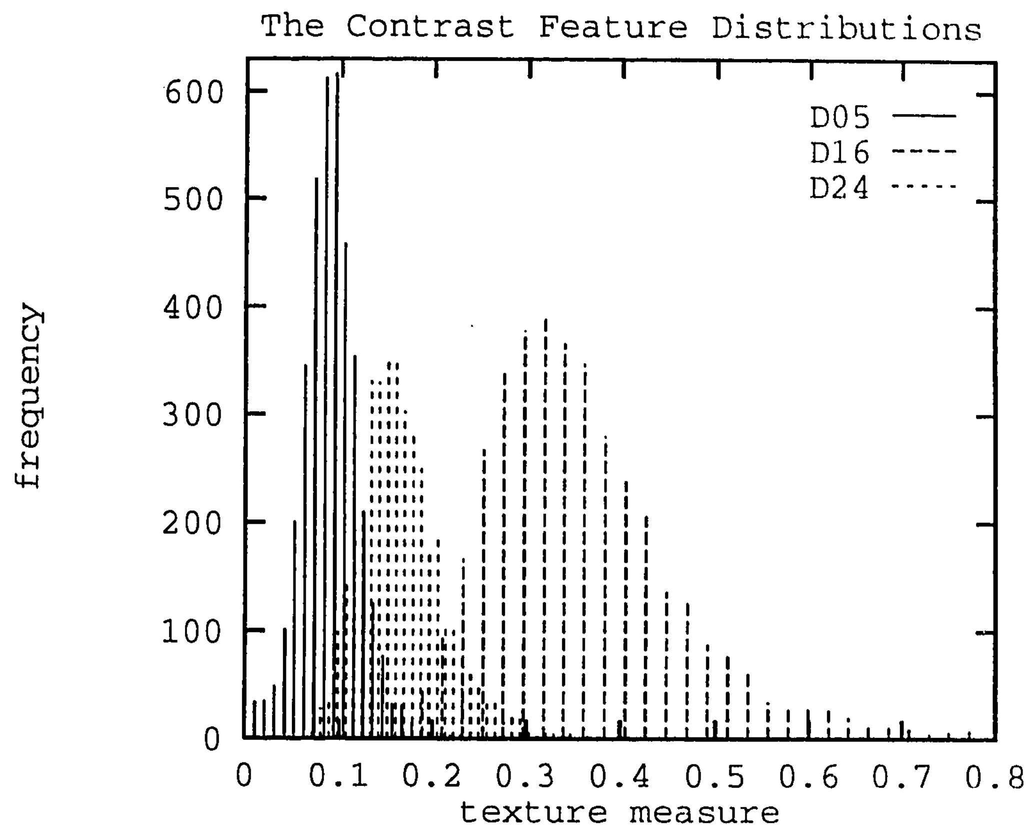


Figure 3.2: Contrast histograms of three textures

Table 3.1: Contrast means and standard deviations of three textures

Texture	mean	s.d.
D05	0.0855	0.0309
D16	0.3493	0.1067
D24	0.1615	0.0444

A diverse degree of region homogeneity as shown in the above example is a common occurrence in real images. For image M1, there may be a feature, different from the contrast, that can result in a better segmentation using the Fixed Threshold method. Nevertheless, if it exists, such a feature can only be found after a thorough analysis on many features of the three composition textures. It would be more fruitful to seek a homogeneity test that can overcome the region feature variation problem directly.

Stated loosely, a threshold specifies the allowance for which feature values within a region can deviate. A segmentation threshold is therefore a feature- and region-dependent parameter. In our Adaptive homogeneity test, we base the homogeneity decision on the feature distributions of the two regions under examination. From the feature histogram of a region, we estimate a range, called the Adaptive range (ℓ_1, ℓ_2), within which the central λ portion of the region feature values lie. The user-defined parameter λ $(0 \leq \lambda \leq 1)$ is called the Adaptive parameter. Suppose a region R_i has feature values $\{X_{i1}, \ldots, X_{in}\}$ where each X_{ij} is identically distributed, and the feature mean of R_i is $\bar{X}_i = \sum_{j=1}^{n} X_{ij}/n$. Given a λ, for each region R_i in the image, we define the region's Adaptive range (ℓ_{i1}, ℓ_{i2}) by the following relations:

$$Pr(\ell_{i1} < X_{ij} < \ell_{i2}) = \lambda \qquad \text{and} \tag{3.4}$$

$$Pr(\ell_{i2} \leq X_{ij}) = Pr(X_{ij} \leq \ell_{i1}) = \frac{1-\lambda}{2}. \tag{3.5}$$

For two regions to be considered homogeneous, we require that each region's feature mean falls within the other region's Adaptive range. The **Adaptive homogeneity test** on regions R_i and R_j is defined as

$$\mathcal{H}(R_i \bigcup R_j) = \begin{cases} \text{true} & \text{if } \ell_{j1} < \bar{X}_i < \ell_{j2} \text{ and } \ell_{i1} < \bar{X}_j < \ell_{i2}, \\ \text{false} & \text{otherwise.} \end{cases} \tag{3.6}$$

In our experiment, it is found that $\lambda \in (0.80, 0.85)$ produce good segmentations. For example, if $\lambda = 0.80$, the Adaptive homogeneity test in Equation 3.6 states that regions R_i and R_j are considered homogeneous if $\bar{X}_i$ falls within the central 80% range of R_j, and $\bar{X}_j$ falls within that of R_i.

In the segmentation example given in Figure 3.1, comparing the M1 segmentations in (a) and (b), it is clear that the result from the Adaptive method is better than that from the Fixed Threshold method. We summarize the advantages of the Adaptive test as follows. A theoretical analysis on the homogeneity tests is given in the next section, and experimental results are given in Chapter 4 to substantiate these claims.

- It results in a robust region growing algorithm.
  Since merge decisions are based on locally computed feature means and

variations, the segmentation algorithm is robust with respect to region feature homogeneity.

- It is easy to use.
  The Adaptive parameter λ can be kept at a fixed value for different images, and the Adaptive range for each region is computed automatically. There is no parameter tuning as in the Fixed Threshold method. Also, no *a priori* knowledge of an image is assumed.

- It is efficient to implement.
  Although the Adaptive test is more complex than the Fixed Threshold test, it is still simple enough to be implemented efficiently.

3.3 Comparing the Fixed Threshold and the Adaptive Tests

In this section, we provide a formal analysis of the Fixed Threshold and the Adaptive homogeneity tests using probability theory.

Consider three regions R_1, R_2, and R_3 with feature values $\{Z_1, \ldots, Z_{n_1}\}$, $\{X_1, \ldots, X_{n_2}\}$, and $\{Y_1, \ldots, Y_{n_3}\}$, respectively. Region R_i is formed by n_i primitive regions, where n_i is also called the size of R_i. Let the feature means of the three regions be $\bar{Z}$, $\bar{X}$, and $\bar{Y}$.

In the following analysis, we assume the feature random variables X_i, Y_i, and Z_i to be independently distributed. In addition, all random variables within the same region are assumed to be identically and normally distributed. The validity of the normality assumption is substantiated by the Shapiro-Wilk test performed on five feature measures over twenty textures. Details of the test are given in Appendix B. Although the normality assumption is required in the following proofs, it should be noted that application of the FAS algorithm is not restricted to normally distributed features.

Definition 3.1 (Normal distribution)

$$X \sim N(\mu, \sigma) \Longleftrightarrow Pr(u < X < v) = \frac{1}{\sigma\sqrt{2\pi}} \int_u^v e^{-\frac{(x-\mu)^2}{2\sigma^2}} dx \overset{\text{def}}{=} P_N(\mu, \sigma, u, v)$$

$$\mu, \sigma, u, v \in \Re, \qquad \sigma > 0.$$

We use the notation $N(\mu,\sigma)$ for a normal distribution with mean μ and variance σ^2. The shorthand notation $P_N(\mu,\sigma,u,v)$ is used to represent the area under the normal distribution $N(\mu,\sigma)$ from u to v.

The following Lemmas are used for proving the latter theorems. The shorthand notation $\overset{[n]}{=}$ is used to indicate that a certain equality holds due to Lemma n. The proofs of these lemmas are given in Appendix C.

Lemma 3.1 (Shift)

$$P_N(\mu,\sigma,u,v) = P_N(\mu-a,\sigma,u-a,v-a), \qquad a\in\Re.$$

Lemma 3.2 (Halves)

$$P_N(\mu,\sigma,\mu-\ell,\mu+\ell) = 2P_N(\mu,\sigma,\mu,\mu+\ell) = 2P_N(0,\sigma,0,\ell).$$

Lemma 3.3 (Negation)

$$P_N(\mu,\sigma,-\ell,\ell) = P_N(-\mu,\sigma,-\ell,\ell).$$

Lemma 3.4 (Conversion to Standard Normal)

$$P_N(0,\sigma,0,\ell) = P_N(0,1,0,\frac{\ell}{\sigma}) \overset{\text{def}}{=} S(\frac{\ell}{\sigma}).$$

Lemma 3.5 (Conversion to Exponential)

$$P_N(\mu,u\sigma,-v\ell,v\ell) = \frac{1}{\sqrt{\pi}}\int_{\frac{-v\ell-\mu}{\sqrt{2}u\sigma}}^{\frac{v\ell-\mu}{\sqrt{2}u\sigma}} e^{-y^2}dy \overset{\text{def}}{=} \frac{1}{\sqrt{\pi}}P_E\left(\frac{-v\ell-\mu}{\sqrt{2}u\sigma},\frac{v\ell-\mu}{\sqrt{2}u\sigma}\right).$$

Lemma 3.6

$$P_N(0,\frac{\sigma}{n},0,\ell) > P_N(0,\sigma,0,\ell), \qquad n>1.$$

Lemma 3.7

If $\sigma_1 > \sigma_2$ *and* $P_N(0, \sigma_1, 0, t_1) = P_N(0, \sigma_2, 0, t_2)$, *then* $t_1 > t_2$.

Lemma 3.8

$$P_E(a, d) > P_E(b, c) \qquad \textit{if} \quad a \leq b \leq c < d.$$

Lemma 3.9

$$P_E(a, b) < P_E(c, d) \qquad \textit{if} \quad a < b \leq c < d \leq 0, \quad d - c \geq b - a.$$

Lemma 3.10

$$P_E(a, b) < P_E(c, d) \qquad \textit{if} \quad a < b \leq 0, \quad c \leq 0 < d, \quad d - c \geq b - a.$$

Lemma 3.11

$$P_E(a, b) < P_E(c, d) \quad \textit{if} \ \ a < b < 0 < c < d, \ \ |b| > |c|, \ \ d - c \geq b - a.$$

Suppose regions R_2 and R_3 are homogeneous and $X_i, Y_j \sim N(\mu_2, \sigma_2)$ for all i, j where $1 \leq i \leq n_2$ and $1 \leq j \leq n_3$. Region R_1, not homogeneous with R_2 or R_3, has a feature distribution $Z_k \sim N(\mu_1, \sigma_1)$ for all k where $1 \leq k \leq n_1$. We say that R_2 and R_3 are two independent and identically distributed (i.i.d.) regions, while R_2 and R_1 are two non-i.i.d. regions. Also, let t be the threshold used in the Fixed Threshold test, and λ be the Adaptive parameter.

We will examine the probabilities for the region pair (R_2, R_3) and (R_2, R_1) to pass the Adaptive and the Fixed Threshold tests. We denote these probabilities using P_{iA}, P_{iF}, P_{nA}, and P_{nF}. The suffixes A and F represent the Adaptive and Fixed Threshold tests respectively; i indicates that feature random variables of the two regions are i.i.d., as in the case of (R_2, R_3); and n indicates that feature random variables of the two regions are not i.i.d., as in the case of (R_2, R_1). Ideally, we would like P_{iF}, P_{iA} to be close to 1, and P_{nF}, P_{nA} to be close to 0.

We want to show that Adaptive is a better homogeneity test than Fixed Threshold by proving the following:

- When feature random variables of two regions are i.i.d. and the homogeneity test should be passed, P_{iF} has a lower bound of 0 while P_{iA} has a lower bound of λ^2. (Theorems 3.1 and 3.2)
- When feature random variables of two regions are not i.i.d. and the homogeneity test should not be passed, for some λ and relation between λ and ϵ so that $P_{iF} \approx P_{iA}$, we have $P_{nF} > P_{nA}$ if some general conditions on the region sizes and feature variances are satisfied. (Theorem 3.3)

From properties of the normal distribution on sample means [19, 39], we have $\bar{X} \sim N(\mu_2, \sigma_2/\sqrt{n_2})$, $\bar{Y} \sim N(\mu_2, \sigma_2/\sqrt{n_3})$, and $\bar{Z} \sim N(\mu_1, \sigma_1/\sqrt{n_1})$. Also, $(\bar{X} - \bar{Y}) \sim N(0, \sigma_2(\frac{1}{\sqrt{n_2}} + \frac{1}{\sqrt{n_3}}))$, and $(\bar{X} - \bar{Z}) \sim N(\mu_2 - \mu_1, \frac{\sigma_2}{\sqrt{n_2}} + \frac{\sigma_1}{\sqrt{n_1}})$. Since the normal distribution is symmetrical about the distribution mean, we can denote the Adaptive range of R_1 using $(\mu_1 - \ell_1, \mu_1 + \ell_1)$, and that of regions R_2 and R_3 as $(\mu_2 - \ell_2, \mu_2 + \ell_2)$.

According to the definitions of the Fixed Threshold and Adaptive tests given in equations 3.3 and 3.6, the probabilities P_{iA}, P_{iF}, P_{nA}, and P_{nF} can be written as follows:

- Probability of passing the Fixed Threshold test on two i.i.d. regions R_2 and R_3:

$$\begin{aligned} P_{iF} &\overset{\text{def}}{=} Pr(|\bar{X} - \bar{Y}| < \epsilon) \\ &= Pr(-\epsilon < \bar{X} - \bar{Y} < \epsilon) \\ &= P_N(0, \sigma_2(\frac{1}{\sqrt{n_2}} + \frac{1}{\sqrt{n_3}}), -\epsilon, \epsilon) \\ &\overset{[3.2]}{=} 2P_N(0, \sigma_2(\frac{1}{\sqrt{n_2}} + \frac{1}{\sqrt{n_3}}), 0, \epsilon). \end{aligned}$$

- Probability of passing the Adaptive test on two i.i.d. regions R_2 and R_3:

$$\begin{aligned} P_{iA} &\overset{\text{def}}{=} Pr(\mu_2 - \ell_2 < \bar{X} < \mu_2 + \ell_2 \;\wedge\; \mu_2 - \ell_2 < \bar{Y} < \mu_2 + \ell_2) \\ &= Pr(\mu_2 - \ell_2 < \bar{X} < \mu_2 + \ell_2) Pr(\mu_2 - \ell_2 < \bar{Y} < \mu_2 + \ell_2) \\ &= P_N(\mu_2, \frac{\sigma_2}{\sqrt{n_2}}, \mu_2 - \ell_2, \mu_2 + \ell_2) P_N(\mu_2, \frac{\sigma_2}{\sqrt{n_3}}, \mu_2 - \ell_2, \mu_2 + \ell_2) \\ &\overset{[3.2]}{=} 2P_N(0, \frac{\sigma_2}{\sqrt{n_2}}, 0, \ell_2) 2P_N(0, \frac{\sigma_2}{\sqrt{n_3}}, 0, \ell_2). \end{aligned}$$

- Probability of passing the Fixed Threshold test on two non-i.i.d. regions R_2 and R_1:

$$P_{nF} \overset{\text{def}}{=} Pr(|\bar{X} - \bar{Z}| < \epsilon)$$

$$\begin{aligned} &= Pr(-\epsilon < \bar{X} - \bar{Z} < \epsilon) \\ &= P_N(\mu_2 - \mu_1, \frac{\sigma_2}{\sqrt{n_2}} + \frac{\sigma_1}{\sqrt{n_1}}, -\epsilon, \epsilon). \end{aligned}$$

- Probability of passing the Adaptive test on two non-i.i.d. regions R_2 and R_1:

$$\begin{aligned} P_{nA} &\overset{\text{def}}{=} Pr(\mu_1 - \ell_1 < \bar{X} < \mu_1 + \ell_1 \ \wedge \ \mu_2 - \ell_2 < \bar{Z} < \mu_2 + \ell_2) \\ &= Pr(\mu_1 - \ell_1 < \bar{X} < \mu_1 + \ell_1) Pr(\mu_2 - \ell_2 < \bar{Z} < \mu_2 + \ell_2) \\ &= P_N(\mu_2, \frac{\sigma_2}{\sqrt{n_2}}, \mu_1 - \ell_1, \mu_1 + \ell_1) P_N(\mu_1, \frac{\sigma_1}{\sqrt{n_1}}, \mu_2 - \ell_2, \mu_2 + \ell_2). \end{aligned}$$

Theorem 3.1 *The probability of two i.i.d. regions passing the Fixed Threshold test decreases when the standard deviation of their feature distribution function increases. That is,* $\lim_{\sigma\to\infty} P_{iF} = 0$.

Proof.

Consider two i.i.d. regions of sizes n_2 and n_3 sharing the feature distribution $N(\mu, \sigma)$.

Let $\sigma(\frac{1}{\sqrt{n_2}} + \frac{1}{\sqrt{n_3}}) = \phi$, then $P_{iF} = 2P_N(0, \phi, 0, \epsilon) \overset{[3.4]}{=} 2S(\frac{\epsilon}{\phi})$,

$\lim_{\sigma\to\infty} S(\frac{\epsilon}{\phi}) = \lim_{\phi\to\infty} S(\frac{\epsilon}{\phi}) = \lim_{\frac{\epsilon}{\phi}\to 0} S(\frac{\epsilon}{\phi}) = 0$.

Therefore, $\lim_{\sigma\to\infty} P_{iF} = 0$. □

Theorem 3.2 *The probability of two i.i.d. regions passing the Adaptive test has a lower bound of* λ^2. *That is,* $P_{iA} > \lambda^2$.

Proof.

Consider two i.i.d. regions of sizes $n_2, n_3 > 1$ sharing the feature distribution $N(\mu, \sigma)$.

$P_N(0, \frac{\sigma}{\sqrt{n_2}}, 0, \ell) \overset{[3.6]}{>} P_N(0, \sigma, 0, \ell) = \frac{\lambda}{2}$,

$P_N(0, \frac{\sigma}{\sqrt{n_3}}, 0, \ell) \overset{[3.6]}{>} P_N(0, \sigma, 0, \ell) = \frac{\lambda}{2}$,

$P_{iA} = 2P_N(0, \frac{\sigma}{\sqrt{n_2}}, 0, \ell) 2P_N(0, \frac{\sigma}{\sqrt{n_3}}, 0, \ell) > 4\ (\frac{\lambda}{2})(\frac{\lambda}{2}) = \lambda^2$. □

Theorem 3.2 says that regardless of the feature distribution parameters, it is guaranteed that the probability of two i.i.d. regions passing the Adaptive test is at least λ^2, or 0.64 ($\lambda = 0.8$) in practice. On the other hand, as stated in Theorem 3.1, the same event can have a probability as low as 0 if the Fixed Threshold test is used; the larger the standard deviation of the feature distribution, the lower is the probability.

We now look at the implications of these two theorems. If an image has a varied degree of region feature variations, and a relatively small fixed threshold needs to be used because of the closeness of some region feature means, then the Fixed Threshold method would have difficulties merging homogeneous regions, especially for those regions with high feature variations. Conversely, the Adaptive method is consistently good in merging homogeneous regions, regardless of their feature variations. This result is consistent with our observation in the segmentation of image M1 (Figure 3.1).

Besides allowing homogeneous regions to merge, it is equally important for a good homogeneity test to ascertain that non-homogeneous regions would not be merged. In the remaining theorem, we will prove that under a general condition on region sizes, if a Fixed Threshold test and an Adaptive test are equally good in merging homogeneous regions, then the Adaptive test would have a lower probability than the Fixed Threshold test of allowing the merging of two non-homogeneous regions.

To prove the next theorem, we need to present the lemmas which specify the condition for a Fixed Threshold and an Adaptive tests to be equally good in merging homogeneous regions.

Lemma 3.12 *If $P_N(0, \sigma, -\ell, \ell) = \lambda$, and $\ell = c\sigma$ where c is a constant, then if $\lambda > 0.8$, we have $c > 1.28$.*

Proof.

Given a fixed $\lambda > 0.80$, $P_N(0, \sigma, 0, \ell) \overset{[3.4]}{=} S(\frac{\ell}{\sigma}) > 0.40$

From the standard normal distribution function (Tables in [19, 39]), $\frac{\ell}{\sigma} > 1.28$.

Therefore, $c > 1.28$. □

Lemma 3.13 *If $\lambda > 0.80$ and $P_N(0, \sigma, -\ell, \ell) = \lambda$, then $P_{iF} \approx P_{iA} \approx 1$ for any two i.i.d. regions with sizes greater than 8 if we set $\epsilon = 3\ell$.*

Proof.

From Lemma 3.12, if $\lambda > 0.8$ and $\ell = c\sigma$, then $c > 1.28$.

Consider two i.i.d. regions of sizes n_2 and n_3 having the feature distribution $N(\mu, \sigma)$.

$P_N(0, \frac{\sigma}{\sqrt{n_2}}, 0, \ell) \overset{[3.4]}{=} S(\frac{\ell\sqrt{n_2}}{\sigma}) = S(\sqrt{n_2}c) \approx \frac{1}{2}$ for $\sqrt{n_2} \geq 3$.

Similarly, $P_N(0, \frac{\sigma}{\sqrt{n_3}}, 0, \ell) \approx \frac{1}{2}$ for $\sqrt{n_3} \geq 3$.

Therefore, $P_{iA} \approx 1$.

$P_{iF} \overset{[3.4]}{=} 2\,S\left(\frac{\epsilon}{\sigma(\frac{1}{\sqrt{n_2}} + \frac{1}{\sqrt{n_3}})}\right)$.

Let $\epsilon = 3\ell = 3c\sigma$, then $P_{iF} = 2\,S\left(\frac{3c}{\sqrt{n_2}+\sqrt{n_3}}\sqrt{n_2 n_3}\right) \approx 1$ for $\sqrt{n_2} \geq 2, \sqrt{n_3} \geq 2$.

□

Theorem 3.3 *If $\lambda > 0.80$, $P_{iA} \approx P_{iF}$ for regions R_2 and R_3, and $\frac{\sigma_2}{n_2} \leq 4\frac{\sigma_1}{n_1}$ or $n_1 \leq 4n_2$, where n_2, σ_2 and n_1, σ_1 are the size and standard deviation of R_2 and R_1 respectively, then $P_{nA} < P_{nF}$ for regions R_2 and R_1.*

Proof.

For $\lambda > 0.80$, and choosing $\epsilon = 3\ell_2$, Lemma 3.13 states that $P_{iF} \approx P_{iA} \approx 1$ for regions R_2 and R_3. Let us now consider the probabilities of merging the two non-i.i.d. regions R_2 and R_1.

Using the Fixed Threshold test with $\epsilon = 3\ell_2$,

$$P_{nF} = P_N(\mu_2 - \mu_1, \frac{\sigma_2}{\sqrt{n_2}} + \frac{\sigma_1}{\sqrt{n_1}}, -3\ell_2, 3\ell_2) \overset{[3.3]}{=} P_N(\mu_1 - \mu_2, \frac{\sigma_2}{\sqrt{n_2}} + \frac{\sigma_1}{\sqrt{n_1}}, -3\ell_2, 3\ell_2).$$

With $\mu_1 - \mu_2 = \mu$, $\ell_2 = \ell$, and $\frac{\sigma_2}{\sqrt{n_2}} + \frac{\sigma_1}{\sqrt{n_1}} = \sigma'$,

$$P_{nF} = P_N(\mu, \sigma', -3\ell, 3\ell) \overset{[3.5]}{=} \frac{1}{\sqrt{\pi}} P_E\left(\frac{-3\ell - \mu}{\sqrt{2}\sigma'}, \frac{3\ell -}{\sqrt{2}\sigma}\right).$$

Using the Adaptive test,

$$P_{nA} = P_N(\mu_2, \frac{\sigma_2}{\sqrt{n_2}}, \mu_1 - \ell_1, \mu_1 + \ell_1) P_N(\mu_1, \frac{\sigma_1}{\sqrt{n_1}}, \mu_2 - \ell_2, \mu_2 + \ell_2).$$

Since $0 \le P_N(.,.,u,v) < 1$, for any finite u and v,

$$\Rightarrow\ P_{nA} < P_N(\mu_2, \frac{\sigma_2}{\sqrt{n_2}}, \mu_1 - \ell_1, \mu_1 + \ell_1) \overset{[3.1]}{=} P_N(\mu_2 - \mu_1, \frac{\sigma_2}{\sqrt{n_2}}, -\ell_1, \ell_1) \quad \text{and}$$
$$P_{nA} < P_N(\mu_1, \frac{\sigma_1}{\sqrt{n_1}}, \mu_2 - \ell_2, \mu_2 + \ell_2) \overset{[3.1]}{=} P_N(\mu_1 - \mu_2, \frac{\sigma_1}{\sqrt{n_1}}, -\ell_2, \ell_2).$$

Consider the following two cases:

Case 1: If $\frac{\sigma_2}{\sqrt{n_2}} \le 2\frac{\sigma_1}{\sqrt{n_1}}$, let $\frac{\sigma_1}{\sqrt{n_1}} = \sigma$.

Case 2: Otherwise, if $2\sqrt{n_2} > \sqrt{n_1}$, let $\frac{\sigma_2}{\sqrt{n_2}} = \sigma$.
$\frac{\sigma_2}{\sqrt{n_2}} > 2\frac{\sigma_1}{\sqrt{n_1}} \Rightarrow \sigma_2 > \frac{2\sqrt{n_2}}{\sqrt{n_1}}\sigma_1$.
If $2\sqrt{n_2} \ge \sqrt{n_1}$, then $\sigma_2 > \sigma_1$.
By Lemma 3.7, we have $\ell_2 > \ell_1$,
$\Rightarrow P_{nA} < P_N(\mu_2 - \mu_1, \sigma, -\ell_1, \ell_1) < P_N(\mu_2 - \mu_1, \sigma, -\ell_2, \ell_2)$.

Recall that $\mu_1 - \mu_2 = \mu$, and $\ell_2 = \ell$.
Also, let $\sigma' = t\sigma$ (recall that $\sigma' = \frac{\sigma_2}{\sqrt{n_2}} + \frac{\sigma_1}{\sqrt{n_1}}$).
For both of the above cases, we have

$$P_{nA} < P_N(\mu, \sigma, -\ell, \ell) \overset{[3.5]}{=} \frac{1}{\sqrt{\pi}} P_E\left(\frac{-\ell-\mu}{\sqrt{2}\sigma}, \frac{\ell-\mu}{\sqrt{2}\sigma}\right).$$

Case 1: $\sigma = \frac{\sigma_1}{\sqrt{n_1}}$ and $\frac{\sigma_2}{\sqrt{n_2}} \le 2\frac{\sigma_1}{\sqrt{n_1}} \quad \Rightarrow 1 < t \le 3$,

Case 2: $\sigma = \frac{\sigma_2}{\sqrt{n_2}}$ and $\frac{\sigma_2}{\sqrt{n_2}} > 2\frac{\sigma_1}{\sqrt{n_1}} \quad \Rightarrow 1 < t < 2$,

$$P_{nA} < \frac{1}{\sqrt{\pi}} P_E\left(\frac{-\ell-\mu}{\sqrt{2}\sigma}, \frac{\ell-\mu}{\sqrt{2}\sigma}\right) = \frac{1}{\sqrt{\pi}} P_E\left(\frac{-t\ell-t\mu}{\sqrt{2}t\sigma}, \frac{t\ell-t\mu}{\sqrt{2}t\sigma}\right).$$

Recall that

$$P_{nF} = P_N(\mu, t\sigma, -3\ell, 3\ell) = \frac{1}{\sqrt{\pi}} P_E\left(\frac{-3\ell-\mu}{\sqrt{2}t\sigma}, \frac{3\ell-\mu}{\sqrt{2}t\sigma}\right).$$

Let

$$p' = \frac{-t\ell-t\mu}{\sqrt{2}t\sigma}, \quad q' = \frac{t\ell-t\mu}{\sqrt{2}t\sigma}, \quad r' = \frac{-3\ell-\mu}{\sqrt{2}t\sigma}, \quad s' = \frac{3\ell-\mu}{\sqrt{2}t\sigma}, \quad \text{and}$$
$$p = -t\ell - t\mu, \quad q = t\ell - t\mu, \quad r = -3\ell - \mu, \quad s = 3\ell - \mu.$$

Since $\sigma > 0$, from Lemma 3.12 we have $\ell > 0$. And, from Lemma 3.3, we can assume $\mu \geq 0$.

We deduce the following relations:

$$p < 0, \qquad r < 0, \qquad p < q, \qquad r < s,$$
$$q - p = 2t\ell \leq 6\ell = s - r, \quad \text{and}$$
$$q < s \quad \text{since } s - q = (3-t)\ell + (t-1)\mu > 0.$$

- If $p \geq r$, i.e., $\ell \geq \frac{t-1}{3-t}\mu$, then $r \leq p < q < s$.
  By Lemma 3.8, $P_E(p', q') < P_E(r', s')$.
  Therefore, $P_{nA} < P_{nF}$.
- If $p < r$, i.e., $\ell < \frac{t-1}{3-t}\mu$, there are the following two cases:
  - If $q \leq r$, i.e., if $\ell \leq \frac{t-1}{t+3}\mu$, then $p < q \leq r < s$.
    If $s \leq 0$, by Lemma 3.9, $P_E(p', q') < P_E(r', s')$.
    Otherwise, by Lemma 3.10, $P_E(p', q') < P_E(r', s')$.
    $\Rightarrow P_{nA} < P_{nF}$.
  - If $q > r$, i.e., if $\ell > \frac{t-1}{t+3}\mu$, then $p < r < q < s$.
    Since $q-p \leq s-r$, $q-r+r-p \leq s-q+q-r \Rightarrow r-p \leq s-q$.
    If $r \leq 0 \leq q$, $|r| = 3\ell + \mu > 3\ell - 3\mu \geq |q|$,
    by Lemma 3.11, $P_E(p', r') < P_E(q', s')$.
    If $q \leq 0 < s$, by Lemma 3.10, $P_E(p', r') < P_E(q', s')$.
    Otherwise, $s \leq 0$, and by Lemma 3.9, $P_E(p', r') < P_E(q', s')$.
    $P_E(p', r') < P_E(q', s')$
    $\Rightarrow P_E(p', r') + P_E(r', q') < P_E(r', q') + P_E(q' + s')$.
    $\Rightarrow P_E(p', q') < P_E(r', s')$.
    $\Rightarrow P_{nA} < P_{nF}$. □

Theorem 3.3 states that when the Fixed Threshold and the Adaptive tests are almost equally likely to merge two i.i.d. regions, the Adaptive test is less likely to merge two non-i.i.d. regions if *either* of the following two conditions is satisfied. The sizes and standard deviations of the two non-homogeneous regions are denoted by n_1, n_2, and σ_1, σ_2.

- $\frac{\sigma_2}{n_2} \leq 4\frac{\sigma_1}{n_1}$ – The ratio of feature standard deviation to region size of one region must not be greater than four times of that of the other region.

- $n_1 \leq 4n_2$ - The size of one region must not be greater than four times the size of the other region.

Equivalently, we can say that Theorem 3.3 may *not* hold if $\frac{\sigma_2}{4n_2} > \frac{\sigma_1}{n_1}$ *and* $\frac{n_1}{4} > n_2$. Obviously, this is a much more restrictive condition than the ones under which the theorem applies. Therefore, only in a minority of the cases, given $P_{iF} \approx P_{iA}$ we cannot conclude that $P_{nF} > P_{nA}$. And in these cases, it is not certain which of the two tests is superior.

The above theorems deal with homogeneity test results on individual region pairs. We will now relate these results to the entire image which is a system of regions.

Suppose we select the threshold for the Fixed Threshold test according to the region in an image which has the largest feature variation. Then, $P_{iF} \approx 1$ for all regions. However, P_{nF} will also be large and may cause non-homogeneous regions to merge. For the Adaptive test, $P_{iA} \approx 1$ is always true if $\lambda > 0.8$. And in most cases, $P_{nA} < P_{nF}$. Unless the distances between feature means of most adjacent region pairs are larger than the threshold, the segmentation produced by the Fixed Threshold method will be over-merged.

On the other hand, if the threshold is selected according to the region with the smallest feature variation, then except for this region, P_{iF} will be too small for all other regions in the image. For the Adaptive test, $P_{iA} \approx 1$ for all regions if $\lambda < 0.8$. The segmentation produced by the Fixed Threshold method is fragmented.

If an intermediate threshold is selected, although $P_{iF} \approx 1$ for regions with small feature variations, we will have $P_{iF} < P_{iA} \approx 1$ for the rest. And, for those regions with small feature variations, $P_{nA} < P_{nF}$. In the segmentation produced by the Fixed Threshold method, certain regions may be fragmented while the others may be over-merged.

We have proved, by probability theory, that the Adaptive test is more promising than the Fixed Threshold test. More segmentation experiments which support the results of this analysis are presented in Chapters 4 and 5.

Chapter 4

FAS – The Fast Adaptive Segmentation Algorithm

The selection test, in conjunction with the homogeneity test, form the merge criterion. In this chapter, we examine several merge criteria in detail. The criteria are assessed based on the quality of segmentations they produce and the timing requirements of their region growing processes. The Adaptive Fast-merge criterion, which gives the best performance, is adopted in our algorithm. We also introduce the graph model, and present the complete Fast Adaptive Segmentation (FAS) algorithm.

4.1 The Graph Model

An image graph model, similar to that described in [67, 70], is adopted in our study. We represent an image as a disjoint set of regions (vertices), and a set of edges connecting adjacent regions. At the beginning of a region growing process, an image is divided into a number of small primitive regions and a feature value is extracted from each one of them. We view a feature value as a random variable (Section 3.1). During the iterative region merging process, adjacent regions may merge thereby increasing the sizes of regions and reducing the total number of regions. When no more merges are possible, each vertex of the graph represents an image region in the final segmentation.

More formally, an image graph G is a quadruple $< V, E, L, f >$, where
V is a finite set of vertices or regions,
$E \subseteq V \times V$ is a set of undirected edges,
L is a set of labels, and
$f : V \cup E \rightarrow L$ is a function that assigns a label to each edge and vertex of G.

In the following discussion, we denote vertices using p, q, r, s, and use e_{pq} to represent the edge connecting vertices p and q. Since the graph is undirected, e_{pq} is the same as e_{qp}. In our study, the vertex and edge labels are real numbers, $L \subseteq \Re$. If region p has n_p feature values $x_1, x_2, \ldots x_{n_p}$, we label p by its feature mean. An edge e_{pq} is labelled by the absolute difference of the labels of p and q. That is,

$$f(p) = \frac{1}{n_p} \sum_{i=1}^{n_p} x_i \qquad p \in V, \tag{4.1}$$

and

$$f(e_{pq}) = |f(p) - f(q)| \qquad e_{pq} \in E. \tag{4.2}$$

A **merge**, $M(p, q)$, is a graph transformation operation that combines the vertices p, q into a new vertex r. It is also called an edge contraction as the edge e_{pq} is removed. We write

$$G \xrightarrow{M(p,q)} G'. \tag{4.3}$$

The graph G' has vertex r and all other vertices of G except p and q. For any vertex s in G, e_{ps} and e_{qs} are merged into the single edge e_{rs} in G', and e_{pq} is removed. Other edges of G are included in G'. In our model, the label of the new vertex r can be written as

$$f(r) = \frac{n_p f(p) + n_q f(q)}{n_p + n_q}. \tag{4.4}$$

Note that a merge always reduces the total number of regions by one, and the total number of edges by at least one.

A decision function, called the **merge criterion**, determines whether two regions should be merged. The decision function is defined to be *false* if two regions p and q are not adjacent ($e_{pq} \notin E$). A detailed discussion on the merge criterion is given in Section 4.3. The region growing models considered in this study differ in the definitions of their merge criteria.

The region growing process is a sequence of merges on the image graph. Coray *et al.* [16] defined the set of terminally order-independent (TOI) problems which different merge sequences will arrive at the same segmentation. In order to be TOI, merge decisions on all region pairs must not be affected by any subsequent merges. In other words, if we assign a *true* or *false* value to every edge in the initial graph according to the decision function on the corresponding region pair, we can form each region of the final segmentation by simply joining all primitive regions connected by *true* edges. Only a trivial labeling function and merge decision can result in a TOI model. TOI region growing is therefore not very useful.

The region growing models in this study are order-dependent, or equivalently merge operations are non-transitive. That is, different merge sequences will produce different segmentations. Depending on the merging criterion, the number of possible merge sequences differ. Since we do not choose among the merge sequences, and there is no way of knowing in advance if one sequence would produce a better segmentation or require a shorter processing time than another, we define our merge criterion in such a way that segmentation produced from any of the possible merge sequences is satisfactory.

4.2 The Region Growing Process

Although we describe the region growing process as a sequence of merges, certain merges can in fact be carried out simultaneously. As each merge is a local operation involving only a small neighborhood of the graph, at any time during the process, many pairs of vertices may satisfy the merging criterion and can be merged independently at the same time. Since all merges have equal priority, they could be performed simultaneously.

Before we examine the merge criterion in detail, a brief understanding of the region growing process is necessary. The following paragraphs give an overview of the three phases of a region growing process. More detailed issues are given in Chapter 5.

- **Graph Building**
  Graph building refers to the initial setup phase of the region growing process. This phase includes the reading of image data from a file, the computation of feature values for all initial primitive regions, and the building of the graph data structures. In a parallel implementation, it also includes

process initialization, workload partitioning among all participating processors, and the data distribution to the processors.

At the beginning of the process, every primitive region is represented by a vertex. The graph is in the form of a grid with each vertex having four neighbors. As the merging process progresses, the numbers of vertices and edges decrease, and the grid is transformed into an irregular planar graph. Figure 4.3 shows the data structures used in various phases of the region growing process.

- **The Merge Phase**
  Region merging is an iterative process; every vertex is regularly checked for a possible merge with one of its neighbors. The repetitions are necessary because every merge changes the topology of the graph and the labels on certain vertices and edges. For example, a merge attempt that failed on vertices p and q may succeed after one of p or q's neighbor has been merged.

  Every region in the image graph is given an equal chance to grow in the merge process. Hence, during an iteration, every vertex is examined exactly once for a possible merge. Even if a merge succeeded in merging p and q into r, the new vertex r will not be examined again in the same iteration. However, r may participate in a later merge if one of its neighbors is examined and that resulted in a merge with r.

  A vertex that is connected to several vertices may be permitted to merge with none, one, or several of its neighbors using a given homogeneity test. In practice, it is too expensive and also unnecessary to perform the homogeneity test on all neighbors of a vertex. It is more efficient to maintain the list of edges of a vertex in ascending order of their labels, and only examine a vertex with its closest neighbor for a possible merge.

  The ordering of merge attempts on vertices depends on the implementation and computational model. For example, in a SIMD implementation, every vertex in the graph is examined at the same instant. On a sequential system, all vertices are examined in the same linear order in each iteration. In a MIMD or a distributed implementation, vertices assigned to the same processor are examined linearly while merges are carried out concurrently on all the processors.

  Pseudo code for the sequential FAS merge phase is given in Figure 4.4.

- **Process Termination**
  The merge phase terminates when no more merges are possible on the

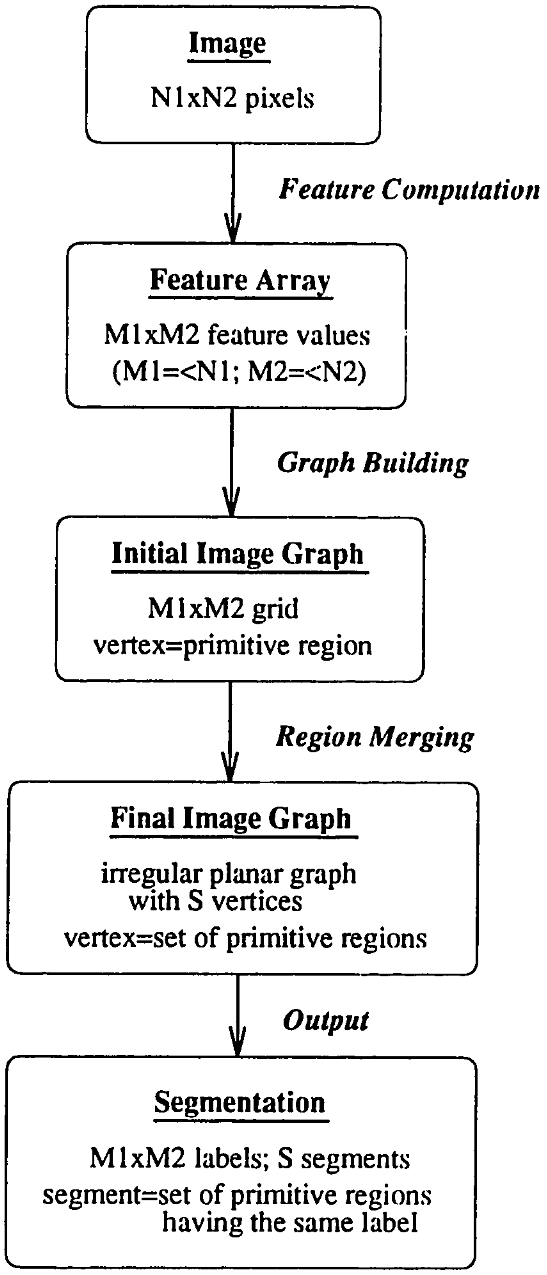


Figure 4.3: Data structure conversions in region growing

V	list of all vertices (regions) in the image graph
E_p	list of all edges, sorted by the edge label, connecting to vertex p
A_p	list of all primitive regions constituting region p

```
merge ← $true$
repeat until merge = $false$  {
    merge ← $false$
    for each $p \in V$  {
        select vertex $q$ where $e_{pq}$ is the first member of $E_p$
        if $p$ and $q$ pass the adaptive test  {
            merge ← $true$
            compute feature mean of region $p \bigcup q$
            $A_p \leftarrow A_p \bigcup A_q$
            $E_{new} \leftarrow null$
            delete $e_{pq}$
            for each $e_{ps} \in E_p$  {
                recompute label of $e_{ps}$
                add $e_{ps}$ to $E_{new}$ in sorted order
            }
            for each $e_{qs} \in E_q$  {
                if $e_{qs} \notin E_{new}$  {
                    recompute label of $e_{qs}$
                    add $e_{qs}$ to $E_{new}$ in sorted order
                }
            }
            delete $q$
            $E_p \leftarrow E_{new}$
        }
    }
}
```

Figure 4.4: The sequential FAS merge phase

graph. At this stage, each vertex in the graph represents a region in the final segmentation. Segmentation is presented as a two-dimensional array of labels, with all entries having the same label constitute a region.

We now take a rough look at the time complexities of the above described region growing process. Let us assume that the cost of inspecting the merge criterion is t_c, and the cost of merging any two regions, t_m, is approximately a constant. (By assuming that region feature distributions are normal, hence using sample variances to estimate the Adaptive ranges, t_m is a constant.) Let M and N be the total numbers of regions at the beginning and the end of the region growing process on a given image, respectively. Using the binary merge paradigm, it is obvious that every merge reduces the total number of regions by one. Therefore, there are $M - N$ merges in the segmentation process.

The best case timing occurs when every merge attempt results in a merge. That is, the minimum cost of the process is $(M - N)(t_c + t_m)$ and the lower bound time complexity is $O(M)$. The worst case scenario happens when every iteration results in only one merge. In this case, the total cost of the process is $(M-N)t_m+t_c\sum_{i=N-1}^{M-1} i$. Thus, the upper bound time complexity of the algorithm is $O(M^2)$. Without knowledge about the distribution of the process convergent rate, a more realistic average cost of the algorithm cannot be expressed in a closed form.

4.3 The Merge Criterion

The main focus of our study is on the merge criterion - the way it affects the quality of segmentation and the processing time. Generally, a merge criterion consists of two parts: a homogeneity test and a selection policy. Two adjacent regions need to pass the homogeneity test in order to be merged. When several neighboring regions pass the homogeneity test to merge with a given region, a selection policy is applied to choose one region among the several to participate in the merge.

In Chapter 3, we have presented the Adaptive homogeneity test, and compared this test with the existing Fixed Threshold test. In this section, we complete the discussion on merge criterion by introducing our Fast-merge selection policy, and evaluate the performance of several homogeneity test and selection policy combinations. The relations among the various terminologies are depicted in Figure 4.5.

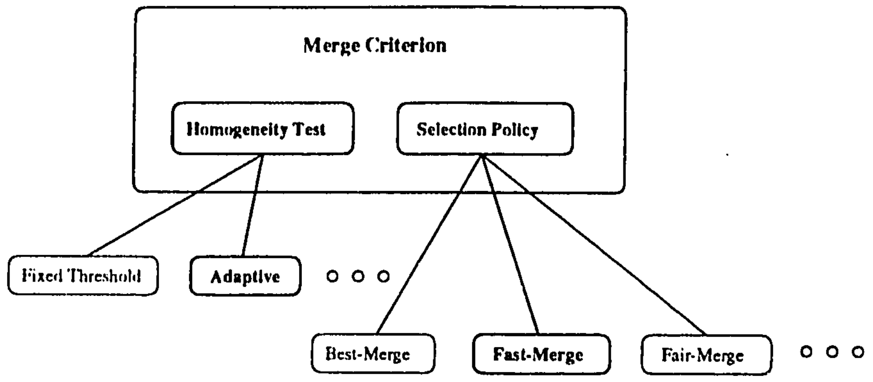


Figure 4.5: Composition of a merge criterion

We consider the following three selection policies:

- **Best-Merge**
  Initial studies on using the merge selection to improve segmentation are presented in [63] and [67]. These parallel region growing algorithms adopt a "Best-merge" paradigm that requires the edge connecting two merging regions to be minimum with respect to *both* regions. Using the earlier notations, we merge regions p and q under the Best-merge paradigm if $f(e_{pq}) = \overset{Min}{i} \{f(e_{pi})\} = \overset{Min}{j} \{f(e_{jq})\}$. In other words, a region only merges with the neighboring region that best satisfies the homogeneity requirement, and the merge choice must be mutual for both regions. The strength of the Best-merge approach is that different parallel runs on the same image would produce the same segmentation due to the ordering imposed on the merge sequence.

- **Fast-Merge**
  Regions in a Best-merge segmentation are very homogeneous because the selection policy minimizes the increase in feature range with each merge. However, on images which merge sequences do not allow for much parallelism, only marginal speedup can be achieved on parallel Best-merge processes. The objective of Fast-merge is to remove this deficiency of the Best-merge paradigm.

  We merge regions p and q using the Fast-merge policy if $f(e_{pq}) = \overset{Min}{i} \{f(e_{pi})\}$ or $f(e_{pq}) = \overset{Min}{j} \{f(e_{jq})\}$. That is, the value of the edge connecting the two merging regions has to be a minimum with respect to *either* (not necessarily both) of the regions. The most important improvement of Fast-merge over that of the existing Best-merge is the reduction in processing time. Also, as shown in our experiments, Fast-merge helps to improve region mergeability.

- **Fair-Merge**
  In Fair-merge, all regions qualified to merge with a given region are considered equally suitable, and one is selected *randomly* for the merge. Fair-merge is included in our discussion for completeness, as it and Best-merge represent two extreme selection policies – Best-merge chooses the locally most similar ("best") region pair to merge while Fair-merge does not give preference to any merge candidates.

To evaluate the different merge criteria, we designed an experiment to investigate the following four aspects of region growing. The first two aspects measure segmentation quality, while the latter two aspects concern processing time.

1. **Region Mergeability**
   Region mergeability refers to the degree of which regions that should be merged are merged. In our experiment, we measure region mergeability using the number of regions in the final segmentation. When each object in a scene is presented by many instead of one or a few regions in a segmentation, we say that the regions are under-merged, and describe the segmentation as *fragmented.* Besides noise and irregularities present in the image data, merge criteria also affect region mergeability.

2. **Boundary Accuracy**
   The other most important aspect of segmentation quality is the accuracy of region boundaries. In region growing, it is important that regions that should not be merged remain as separate regions throughout the process. In our experiment where the exact locations of region boundaries are known, boundary accuracy is measured in the number of boundary units found in these expected locations.

3. **Merge Rejections**
   In binary merge region growing, every merge reduces the number of regions by one. Assuming that the cost for merging two regions is a constant and we begin with the same number of primitive regions, different region growing processes on the same image will require similar numbers of merges. As a result, to compare the processing speed of different region growing algorithms, the number of successful merges is not a key measure. Instead, the number of merge rejections should be used.

If we process the same image using different merge criteria, the differences in merge rejection costs are often substantial. Among the region growing algorithms in our study, the number of merge rejections is the only major factor that accounts for the timing performance differences of the algorithms.

4. **Process Iterations**
Ideally, in parallel region growing, merges in the same iteration are performed at the same time. Therefore, the number of iterations required in the region growing process is an indicator on the length of its parallel processing time. The algorithm that requires fewer iterations will have a shorter parallel processing time.

In our experiment, we use simple 50 × 50 images. Each image has two equal sized regions that share a simple straight 50-unit boundary. We randomly generate data for the two regions from a pair of normal distributions having different means and variances. That is, all values in a 25 × 50 region are normally, independently, and identically distributed. For each pair of normal distributions, we generate 100 test images. Three sets of distribution pairs with different distances between the two means are used. We named the three image sets A, B and C. The distance between the two means are largest in A and smallest in C. Therefore, the segmentations in C are expected to have the least accurate boundaries. In the Fixed Threshold segmentations, different thresholds are tested and results of the best segmentations are reported. All six combinations of the two homogeneity tests and three selection policies are examined. Detailed results on the four aspects of investigation are given in Tables 4.2 through 4.5. A summary is given in Table 4.6.

Table 4.2 gives the percentages of segmentations having various numbers of result regions. Best-merge produces very fragmented segmentations. In all cases, over 80% of best-merge segmentations have over 100 regions. Using Fair-merge and Fast-merge, the majority of Fixed Threshold segmentations have 11 to 50 regions, while almost all Adaptive segmentations have fewer than 5 regions. Mergeabilities are consistent for sets A, B, and C.

In Table 4.3, we present the percentages of segmentation with boundary accuracies over certain scores. The score for a perfect boundary is 50. Since features of the two regions are most distinctive in case A, segmentations of set A have the highest boundary accuracies. Although Best-merge segmentations have high scores, the apparently good results are not significant due to the many within-segment false boundaries present in the segmentations. The Adaptive Fast-merge criterion produces the best boundaries, while Adaptive Fair-merge

Table 4.2: Mergeability

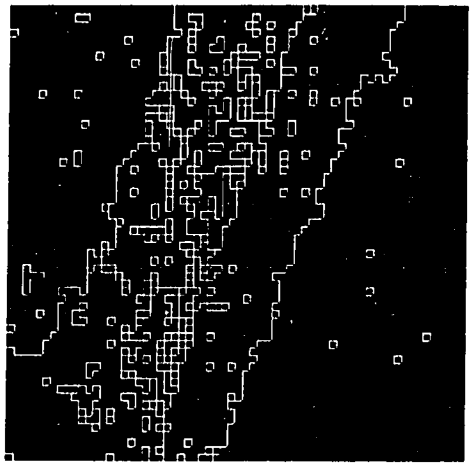
Number of Regions →		Fixed Threshold (%)					Adaptive (%)					
		< 10	-50	-100	-150	> 150	< 5	-10	-50	-100	-150	> 150
Fair	A	0	76	24	0	0	100	0	0	0	0	0
	B	0	79	21	0	0	100	0	0	0	0	0
	C	0	76	24	0	0	100	0	0	0	0	0
Fast	A	0	96	4	0	0	86	14	0	0	0	0
	B	0	96	4	0	0	86	14	0	0	0	0
	C	0	97	3	0	0	90	10	0	0	0	0
Best	A	0	0	14	78	8	0	3	11	7	31	48
	B	0	0	16	77	7	0	1	7	10	33	49
	C	0	0	13	82	50	0	0	9	8	34	49

gives the poorest boundaries.

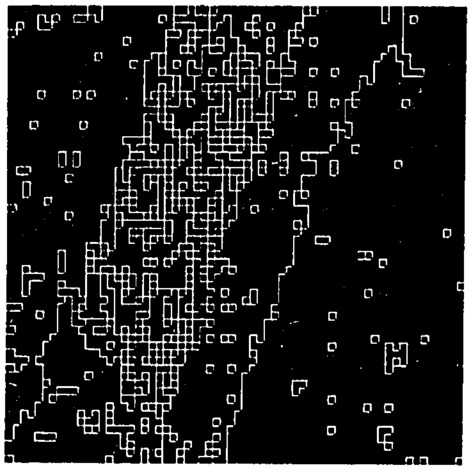
There are two types of merge rejections: Type I rejections happen when the homogeneity test is failed, and Type II rejections are caused by the selection policy. In our Fast-merge and Best-merge implementations, the list of edges of a region vertex is maintained in ascending order of the edge labels. Therefore, when a vertex and its first neighbor are tested for a possible merge, the Fast-merge criterion is always satisfied whereas the Best-merge criterion may fail if this neighbor is closer to a third vertex. In Table 4.4, compared to the other two selection policies, Best-merge also results in a much higher rejection rate in the homogeneity test. This will be elaborated on later in this section. The edge lists in Fair-merge are not sorted. As a result, the Fair-merge algorithm may not be testing a vertex with its neighbor that is most likely to pass the homogeneity test. This explains the higher rejection rates compared to Fast-merge.

As shown in Table 4.5, Best-merge requires about 4 times as many iterations as the other two selection policies. If Best-merge had provided better mergeability, even more iterations would have been required to complete the segmentation processes. Similarly, Fixed Threshold Fast-merge requires fewer iterations than Adaptive Fast-merge because segmentations using the former criterion are more fragmented. Naturally, the most relaxed Fair-merge policy requires the fewest iterations.

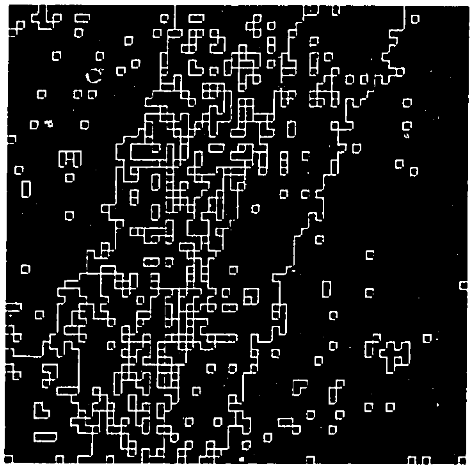
In Figures 4.6 and 4.7, we show the six Fixed Threshold and Adaptive segmentations of a 4-region textured image M4. The test image M4 is given in



(a) Fast-merge segmentation

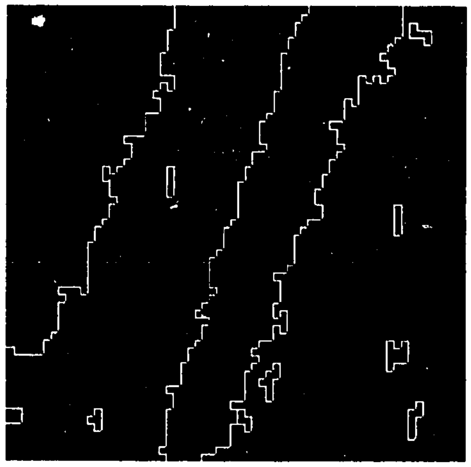


(b) Fair-merge segmentation

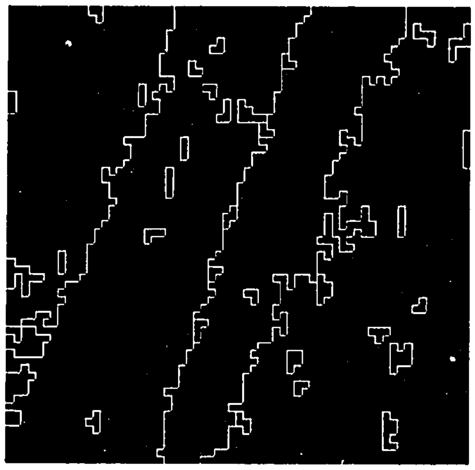


(c) Best-merge segmentation

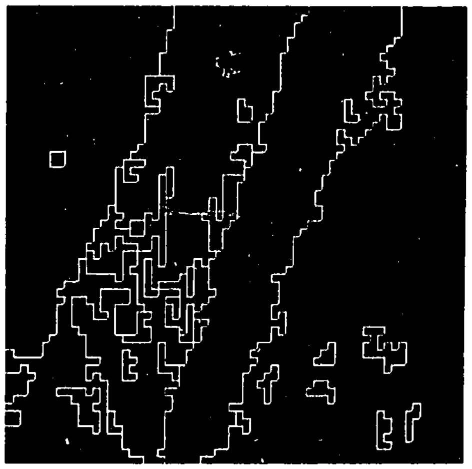
Figure 4.6: Fixed Threshold segmentations on M4



(a) Fast-merge segmentation



(b) Fair-merge segmentation



(c) Best-merge segmentation

Figure 4.7: Adaptive segmentations on M4

Table 4.3: Boundary accuracy

Scores (Total=50)	→	Fixed Threshold (%) > 40	> 42	> 44	> 46	> 48	Adaptive (%) > 40	> 42	> 44	> 46	> 48
Fair	A	100	84	74	47	14	78	61	36	17	2
	B	88	70	49	19	4	43	24	8	4	1
	C	59	24	14	5	0	6	4	1	0	0
Fast	A	100	94	87	57	20	100	100	97	82	34
	B	88	70	47	15	1	99	94	74	48	7
	C	57	34	13	3	0	79	69	35	5	0
Best	A	100	100	96	81	32	97	97	97	83	45
	B	92	82	56	29	3	96	93	90	44	14
	C	62	33	15	3	1	93	76	29	8	0

Figure 4.13. The best thresholds are used to obtain the segmentations in Figure 4.6. Any larger thresholds will result in the two regions on the right being merged into one.

Table 4.6 provides a performance summary of the different merge criteria. The "+" symbol indicates that a merge criterion performs well in a specified aspect; the "–" symbol indicates that the performance is unacceptable. When the performance is moderate, the "o" label is used. The Adaptive Fast-Merge combination has shown to be consistently superior in the four aspects of investigations: region mergeability, boundary accuracy, savings on merge rejections and process iterations. Fixed Threshold Fast-merge criterion also provides reasonable results although segmentations are more fragmented than using the Adaptive test. We will now further examine the causes of several undesirable effects on the segmentations.

In the following discussion, a segment refers to a connected area of an image which corresponds to an object in a scene. In other words, a segment is a region in the correct, final segmentation of an image. And all feature values in a segment are identically distributed.

The high merge rejections and process iterations of Best-merge can be explained by examining the probabilities of merging two regions using the different selection policies. Let us consider a set of regions belonging to the same image segment. Let p be the probability of a given region R_i passing a certain homogeneity test with one of its neighbors R_j. Since the merge criterion is composed

Table 4.4: Average number of merge rejections

		Fixed Threshold			Adaptive		
		Type I	Type II	Total	Type I	Type II	Total
Fair	A	1032	–	1032	499	–	499
	B	1023	–	1023	498	–	498
	C	1028	–	1028	500	–	500
Fast	A	489	–	489	353	–	353
	B	482	–	482	358	–	358
	C	480	–	480	361	–	361
Best	A	5784	7323	13107	3560	6056	9617
	B	5731	7290	13021	3710	6280	9989
	C	5700	7257	12957	2646	5947	9593

Type I: Failed homogeneity test Type II: Failed selection test

of two tests, the probability of merging R_i and R_j is

$$\begin{aligned} Pr & \text{ (merge } R_i \text{ with } R_j) \\ = & \; Pr(\text{homogeneity test passed} \wedge \text{selection test passed}) \\ = & \; p \times Pr(\text{selection test passed/homogeneity test passed}). \end{aligned}$$

Let d_1 and d_2 be the numbers of neighboring regions which satisfy the homogeneity test with R_i and R_j, respectively. Then,

$$\begin{aligned} Pr(\text{Best-merge passed}) &= \frac{1}{d_1} \times \frac{1}{d_2}, \\ Pr(\text{Fast-merge passed}) &= \frac{1}{d_1} + \frac{1}{d_2} - \frac{1}{d_1 d_2}, \\ Pr(\text{Fair-merge passed}) &= 1. \end{aligned}$$

Suppose R_i and R_j satisfy the homogeneity test, that is $d_1, d_2 \geq 1$. We have

$$\begin{aligned} Pr & \text{ (merge } R_i, R_j \text{ using Best-merge}) = \frac{p}{d_1 d_2} \\ \leq & \; Pr(\text{merge } R_i, R_j \text{ using Fast-merge}) = \frac{p(d_1 + d_2 - 1)}{d_1 d_2} \\ \leq & \; Pr(\text{merge } R_i, R_j \text{ using Fair-merge}) = p. \end{aligned}$$

Table 4.5: Average number of iterations

		Fixed Threshold	Adaptive
Fair	A	8.4	8.0
	B	8.4	8.0
	C	8.5	8.0
Fast	A	9.1	12.5
	B	9.1	12.5
	C	9.1	12.6
Best	A	45.6	44.4
	B	45.6	45.9
	C	45.7	43.3

In our Best-merge and Fast-merge implementations, the list of edges connected to a region node is maintained in sorted order. That is, we always examine a given region and its (first) closest neighbor. In this case,

$$\begin{aligned} Pr \quad & (\text{merge } R_i,\ R_j \text{ using Best-merge}) = \frac{p}{d_2} \\ \leq \quad & Pr(\text{merge } R_i,\ R_j \text{ using Fast-merge}) = p. \end{aligned}$$

The above expression implies that if an average region has four neighbors, then Best-merge is 4 times less likely to merge two regions than Fast-merge. Suppose there are 10,000 merge attempts during an iteration of the region growing process, and assume that the probability of passing the homogeneity test remains at 0.7. Using Best-merge there would be about 1750 merges and 8250 rejections, while Fast-merge may result in 7000 merges and 3000 rejections. Since the total number of merges required to arrive at a certain segmentation is the same, Best-merge will need many more iterations than Fast-merge to complete the segmentation.

In a parallel region growing process, compared to Best-merge, Fast-merge will allow many more merges to be carried out at the same time, encounter much fewer merge rejections and require a much shorter time to produce a segmentation.

We have discussed the probability of passing the selection test using Best-merge. In practice, Best-merge also reduces the probability of passing the ho-

Table 4.6: Summary of merge criteria performance

	Fixed Threshold		Adaptive	
Fair	o	Fragmented	+	Merge well
	o	Fair boundary	-	Bad boundary
	o	Fairly low merge rejections	+	Low merge rejections
	+	Few iterations	+	Few iterations
Fast	o	Fragmented	+	Merge well
	o	Fair boundary	+	Good boundary
	+	Low merge rejections	+	Low merge rejections
	+	Few iterations	+	Few iterations
Best	-	Very fragmented	-	Very fragmented
	+	Good boundary	+	Good boundary
	-	High merge rejections	-	High merge rejections
	-	Many iterations	-	Many iterations

+ desirable   - undesirable   o tolerable

mogeneity test, and this causes fragmented segmentations. To explain the effect of Best-merge on the homogeneity test, we will first show that minimizing the within-region variation is equivalent to maximizing the between-region variation. The following scatter matrix analysis can be easily generalized to apply on any d-dimensional feature vector. A similar analysis on partitional clustering is given in [34].

During the region growing process, suppose the feature values in K connected regions are identically distributed and belong to the same image segment. Let $\{x_i | 1 \leq i \leq n_k\}$ denote the set of feature values of a region with size n_k. Let the feature mean of the kth region be $m_k = (1/n_k)\sum_j^{n_k} x_j$. And the grand feature mean be $m = (1/n)\sum_{k=1}^{K} n_k m_k$, where $n = \sum_{k=1}^{K} n_k$.

Then the scatter, S, for the segment is defined as

$$S = \sum_{k=1}^{K}\sum_{j=1}^{n_k}(x_j - m)^2. \qquad (4.5)$$

Note that S is a constant regardless of the final segmentation. We define the scatter for the kth region as

$$\mathcal{S}^{(k)} = \sum_{j=1}^{n_k}(x_j - m_k)^2. \tag{4.6}$$

The within-region scatter, $\mathcal{S}_W$, is defined as the sum of the region scatters,

$$\mathcal{S}_W = \sum_{k=1}^{K}\mathcal{S}^{(k)}. \tag{4.7}$$

Finally, the between-region scatter, $\mathcal{S}_B$, is defined as the scatter for the group means,

$$\mathcal{S}_B = \sum_{k=1}^{K}\sum_{j=1}^{n_k}(m_k - m)^2. \tag{4.8}$$

Writing $x_j - m = (m_k - m) + (x_j - m_k)$, we have

$$\begin{aligned}
\mathcal{S} &= \sum_{k=1}^{K}\sum_{j=1}^{n_k}\{(m_k - m) + (x_j - m_k)\}^2 \\
&= \sum_{k=1}^{K}\sum_{j=1}^{n_k}(m_k - m)^2 + \sum_{k=1}^{K}\sum_{j=1}^{n_k}(x_j - m_k)^2 - 2\sum_{k=1}^{K}\sum_{j=1}^{n_k}(m_k - m)(x_j - m_k) \\
&= \mathcal{S}_B + \mathcal{S}_W - 2\sum_{k=1}^{K}\sum_{j=1}^{n_k}(m_k x_j - m x_j - m_k^2 + m m_k) \\
&= \mathcal{S}_B + \mathcal{S}_W - 2(\sum_{k=1}^{K} n_k m_k^2 - m\sum_{k=1}^{K} n_k m_k - \sum_{k=1}^{K} n_k m_k^2 + mnm) \\
&= \mathcal{S}_B + \mathcal{S}_W - 2(-mnm + mnm) \\
&= \mathcal{S}_B + \mathcal{S}_W.
\end{aligned}$$

Therefore, the total scatter of an image segment is divided into the within-region scatter and the between-region scatter. Since $\mathcal{S}$ is a constant for a given segment, minimizing $\mathcal{S}_W$ is the same as maximizing $\mathcal{S}_B$. Intuitively, a small $\mathcal{S}_W$ indicates that each region in the set is compact and has a small within-region variation, and a large $\mathcal{S}_B$ implies that the mean values of the regions form a loosely bound set. The more compact are the individual regions, the more dispersed are the region means.

Comparing the three selection policies, it is obvious that Best-merge will give the most compact regions and hence the most dispersed region means. Our homogeneity tests based merging decision on the difference between two region means; the greater the difference, the less likely will the regions be allowed to merge. As a result, compared to the other selection policies, a higher failure rate on the homogeneity test is expected on Best-merge. When regions that should be merged stop growing at an early stage, a fragmented segment will result.

Let us generalize the above scatter analysis from a single segment to the entire image. The pulling apart of feature means observed in Best-merge indiscriminately applies within segments as well as among segments. Fragmented segments are no doubt undesirable, but the separation of feature means belonging to different segments is often desirable, as it helps to resolve boundary ambiguities and produce accurate boundaries.

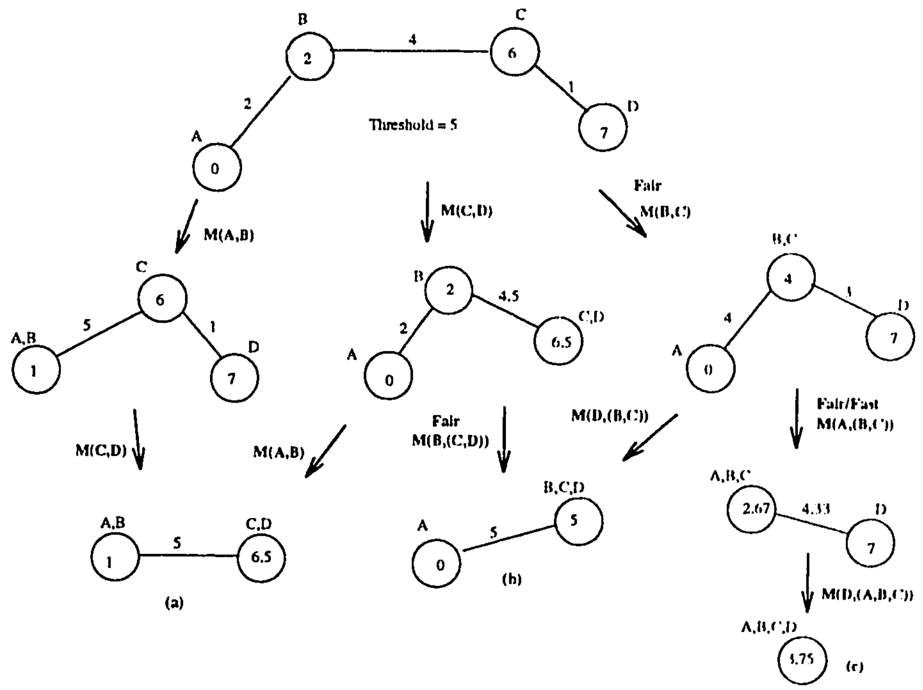


Figure 4.8: Selection policies and segmentations

We illustrate the effect of selection policies on boundary accuracies using a simple example given in Figure 4.8. A, B, C and D represent four regions. The feature mean of a region is written on the node. The value on an edge gives the difference between the two connected region means. For convenience, we assume the four regions to be the same size, and use the Fixed Threshold test with the threshold being 5. Each transformation arrow is labelled with the merge $M(p,q)$, where p being the initiatory region. When a merge is only possible under certain policies, names of these policies are specified next to the transformation arrow. Otherwise, a merge is achievable with either of the three selection policies. To allow regions to have an equal chance to grow, a region never initiates consecutive merges. The three possible segmentations are labelled as (a), (b) and (c) in the figure. Best-merge and Fast-merge will produce (a) only. Depending on the order in which regions are tested, Fair-merge may produce any of the three segmentations.

Suppose there should be a boundary between regions B and C, although the difference between the two feature means is too small to be rejected by the homogeneity test. Also, regions A and B belong to one segment, while C and D belong to a different segment. In this situation, Best-merge and Fast-merge will still produce the correct segmentation given in 4.8(a), but Fair-merge is likely to produce the less preferred segmentations given in (b) and (c). This explains the poor boundaries by Fair-merge presented in Table 4.3.

The severe drawbacks of Best-merge are the fragmented segmentation it produces and its slow processing speed. In addition, good parallel timing is not achievable using Best-merge. Fair-merge does not have these disadvantages, but it does not always produce good boundaries. Fast-merge combines the merits of both Best-merge and Fair-merge to overcome their respective limitations. The results of a more comprehensive experiment on Adaptive Fast-merge region growing is given in Chapter 5.

4.4 More Segmentation Results

Segmentation experiments were performed on textured and grayscale images to assess the performance of the FAS algorithm, and to substantiate the results of our analysis. Some of these results have been presented in the earlier chapters. In this section, several more tests are presented. A test is designed to appraise a specific attribute of a segmentation algorithm, such as its sensitivity to small regions and robustness to noise. In these tests, segmentations produced by the existing Best-merge Fixed Threshold segmentation algorithm (BFS) [67] are compared to that of our Fast-merge Adaptive segmentation algorithm (FAS).

The textured images used in our experiments were created using images of natural textures given in the Brodatz album [4]. The co-occurrence features [29, 26], demonstrated as being effective measures for natural textures in a recent study by Ohanian and Dubes [46], are used in our experiments. Definitions of these co-occurrence features are given in Appendix A. Other texture measures, for examples Gabor filters, Law's energy masks, fractal measures, can also be applied in our algorithm.

4.4.1 Sensitivity to Small Regions

Image regions that contain useful information for a vision task may be of any size. Therefore, it is important for a good segmentation algorithm to be able to produce accurate segmentations even on very small regions.

The two test images M2 and M3 used in this experiment are given in Figures 4.10 and 4.11. Figure 4.9 gives the Brodatz texture codes of the regions in M2 and M3. We will henceforth refer to the regions by their texture codes. Each of these two images has four textured regions, of which one is a small region. The areas of regions D21 in M2 and M3 are about 1% and 0.5% of the total image size. The co-occurrence contrast feature is used in the segmentation of these images. The primitive regions are 8×8 pixels.

The segmentations of M2 and M3 by the FAS algorithm are given in Figure 4.12(a) and (b). The small regions D21 are evident in the segmentations. Due to the coarse texture granularity of D05, several small regions also present in the segmentations of that region. As expected from the analysis in Chapter 3, the M2 and M3 segmentations produced by the BFS algorithm are fragmented. Region D21 in the M2 segmentation is reasonably evident (Figure 4.12(c)). However, the smaller D21 region in the M3 segmentation appears to have been overly merged with neighboring pixels. (Figure 4.12(d)).

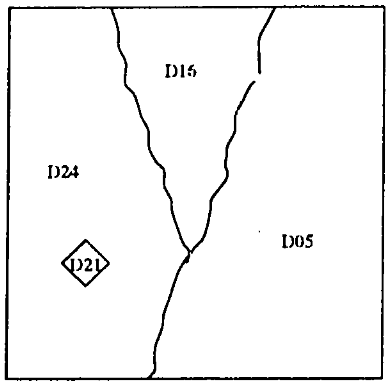


Figure 4.9: Texture codes of M2 and M3

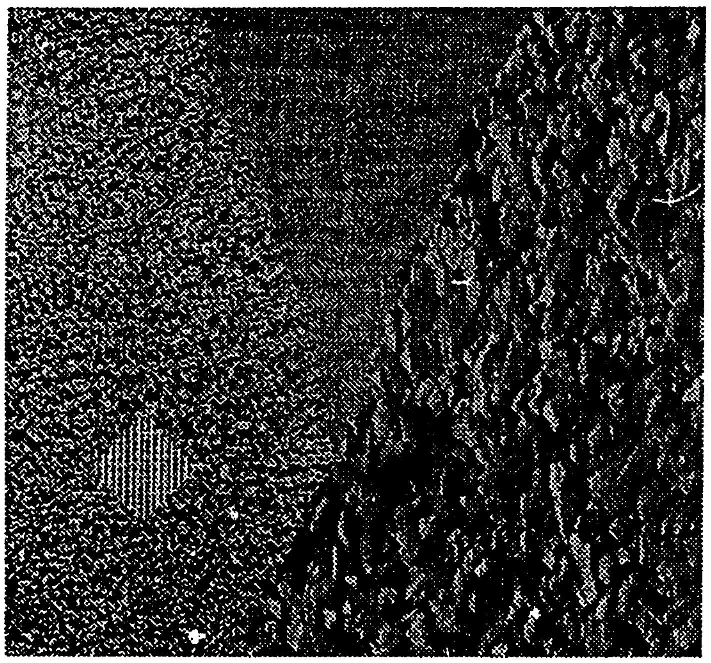


Figure 4.10: Image M2

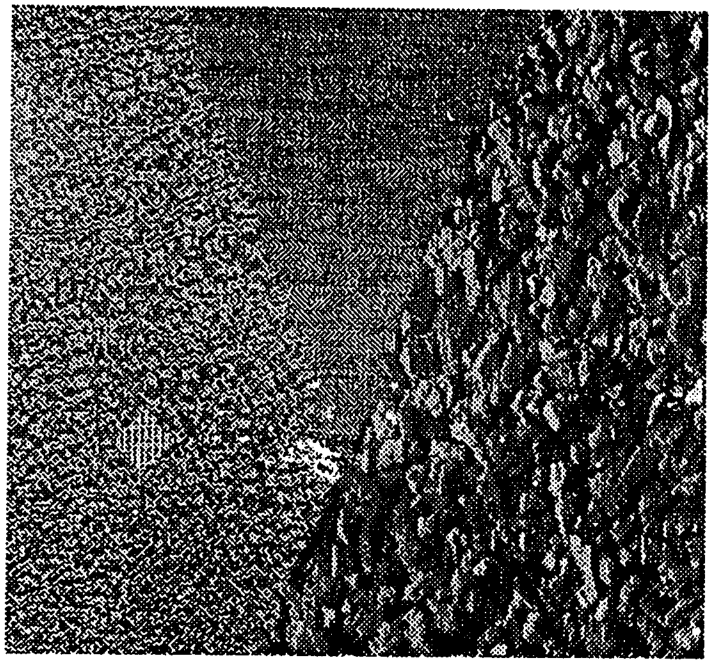


Figure 4.11: Image M3

4.4.2 Robustness on Noisy Images

Noise may be introduced into an image at various stages of a vision process. Although noise from known sources may be removable by suitable image restoration techniques, such procedures take time and it is desirable for a segmentation algorithm to be able to produce good segmentations regardless of the presence of noise. In this test, we evaluate the performance of the FAS algorithm on images corrupted by a small degree of random noise.

The test images M4, M5, and M6, shown in Figures 4.13, 4.15, and 4.16, are four-region textured images. Images M5 and M6 are produced from M4 with random noise added with increasing intensities in the top-down and left-right directions, respectively. We use the correlation feature on 16×16 pixel blocks in this experiment.

Both the FAS and BFS algorithms work well on the original image M4. The FAS and BFS segmentations of M4 are given in Figure 4.14. Figures 4.17 and 4.18 show the segmentations of M5 and M6, respectively. On the noisy images M5 and M6, we can see that the segmentations from the BFS algorithm degrade more severely with noise intensity compared to that from the FAS algorithm. On the FAS segmentations, only the image sections with high noise intensities suffer from poor region boundaries. For example, the lower section in Figure 4.17(a) and the right section of Figure 4.18(a). Other parts of these segmentations are close to that of the original image given in Figure 4.14(a). On the other hand, the entire BFS segmentations on M5 and M6 (Figures 4.17(b) and 4.18(b)) are affected by noise, even for the parts where the noise intensities are low.

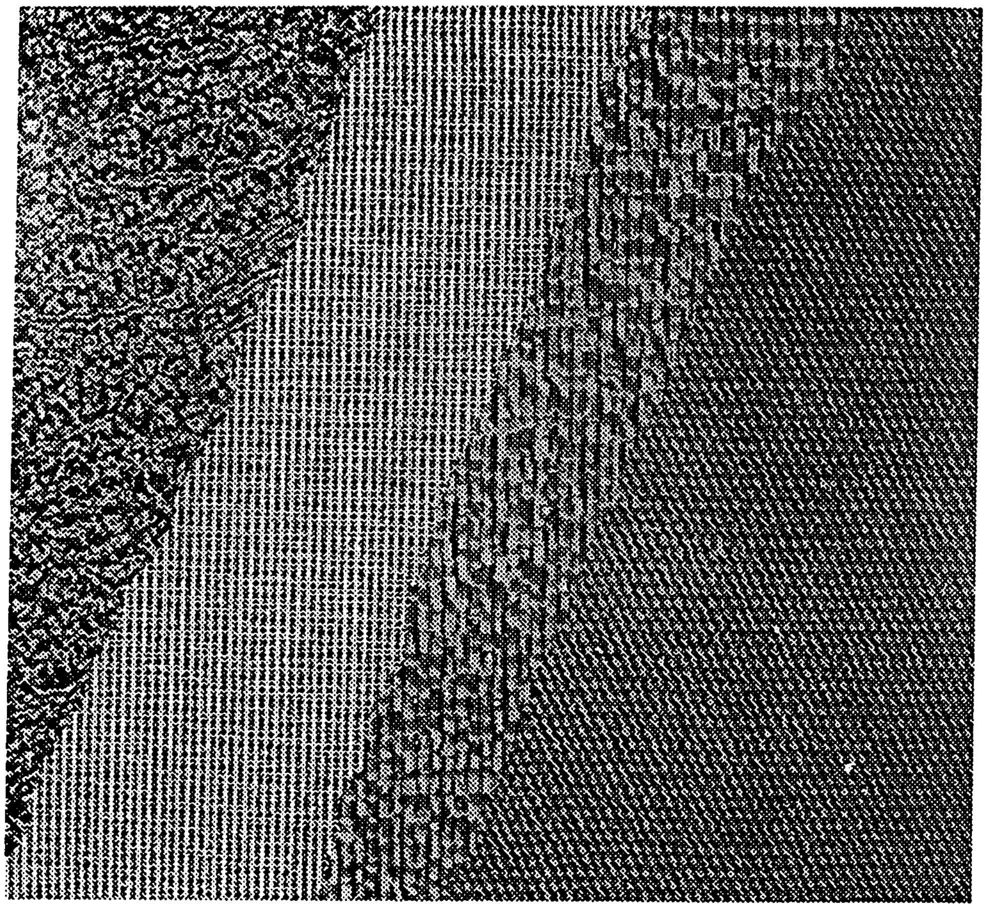
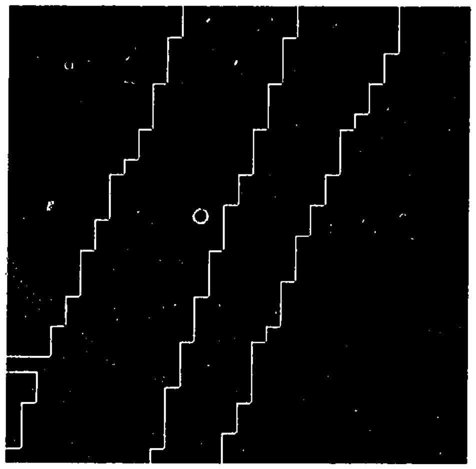
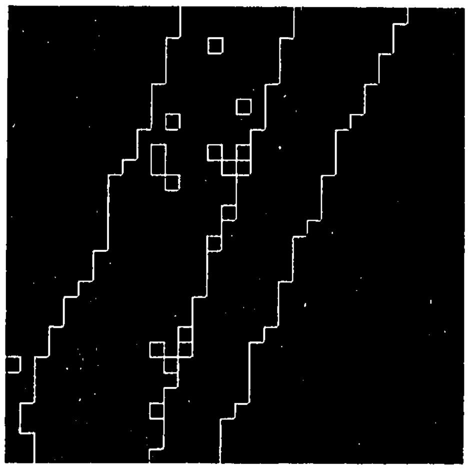


Figure 4.13: Image M4 (no noise added)



(a) FAS segmentation of M4



(b) BFS segmentation of M4

Figure 4.14: Segmentations of M4

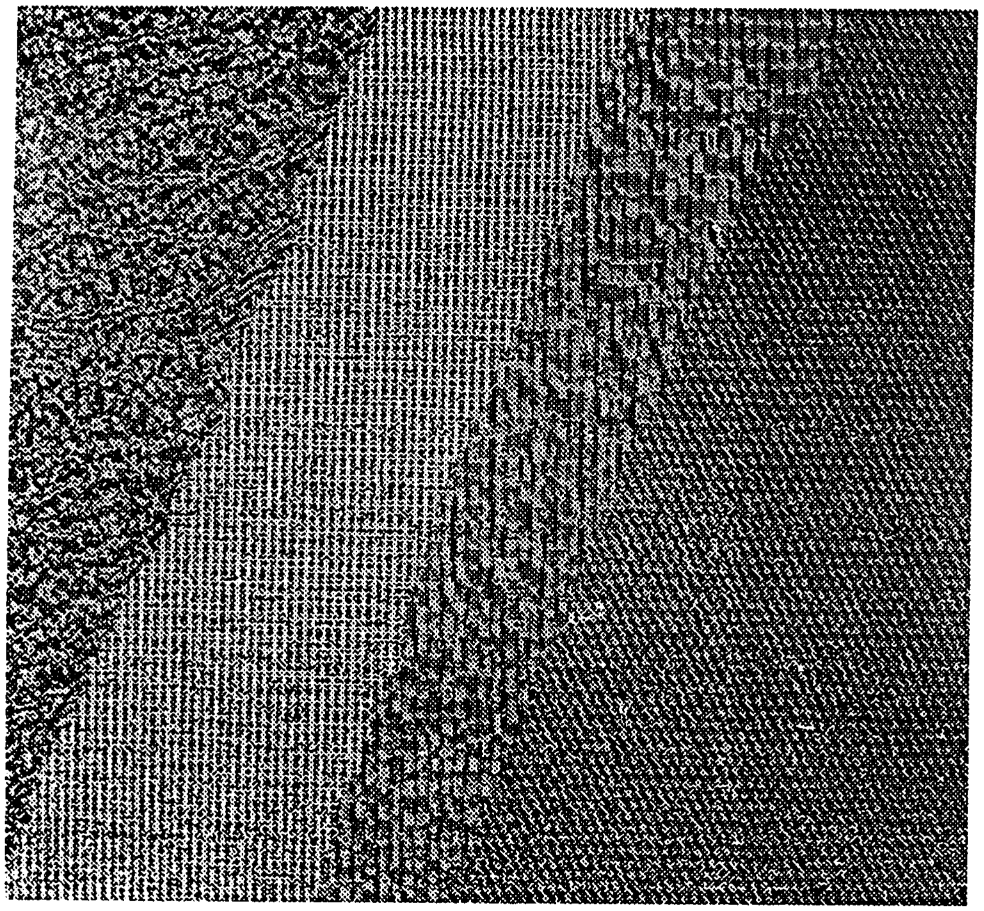


Figure 4.15: Image M5, increased noise level top-down

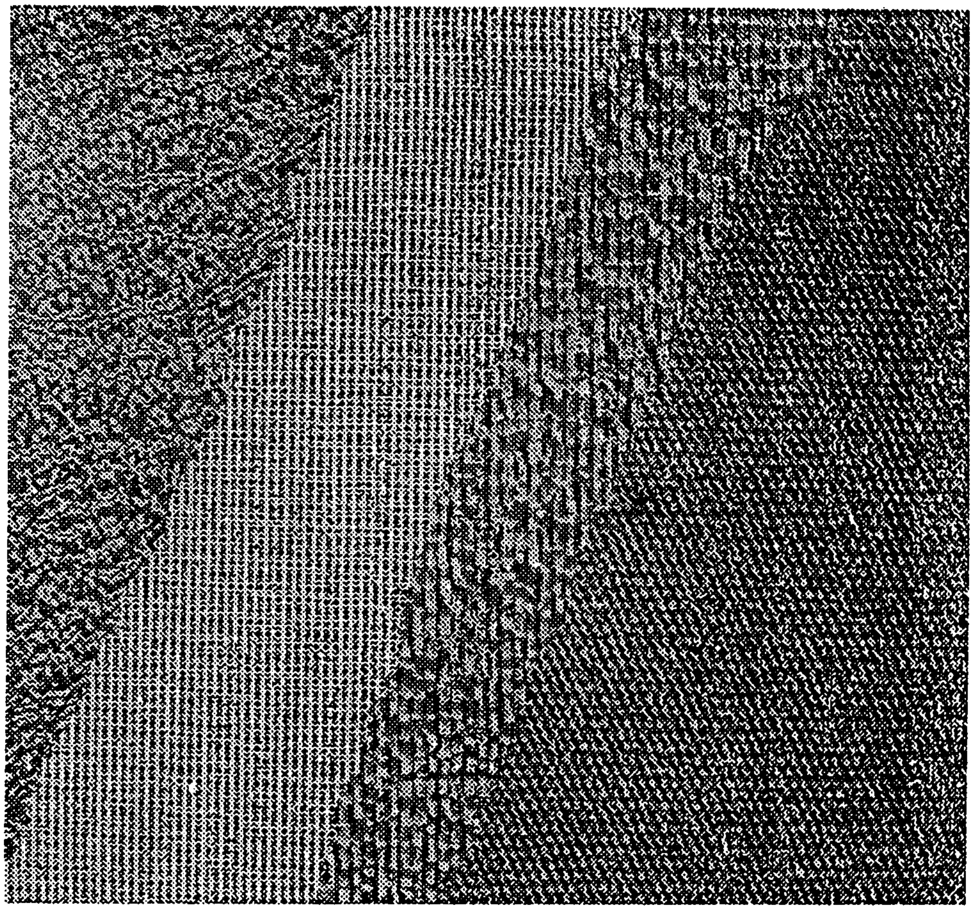
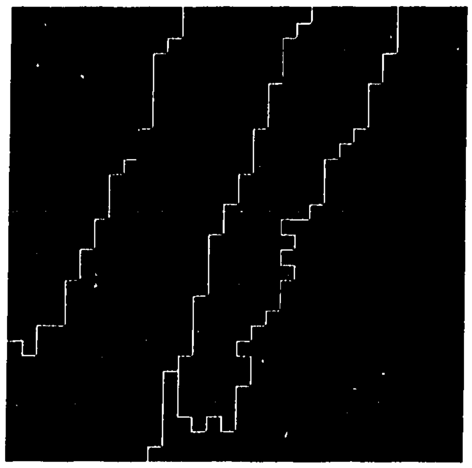
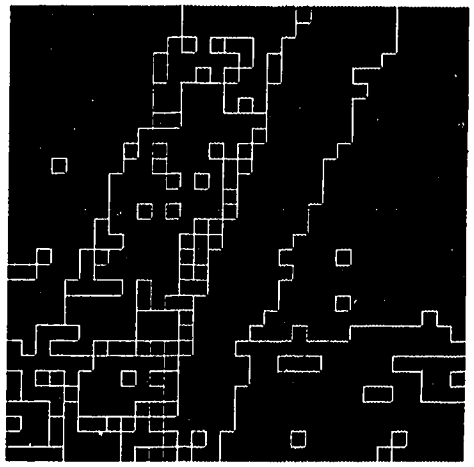


Figure 4.16: Image M6, increased noise level left-right

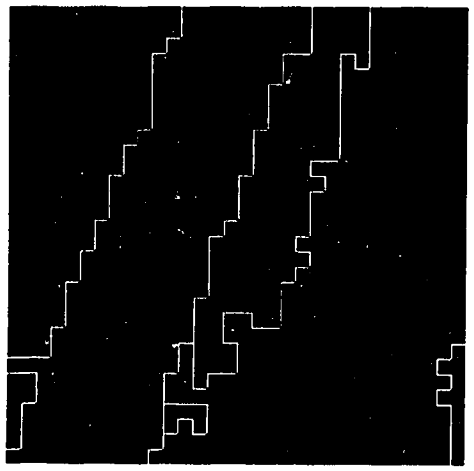


(a) FAS segmentation of M5

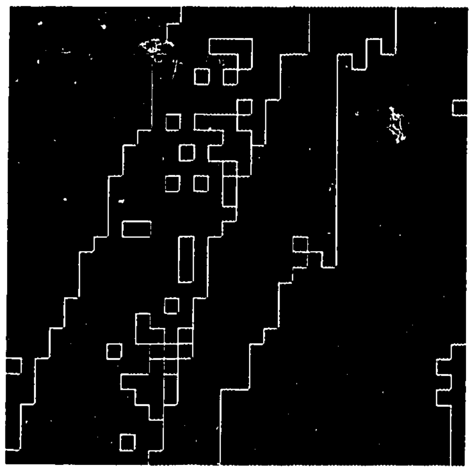


(b) BFS segmentation of M5

Figure 4.17: Segmentations of M5



(a) FAS segmentation of M6



(b) BFS segmentation of M6

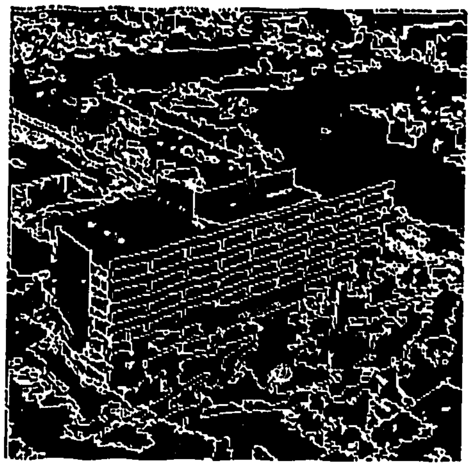
Figure 4.18: Segmentations of M6

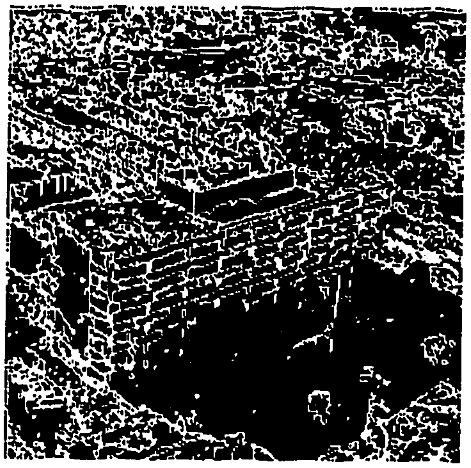
4.4.3 Applicability on Natural Grayscale Images

The objective of this project is to provide a general segmentation framework that can be easily customized to work on different types of images. We have so far presented several segmentations on textured images. In this experiment, we show that the FAS algorithm can also produce good segmentation on a wide range of grayscale images.

Four natural 256-level grayscale images, M19 to M22, are given in Figures 4.19 to 4.22. In this experiment, we use pixel intensity as the feature measure; each pixel is a primitive region in the region growing process. The FAS and BFS segmentation results on this set of images are given in Figures 4.19 to 4.22. The FAS algorithm produces consistently good segmentations on all four images. All four FAS segmentations are obtained using $\lambda = 0.80$. Applying the BFS algorithm on an image, we have to invoke the segmentation process using different threshold values until a satisfactory segmentation is obtained. The most satisfactory BFS results are given in the figures. For some images, no satisfactory segmentation can be obtained using the BFS algorithm. For example, on the BFS segmentations of image M19, (Figure 4.19(c) and (d)) the background is fragmented regardless of the threshold applied; relaxing the threshold (Figure 4.19(d)) causes details on the building to be lost. The thresholds used in segmenting the other three images are in the range of 15 to 30.

(a) Image M19


(b) FAS segmentation of M19


(c) BFS segmentation of M19
(threshold = 15)

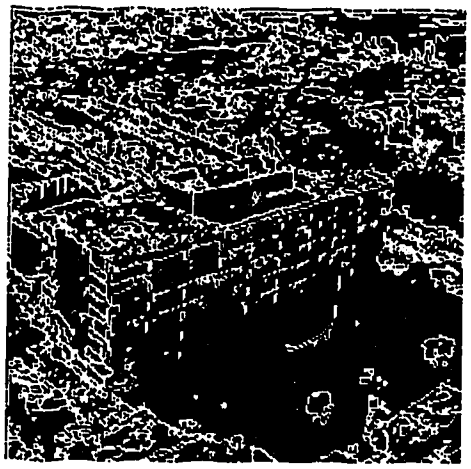
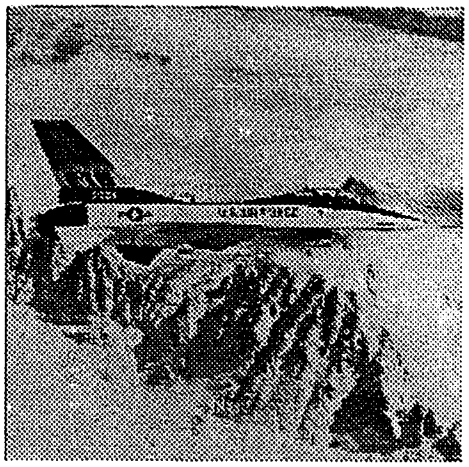
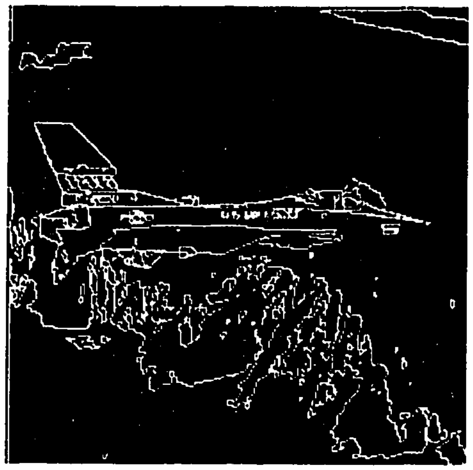

(d) BFS segmentation of M19
(threshold = 20)

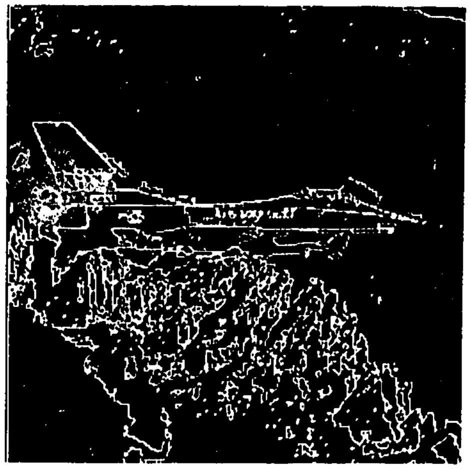
Figure 4.19: M19 and segmentations



(a) Image M20

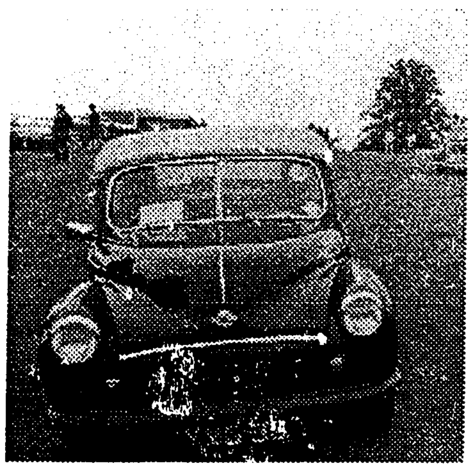


(b) FAS segmentation of M20

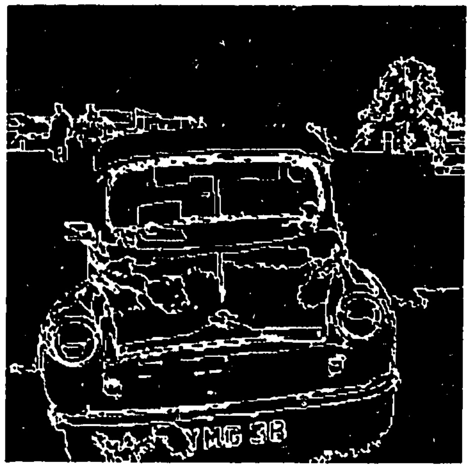


(c) BFS segmentation of M20

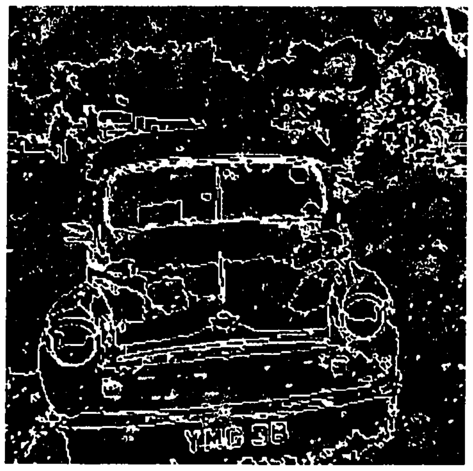
Figure 4.20: M20 and segmentations



(a) Image M21

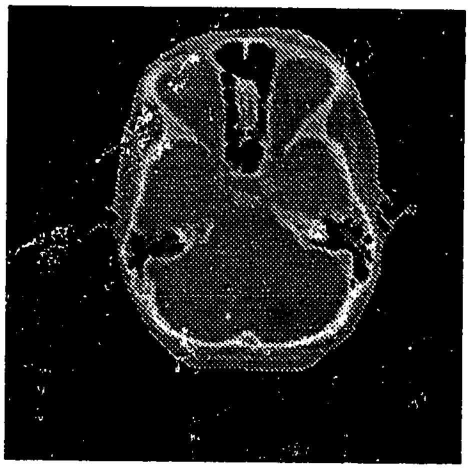


(b) FAS segmentation of M21

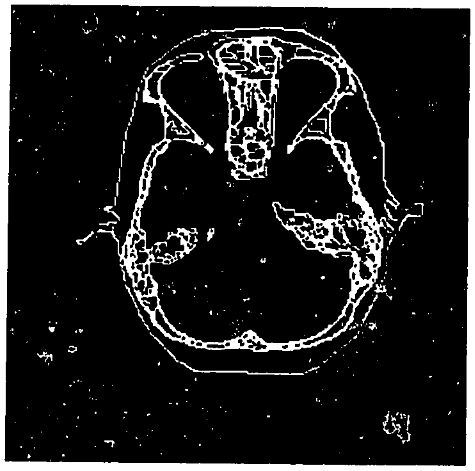


(c) BFS segmentation of M21

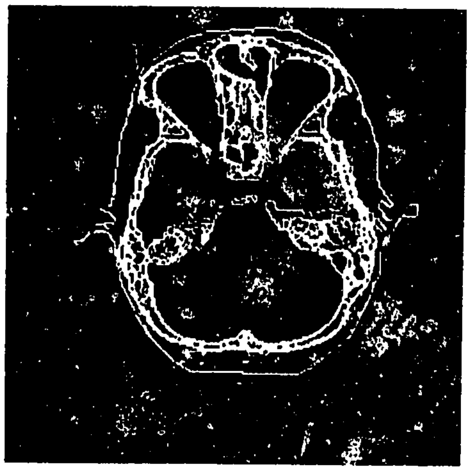
Figure 4.21: M21 and segmentations



(a) Image M22



(b) FAS segmentation of M22



(c) BFS segmentation of M22

Figure 4.22: M22 and segmentations

4.4.4 Using Multiple Features

Single features may not always be sufficient to produce satisfactory segmentations, especially on complex textured images. In such cases, several features, each capable of differentiating different subsets of textures, need to be used to improve on the segmentations. In this test, we demonstrate that multiple features can be incorporated quite easily and successfully in the FAS algorithm.

Test image M12 consists of sixteen equal sized texture regions. Figure 4.23 shows the sixteen textures, magnified and arranged in the same layout as in M12. Three co-occurrence features, the contrast, entropy, and correlation measures, are computed from 16 × 16 pixel blocks in the segmentation processes. The segmentations obtained using single features, combinations of two-feature, and three-feature are given in Figures 4.24 to 4.26. In all three figures, the left sub-figures are segmentations from the FAS algorithm, and the right sub-figures are results from the BFS algorithm.

Due to the diversity of textures in M12, both the FAS and the BFS algorithms cannot successfully segment the image with any one of the three features. Adjacent textures that cannot be differentiated using a particular feature are presented as single regions. When two features are used, some improvements are shown on the segmentations. For both algorithms, the three-feature processes give the best segmentations. From the figures, it is obvious that the FAS segmentations are of a higher quality than the corresponding BFS segmentations.

For the BFS algorithm, we apply the conventional Euclidean distance measure in the consolidation of multiple features. That is, the square of individual features of a region are added together to produce a single value. And this value is then used as *the* feature value of the region. The main advantage of this approach is that only one threshold is necessary in the BFS process. Since the BFS algorithm requires careful adjustments on the threshold, multiple-threshold BFS can be formidable. In this approach, feature normalization may be required prior to the computation of the Euclidean distance.

A different feature consolidation approach is used in the FAS algorithm. In this approach, we treat each feature independently. The homogeneity test is a composite test in the multiple-feature case – the composite homogeneity test is passed only if the simple homogeneity test is passed on every feature. In other words, we consider two regions to be homogeneous only if they are homogeneous with respect to every feature that are used in the segmentation process. Therefore, for any two adjacent regions that should not be merged, if any one feature in the set is capable of differentiating them, then these two regions will not be incorrectly merged.

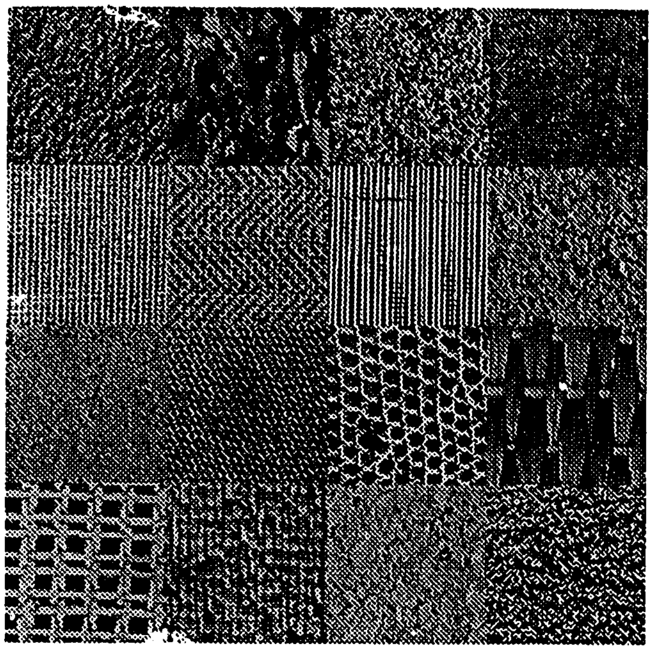
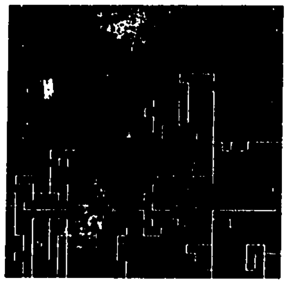
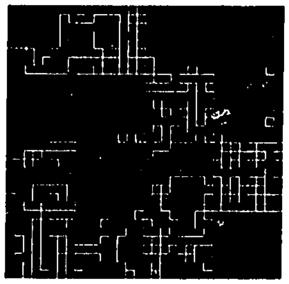
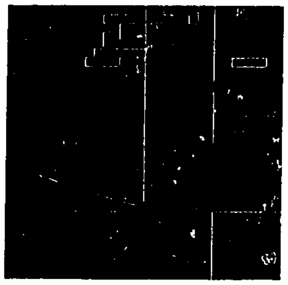
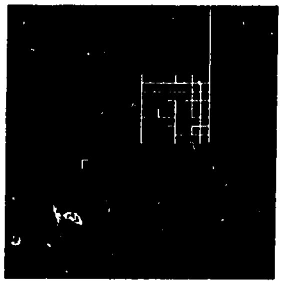


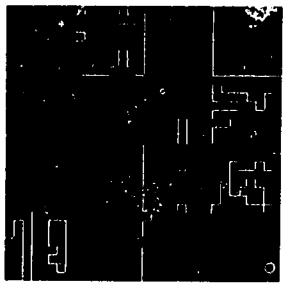
Figure 4.23: Texture layout in image M12


(a) FAS, Entropy


(b) BFS, Entropy


(c) FAS, Contrast


(d) BFS, Contrast


(e) FAS, Correlation

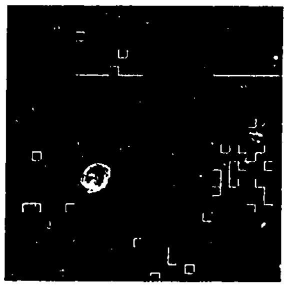
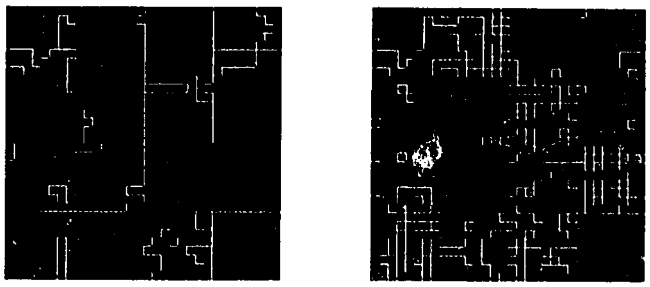
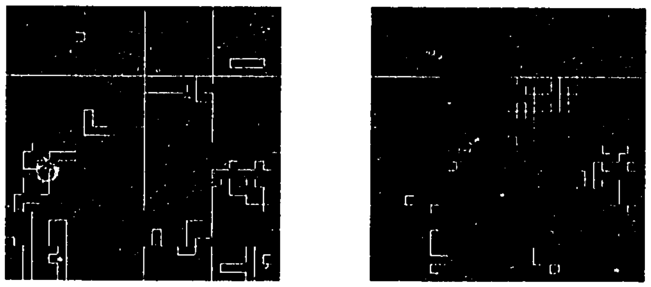

(f) BFS, Correlation

Figure 4.24: Single-feature segmentations of M12

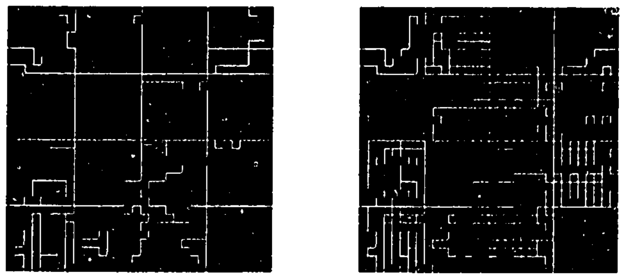


(a) FAS, Entropy-Contrast . (b) BFS, Entropy-Contrast



(c) FAS, Contrast-Correlation (d) BFS, Contrast-Correlation

Figure 4.25: Two-feature segmentations of M12



(a) FAS, Entropy-Contrast-Correlation (b) BFS, Entropy-Contrast-Correlation

Figure 4.26: Three-feature segmentations of M12

Chapter 5

A Shared Memory SPMD Implementation

We have conducted a detailed study on a SPMD (Single Program, Multiple Data) region growing model on the shared memory TC2000 system. In this chapter, we present the results of this study, and examine implementation issues of the model. As in the previous chapters, we compare the performance of our FAS algorithm to that of the existing BFS algorithm given in [67].

5.1 The TC2000 Environment

The BBN TC2000 [3] is a multiprocessor machine that employs shared memory to store information. The TC2000 processors access the shared memory through an interconnection network called the Butterfly switch. The processor is a Motorola 88000 chip group comprised of an 88100 CPU chip and at least two 88200 cache/memory management unit chips. The system uses 34-bit addresses.

The TC2000 system which we use for our experiments was managed by the Massively Parallel Computing Initiative (MPCI) project at the Lawrence Livermore National Laboratory. The system has 128 processors; each processor has 16 megabytes of memory constituting 2 gigabytes of global memory space.

The PCP [5, 25] programming model is used for our implementation on the TC2000. PCP is a parallel extension of the C programming language which employs the split-join parallel model. All processors start and end a program execution at the same time, and sections of the code can be labelled to be executed by a single master process or subteams of processes. There are also primitives for process synchronization, shared and private data management.

The following describes the way our timing results were recorded in the TC2000 environment:

- Unless otherwise specified, "processing time" refers to the amount of time taken from the initiation of the segmentation process until the segmentation results have been stored in a file. "Merge time" refers to the time taken by the merge phase only.
- Timing results are obtained from the UNIX system function "getrusage". In each case, the sum of "system" and "user" times is reported.
- Timing results were obtained using the "benchmark" (single-job, uninterrupted) mode supported by the Gang Scheduler [24].
- A reported processing time is the average of three consecutive runs performed on an image using the same number of processors.
- When parallel processes require different amounts of time to complete a segmentation phase, the time taken by the slowest process is used to compute the total processing time.
- The FAS and the BFS programs are identical except for the [illegible] implementing the merge criteria. This applies to both the sequential and parallel implementations.
- A sequential segmentation time of an image always refers to the best TC2000 sequential time achievable on a given algorithm. Due to the large image graphs, (256K vertices) the best sequential times of grayscale segmentations in our experiment were achieved using the interleaved and cache memories. For the set of textured images, (16K vertices) the best sequential times were obtained using the cached local node memory.
- Due to system problems related to cache memory usage for parallel processes, the parallel timing results reported in the following sections were obtained using the shared memory without the use of cache.

5.2 Sequential Region Growing

A sequential region growing algorithm is implemented so that results can be used for the evaluation of the parallel implementation. Region merging in the sequential algorithm is an iterative process. All existing vertices in the graph are maintained in a doubly linked list. During each iteration, every vertex in the list is checked once for a possible merge. Once a vertex passes the merge criterion, a merge will be performed before the next vertex in the list is examined. The process continues until no more merges are possible in an iteration.

On systems with small memory capacities, paging often takes up a significant amount of time due to the large memory requirement of the process. Careful memory management, such as allocating heap memory in large blocks, does alleviate but does not solve the paging problem. On the TC2000, paging does not pose a problem because a single processor can access the large shared memory. In addition, unlike the SPMD version, fast cache memory can be used in a TC2000 sequential process.

Tables 5.7 and 5.8 give the TC2000 sequential timing results on a set of 512 × 512 grayscale images. The test images M7 to M11 are shown in D.50 to D.54 in Appendix D. Processing time obtained by using different memory policies available on the TC2000 system are presented. "Local memory" refers to the use of the memory subsystem local to the processor where the sequential process resides, and the shared memory is unavailable to the process. When the "interleaved memory" is used, the TC2000 system maps logically contiguous addresses to different memory subsystems. In particular, the interleaver maps each clump of 16 bytes to a different switch port, and therefore to a different memory subsystem. "Copy-back", "write-through", and "uncached" are different modes of cache use. The former two modes differ in whether a write operation on any cache line will immediately initiate an update on the memory system. Naturally, "uncached" inhibits the use of the cache memory.

Although each memory subsystem in the TC2000 has 16 megabytes of memory, only about 8 megabytes of it is available to the user program. Therefore, when only the local memory is used in the segmentation of a 512 × 512 image, a high rate of pagefaults is incurred and this results in a slow process. In our experiment, the use of the interleaved memory with the copy-back cache mode provides the best sequential timings.

Comparing the FAS and the BFS sequential results given in Tables 5.7 and 5.8, we can conclude that the FAS algorithm provides faster segmentation processes. Also, the variation on processing times among images is much smaller using the FAS algorithm. The FAS and the BFS processing time variations on the set of images are depicted in Figure 5.27. It is clear that the BFS timings are highly image-dependent.

Table 5.7: FAS sequential timing on the grayscale images (in seconds)

Image	Local Memory	Interleaved Memory		
	Cached	Copy-back	Write-through	Uncached
M7	501.43	134.98	158.38	363.87
M8	1121.30	147.84	170.29	419.01
M9	467.20	117.49	144.27	309.65
M10	769.10	124.32	149.10	314.29
M11	489.43	112.86	141.71	332.07

Table 5.8: BFS sequential timing on the grayscale images (in seconds)

Image	Local Memory	Interleaved Memory		
	Cached	Copy-back	Write-through	Uncached
M7	981.50	175.42	203.22	613.25
M8	1274.59	210.51	238.87	873.45
M9	1657.50	275.76	318.94	1274.70
M10	1095.88	144.41	181.56	416.09
M11	1453.13	446.59	500.26	2602.26

5.3 The Parallel Process

Mapping a region growing process onto a set of shared memory processors involves the partitioning of workload among the processors. Since the computation requirement of region growing is data-dependent and cannot be determined at compile time, we adopt a simple workload partitioning scheme that assigns an approximately equal division of an image to each processor. A sample of the initial load distribution on a set of processors are shown in Figure 5.28. After individual processors have obtained their sub-images, the image graph is built in parallel, with cooperation between processors sharing image division boundaries.

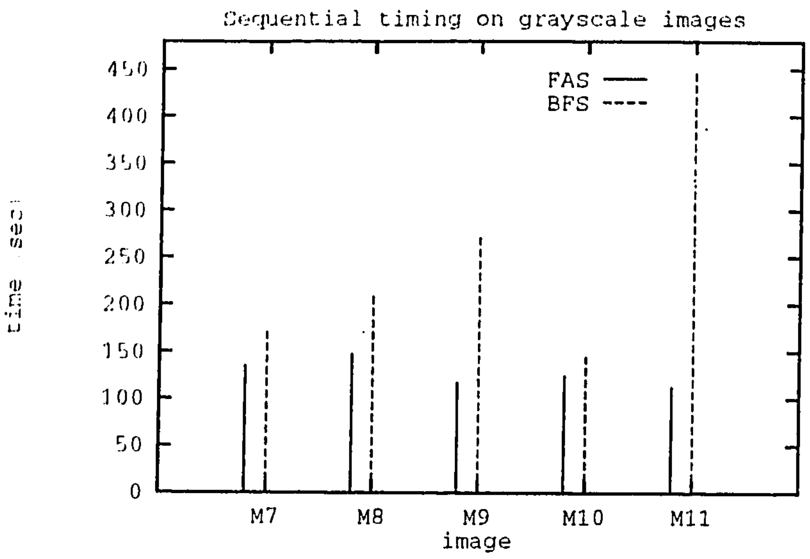


Figure 5.27: Sequential timing on a set of grayscale images

The subsequent region merging phase is an iterative process on individual processors. In the merging phase, the team of processors perform independent merges which update the global image graph at the same time. When no more merges are possible, segmentation results are gathered from the processors and then written to a file. Figure 5.29 depicts an overview of the parallel region growing process on the TC2000.

In the following sections, we examine in detail the implementation issues concerning each of the region growing phases.

5.3.1 Data Distribution and Results Gathering

In the distribution of image data to the multiple processors, we have found that the most efficient method is to allow several processors to read different portions of the image file into the shared memory, then the rest of the processors copy the relevant sub-portions for processing.

In region growing, the virtual processor network has a grid layout. In each

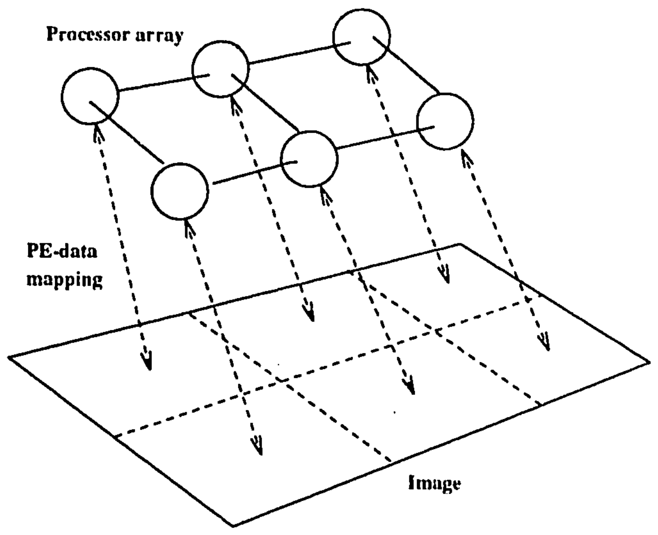


Figure 5.28: Initial load distribution on the TC2000

row of the grid, we assign the left most processor to read a horizontal strip of the image data from a file. All other members in the row then copy the relevant data from the first processor to build their local subgraphs.

After the merging process has completed, individual processors copy their results to a shared memory array. The master process will then store the results in a file. This approach is found to be more efficient than having the master process gather partial results from separate locations of the shared memory.

In Figures 5.30 and 5.31, we show a breakdown of the processing times on images M7 (512 × 512, grayscale) and M13 (1024 × 1024, textured) respectively. In each figure, the curve closest to the X-axis gives the total time required to read the image data, distribute this data among the processors, and build the complete image graph. The next curve above the X-axis indicates the time taken from the start of segmentation to completion of the region merging phase. The vertical difference between the two higher curves gives the time taken to gather and save the segmentation results.

In the segmentation of a 512 × 512 grayscale image, the total time taken for data distribution and results gathering is about 6.5 seconds, of which, about 4.5 seconds are required for writing the results onto a secondary storage. The amount of time increases slightly when more than 25 processors are used in a

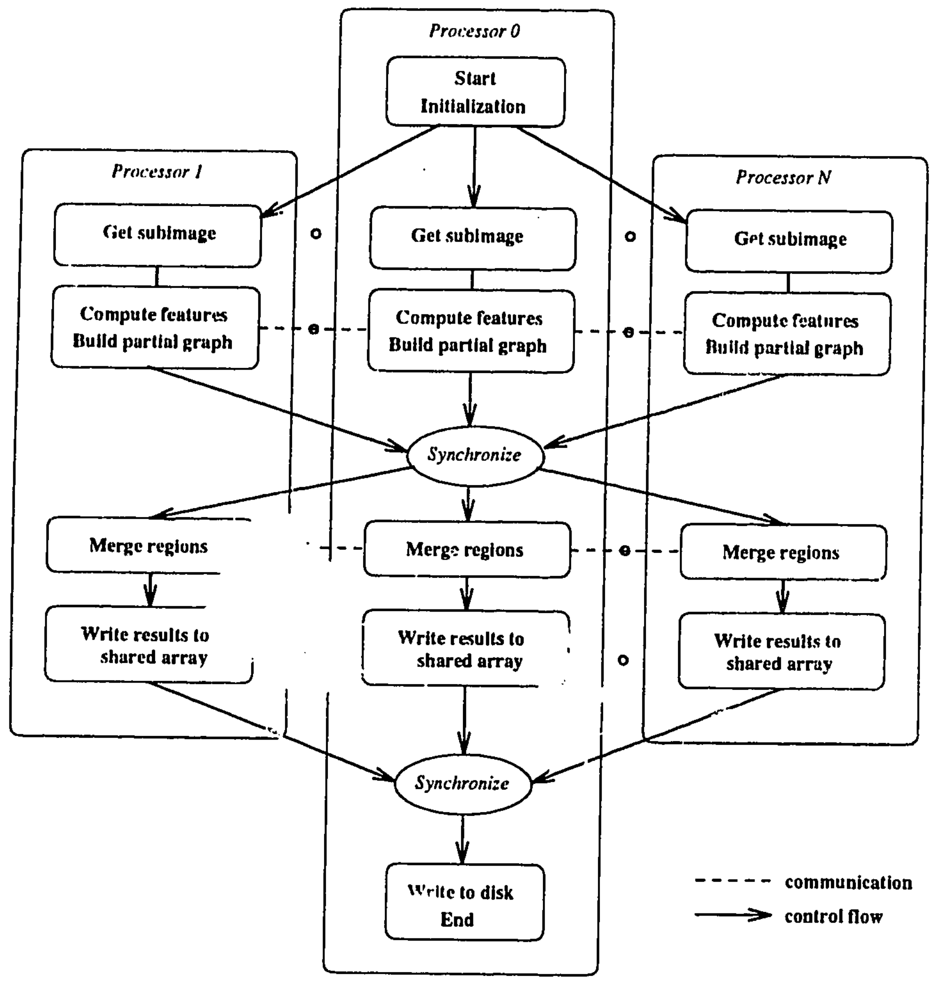


Figure 5.29: A logical view of the TC2000 parallel region growing process

segmentation. Although 4 times larger in size, only about 0.3 second is required to save the segmentation results of a 1024 × 1024 textured image. This is because the segmentation array of the textured image consists of 128 × 128 numbers, compared to the 512 × 512 segmentation array obtained from a grayscale image.

5.3.2 Feature Computation and Graph Building

Feature computation refers to the extraction of a feature value from a pixel or a neighborhood of pixels. In grayscale image segmentation, if the grayscale intensity of a pixel is used as the feature value, feature computation is a trivial task. However, in textured image segmentation where a feature value is always computed from a neighborhood of pixels, the cost of computing all features is a significant portion of the total segmentation cost. As an example, consid a sequential segmentation process on a 1024 × 1024 textured image. If 8 × non-overlapping pixel blocks are used, the feature computation cost of a single co-occurrence measure on the entire image is about 24 seconds. If three co-occurrence measures are used, it will take about 47 seconds to compute all the feature values. In each case, the computation of feature values constitutes 79% and 65% of the total segmentation cost, respectively. Although texture feature computation is expensive, the operations can be efficiently parallelized because each computation involves only local data.

In our TC2000 implementations, feature computation is incorporated in the graph building phase. A processor builds a partial graph on the rectangular image division it has been assigned; feature values are computed and stored in the vertices during the graph building process. Adjacent image divisions have small overlaps to facilitate the seaming of partial graphs. In our graph model, an edge is labelled by the difference between the feature values of its connecting vertices. Although the label of an edge connecting vertices belonging to two adjacent divisions can be computed from the overlap, the pointers to these vertices need to be shared in the seaming process. After attempting several methods to allow the sharing of pointer information, we find that the most efficient way is to let each processor copy into a shared array the vertex pointers other processors will require, and if already available, copy from the shared array those pointer information it needs. Since we adopt the 4-neighbor scheme (each image pixel has a north, south, east, and west neighbors), the initial image graph has a regular grid layout.

The times taken for the graph building phases in the segmentations of M7 :d M13 are given in Figures 5.30 and 5.31. In each figure, there is a notable increase in graph building time when more than 25 processors are used in the process. This undesirable effect cannot be eliminated or lessened by distributing the shared pointer array on different memory subsystems. From our observation, a long graph building time is often caused by a few processors which take exceptionally long times to complete their graph building phases. Since the partial graphs are approximately equal in size, memory switch contention is suspected to have caused the delay. If there was some user control on application communication graph and system network mapping, a better solution may be possible.

5.3.3 Region Merging and Shared Memory Access

A merge operation on the region growing graph involves updates of the merging vertices, edges connecting to them, and all vertices connecting to these edges. In a parallel implementation where merges can be performed at the same time, it is possible for a data field to be corrupted from simultaneous updates. To preserve data integrity under this hazardous situation, we apply locks on shared memory data items.

In the graph data structure, every vertex is assigned a unique identification which is a positive number with the most significant bit serving as a lock. We implement the lock using several atomic instructions supported by the TC2000 hardware. Before proceeding with a merge, a process has to obtain the locks on all involved vertices. Deadlock can occur if two processes locked some vertices and wait for the release of vertices which happened to be held by the other processor. To prevent deadlock from occurring, a process always locks the vertices in ascending order of their identification numbers. Sometimes, while trying to lock a list of vertices, a process may encounter a vertex that has been locked by another process. In this situation, the first process will relinquish all previously locked vertices, sleep for a short interval of time, then repeat the merge attempt. Only after a certain number of attempts have failed will the process proceed to merge a different pair of vertices. In our experiment, a failed merge attempt due to locked vertices is found to be a rare occurrence.

Although the share memory consists of a number of memory units, the actual locations of data items are transparent to the user process. Therefore, unlike the distributed implementation described in Chapter 6, a user program on the shared memory system need not differentiate between inter- and intra-processor merges. Despite this convenience, to ensure an even distribution of data in the shared memory so that network switch contention can be minimized, our algorithm builds the initial subgraph on the memory subsystem local to its processor. Since every process makes the most frequent accesses to its own subgraph, placing its subgraph in the local memory unit is logical.

The TC2000 system does not support cache memory usage in parallel processes. Maintaining cache coherency is therefore the responsibility of the programmer. Our parallel implementation does not make use of the cache memory. However, to provide a conservative study on timing performance ratio, the sequential implementation uses interleaved memory with cache which provides the best timing results.

In Figure 5.32, we compare the 24-processor parallel segmentations using the FAS and the BFS algorithms. Again, results on the grayscale images M7 to M11 are shown. It is clear that compared to the BFS algorithm, the FAS algorithm provides good, consistent timings on the set of images. More detailed results are given in the next section.

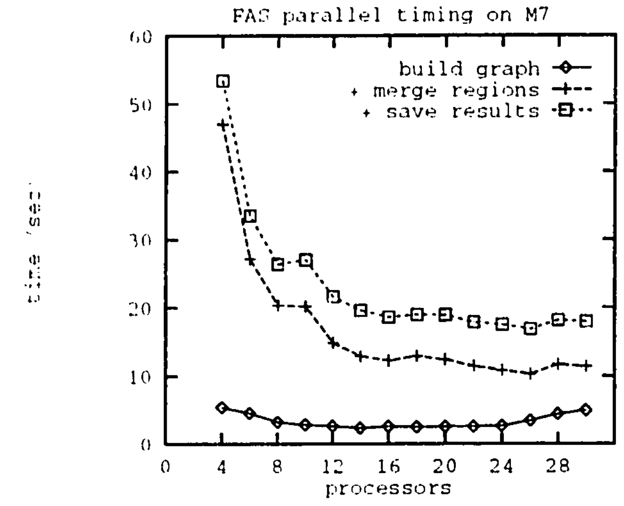


Figure 5.30: Parallel timing on grayscale image M7

5.4 Experimental Results

In this section, we present the detailed parallel timing results of the set of grayscale images. We also examine the distributions of workload on processors

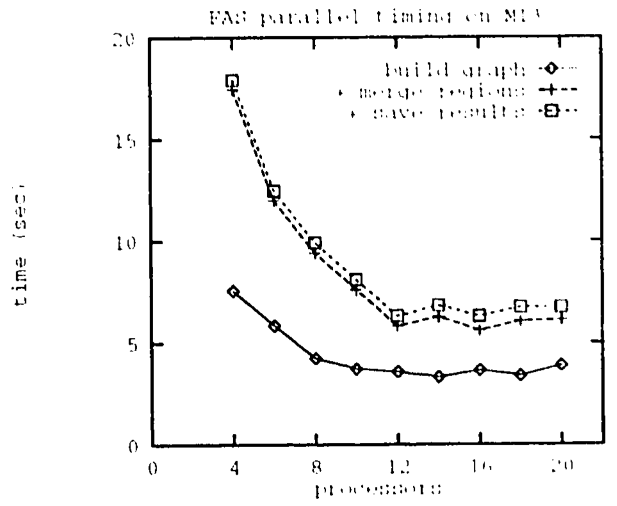


Figure 5.31: Parallel timing on textured image M13

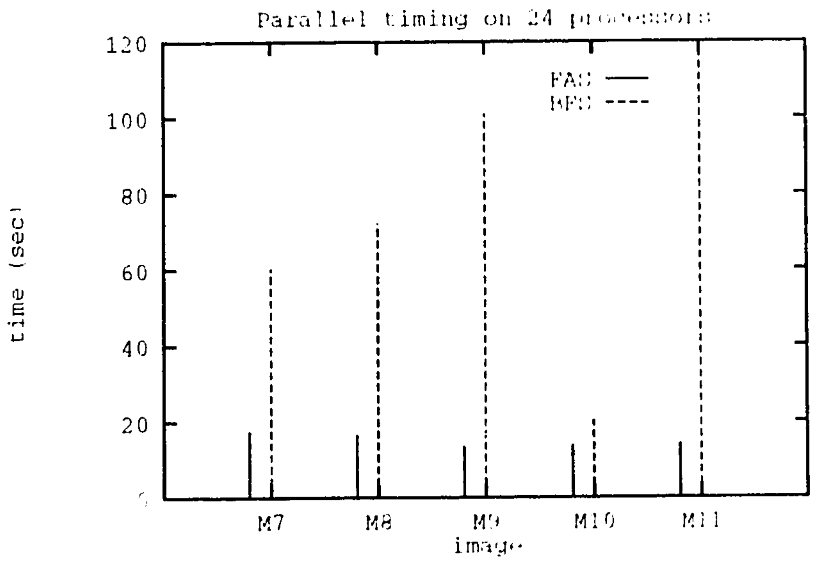


Figure 5.32: Parallel timing on a set of grayscale images

using the FAS and the BFS algorithms, and suggest the appropriate number of processors to use on our given problem size.

An important measure for assessing the performance of a parallel algorithm is the timing ratio between the best sequential algorithm and the parallel algorithm. This measure, known as "speedup", has many definitions in the literature [2, 33, 60]. For example, the sequential and parallel runs may or may not be carried out on the same machine, the sequential algorithm may or may not be the fastest. In this study, we measure the ratio between the best sequential and the parallel processing times on the same machine using the same merge criterion. More specifically, the best sequential time is the shortest runtime achievable on a single-processor given the available memory access options on the TC2000. We simply call our measure the **timing ratio** or the **merge time ratio**, depending on whether the entire segmentation process or only the merge phase is being evaluated. The term "speedup" is not used to avoid possible confusion.

5.4.1 Parallel Segmentation Time

Table 5.9 presents the FAS and the BFS parallel timings of images M7 to M11 using different numbers of processors. To provide a clearer comparison, the FAS and BFS timings on image M7 are depicted in Figure 5.33.

In the experiment, it is shown that the FAS algorithm gives good, consistent parallel timing results on all five images, but parallel timings from the BFS algorithm are image-dependent. In addition, the change in BFS timings using different numbers of processors is inconsistent and unpredictable. In conjunction with our earlier analysis (in Chapter 4) on the Best-merge selection policy, we conclude that the BFS algorithm results in highly ordered and sequential segmentation processes on some images - before a processor carries out a certain obstructive merge, another processor cannot perform its merges.

Increasing the number of processors in a BFS process may help to reduce the parallel processing time marginally. This is because by reducing the workload on individual processors, size of their partial graphs are also reduced; consequently, obstructive merges have higher chances of being performed sooner. Nevertheless, further timing improvement is severely limited by the sequential nature of the merge sequence.

As is the convention in the literature on parallel timing performance [59], times required for initial setup and final results gathering are not included in the comparison of the sequential and parallel timings. In region growing segmentation, we report the merge time ratio obtained by dividing the sequential merge time by the parallel merge time on an image. The merge time ratios on the test images M7 to M11 are given in Table 5.10. As in the presentation of parallel timing, a graphical illustration on the performance ratios of image M7 is given in Figure 5.34.

The excellent FAS merge time ratios for segmentations using below 12 processors were achieved because most merges in the parallel processes involve local memory updates; in the associated sequential segmentation which employs the interleaved memory, a merge involves updating memory locations on different memory subsystems. By increasing the number of processors, task granularity decreases and more merges require the updates on non-local memory locations. As a result, the merge time ratios become less desirable.

More sequential and parallel timing results on textured images are given in Section 5.5.

Table 5.9: Parallel timing on the grayscale images

PEs	FAS (sec.)					BFS (sec.)				
	M7	M8	M9	M10	M11	M7	M8	M9	M10	M11
1	135.0	147.8	117.5	124.3	112.9	175.4	210.5	275.7	144.4	446.6
4	53.4	39.9	39.8	37.7	38.3	187.2	587.3	829.8	59.5	8119.9
6	33.5	33.1	26.7	27.0	29.6	164.4	520.4	442.4	44.9	4515.8
8	26.4	26.4	21.6	22.3	23.6	142.2	499.5	312.1	39.3	1032.8
10	27.0	21.3	19.2	20.1	20.8	117.7	159.5	124.5	31.9	1524.6
12	21.6	20.0	19.1	17.5	18.8	80.3	244.0	200.7	30.5	836.8
14	19.6	19.2	16.2	16.2	17.0	56.7	154.6	88.9	28.4	1252.2
16	18.6	17.0	15.2	15.1	16.2	84.0	175.0	149.0	24.2	742.0
18	18.9	17.7	14.9	15.2	16.4	82.0	172.0	90.2	23.7	402.6
20	18.9	16.2	14.3	14.3	15.2	59.3	128.9	85.8	21.5	782.4
22	17.8	17.5	13.8	14.4	14.7	49.7	67.0	87.9	20.4	538.6
24	17.5	16.4	13.4	13.8	14.2	60.2	72.2	101.0	20.4	501.9
26	16.8	15.0	13.9	14.3	15.0	37.6	80.4	88.9	21.1	332.9
28	18.1	18.0	14.8	14.5	14.6	46.7	88.5	83.0	21.3	260.2
30	17.9	19.0	15.0	15.8	14.9	64.5	78.7	97.2	20.8	250.0

Table 5.10: Sequential to parallel merge time ratio on the grayscale images

PEs	FAS					BFS				
	M7	M8	M9	M10	M11	M7	M8	M9	M10	M11
4	2.7	3.8	3.3	3.8	3.3	0.9	0.3	0.3	2.6	0.1
6	4.7	5.1	5.4	5.7	4.5	1.1	0.4	0.6	3.5	0.1
8	6.3	7.2	7.3	7.4	6.1	1.2	0.4	0.9	4.2	0.4
10	6.4	9.1	8.6	8.7	7.3	1.5	1.3	2.3	5.4	0.3
12	8.7	10.1	8.7	10.9	8.6	2.3	0.5	1.4	5.6	0.5
14	10.0	11.0	11.3	12.1	9.8	3.3	1.4	3.4	6.7	0.4
16	10.5	12.8	12.6	13.6	10.8	2.2	1.2	1.9	7.8	0.6
18	10.0	12.3	13.3	13.5	11.2	2.2	1.2	3.3	8.0	1.1
20	10.4	14.3	14.0	15.0	12.6	3.2	1.7	3.5	9.1	0.6
22	11.3	12.4	14.2	15.1	12.7	3.8	3.4	3.4	9.7	0.8
24	11.9	14.3	15.4	16.1	13.6	3.1	3.1	2.9	9.8	0.9
26	12.5	16.4	14.9	15.2	12.1	5.4	2.8	3.3	9.0	1.4
28	11.0	12.2	15.2	15.4	12.8	4.1	2.5	3.6	9.2	1.7
30	11.3	11.2	13.0	12.4	12.5	2.9	2.8	3.0	9.1	1.8

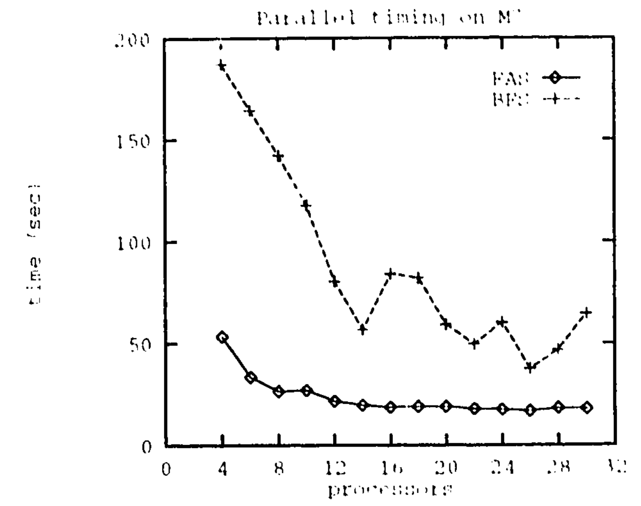


Figure 5.33: FAS and BFS parallel timing on M7

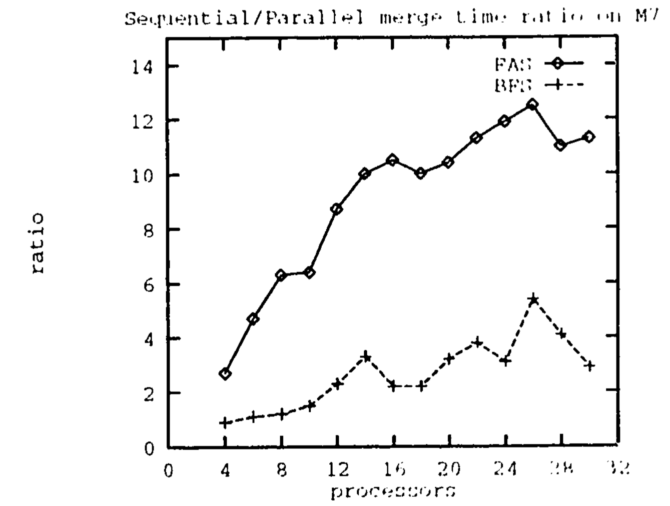


Figure 5.34: FAS and BFS merge time ratio on M7

5.4.2 Load Partitioning

If present, the load imbalance problem in region growing is difficult to solve. This is because region growing is a data-dependent problem in which behavior and computational requirement cannot be determined at compile time. Unless load imbalance is critical, dynamic load balancing is not feasible due to the high cost and difficulty in finding an optimal load transfer solution.

Load imbalance is a serious performance problem in a region growing process employing the Best-merge selection policy. The Best-merge policy has the characteristic of imposing an order on the merges. In a parallel implementation, this results in a significant amount of time being spent on processing failed merge attempts. Since the distribution of failed merges is highly uneven and image-dependent, times taken for the parallel merge processes are also highly uneven.

However, by using the Fast-merge policy and a sufficient number of processors, we are able to obtain a fairly balanced load by evenly dividing an image among the processors. That is, every processor works on an equally sized subimage at the beginning of the merge phase. In our implementation, the merge process is so short that any small load imbalance has insignificant negative impact on the overall performance.

Tables 5.11 and 5.12 give an example of processor load distribution using the FAS and the BFS algorithms on the test image M7. Each row of the tables gives three load measures that are recorded when the algorithms were run using a specified number of processors. The three load measures are the amount of time spent on the merge phase, the number of successful merges performed, and the number of merge rejections encountered. For each measure, we present the maximum, minimum and mean values of all the processors in a run. The standard deviation, which indicates the variation of this measure among the processors, is given in the fourth column. Note that the numbers of successful merges in both tables are given in thousands, so are the numbers of merge rejections in the BFS table.

From the two tables, it is clear that the distributions of successful merges on the FAS and the BFS processes are even and similar. However, the distributions of merge rejections on the BFS processes are highly uneven compared to those on the FAS processes. Consequently, the BFS merge times are much longer because processes in a run have very different workload.

The graph in Figure 5.35 shows the individual processor merge times of the FAS and the BFS processes on 16 processors. The same test image M7 is used in both runs. It is clear that the BFS processes suffer a severe load imbalance, and the FAS processes have a well balanced load.

Table 5.11: PE Load distribution using FAS on M7

PEs	Merge Time (sec.)				Successful Merges (K)				Merge Rejections			
	min	max	mean	σ	min	max	mean	σ	min	max	mean	σ
4	36.1	41.5	38.5	2.0	64.9	65.3	65.2	0.17	430	2251	1254	658
6	20.2	22.6	21.3	0.8	43.1	43.6	43.4	0.14	308	837	519	216
8	15.2	17.1	16.2	0.6	32.4	32.7	32.6	0.09	143	829	368	226
10	12.4	15.6	13.3	0.8	25.7	26.4	26.0	0.19	24	721	279	204
12	10.2	12.2	11.0	0.5	21.5	21.9	21.7	0.09	21	755	265	214
14	8.7	10.5	9.5	0.4	18.4	18.7	18.6	0.10	42	741	205	191
16	7.7	9.7	8.3	0.4	16.2	16.4	16.3	0.05	12	648	188	160
18	6.8	9.0	7.6	0.4	14.2	14.5	14.4	0.09	12	1054	200	231
20	6.0	8.8	6.8	0.5	12.8	13.2	13.0	0.10	2	461	143	131
22	5.7	8.5	6.4	0.5	11.5	12.0	11.8	0.14	10	491	138	112
24	5.2	8.1	5.7	0.5	10.7	11.0	10.8	0.09	5	443	118	121
25	5.0	7.1	5.5	0.4	10.2	10.6	10.4	0.09	2	596	137	167
26	4.8	6.8	5.5	0.4	9.8	10.2	10.0	0.13	5	396	112	98
28	4.5	7.2	5.0	0.4	9.1	9.4	9.3	0.06	10	958	124	183
30	4.3	6.4	4.6	0.3	8.5	8.8	8.6	0.08	4	591	116	159

5.4.3 Number of Processors

For any given computation problem, no matter how many processors are applied, its parallel timing performance is limited by it sequential, non-parallelizable segment – this fact is stated in the well known Amdahl's Law. Besides the sequential segment of an algorithm, it is obvious that the size of a problem instance also limits the maximum number of processors to use. Depending on system architecture, certain parallel overheads (e.g. process communication) may also grow with the number of processors, thus putting a limit on the number of processors used for solving the problem.

On the given problem size of a 512×512 grayscale image, it is shown in our experiment that the maximum merge time ratio on the TC2000 is achieved on about 25 processors.

Table 5.12: PE load distribution using BFS on M7

PEs	Merge Time (sec.)				Successful Merges (K)				Merge Rejections (K)			
	min	max	mean	σ	min	max	mean	σ	min	max	mean	σ
4	58.1	174.3	112.0	50.2	64.4	64.8	64.6	0.17	53.9	64.1	57.9	3.83
6	32.7	153.8	75.4	51.5	42.6	43.3	43.1	0.28	33.4	46.1	38.9	4.08
8	26.6	132.5	63.0	43.5	32.0	32.5	32.3	0.15	24.1	36.2	29.5	3.42
10	19.7	108.5	48.4	35.2	25.4	26.1	25.8	0.20	19.0	27.5	23.4	2.56
12	16.7	71.3	32.7	19.6	21.2	21.7	21.5	0.14	13.1	23.7	19.5	2.58
14	14.1	47.4	25.0	12.6	18.1	18.7	18.4	0.14	12.8	19.6	16.7	1.72
16	12.0	74.7	25.6	18.4	15.9	16.3	16.1	0.11	10.3	17.9	14.8	1.74
18	10.1	72.6	24.1	18.4	14.1	14.6	14.3	0.13	9.0	16.4	13.3	1.88
20	9.6	50.1	19.3	13.1	12.6	13.1	12.9	0.12	7.7	14.6	11.8	1.62
22	8.3	40.6	17.7	11.6	11.4	12.1	11.7	0.20	7.8	15.4	10.8	1.67
24	7.6	50.2	17.7	12.8	10.5	10.9	10.7	0.10	5.7	12.3	9.9	1.49
25	7.4	44.7	15.4	10.9	10.0	10.5	10.3	0.13	5.6	11.5	9.5	1.39
26	7.4	27.4	14.1	6.8	9.6	10.1	9.9	0.15	6.5	12.0	9.1	1.27
28	6.8	35.7	14.1	9.6	9.0	9.4	9.2	0.08	4.9	10.6	8.5	1.32
30	6.1	52.2	15.0	13.5	8.3	8.8	8.6	0.11	4.2	10.2	7.9	1.35

5.5 Effects of Image Characteristics on Timing

The objective of the following experiment is to examine the effects of several image characteristics on timing performance. It is important for the timing requirement of a segmentation algorithm to be predictable. That is, the timing performance should be relatively insensitive to image characteristics such as region sizes, shapes, and feature distributions. In the previous sections, it has been shown that the FAS timing requirements on natural images M7 to M11 are consistent. In this section, we want to further verify through a control experiment that the timing performance of the FAS algorithm is relatively image-independent.

The test images used in this experiment are constructed using textures from the Brodatz album [4]. The same texture codes in [4] are used here to identify the textures. Each image is 1024 × 1024 pixels in size. They are divided into three groups:

- **Group A:** It consists of three textured images M13, M14, and M15. Each image has only one region. Three textures with different degrees of coarse-

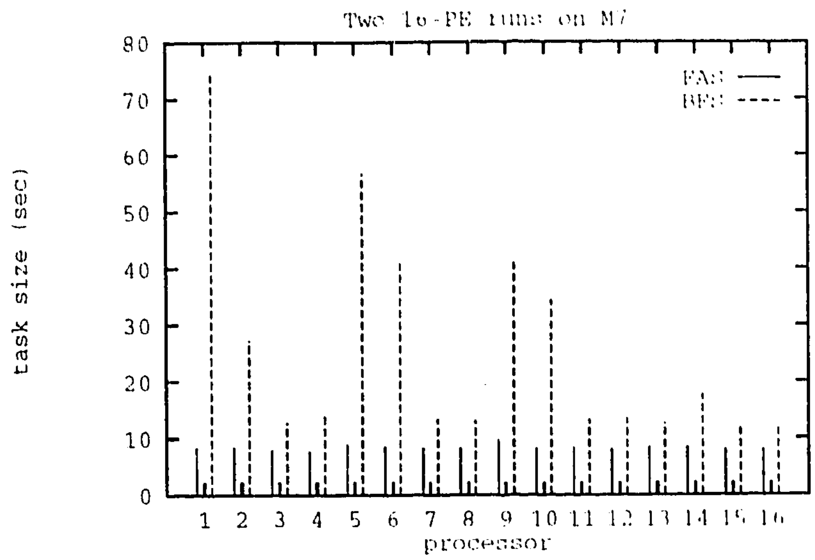


Figure 5.35: Load distribution by FAS and BFS

ness are chosen. The textures are shown in Figures 5.36, 5.37, and 5.38.

- **Group B:** The two images, M16 and M17, in this group are given in Figures 5.39 and 5.40. Each image is made up of the same two textures D77 and D57. The region made up of texture D57 is a trapezium in M16 but has a more complex spiral shape in image M17. The trapezium and spiral regions have the same size.

- **Group C:** A 4×4 and an 8×8 texture mosaics are included in this group. The images are named M12 and M18 respectively. The texture layouts of these two images are shown in Figures 4.23 and 5.41.

We present the sequential segmentation times of all three groups of images in Table 5.13. For group A and B images, processing times using one and three co-occurrence features are reported. Only the 3-feature timings are given for images M12 and M18 because these images can only be successfully segmented using three features. In the experiment, the same features are used in the segmentation of images within the same group. We can therefore assume that the feature computation times of these processes are equal. In all segmentations, texture windows of size 8×8 were used. That is, the size of the initial image graph has 128×128 vertices in each process.

Following are some observations of the effects of image characteristics on timing performance:

1. **Texture Feature Distribution**
   Among the three images in group A, image M13 has the longest segmentation times using the FAS algorithm. The segmentation times on image M13 is 4 and 13% longer than the times required on the other two images. Also, the segmentation processes on M14 are about 3 and 12% shorter than that on M13 and M15.

   Using the BFS algorithm, image M13 also requires more time to segment. Its segmentations take 92 and 139% longer than those of M14 and M15 in the single feature case, and 100% longer than the 3-feature segmentation of M14. Except for the single-feature BFS segmentation, image M14 always has the shortest segmentation times. Its BFS segmentation processes are are 92 and 102% shorter than the others.

   Since the only difference among the three images is the texture, it is reasonable to assume that texture feature distribution does have an effect on the timing of both algorithms. Obviously, the BFS algorithm timing performance is much more sensitive to texture feature distribution than the FAS algorithm.

2. **Region Shape**
   Using the FAS algorithm, the segmentations of image M16 are 5 and 7% shorter than that of image M17. However, using the BFS algorithm, the segmentation processes on M16 are 28 and 30% longer. The only difference between images M16 and M17 is the shapes of their regions; image M16 has a simple boundary between its two regions, while the regions in M17 have a more complex and irregular boundary. Region shape appears to have an opposite effect on the two algorithms. Overall, the effect of region shape on the timing of the FAS algorithm is less significant than that on the BFS algorithm.

3. **Region Size**
   The size of each region in M12 is 4 times bigger than that in M18. Otherwise, the two images are made up of the same sixteen textures arranged in very similar layouts. Using the FAS and the BFS algorithms, image M18 requires 3 and 10% less time to be segmented, respectively. Intuitively, a larger region will require more merges to form than a smaller region. Again, the experiment shows that the effect of region size on the FAS algorithm is less significant than that on the BFS algorithm.

For completeness, we give the FAS parallel timing results of the three image groups in Table 5.14. The parallel processing time t_n obtained on using n processors is recorded in seconds. Only single-feature results are reported for images M13 to M17. Although each textured image is 1024 × 1024 pixels in size, the initial graph has only 16K vertices (compared to 256K on a 512 × 512 grayscale image) because texture features are extracted from 8 × 8 non-overlapping windows. Due to the small problem size, only up to 16 processors were used in this experiment. The timing ratios, $\frac{t_1}{t_n}$, of the overall segmentation processes are also given in Table 5.14.

Comparing the timing results of images within the same group, it is evident that as in sequential segmentation, image characteristics have relatively small effects on the FAS parallel timing performance.

Table 5.13: Sequential timing on a set of textured images

Image	FAS (t_1 sec.)		BFS (t_1 sec.)	
	1-Feature	3-Feature	1-Feature	3-Feature
M13	37.49	59.30	162.35	126.01
M14	33.22	56.47	84.37	62.81
M15	34.75	56.81	67.90	127.26
M16	32.59	55.18	64.36	96.19
M17	34.23	58.96	46.08	67.30
M12	–	54.78	–	66.28
M18	–	53.43	–	60.48

Table 5.14: FAS parallel timing on the textured images

PEs	M13		M14		M15		M16		M17		M12		M18	
n	t_n	$\frac{t_1}{t_n}$	t_n	$\frac{t_1}{t_n}$	t_n	$\frac{t_1}{t_n}$	t_n	$\frac{t_1}{t_n}$	t_n	$\frac{t_1}{t_n}$	t_n	$\frac{t_1}{t_n}$	t_n	$\frac{t_1}{t_n}$
4	17.9	2.1	15.4	2.2	16.5	2.1	12.4	2.6	15.0	2.3	16.4	3.3	17.4	3.1
6	12.5	3.0	11.0	3.0	11.7	3.0	9.0	3.6	9.9	3.5	12.8	4.3	12.5	4.3
8	9.9	3.8	9.2	3.6	9.0	3.9	7.0	4.7	9.3	3.7	9.7	5.7	10.2	5.2
10	8.1	4.6	7.2	4.6	6.7	5.2	6.2	5.2	7.8	4.4	8.6	6.4	9.5	5.6
12	6.3	5.9	6.1	5.5	6.7	5.2	5.8	5.6	7.2	4.8	8.2	6.7	7.8	6.8
14	6.8	5.5	6.6	5.0	7.0	4.9	5.6	5.9	6.8	5.1	8.4	6.5	7.4	7.2
16	6.3	6.0	6.9	4.8	7.1	4.9	5.7	5.8	6.5	5.2	7.3	7.5	8.0	6.7

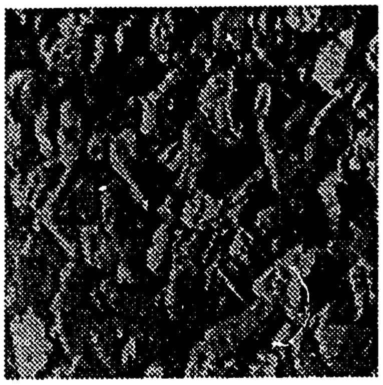


Figure 5.36: Texture (D05) of image M13

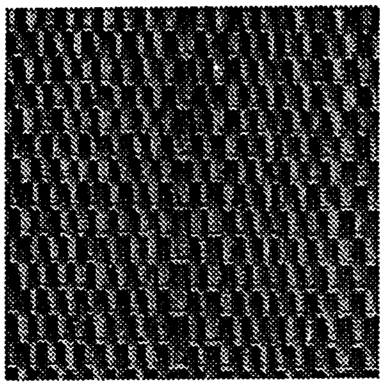


Figure 5.37: Texture (D55) of image M14

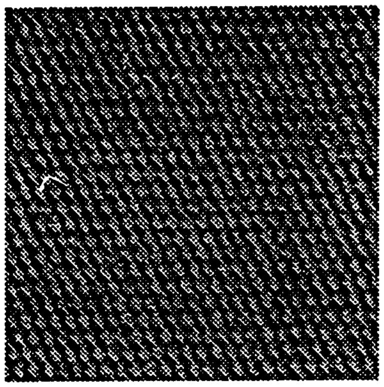


Figure 5.38: Texture (D77) of image M15

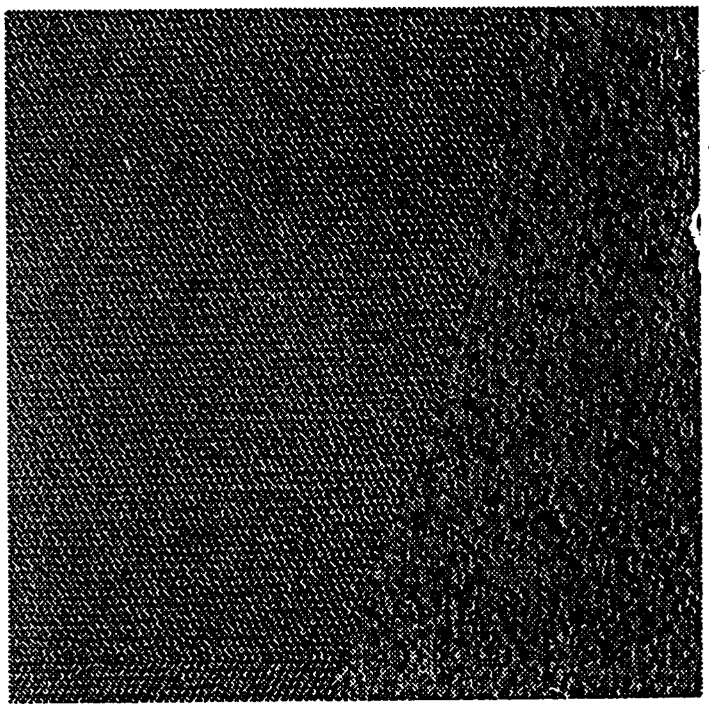


Figure 5.39: Image M16

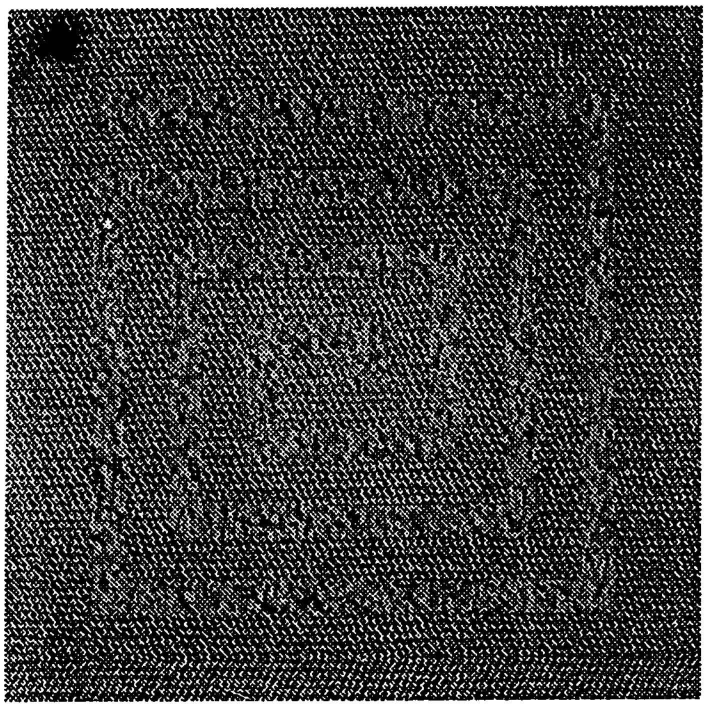


Figure 5.40: Image M17

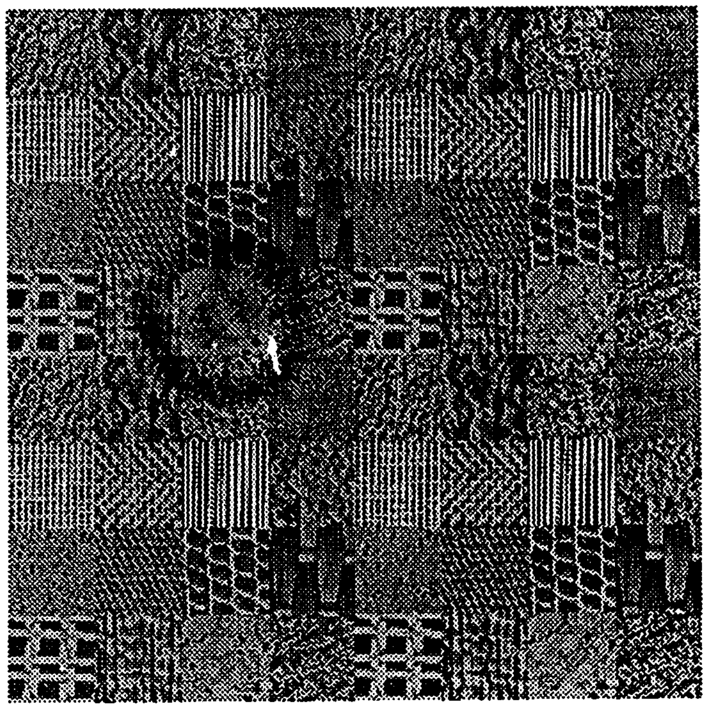


Figure 5.41: Texture layout in image M18

Chapter 6

Other Implementations

We have investigated the shared memory SPMD region growing model in Chapter 5. In this chapter, we present and compare several other possible implementations of the region growing algorithms. The strengths and weaknesses of these different approaches are discussed. Among the approaches, the MIMD image seaming, split-and-merge, and SIMD approaches have been studied in [9, 10, 63, 67]. We implemented the distributed region growing model on a network of workstations. The detailed distributed timing results are presented in Section 6.4.3.

6.1 Image Seaming

An intuitive parallel approach, named image seaming, was proposed by Chen and Pavlidis [9]. Image seaming is a direct divide-and-conquer solution to segmentation. In this method, we divide an image among all the available processors in the parallel system; each processor performs a segmentation on its sub-image independently using the region growing or any other segmentation technique; we then obtain the final segmentation by combining individual segmentations from the processors.

Image seaming may appear to be an attractive approach because besides the initial data distribution and the final result gathering, there is no process communication or shared data access during the segmentation process. However, a closer examination on the algorithm would reveal the complexities of the results combining (seaming) process.

Suppose we divide the image into rectangular sub-images with overlapping boundaries. We map these sub-images to processors arranged in a grid layout in such a way where the spatial relations among sub-images are preserved. After obtaining the sub-image segmentations, label information at the overlap areas are used to identify regions that span processors and produce a coherent segmentation. There are two major difficulties in the seaming process:

- The labeling of overlap areas by different processors often do not agree because each processor labels the same area based on different image divisions. There is no absolute solution to resolve these boundary ambiguities, and a reliable heuristic for resolving label conflicts is difficult to define without *a priori* knowledge about the image.
- Using the rectangular image partition scheme, every processor has either two or four common boundaries to update, depending on the location of the processor in the grid layout. If each common boundary is to be processed only once, all boundary updates will need to be performed in some sequential order. This is because an update involves the identification of regions spanning more than one processors, and the labels have to propagate along these spanned processors to result in a unique label for each region. Therefore, result-combining is a sequential process that is very difficult to parallelize. We may attempt a relaxation result-combining technique that iteratively updates the common boundaries until there are no more changes in region label. Although it will reduce processor idle time, such a method will still require the propagation of region labels among processors and may not always be faster than sequential seaming.

Though intuitive, image seaming is not a practical approach on complex images, and images for which we have no *a priori* knowledge.

6.2 The SIMD Approach

The single-instruction multiple-data stream (SIMD) processor array machines is a successful computational model in low-level image analysis [45, 51, 53]. A typical SIMD processor machine consists of a regular lattice interconnection of a large number of processing elements. Each processing element, ordinarily of modest computing capabilities, has access to its own local memory. The array of processors compute in unison at all times, making these machines SIMD in nature. Many low-level image operations, such as histogramming, convolution, rank order filtering, have been efficiently implemented on SIMD architectures using replicated data [45] and other techniques [18, 42]. High level model based image interpretation has also been attempted using the SIMD model [45].

Although also a low-level image operation, the region growing process has a monotonically decreasing degree of parallelism. There is a high degree of parallelism at the beginning of the process because the number of regions is largest. When the process progresses, the number of regions decreases and so does the degree of parallelism.

In the SIMD computational model, execution is synchronized at the instruction level. At any instant, a processor either executes the same instruction as other processors in the system, or it remains idle. Because of this restriction, it is implausible to assign a group of primitive regions with an undetermined computation requirement to each processor. If we assign one primitive region to each processor, the system will assume the maximum degree of parallelism of the region growing problem. While the number of regions decreases in the process, the number of processors remains a constant. If we maintain one region per processor by transferring image data to an active processor in each merge operation, there will be an increasing number of inactive processors in the system. Alternatively, if region data are distributed among processors, later merges will become more expensive due to high communication costs which are proportional to region size. In terms of processor utilization and efficiency, SIMD is not an appropriate model for region growing.

Willebeek-LeMair and Reeves [67] described a SIMD region growing approach on the MPP. Some of their results are given in Section 2.7.

6.3 Split-and-Merge

In the split-and-merge approach, the main objective of the split process is to reduce the number of initial regions for the latter merge process so that the total processing time can in turn be reduced.

The most efficient data structures for the split and merge processes are the quadtree and graph respectively. The top-down split process recursively divides an image into homogeneous regions represented by the leaf nodes of a quadtree. The quadtree is then converted to an adjacency graph, and a merge process combines adjacent regions to produce the final segmentation. The merge process is identical to the one in a our pure merge (bottom-up) region growing approach.

We have performed a small experiment to compare the timing performance of sequential pure merge region growing and sequential split-and-merge. In the split process, a parent region is divided into four equal sized child regions, and a feature mean is computed on each child region. If the difference between the maximum and minimum feature means exceeds a predefined threshold, the parent region is split into four, and the four child regions are promoted to parents and the split operation is recursively applied on each of them. Otherwise, if the difference between the maximum and minimum feature means is not greater than the threshold, then the parent region remains as a whole region. The Adaptive Fast-merge merge criterion is used in the merge process.

Our test image M2 (given in Chapter 4) is a 4-region textured image that is 512 × 512 pixels in size. Using 8 × 8 pixel blocks to compute feature values, the initial graph for the pure merge approach has 4096 vertices. After the split process in the split-and-merge approach, we have 1294 regions. That is, on this simple image that has only a few large regions, the split process has reduced the number of vertices by about 3 times. On any real images with many moderately sized regions, we would expect a much smaller reduction of graph size. Also, the smaller the reduction on vertices, the longer are the split process and the conversion of quadtree to graph. This is because it requires more split operations to result in a deep quadtree, and the resulted large graph would also take a longer time to build. Unless the rate of split is many times higher than the rate of merge, the split phase may not serve to reduce the overall segmentation time. In our algorithm where the speed of the merge process has been greatly improved by the Adaptive Fast-merge criterion, a split process is less useful and may even slow down the segmentation process.

In our experiment, both approaches take about 8.3 seconds to compute the feature values, and 0.1 second to save the results. In the pure merge approach, graph building is incorporated into the feature computation process. The merge process takes about 0.68 second to complete. As a result, the total segmentation time for the pure merge approach is about 9.08 seconds. In split-and-merge, the split process requires 0.19 second, it then takes 0.30 second to convert the quadtree to a graph, and the merge process requires a further 0.34 second to complete. Therefore, a total of 9.23 seconds is needed for the split-and-merge process.

Although the above result is not sufficient to conclude that split-and-merge is always slower than pure merge region growing, the following two observations are noteworthy:

- Feature computation in textured image segmentation takes up a large proportion of the total processing time. Unless the split process can also lower feature computation cost, it will have little positive impact on the total segmentation time.
- The conversion of quadtree to graph is a nontrivial process. To reduce total processing time, any speed gained by the split process would have to be large enough to offset the time taken for data structure conversion.

The above comparison on sequential pure merge and split-and-merge processes is also applicable to their parallel implementations. In addition, a multiple-instruction multiple-data stream (MIMD) split-and-merge process has the following drawbacks:

In a MIMD model, no communication among processors or access to shared memory is required in the split phase. However, converting a forest of quadtrees (one quadtree per processor) to a global graph is a nontrivial task that requires communication between processors sharing common image division boundaries. This data structure conversion process is more expensive than the building of a global graph from the image, because in the latter, all primitive regions have the same size and shape, and the resulting graph always has a regular grid topology.

Also, compared to the pure merge approach, an extra process synchronization point is required after the split phase in split-and-merge. In parallel processing, such a global synchronization almost always increases processor idle times and the total processing time.

If the merge process is well parallelized as in our algorithm, and a sufficient number of processors is used so that the size of the partial graph at each processor is kept small, then little can be gained by introducing the split process. This is because a large initial graph is not a cause of long processing time in an efficient MIMD pure merge approach. In this case, a split phase is likely to slow down the segmentation process.

6.4 The Distributed Approach

A distributed system is a network of homogeneous or heterogeneous processors with each processor having its own memory unit that is not accessible by others. Communication between any two processors is achieved through message passing.

Nowadays, a distributed network of machines is often the most accessible system in a computing environment. Although it has a high processor communication cost compared to a shared memory or a massively parallel system, a distributed algorithm on such a network is useful because the supporting architecture is readily available. Besides harnessing the otherwise unused CPU cycles, problems with memory requirements which cannot be satisfied on single machines can take advantage of the large distributed memory space. In our study, two distributed region growing algorithms were implemented on the local network before the TC2000 shared memory version was implemented via wide area network access. Through the early distributed experiment, we gained a better understanding of the region growing problem which facilitated the later experiment on the TC2000.

In our implementation, one of the processors in the distributed system, called the host processor, is responsible for system initialization, input and output, load partitioning, data distribution, and results gathering. After reading an image file, the host partitions the image data into $N_1 \times N_2$ rectangular sub-images with small overlaps at the boundaries. Each sub-image is then sent to the corresponding processor in a grid layout. Every processor builds a subgraph on its image data, and exchanges edge information with processors having adjacent sub-images to setup the global graph. The resulting graph has two types of edges: local edges which connect vertices residing on the same processors, and inter-processor edges which connect vertices of different processors. Each processor maintains its vertex list. Region merging is an iterative process on individual processors. At the end of the merging phase, node information is gathered by the host and then written to a file. Due to the high communication cost in a distributed system, the sizes of initial partial graphs in individual processors have to be reasonably large in order to benefit from distributed processing.

In the next sections, we describe two distributed approaches: a complex but naive implementation and its improved version. In the following discussion, a merge that involves the update of at least one inter-processor edge is called a inter-processor merge. Otherwise, it is called a local merge. A local merge involves only vertices and edges residing on a single processor.

6.4.1 Inter-processor Merge Approach

Both inter-processor and local merges are performed with equal priority in this approach. It is a straightforward mapping of the region growing problem on a system of distributed processors.

To allow inter-processor merges in a distributed system, a sophisticated cooperation scheme is required to ensure deadlock-free processes. The host processor maintains a reservation table for keeping track of the status of all processors. A processor is either *available*, or it is *busy* participating in an inter-processor merge. Whenever a processor finds a pair of vertices that satisfy the merge criterion, it will check for any inter-processor edges that need to be updated in the merge. If inter-processor edges are involved, then a request with a list of participating processor names will be sent to the host. Depending on the availabilities of these processors, the host will either update the reservation table and grant the merge, or it will deny the request. The host is also responsible for informing all participating processors to get ready for the merge. The inter-processor merge will begin when all processors involved have completed their local merges in progress. Upon completing the inter-processor merge, the initiatory processor will inform the host to release all participated processors. This centralized communication scheme is adopted to prevent deadlock from occurring.

As expected, an inter-processor merge is very expensive compared to a local, intra-processor merge. The complexity of such a scheme makes the algorithm difficult to debug and implement. Also, due to the high communication cost on a network, the resulting program is very inefficient.

Although it is obvious that minimizing inter-processor edges will help to improve performance, finding a minimum cost partition for an irregular graph is a very difficult problem. In region growing, since the graph topology changes with every merge operation, maintaining a low cost partition is even more intractable.

6.4.2 Local Merge Approach

The local merge approach compromises the fair region growth policy for processing speed. Only local merges are allowed in this approach. It should be noted that each pair of merging regions would still have to satisfy the Fixed Threshold test and a selection policy. Experimental results show no visible differences between segmentations from the sequential and the local merge distributed processes.

Whenever a processor encounters a possible merge that involves inter-processor edges, the merge will be abandoned and the next vertex in the list will be examined. The iterative process continues until no more local merges are possible. The host is notified of the completion of a local process, then the partial graph of this local process is transferred to the host. After all processors have completed their local merges and data transfers, a sequential merge process will be initiated at the host to complete the segmentation.

This approach is very simple and efficient compared to the inter-processor merge approach. There is no communication among peer processors. The host communicates with the other processors only during the initial data distribution and the final results gathering phases. The local merge approach improves over the inter-processor merge approach by minimizing on communication cost.

6.4.3 Distributed Timing Results

In this section, we examine the timing performance of the local merge distributed approach on a network of workstations. Each workstation is a SPARC IPC with 8 megabytes of memory. The workstation network is connected by the Ethernet. Two local merge algorithms, using the Fast-merge Fixed Threshold and the existing Best-merge Fixed Threshold [67] merge criteria, are compared in this experiment. In this section, we will simply refer to them as the Fast-merge and the Best-merge algorithms.

Two sets of grayscale images are used, with five images in each set. Those in the first set are named M7 to M11, and each of them is 512 × 512 pixels in size. These image are shown in Figures D.50 to D.54 in Appendix D. Images in the second set, MS7 to MS11, have the same contents as those in the first. However, each image in this set is 256 × 256 in size. In the later discussion, we will refer to the two sets as the large and small images.

Due to the high memory usage of the region growing algorithms, a sequential or even a two-processor distributed process on a 512 × 512 image will cause a high degree of paging. Therefore, the second set of 256 × 256 images is included in the experiment so that timing results that are not effected by paging can be assessed. Nevertheless, since images larger than 512 × 512 pixels are widely used in image applications, we need to refer to the large image set for realistic timing results.

Tables 6.15 to 6.18 give the distributed processing times (t_n, in seconds) on the images using different numbers of processors (n). To further examine the timing performance gain of the algorithms, we provide the sequential to distributed timing ratio ($\frac{t_1}{t_n}$) on the given image set. The results indicate that the processing time of a Best-merge segmentation is image-dependent – different images require very different processing times, and segmentations of some images are more parallelizable than others. On the contrary, the Fast-merge timing differences between images are small. Also, the change in timing with the number of processors is steady and predictable using the Fast-merge algorithm.

In Figures 6.42 to 6.45, we show the change in processing time and timing ratio with the number of processors. Figures 6.42 and 6.43 present the Fast-merge and Best-merge results on image M7. The latter two figures present those on the small image MS7.

For the large images, the excellent timing ratios achieved on small numbers of processors are partly due to the high levels of paging incurred in the single-processor segmentations. The paging costs are especially high on the sequential Best-merge processes due to the large number of merge rejections. Using more than six processors would not further improve processing speed because the communication cost also increases with the number of processors. For our given problem size, the appropriate number of processors to use is six.

Tables 6.19 and 6.20 give the breakdowns on processing time of the local merge approach. By categorizing every phase or activity of a distributed process into distributed, serial, and communication, it helps to provide a better understanding on the behavior of the algorithms. In the local merge approach, only the distributed merge phase is categorized as "distributed". "Serial" activities include file input and output, and the final merge process performed by the host. The initial data distribution by the host and the final results gathering from the distributed processes are classified as "communication". In the two tables, the percentages of time spent on these three types of activities are denoted by p_d, p_s, and p_c, respectively. Figures 6.46 and 6.47 illustrate the p_d, p_s, and p_c timings of two 4-processor runs of the Best-merge and Fast-merge algorithms, respectively.

From Table 6.20, it is clear that most Best-merge processes spend higher percentages of time on the sequential merge phase than on the distributed merge phase. This re-confirms our earlier conclusion about Best-merge in Chapter 4, that the Best-merge merge sequence is highly ordered and few merges can be performed independently at the same time.

In Table 6.19, a Fast-merge process has a relatively long distributed merge phase compared to its sequential merge phase. This implies that a large proportion of the segmentation work is done by the distributed merge processes using the Fast-merge approach. Nevertheless, due to the high communication cost in the distributed system, as the number of processors increases, the communication overhead will gradually offset the performance gain of the distributed merge process. Also, since the local merge distributed algorithm is not completely parallelizable (there is always a sequential merge phase), the achievable performance gain is limited.

On the Fast-merge and Best-merge segmentations of image M7, the change in p_d, p_s, and p_c with the number of processors can be observed in Figures 6.48 and 6.49.

Table 6.15: Fast-merge timing (in seconds) on a set of 512×512 grayscale images

PEs	M7		M8		M9		M10		M11	
n	t_n	$\frac{t_1}{t_n}$	t_n	$\frac{t_1}{t_n}$	t_n	$\frac{t_1}{t_n}$	t_n	$\frac{t_1}{t_n}$	t_n	$\frac{t_1}{t_n}$
1	91.6	-	96.6	-	89.4	-	86.7	-	92.1	-
2	59.1	1.5	60.5	1.6	60.5	1.5	55.1	1.6	63.2	1.5
4	26.0	3.5	26.5	3.6	25.6	3.5	23.6	3.7	29.1	3.2
6	23.8	3.8	24.5	3.9	23.5	3.8	21.4	4.1	26.9	3.4
8	23.8	3.8	24.8	3.9	24.1	3.7	22.8	3.8	26.7	3.4
12	28.3	3.2	27.7	3.5	28.3	3.2	26.5	3.3	29.0	3.2
16	27.5	3.3	27.6	3.5	27.5	3.3	26.5	3.3	27.8	3.3

Table 6.16: Best-merge timing (in seconds) on a set of 512×512 grayscale images

PEs	M7		M8		M9		M10		M11	
n	t_n	$\frac{t_1}{t_n}$	t_n	$\frac{t_1}{t_n}$	t_n	$\frac{t_1}{t_n}$	t_n	$\frac{t_1}{t_n}$	t_n	$\frac{t_1}{t_n}$
1	683.9	-	401.4	-	218.7	-	171.1	-	8324.4	-
2	346.6	2.0	190.5	2.1	147.0	1.5	112.6	1.5	3456.9	2.4
4	220.0	3.1	196.5	2.0	89.5	2.4	36.4	4.7	3095.3	2.7
6	416.4	1.6	157.5	2.5	117.6	1.9	37.2	4.6	3268.5	2.5
8	206.6	3.3	162.5	2.5	87.2	2.5	41.0	4.2	5940.4	1.4
12	182.2	3.8	76.4	5.3	125.4	1.7	49.4	3.5	3200.1	2.6
16	313.7	2.2	96.8	4.1	93.1	2.3	51.7	3.3	8993.8	0.9

Table 6.17: Fast-merge timing (in seconds) on a set of 256×256 grayscale images

PEs	MS7		MS8		MS9		MS10		MS11	
n	t_n	$\frac{t_1}{t_n}$	t_n	$\frac{t_1}{t_n}$	t_n	$\frac{t_1}{t_n}$	t_n	$\frac{t_1}{t_n}$	t_n	$\frac{t_1}{t_n}$
1	12.4	-	11.8	-	9.5	-	9.1	-	13.3	-
2	8.3	1.5	8.9	1.3	10.0	0.9	9.3	1.0	9.8	1.4
4	6.6	1.9	7.3	1.6	8.5	1.1	7.6	1.2	8.1	1.6
6	6.4	1.9	7.4	1.6	8.3	1.1	7.8	1.2	7.9	1.7
8	6.9	1.8	7.8	1.5	8.2	1.2	8.4	1.1	8.4	1.6

Table 6.18: Best-merge timing (in seconds) on a set of 256×256 grayscale images

PEs	MS7		MS8		MS9		MS10		MS11	
n	t_n	$\frac{t_1}{t_n}$	t_n	$\frac{t_1}{t_n}$	t_n	$\frac{t_1}{t_n}$	t_n	$\frac{t_1}{t_n}$	t_n	$\frac{t_1}{t_n}$
1	44.4	-	104.4	-	13.9	-	11.7	-	134.2	-
2	42.0	1.1	90.7	1.2	14.0	1.0	11.8	1.0	81.6	1.6
4	25.4	1.7	86.9	1.2	12.3	1.1	10.3	1.1	50.4	2.7
6	35.6	1.2	82.4	1.3	12.4	1.1	11.5	1.0	61.0	2.2
8	33.3	1.3	86.3	1.2	13.0	1.1	13.1	0.9	54.8	2.4

Table 6.19: Fast-merge timing percentages on distributed (p_d), serial (p_s), and communication (p_c) subprocesses

PEs	M7			M8			M9			M10			M11		
	p_d	p_s	p_c	p_d	p_s	p_c	p_d	p_s	p_c	p_d	p_s	p_c	p_d	p_s	p_c
2	94	5	1	88	6	6	87	5	8	84	5	11	82	5	13
4	57	14	29	52	15	33	50	13	37	62	11	27	49	13	38
6	38	16	46	35	18	47	38	16	46	36	13	51	34	15	51
8	28	17	55	26	19	55	27	16	57	25	13	62	26	16	58
12	16	16	68	16	18	66	16	15	69	15	13	72	16	17	67
16	12	16	72	12	18	70	12	15	73	11	13	76	13	16	71

Table 6.20: Best-merge timing percentages on distributed (p_d), serial (p_s), and communication (p_c) subprocesses

PEs	M7			M8			M9			M10			M11		
	p_d	p_s	p_c	p_d	p_s	p_c	p_d	p_s	p_c	p_d	p_s	p_c	p_d	p_s	p_c
2	58	41	1	65	28	7	92	4	4	92	3	5	20	79	1
4	12	78	10	32	55	13	45	29	26	46	12	42	10	88	2
6	4	91	5	36	44	20	41	33	26	27	16	57	3	96	1
8	6	82	12	7	73	20	34	36	30	18	16	66	2	98	0
12	5	77	18	10	38	52	9	60	31	10	17	73	1	98	1
16	2	86	12	7	50	43	14	45	41	7	15	78	0	99	1

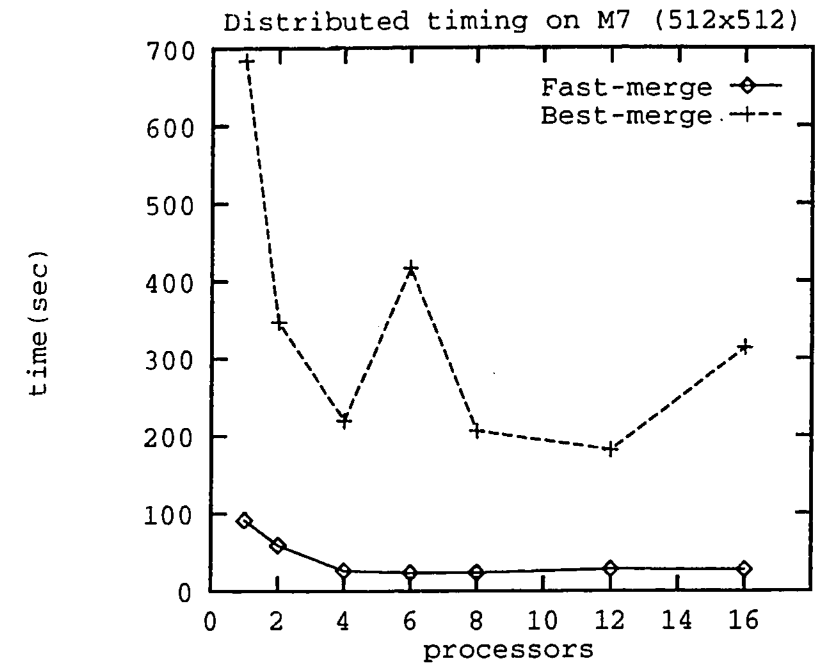


Figure 6.42: Distributed processing time on M7

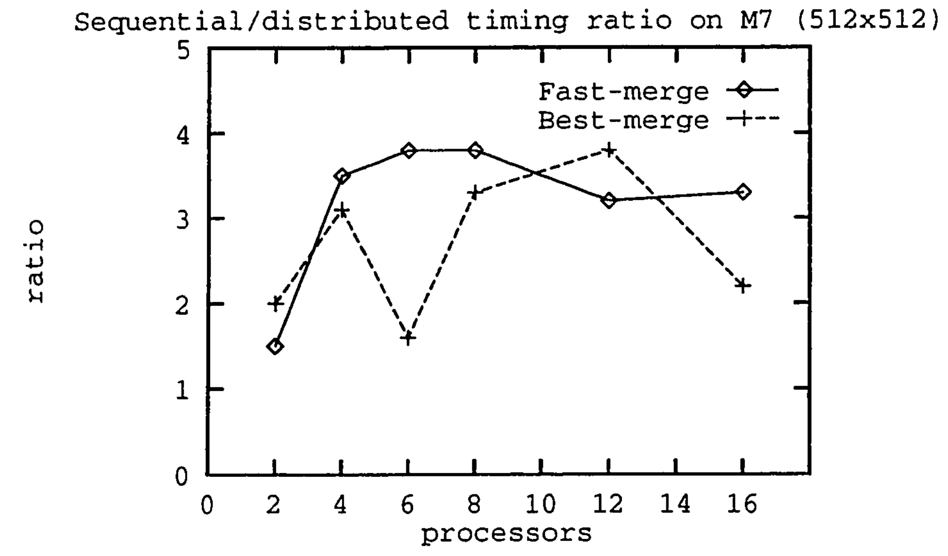


Figure 6.43: Distributed timing ratio on M7

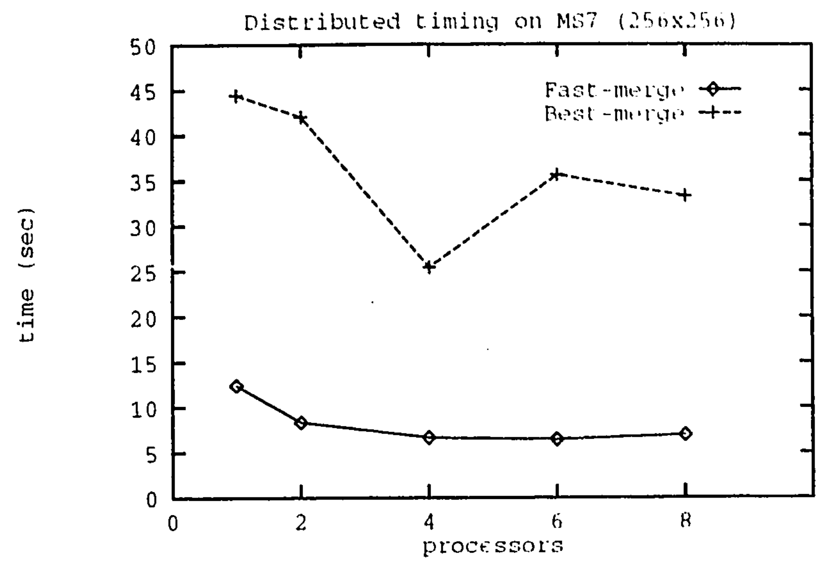


Figure 6.44: Distributed processing time on MS7

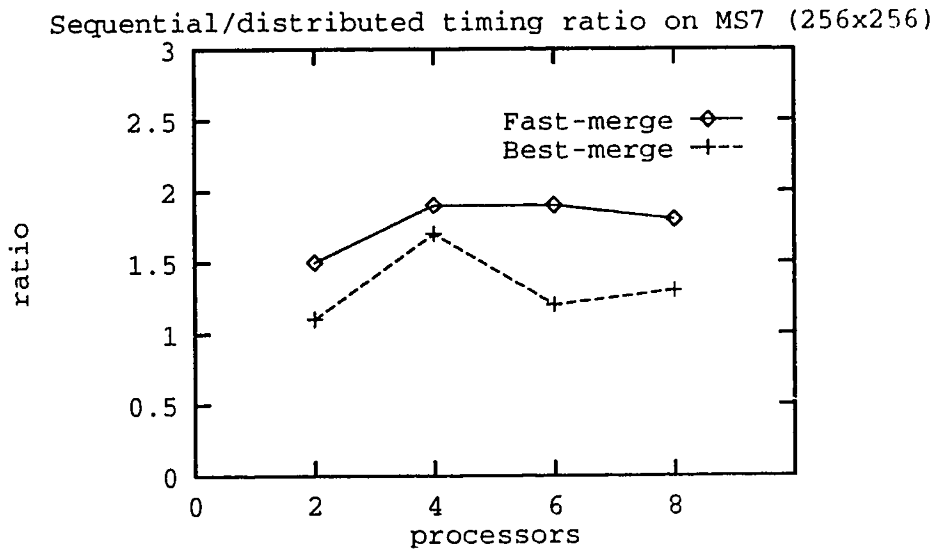


Figure 6.45: Distributed timing ratio on MS7

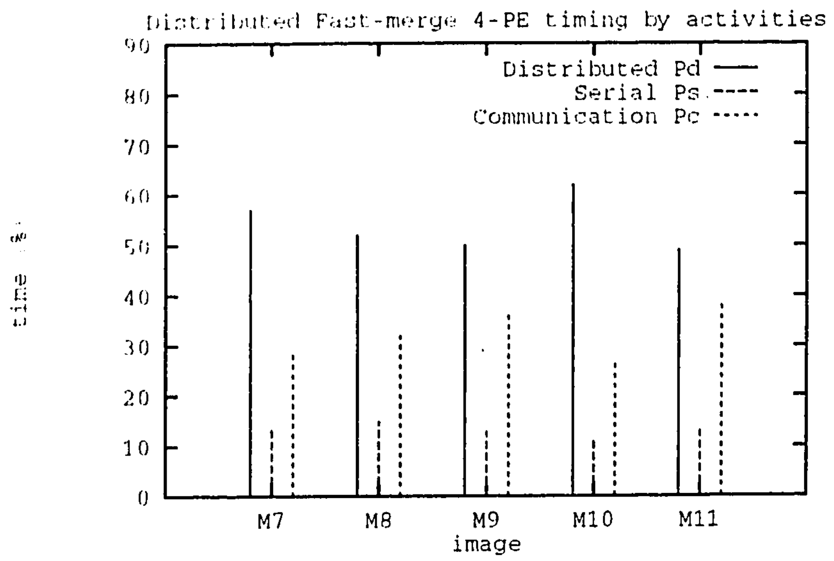


Figure 6.46: Distributed Fast-merge timing analysis

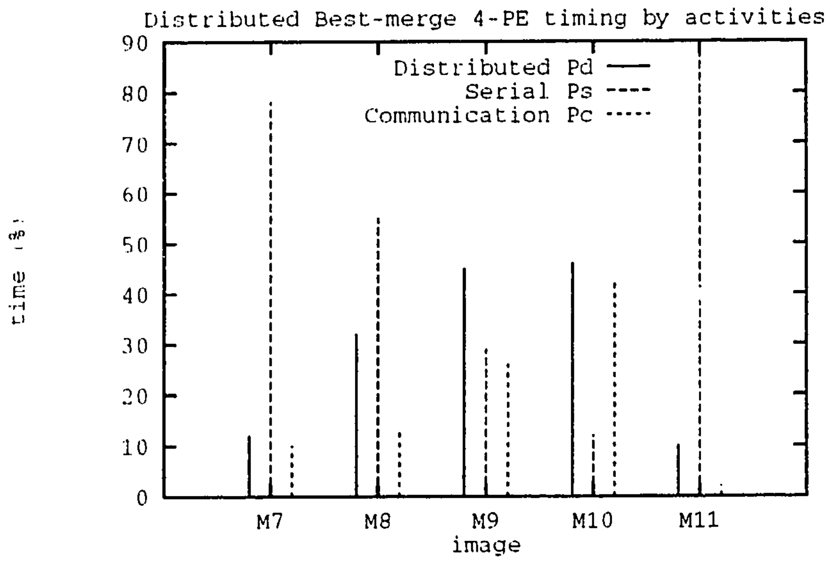


Figure 6.47: Distributed Best-merge timing analysis

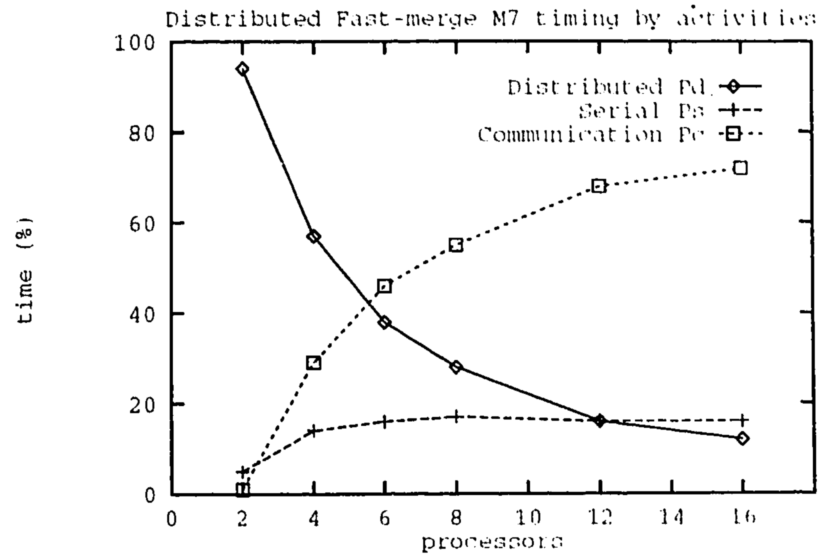


Figure 6.48: Distributed Fast-merge timing analysis on M7

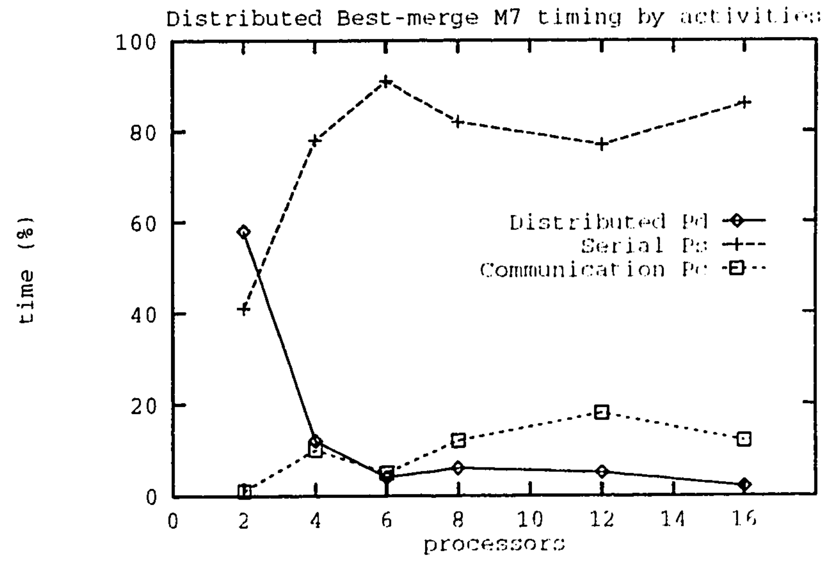


Figure 6.49: Distributed Best-merge timing analysis on M7

Chapter 7

Conclusions and Future Work

In this study, we proposed a new region merging decision test called the Fast-merge Adaptive criterion. Our region growing method, called the Fast Adaptive Segmentation (FAS) algorithm, is a simple and general method that has proven to be superior to the existing Best-merge Fixed Threshold algorithm. Through theoretical analysis and empirical studies, we have shown that the FAS algorithm is robust and produces high quality segmentations.

In terms of processing speed, regardless of image characteristics such as region sizes, shapes and feature distributions, the FAS algorithm provides consistently fast segmentations. In addition, the FAS algorithm is amenable to parallelization. The shared memory SPMD implementation of the algorithm has produced good timing performance.

We summarize the main contributions of this study as follows:

- *Proposed a simple and effective adaptive homogeneity test for region growing.*
  The proposed Adaptive homogeneity test based merge decision on feature distributions of the merging regions. The performance of the Adaptive test has been proven to be superior to the widely used Fixed Threshold test. Its effectiveness is also substantiated by extensive experiments on grayscale and textured images. The Adaptive test is simple, easy to apply, and efficient to implement compared to existing methods that use fixed thresholds or varying thresholds in their segmentation processes.

- *Achieved consistent, fast sequential segmentations on a wide range of images.*
  We identified the performance bottleneck of the existing Best-merge selection policy, and proposed a new Fast-merge policy that not only drastically improves on segmentation speed, but also helps to produce segmentations of better quality. Moreover, experimental results showed that the processing speed of our algorithm is not sensitive to image characteristics.

- *Proposed an efficient parallel region growing algorithm*
  Due to the Fast-merge policy, our region growing algorithm results in a segmentation process with a high degree of parallelism, and a well balanced load among parallel processes.

- *Provided detailed, comprehensive studies of merge criteria and parallel region growing.*
  Although the segmentation problem has been studied for decades, there has not been a detailed study of region growing merge criteria. Studies on adaptive segmentation have mostly been restricted to domain-specific images. Also, existing experiments on parallel segmentation have provided little insight into the timing performance and segmentation quality of their algorithms.

- *Provided a general region growing framework that is easy to use and can be easily adapted for specific applications.*
  The segmentation algorithm that we have proposed does not apply only to specific types of images. It is a general segmentation framework which can be adapted to different image applications. Unlike methods which require the use of user-defined thresholds, there is no need for parameter tuning in our algorithm.

A limitation of the FAS algorithm is the applicability of the Adaptive homogeneity test on very small regions. The Adaptive ranges for merging two regions are determined from the regions' feature histograms. On very small regions containing very few feature values, reliable Adaptive ranges cannot be determined. In our implementation, we use only the Fast-merge policy to merge regions to a minimum size, then apply the nonparametric Mann-Whitney test [15] to merge regions to another minimum size where the Adaptive test can be reliably employed. Another undesirable characteristic of the FAS algorithm is the order dependency of its segmentation results. Different parallel executions on the same image may result in slightly different segmentations due to the different merging orders adopted by the processes. In our study, however, such differences in segmentation results have shown to be minute and insignificant.

In the course of this project, we have directly or indirectly come across several related problems that worth further investigation. These problems are listed as follows:

- Although image segmentation is usually performed at the low-level processing stage where *a priori* knowledge about objects in a scene is not available, in certain applications, such as the inspection of X-ray images, some domain-specific information is available before the segmentation process. The incorporation of *a priori* knowledge in the region growing algorithm can help to produce more accurate segmentations.

- In order to achieve a good control on the process, region growing has been restricted to binary merges. If possible, multiple-region merges will likely to improve on processing speed.

- We have performed a preliminary experiment on using multiple-features in region growing. A more comprehensive study on multiple-feature merge criterion will further generalize the application of our region growing algorithm.

- The memory requirement of a region growing process is high. Therefore, for the segmentation of a moderately sized image, the degree of paging between memory hierarchy levels can severely degrade timing performance. The paging problem may be alleviated by defining a suitable mapping from a two-dimensional image to the linear memory address space, coordinated with an appropriate merging order.

Appendix A

The Co-occurrence Texture Measure

Due to its good discrimination power for many texture types, the co-occurrence texture measure is widely used in segmentation implementations [10, 40, 55]. The use of spatial distribution and dependence of pixel gray-tone to measure texture was first proposed by Julesz in 1962 [36]. Haralick *et al.* later extended the idea to two-dimensional spatial dependence of gray-tones in a co-occurrence matrix for each fixed distance and angular spatial relationship [29, 26]. In a recent study, Ohanian and Dubes [46] demonstrated that co-occurrence features are more effective than Gabor filters, Markov Random Field, and Fractal features for the classification of natural textures such as those from the Brodatz album [4]. Other performance comparisons of co-occurrence and other texture features can be found in [21, 58].

We now give the definition of the co-occurrence matrix [26]. Each entry $p(i,j)$ in a matrix $P_{d,\theta}$ denotes the number of pixel pairs with gray-tone $i,j = 0,...,m-1$, separated at a spatial distance d at angle θ. A set of fourteen textural features based on visual textural characteristics, statistics, information theory and information measures of correlation are then derived from matrices with various d and θ values.

Let $p'(i,j) = \frac{1}{M}p(i,j)$ be an entry of the normalized co-occurrence matrix, where $M = \sum_{i=0}^{m-1} \sum_{j=0}^{m-1} p(i,j)$ is the normalization constant. Definitions of the features used in this study are given below.

- **Angular second-moment**

$$f1 = \sum_{i=0}^{m-1} \sum_{j=0}^{m-1} (p'(i,j))^2. \tag{A.1}$$

- **Contrast**

$$f2 = \sum_{n=0}^{m-1} n^2 (\sum_{i=0}^{m-1} \sum_{j=0}^{m-1} (p'(i,j))^2), \qquad |i-j| = n. \tag{A.2}$$

- **Correlation**

$$f3 = \frac{\sum_{i=0}^{m-1} \sum_{j=0}^{m-1} ijp'(i,j) - \mu_x \mu_y}{\sigma_x \sigma_y}, \tag{A.3}$$

  where μ_x, μ_y, σ_x, and σ_y are the means and standard deviations of the marginal distributions associated with $p'(i,j)$.

- **Inverse difference moment**

$$f4 = \sum_{i=0}^{m-1} \sum_{j=0}^{m-1} (1 + (i-j)^2)^{-1} p'(i,j). \tag{A.4}$$

- **Entropy**

$$f5 = \sum_{i=0}^{m-1} \sum_{j=0}^{m-1} p'(i,j) \log(p'(i,j)). \tag{A.5}$$

The textured images used in the experiments of this project are generated from the Brodatz texture album. The original images have 256 grayscale levels. We quantize them to 8 levels before computing a co-occurrence matrix from each non-overlapping window of size 8 × 8 or 16 × 16 pixels. Horizontal, vertical, diagonal ($\theta \in \{0, 45, 90, 135, 180, 225, 270, 315\}$) distance-1 ($d = 1$) neighbors were used in our experiments.

Appendix B

Normality Test on Texture Features

To substantiate the normal feature distribution assumption made in the homogeneity test analysis (Section 3.3), we perform the Shapiro-Wilk test for normality on texture features computed from a set of natural textures.

The Shapiro-Wilk test [57] is a sensitive goodness-of-fit test for normality. Some empirical studies [15] indicate that this test has good power in many situations when compared with many other tests of the composite hypothesis of normality, including the Lilliefors test and the chi-square test.

The data used in the Shapiro-Wilk test consists of a random sample $\{x_1, x_2, \ldots, x_n\}$ of size n associated with some unknown distribution function $F(x)$.

The test hypotheses are defined as follows:
$H_0 : F(x)$ is a normal distribution function with unspecified mean and variance.
$H_1 : F(x)$ is non normal.

The following procedure is used to compute the test statistic W:

1. Order the observations to obtain an ordered sample $y_1 \leq y_2 \leq \cdots \leq y_n$.
2. Compute $S^2 = \Sigma_{i=1}^{n}(y_i - \bar{y})^2 = \Sigma_{i=1}^{n}(x_i - \bar{x})^2$.

3. Compute $b = \Sigma_{i=1}^{n} a_i(y_{i+1} - y_i)$.
   The a_i's are coefficients given in a table.

4. Compute $W = b^2/S^2$.

The decision rule of the test is to reject H_0 at the level of significance α if W is less than the α quantile given in a table. For the a_i and W tables and other details of this test, please refer to [15] and [57].

A total of one hundred Shapiro-Wilk tests on texture features were performed; we examined five features on each of twenty textures from the Brodatz album [4]. The features tested are the co-occurrence energy, entropy, contrast, angular second-moment, and correlation (see Appendix A and [46]). In each test, fifty non-adjacent 16x16 pixel blocks are used to compute the feature measures. Since the objective of the normality test is to verify the "shape" of the distribution without the interference of noise in the texture data, these fifty values were ordered and a few outliers from each end of the sequence were excluded before the statistic W is computed. Table B.21 gives the number of rejections on each feature at two levels of significance. Each entry in the table is the result of twenty tests on the same feature from the different textures. We present two sets of results – with three and five outliers removed from each end of a sorted list of 50 values, leaving 44 and 40 samples for each test.

Table B.21: Shapiro-Wilk test results on texture features

Feature	Number of rejections*			
	44 samples		40 samples	
	$\alpha = 5\%$	$\alpha = 10\%$	$\alpha = 5\%$	$\alpha = 10\%$
Energy	0	1	0	0
Entropy	10	12	5	6
Contrast	0	0	0	0
Angular second-moment	2	2	2	2
Correlation	0	0	0	0

* Each column contains results of 100 tests.

From the test results, we conclude that among the five features tested, only the entropy feature has considerable high rejection rates. We also deduce that many of the rejected entropy distributions failed the test due to irregularities or noise in the textures rather than the "shape" of the distributions, since by increasing the number of excluded outliers, the number of rejections is reduced drastically.

Appendix C

Lemmas and Proofs

The following are the proofs of the lemmas given in Chapter 3:

Lemma 3.1 (Shift)

$$P_N(\mu, \sigma, u, v) = P_N(\mu - a, \sigma, u - a, v - a), \qquad a \in \Re.$$

Proof.

$$\begin{aligned} P_N(\mu, \sigma, u, v) &= \frac{1}{\sigma\sqrt{2\pi}} \int_u^v e^{-\frac{(x-\mu)^2}{2\sigma^2}} dx \\ &= \frac{1}{\sigma\sqrt{2\pi}} \int_{u-a}^{v-a} e^{-\frac{(y+a-\mu)^2}{2\sigma^2}} dy, \qquad x = y + a, \quad dx = dy, \\ &= \frac{1}{\sigma\sqrt{2\pi}} \int_{u-a}^{v-a} e^{-\frac{(y-(\mu-a))^2}{2\sigma^2}} dy \\ &= P_N(\mu - a, \sigma, u - a, v - a). \end{aligned}$$

□

Lemma 3.2 (Halves)

$$P_N(\mu, \sigma, \mu - \ell, \mu + \ell) = 2P_N(\mu, \sigma, \mu, \mu + \ell) = 2P_N(0, \sigma, 0, \ell).$$

Proof.

$$P_N(\mu, \sigma, \mu - \ell, \mu + \ell) = \frac{1}{\sigma\sqrt{2\pi}} \int_{\mu-\ell}^{\mu+\ell} e^{-\frac{(x-\mu)^2}{2\sigma^2}} dx$$

$$
\begin{aligned}
&= \frac{1}{\sigma\sqrt{2\pi}}\left(\int_{\mu-\ell}^{\mu} e^{-\frac{(x-\mu)^2}{2\sigma^2}}dx + \int_{\mu}^{\mu+\ell} e^{-\frac{(x-\mu)^2}{2\sigma^2}}dx\right)\\
&= P_N(\mu,\sigma,\mu-\ell,\mu) + P_N(\mu,\sigma,\mu,\mu+\ell)\\
&\overset{[3.1]}{=} P_N(0,\sigma,-\ell,0) + P_N(0,\sigma,0,\ell)\\
&= \frac{1}{\sigma\sqrt{2\pi}}\int_{-\ell}^{0} e^{-\frac{x^2}{2\sigma^2}}dx + P_N(0,\sigma,0,\ell)\\
&= -\frac{1}{\sigma\sqrt{2\pi}}\int_{0}^{-\ell} e^{-\frac{x^2}{2\sigma^2}}dx + P_N(0,\sigma,0,\ell)\\
&= \frac{1}{\sigma\sqrt{2\pi}}\int_{0}^{\ell} e^{-\frac{y^2}{2\sigma^2}}dy + P_N(0,\sigma,0,\ell)\\
&= 2P_N(0,\sigma,0,\ell).
\end{aligned}
$$

□

Lemma 3.3 (Negation)

$$P_N(\mu,\sigma,-\ell,\ell) = P_N(-\mu,\sigma,-\ell,\ell).$$

Proof.

$$
\begin{aligned}
P_N(\mu,\sigma,-\ell,\ell) &\overset{[3.1]}{=} P_N(0,\sigma,-\ell-\mu,\ell-\mu)\\
P_N(-\mu,\sigma,-\ell,\ell) &\overset{[3.1]}{=} P_N(0,\sigma,-\ell+\mu,\ell+\mu)\\
&= \frac{1}{\sigma\sqrt{2\pi}}\int_{-\ell+\mu}^{\ell+\mu} e^{-\frac{x^2}{2\sigma^2}}dx\\
&= -\frac{1}{\sigma\sqrt{2\pi}}\int_{\ell-\mu}^{-\ell-\mu} e^{-\frac{y^2}{2\sigma^2}}dy, \qquad y=-x, \quad dy=-dx,\\
&= \frac{1}{\sigma\sqrt{2\pi}}\int_{-\ell-\mu}^{\ell-\mu} e^{-\frac{y^2}{2\sigma^2}}dy\\
&= P_N(0,\sigma,-\ell-\mu,\ell-\mu).
\end{aligned}
$$

□

Lemma 3.4 (Conversion to Standard Normal)

$$P_N(0,\sigma,0,\ell) = P_N(0,1,0,\frac{\ell}{\sigma}) \overset{\text{def}}{=} S(\frac{\ell}{\sigma}).$$

Proof.

$$P_N(0,\sigma,0,\ell) = \frac{1}{\sigma\sqrt{2\pi}}\int_{0}^{\ell} e^{-\frac{x^2}{2\sigma^2}}dx$$

$$
\begin{aligned}
&= \frac{1}{\sigma\sqrt{2\pi}}\int_0^{\ell/\sigma} e^{-\frac{(\sigma y)^2}{2\sigma^2}}\sigma dy, && x = \sigma y, \quad dx = \sigma dy, \\
&= \frac{1}{\sqrt{2\pi}}\int_0^{\ell/\sigma} e^{-\frac{y^2}{2}}dy \\
&= P_N(0,1,0,\ell/\sigma).
\end{aligned}
$$

□

Lemma 3.5 (Conversion to Exponential)

$$
P_N(\mu, u\sigma, -v\ell, v\ell) = \frac{1}{\sqrt{\pi}}\int_{\frac{-v\ell-\mu}{\sqrt{2}u\sigma}}^{\frac{v\ell-\mu}{\sqrt{2}u\sigma}} e^{-y^2}dy \overset{\text{def}}{=} \frac{1}{\sqrt{\pi}}P_E\left(\frac{-v\ell-\mu}{\sqrt{2}u\sigma}, \frac{v\ell-\mu}{\sqrt{2}u\sigma}\right).
$$

Proof.

$$
\begin{aligned}
P_N(\mu, u\sigma, -v\ell, v\ell) &= \frac{1}{u\sigma\sqrt{2\pi}}\int_{-v\ell}^{v\ell} e^{-\frac{(x-\mu)^2}{2(u\sigma)^2}}dx \\
&= \frac{1}{\sqrt{\pi}}\int_{-v\ell}^{v\ell} e^{-(\frac{x-\mu}{\sqrt{2}u\sigma})^2}\frac{1}{u\sigma\sqrt{2}}dx \\
&= \frac{1}{\sqrt{\pi}}\int_{\frac{-v\ell-\mu}{\sqrt{2}u\sigma}}^{\frac{v\ell-\mu}{\sqrt{2}u\sigma}} e^{-y^2}dy, && y = \frac{x-\mu}{u\sigma\sqrt{2}}, \quad dy = \frac{1}{u\sigma\sqrt{2}}dx.
\end{aligned}
$$

□

Definition C.1 (Exponential Function)

Let $f(x) = e^{-x^2}, \qquad x \in \Re.$

We state without proof the following properties of $f(x)$:

1. *$f(x)$ is continuous for $-\infty < x < \infty$.*
2. *$f(x) > 0$ for $-\infty < x < \infty$.*
3. $\lim_{x\to\infty} f(x) = \lim_{x\to-\infty} f(x) = 0$.
4. *$f(x) < f(0) = 1$ for any $x \neq 0$.*
5. *$f(x)$ is monotonously increasing over $x \in (-\infty, 0)$.*
6. *$f(x)$ is monotonously decreasing over $x \in (0, \infty)$.*
7. *$f(x) = f(-x)$ for $-\infty < x < \infty$.*

Lemma 3.6

$$P_N(0, \frac{\sigma}{n}, 0, \ell) > P_N(0, \sigma, 0, \ell), \qquad n > 1.$$

Proof.

$$\begin{aligned} P_N(0, \frac{\sigma}{n}, 0, \ell) &\overset{[3.5]}{=} \frac{1}{\sqrt{\pi}} P_E(0, \frac{n\ell}{\sqrt{2}\sigma}) = \frac{1}{\sqrt{\pi}} \int_0^{\frac{n\ell}{\sqrt{2}\sigma}} e^{-x^2} dx. \\ P_N(0, \sigma, 0, \ell) &\overset{[3.5]}{=} \frac{1}{\sqrt{\pi}} P_E(0, \frac{\ell}{\sqrt{2}\sigma}) = \frac{1}{\sqrt{\pi}} \int_0^{\frac{\ell}{\sqrt{2}\sigma}} e^{-x^2} dx. \end{aligned}$$

Therefore,

$$\begin{aligned} \frac{n\ell}{\sqrt{2}\sigma} &> \frac{\ell}{\sqrt{2}\sigma}, \qquad n > 1. \\ \Rightarrow \int_0^{\frac{n\ell}{\sqrt{2}\sigma}} e^{-x^2} dx &> \int_0^{\frac{\ell}{\sqrt{2}\sigma}} e^{-x^2} dx, \\ \Rightarrow P_N(0, \frac{\sigma}{n}, 0, \ell) &> P_N(0, \sigma, 0, \ell). \end{aligned}$$

□

Lemma 3.7

If $\sigma_1 > \sigma_2$ and $P_N(0, \sigma_1, 0, \ell_1) = P_N(0, \sigma_2, 0, \ell_2)$, then $\ell_1 > \ell_2$.

Proof.

If $\sigma_1 > \sigma_2$, by Lemma 3.6 $P_N(0, \sigma_2, 0, \ell_1) > P_N(0, \sigma_1, 0, \ell_1)$.
$\Rightarrow P_N(0, \sigma_2, 0, \ell_1) > P_N(0, \sigma_2, 0, \ell_2)$
$\Rightarrow \ell_1 > \ell_2$.

□

Lemma 3.8

$$P_E(a, d) > P_E(b, c) \qquad \text{if} \quad a \le b \le c < d.$$

Proof.

$$\begin{aligned} P_E(a, d) - P_E(b, c) &= \int_a^d e^{-x^2} dx - \int_b^c e^{-x^2} dx \\ &= \int_a^b e^{-x^2} dx + \int_c^d e^{-x^2} dx \qquad \text{by Def. C.1.1, C.1.2,} \\ &> 0. \end{aligned}$$

□

Lemma 3.9

$$P_E(a,b) < P_E(c,d) \qquad \text{if} \quad a < b \le c < d \le 0, \quad d-c \ge b-a.$$

Proof.

Let $c < e < d$ such that $d - e = b - a$.

From Def. C.1.1, C.1.5, and since $a < b \le c < d \le 0$ we have $\int_a^b e^{-x^2}dx < \int_e^d e^{-x^2}dx$.

$$\begin{aligned} P_E(c,d) - P_E(a,b) &= \int_c^d e^{-x^2}dx - \int_a^b e^{-x^2}dx \\ &= \int_c^e e^{-x^2}dx + \int_e^d e^{-x^2}dx - \int_a^b e^{-x^2}dx \\ &> \int_c^e e^{-x^2}dx > 0. \end{aligned}$$

□

Lemma 3.10

$$P_E(a,b) < P_E(c,d) \qquad \text{if} \quad a < b \le 0, \quad c \le 0 < d, \quad d-c \ge b-a.$$

Proof.

Let $e < c$ such that $d = c - e$, i.e. $0 - e = d - c$.

$$\begin{aligned} P_E(c,d) - P_E(e,0) &= \int_c^d e^{-x^2}dx - \int_e^0 e^{-x^2}dx \\ &= (\int_c^0 e^{-x^2}dx + \int_0^d e^{-x^2}dx) - (\int_e^c e^{-x^2}dx + \int_c^0 e^{-x^2}dx) \\ &= \int_{-d}^0 e^{-x^2}dx - \int_e^c e^{-x^2}dx && \text{by Def. C.1.7,} \\ &\ge 0 && \text{by Def. C.1.5,} \ d = c - e. \end{aligned}$$

By Lemma 3.9, $P_E(e,0) > P_E(a,b) \quad \Rightarrow P_E(c,d) > P_E(a,b)$. □

Lemma 3.11

$$P_E(a,b) < P_E(c,d) \ \text{ if } \ a < b < 0 < c < d, \ |b| > |c|, \ d-c \ge b-a.$$

Proof.

By Def. C.1.7, $P_E(c,d) = P_E(-d,-c)$.

We have $a < b < 0, \quad -d < -c < 0, \quad b < -c, \quad d - c \geq b - a.$

By Lemma 3.9, $P_E(-d, -c) > P_E(a, b).$

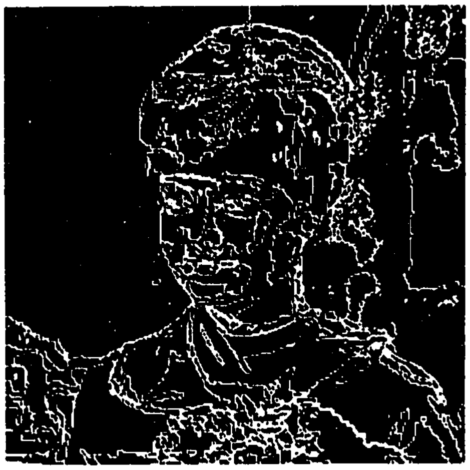
$\Rightarrow P_E(c, d) > P_E(a, b).$ □

Appendix D

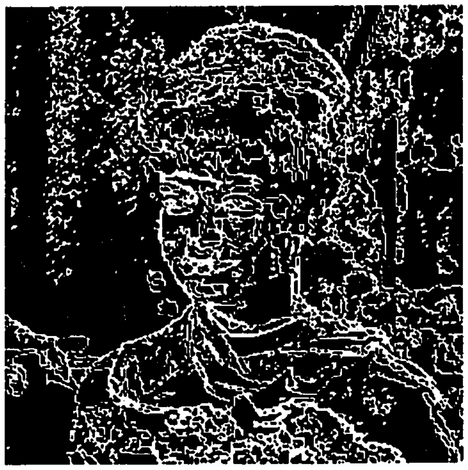
More Images and Segmentations

A set of five 512 × 512 images, which represents different domains of natural scences, were selected for our timing experiments on the TC2000 and the distributed SPARC workstation network described in Chapter 5 and Section 6.4.3 of Chapter 6, respectively. These images, M7 to M11, are given in Figures D.50 to D.54. They are natural 256-level grayscale images of a portrait (M7), the moon's surface (M8), a chest X-ray (M9), an urban aerial view (M10), and text (M11). Pixel intensity is the only feature used in the segmentations of these images. The FAS and BFS segmentation results on this set of images are given in Figures D.50 to D.54. The BFS segmentations reported are the most satisfactory results among segmentations obtained by invoking the BFS process using different thresholds.

(a) Image M7

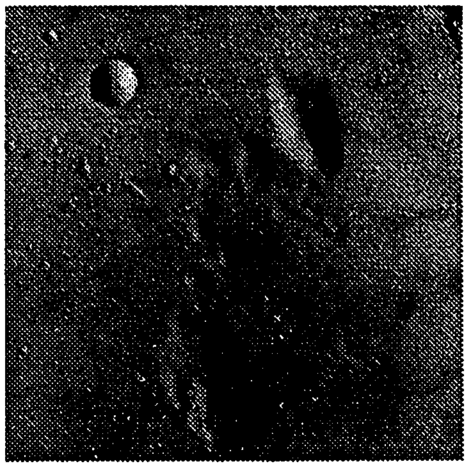


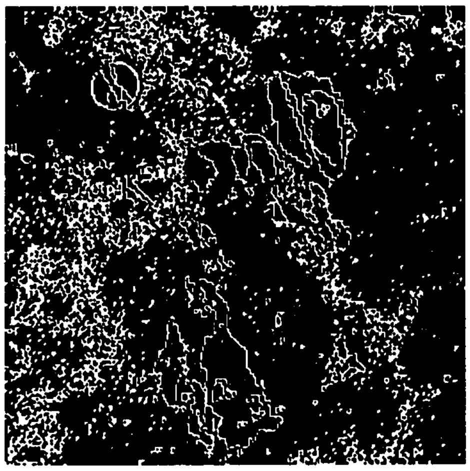
(b) FAS segmentation of M7



(c) BFS segmentation of M7

Figure D.50: M7 and segmentations


(a) Image M8


(b) FAS segmentation of M8

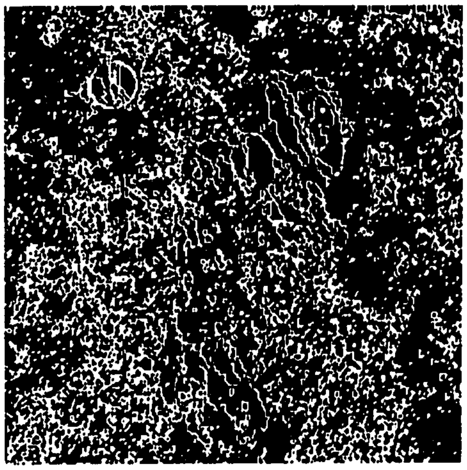
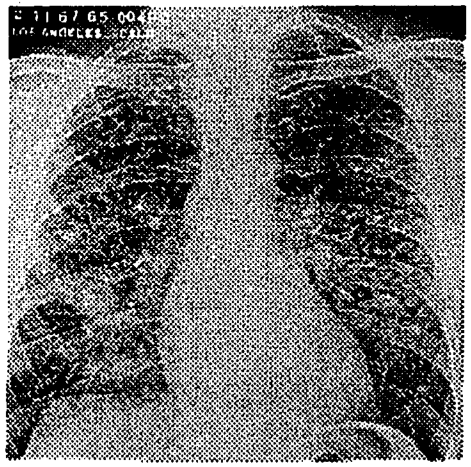
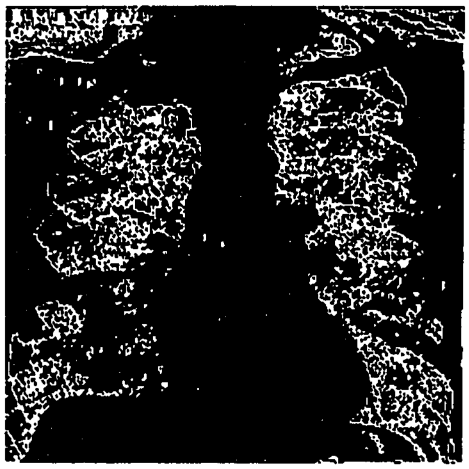

(c) BFS segmentation of M8

Figure D.51: M8 and segmentations


(a) Image M9


(b) FAS segmentation of M9

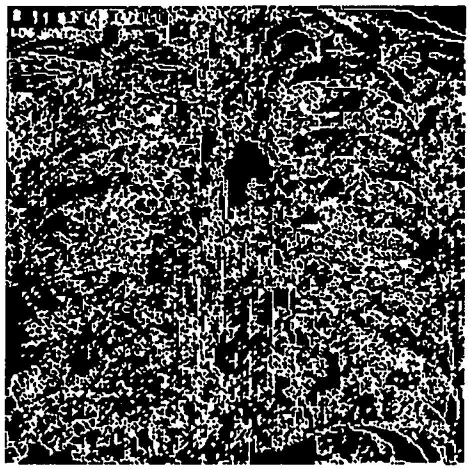
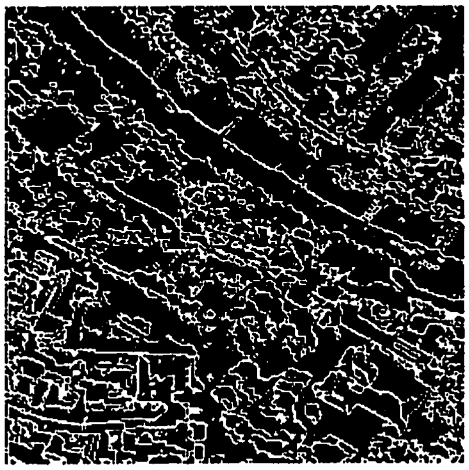
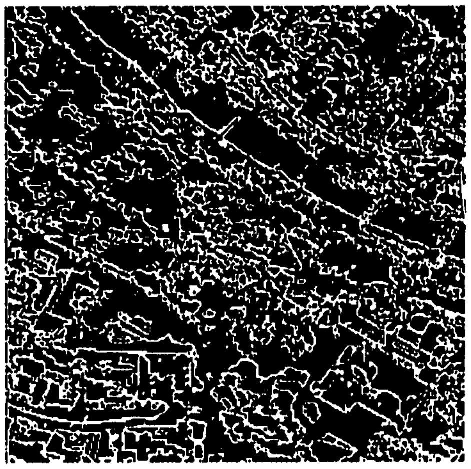

(c) BFS segmentation of M9

Figure D.52: M9 and segmentations

(a) Image M10

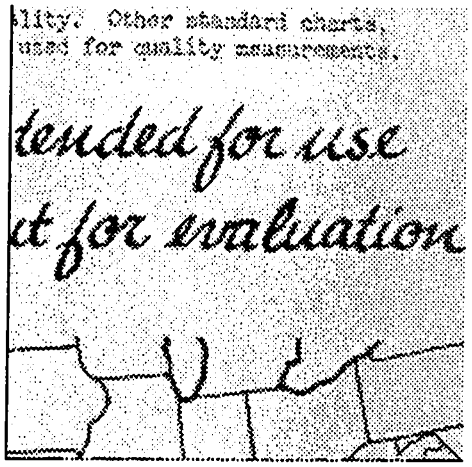


(b) FAS segmentation of M10

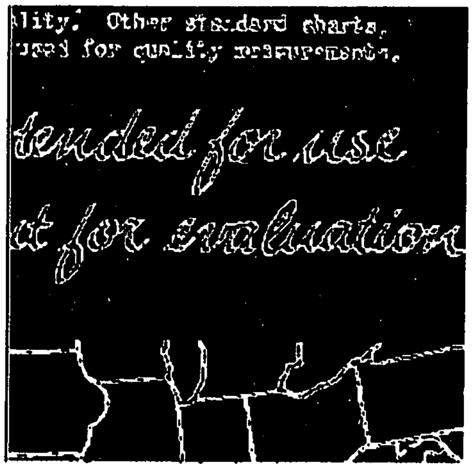


(c) BFS segmentation of M10

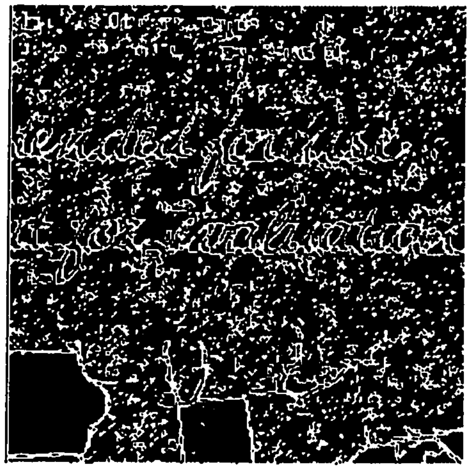
Figure D.53: M10 and segmentations



(a) Image M11



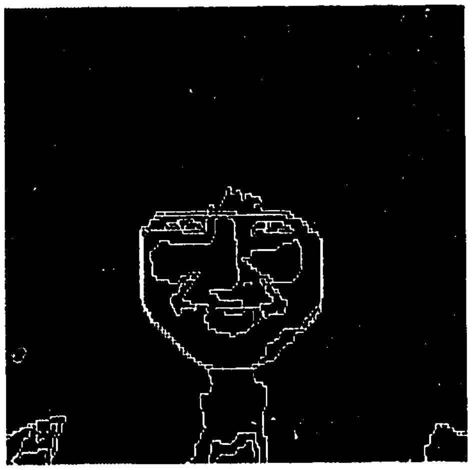
(b) FAS segmentation of M11



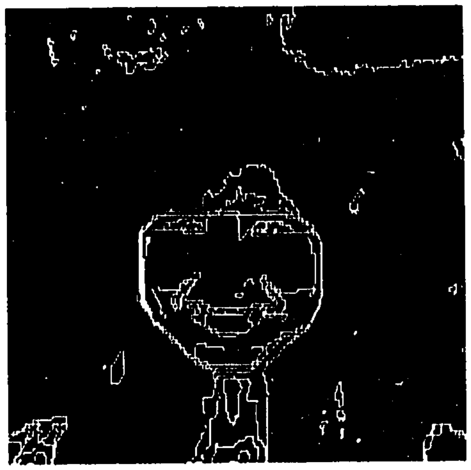
(c) BFS segmentation of M11

Figure D.54: M11 and segmentations

(a) Image M23, the author



(b) FAS segmentation of M23



(c) BFS segmentation of M23

Figure D.55: M23 and segmentations

Bibliography

[1] Aggarwal, R. K., and Bacus, J. W. A multi-spectral approach for scene analysis of cervical cytology smears. *J. Histochem. Cytochem.*, 25:668–680, 1977.

[2] Almasi, G. S., and Gottlieb, A. *Highly parallel computing.* Benjamin/Cummings, 1989.

[3] BBN Advanced Computers Inc. *Inside the TC2000 computer.* BBN Reference manual, 1990.

[4] Brodatz, P. *Textures.* Reinhold, NewYork, 1968.

[5] Brooks III, E. D., Gorda, B. C., and Warren, K. W. The parallel C preprocessor. *The 1992 MPCI Yearly Report: Harnessing the Killer Micros*, pages 58–68, 1992.

[6] Burt, P. J., and Rosenfeld, A. Segmentation and estimation of image region properties through cooperative hierarchical computation. *IEEE Trans. Systems Man Cybernet.*, 11(12):802–809, December 1981.

[7] Cahn, R. L., Poulsen, R. S., and Toussaint, G. Segmentation of cervical cell images. *J. Histochem. Cytochem.*, 25:681–688, 1977.

[8] Chellappa, R., and Rosenfeld, A. Computer vision: attitudes, barriers, counseling. *Proc. Vision Interface*, pages 1–7, 1992.

[9] Chen, M.-H., and Pavlidis, H. Image seaming for segmentation on parallel architecture. *IEEE Trans. Pattern Analy. Mach. Intell.*, 12(6):588–594, 1990.

[10] Chen, P. C., and Pavlidis, T. Segmentation by texture using a coocurrence matrix and a split-and-merge algorithm. *Comput. Graphics Image Process.*, 10:172–182, 1979.

[11] Chen, P. C., and Pavlidis, T. Image segmentation as an estimation problem. *Comput. Graphics Image Process.*, 12:153–172, 1980.

[12] Chen, S-Y, Lin, W-C, and Chen, C-T. Split-and-merge image segmentation based on localized feature analysis and statistical tests. *Comput. Vision Graphics Image Process.*, 53:457–475, 1991.

[13] Cohen, F. S., and Cooper A. B. Simple parallel hierarchical and relaxation algorithms for segmenting noncausal Markovian random fields. *IEEE Trans. Pattern Anal. Machine Intell.*, 9(2):195–219, March 1987.

[14] Coleman, G. B., and Andrews, H. C. Image segmentation by clustering. *Proc. IEEE*, pages 773–785, 1979.

[15] Conover, W. J. *Practical nonparametric statistics.* John Wiley & Sons, 1980.

[16] Coray, G., Noetzel, A., and Selkow, S. M. Order independence in local clustering algorithms. *Comput. Graphics Image Process.*, 4:120–132, 1975.

[17] Cross, G. R., and Jain, A. K. Markov random field texture models. *IEEE Trans. Pattern Anal. Machine Intell.*, 5(1):25–39, January 1983.

[18] Cypher, R., and Sanz, J. L. C. SIMD achitectures and algorithms for image processing. *IEEE Trans. Acoustics, Speech and Signal Proc.*, 37:2158–2173, 1989.

[19] DeGroot, M. H. *Probability and statistics.* Addison-Wesley, 1986.

[20] Derin, H., and Elliott, H. Modeling and segmentation of noisy and textured images using gibbs random fields. *IEEE Trans. Pattern Anal. Machine Intell.*, 9(1):39–55, January 1987.

[21] Du Buf, J. M. H., Kardan M., and Spann, M. Texture feature performance for image segmentation. *Pattern Recognition*, 23(3/4):291–309, 1990.

[22] Dubes, R. C., Jain, A. K., Nadabar, S. G., and Chen, C. C. MRF model-based algorithms for image segmentation. *Proc. ICPR*, pages 808–814, 1990.

[23] Fu, K. S., and Mui, J. K. A survey on image segmentation. *Pattern Recognition*, 13:3–16, 1981.

[24] Gorda, B. C., and Brooks III, E. D. The MPCI gang scheduler. *The 1991 MPCI Yearly Report: The Attack of the Killer Micros*, pages 183–187, 1991.

[25] Gorda, B. C., Warren, K. W, and Brooks III, E. D. Programming in PCP. Technical Report UCRL-MA107029, Lawrence Livermore National Laboratory, 1991.

[26] Haralick, R. M. Statistical and structural approaches to texture. *Proc. IEEE*, 67(5):786-804, May 1979.

[27] Haralick, R. M. Edge and region analysis for digital image data. *Comput. Vision Graphics Image Process.*, 12:60–73, 1980.

[28] Haralick, R. M., and Shapiro, L. G. Image segmentation techniques. *Comput. Vision Graphics Image Process.*, 29(1):100–132, January 1985.

[29] Haralick, R. M., Shanmugam K., and Dinstein I. Textural features for image classification. *IEEE Trans. Systems Man Cybernet.*, 3(6):610–621, November 1973.

[30] Harlow, C. A., and Eisenbeis, S. A. The analysis of radiographic images. *IEEE Trans. Computers*, C-22:678–688, 1973.

[31] Horowitz, S. L., and Pavlidis, T. A graph-theoretic approach to picture processing. *Comput. Graphics Image Process.*, 7:282–291, 1978.

[32] Huang, J. S., and Tseng, D. H. Statistical theory of edge detection. *Comput. Vision Graphics Image Process.*, 43(3):337–346, 1988.

[33] Hwang, K., and Briggs, F. A. *Computer architecture and parallel processing.* McGraw Hill, New York, 1984.

[34] Jain, A. K., and Dubes, R. C. *Algorithms for clustering data.* Prentice Hall, 1988.

[35] Ji, L. Intelligent splitting in the chromosome domain. *Pattern Recognition*, 22(5):519–532, 1989.

[36] Julesz B. Visual pattern discrimination. *IRE Trans. Inform. Theory*, 8(2):84–92, February 1962.

[37] Keller, J. M., Chen, S., and Crownover, R. M. Texture description and segmentation through fractal geometry. *Comput. Vision Graphics Image Process.*, 45(2):150–166, February 1989.

[38] Lakshmanan, S., and Derin, H. Simultaneous parameter estimation and segmentation of gibbs random fields using simulated annealing. *IEEE Trans. Pattern Anal. Machine Intell.*, 11(8):799–813, August 1989.

[39] Larsen, R. J., and Marx, M. L. *An introduction to mathematical statistics and its applications.* Prentice Hall, 1981.

[40] Lee, H. S., Hodgson, R. M., and Wood, E. J. Texture measures for carpet wear assessment. *IEEE Trans. Pattern Anal. Mach. Intell.*, 10(1):92–105, January 1988.

[41] Lee, S. U., Chung, S. Y., and Park, R. H. A comparative performance study of several global thresholding techniques for segmentation. *Comput. Vision Graphics Image Process.*, 52(2):171–190, 1990.

[42] Little, J. J., Blelloch, G. E., and Cass, T. A. Algorithmic techniques for computer vision on a fine-grained parallel machine. *IEEE Trans. Pattern Anal. Mach. Intell.*, 11(3):244–257, March 1989.

[43] Muerle, J. L., and Allen, D. C. Experimental evaluation of techniques for automatic segmentation of objects in a complex scene. *Pictorial Pattern Recognition*, pages 3–13, 1968.

[44] Mui, J. K., Bacus, J. W., and Fu, K. S. A scene segmentation technique for microscopic cell images. *Proc. Symp. Computer Aided Diagnosis of Medical Images*, 76 CH1170-OC:99–106, 1976.

[45] Narayanan, P. J. Effective use of SIMD machines for image analysis. Technical Report CAR-TR-635/CS-TR-2945, Center for Automation Research, Univ. of Maryland, August 1992.

[46] Ohanian, P. P., and Dubes, R. C. Performance evaluation for four classes of textural features. *Pattern Recognition*, 25(8):819–833, 1992.

[47] Ohlander, R., Price, K., and Reddy, D. R. Picture segmentation using a recursive region splitting method. *Comput. Graphics Image Process.*, 8:313–333, 1978.

[48] Perry, A., and Lowe, D. G. Segmentation of textured images. *Proc. CVPR*, pages 319–325, 1989.

[49] Pietikainen, M., and Rosenfeld, A. Image segmentation by texture using pyramid node linking. *IEEE Trans. Systems Man Cybernet.*, 11(12):822–825, December 1981.

[50] Raafat, H. M., and Wong, A. K. C. Textured-based image segmentation. *Proc. CVPR*, pages 1–7, 1986.

[51] Reddaway, S. F., and Kruskal, C. P. A distributed array processor. *First Annual Symposium on Computer Achitecture*, pages 61–65, 1973.

[52] Robertson, T. V., Swain, P. H., and Fu, K. S. Multispectral image partitioning. Technical Report TR-EE 73-26, School of Engineering, Purdue Univ., August 1973.

[53] Rushton, A. *Reconfigurable processor array: a bit-sliced parallel computer.* MIT Press, Cambridge, MA, 1989.

[54] Sahoo, P. K., Soltani, S., Wong, A. K. C., Chen, Y. C. A survey of thresholding techniques. *Comput. Vision Graphics Image Process.*, 41:233–260, 1988.

[55] Sanz, J. L. C. Computing image texture features in parallel computers. *Proc. IEEE*, 76(3):292–294, March 1988.

[56] Schoenmakers, R. P. H. M., Wilkinson, G. G., and Schouten, T. E. Segmentation of remotely-sensed images: a re-definition for operational applications. *International Geoscience and Remote Sensing Symposium*, pages 1087–1090, 1991.

[57] Shapiro, S. S., and Wilk, M. B. An analysis of variance test for normality. *Biometrika*, 52(3 & 4):591–611, 1965.

[58] Shen, H. C., and Bie, C. Y. C. Feature frequency matrixes as texture image representation. *Pattern Recognition Letters*, 13:195–205, 1992.

[59] Shi, H., and Schaeffer, J. Parallel sorting by regular sampling. *J. Parallel Distrib. Comput.*, 14(4):361–372, 1992.

[60] Stone, H. S. *High-performance computer architecture.* Addison-Wesley, 1990.

[61] Swain, P. H., and Fu, K. S. On the applications of nonparametric techniques to crop classification problems. *National Electronics Conf. Proc.*, 24:14–19, 1968.

[62] Taylor, R. W., Savini, M., and Reeves, A. P. Fast segmentation of range imagery into planar regions. *Comput. Vision Graphics Image Process.*, 45:42–60, 1989.

[63] Tilton, J. C., and Cox, S. C. Segmentation of remotely sensed data using parallel region growing. *9th Int. Sym. on Machine Processing of Remotely Sensed Data*, pages 130–137, 1983.

[64] Tsuji, S, and Tomita, F. A structural analyzer for a class of textures. *Comput. Graphics Image Process.*, 2:216–231, 1973.

[65] Van Ryzin, J. *Classification and clustering.* Academic press, 1977.

[66] Weszka, J. S. A survey of threshold selection techniques. *Comput. Vision Graphics Image Process.*, 9:259–265, 1978.

[67] Willebeek-LeMair, M., and Reeves, A. P. Solving nonuniform problems on SIMD computers: case study on region growing. *J. Parallel and Dist. Computing*, 8:135–149, 1990.

[68] Wolf, G. Use of global information and a priori knowledge for segmentation of objects: algorithms and applications. *Biomedical Image Processing and Three-Dimensional Microscopy*, SPIE 1660, pt. 1:397–408, 1992.

[69] Yokoya, N., and Levine, M. D. Range image segmentation based on differential geometry: a hybrid approach. *IEEE Trans. Pattern Analy. Mach. Intell.*, 11(6):643–649, 1989.

[70] Zucker, S. W. Survey, region growing: childhood and adolescence. *Comput. Graphics Image Process.*, 5:382–399, 1976.