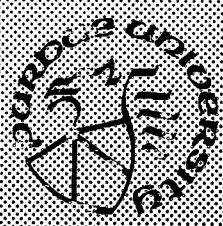
12-1-1988

Edge Detection by Cost Minimization

Hin Leong Tan
Purdue University

Edward J. Delp
Purdue University



Edge Detection by Cost Minimization

Hin Leong Tan
Edward J. Delp

TR-EE 88-49
December 1988

School of Electrical Engineering
Purdue University
West Lafayette, Indiana 47907

EDGE DETECTION BY COST MINIMIZATION

Hin Leong Tan

Edward J. Delp

School of Electrical Engineering

Purdue University

West Lafayette, Indiana 47906

TR-EE 88-49

December 1988

ACKNOWLEDGMENTS

The authors would like to thank Professor Saul B. Gelfand for his many insightful comments and suggestions relating to the contents of this report.

TABLE OF CONTENTS

Page

Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

Figure Page

Figure Page

Figure Page

ABSTRACT

Edge detection is cast as a problem in cost minimization. This is achieved by the formulation of two cost functions which evaluate the quality of edge configurations. The first is a comparative cost function (CCF), which is a linear sum of weighted cost factors. It is heuristic in nature and can be applied only to pairs of similar edge configurations. It measures the relative quality between the configurations. The detection of edges is accomplished by a heuristic iterative search algorithm which uses the CCF to evaluate edge quality.

The second cost function is the absolute cost function (ACF), which is also a linear sum of weighted cost factors. The cost factors capture desirable characteristics of edges such as accuracy in localization, thinness, and continuity. Edges are detected by finding the edge configurations that minimize the ACF. We have analyzed the function in terms of the characteristics of the edges in minimum cost configurations. These characteristics are directly dependent of the associated weight of each cost factor. Through the analysis of the ACF, we provide guidelines on the choice of weights to achieve certain characteristics of the detected edges.

Minimizing the ACF is accomplished by the use of Simulated Annealing. We have developed a set of strategies for generating next states for the annealing process. The method of generating next states allows the annealing process to be executed largely in parallel.

Experimental results are shown which verify the usefulness of the CCF and ACF techniques for edge detection. In comparison, the ACF technique produces better edges than the CCF or other current detection techniques.

CHAPTER 1
INTRODUCTION

1.1 Overview of Edge Detection

The detection of edges in an image is an important task in image processing. Its importance cannot be over-emphasized as it is often the front end processing stage in object reconstruction and image understanding systems [1, 2]; the accuracy in which this task can be performed is a crucial factor in determining the overall system performance. Edge detection is sometimes viewed as the dual of image segmentation; edges are boundaries between regions that have significantly different characteristics. The measure of difference in characteristics may be based on texture involving statistical [3] or structural properties in the gray levels, or they may be based on changes in the image intensity profile of the scene. A great deal of literature has been written on edge detection (see [4-6] for an overview) and the majority of these have concentrated on detecting edges that are caused by changes in the image intensity profile. They have defined edges to be located at points of intensity discontinuity in the image and have traditionally defined three categories of ideal edges; these are the step, ramp and roof edges as shown in Figure 1.1. Detection algorithms based on intensity discontinuity usually result in estimating the degree of slope in the intensity profile at each point in the image.

The classical edge detectors emphasize the use of difference operators which are the digital approximations to the derivative operators in the continuous domain. A major difficulty with differentiation is that it is not robust with respect to noise and the end result of applying difference operators to real images inevitably produce a high degree of false and fragmented edges. Torre and Poggio [7, 8] showed that differentiation is an ill-posed problem (in the sense of Hadmard) and that it can be transformed to a well-posed problem by applying regularizing filters to the image prior to differentiation. The regularizing filters are essentially low pass filters that minimize a given stabilizing functional. There is a good intuitive basis for this since low pass

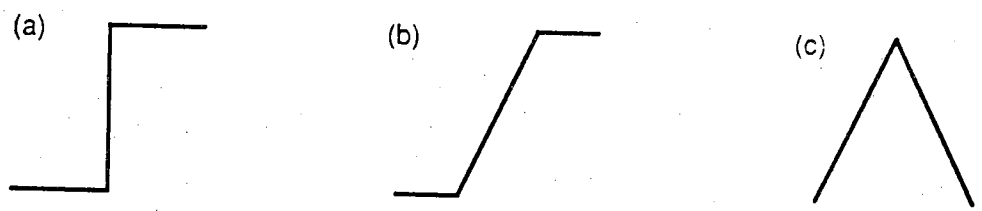


Figure 1.1. Examples of intensity edges. (a) Step edge. (b) Ramp edge. (c) Roof edge.

filtering essentially suppresses high frequency noise in images, and will tend to produce better edges at the differentiation stage.

Optimal filtering techniques have been used in the design of filters for edge detection. Dickey and Shanmugam [9] defined an edge to be a step discontinuity and showed that the ideal bandlimited filter to optimally localize its response about the edge is given by a prolate spheroidal wave function. Canny [10] approached the problem of detecting edges by designing one dimensional optimal filters that satisfy a set of performance criteria. The optimal filter was then approximated by the first derivative of a Gaussian function. It was implemented by convolving the image with a Gaussian operator and then finding the gradient of the smoothed image. Instead of the gradient, Marr and Hildreth [11] used a rotationally invariant second derivative operator, the Laplacian, on Gaussian smoothed images. The edges were found by locating the zero crossings in the output of the $\nabla^2 G$ operator. A detailed discussion of the motivation for using the $\nabla^2 G$ operator is given by Marr [12].

Other approaches have used surface fitting techniques to find changes in the image intensity profile. These techniques are based on the use of various sets of basis functions to describe the shape of the intensity surface. Each basis function has an associated weight and the goal of surface fitting is to estimate the weight values such that the sum of the weighted basis functions produce a minimal error analytic description of the intensity surface of the image. The presence of edges is based on the obtained description. Some of the classical digital derivative operators are based on derivatives of best surface fit models [13]. Hueckel [14] fitted ideal step edges to the image intensity and minimized the error of fitting by using a set of 8 basis functions defined on a circular disk. Haralick [15-17] used a model in which he fitted polynomial surfaces over small neighborhoods of each pixel, and derived expressions for the directional second derivative based on the polynomial coefficients. The pixel at the center of the fitted neighborhood was declared to be an edge if a negatively sloped zero crossing of the second derivative (taken in the direction of the gradient from the pixel center) is found within the pixel area. Nalwa and Binford [18] looked for significant step edges by fitting one dimensional hyperbolic tangent functions over every possible fixed square neighborhood in the image.

Other approaches to edge detection include the use of moment operators [19, 20]. However, these were shown to be essentially equivalent to the standard gradient operators. Sequential techniques for contour tracing or edge linking[21-23] have been used. Such techniques usually involve tracing along a

path in an image in search of thin continuous edges. They have been shown to be fairly insensitive to noise. Nevatia and Babu [24] extracted linear features in an image by convolving the image with masks corresponding to ideal step edges in different directions. The output was then thresholded and thinned and approximated by piecewise linear segments.

1.2 Some Recent Techniques in Edge Detection

We now describe in more detail four of the more recent approaches to edge detection. They are samples from the optimal filtering, surface fitting and sequential edge detection techniques mentioned above.

Derivative of Gaussian

The derivative of Gaussian operator [10], denoted by ∇G, has been proposed as an approximation to the optimal filter for detecting ideal step edges. The optimality is based on a set of three performance criteria: (1) good detection, (2) good localization, and (3) single response to an edge. This method of detecting edges involve smoothing the image with a Gaussian function

$$G(x,y) = k \exp\left[-\frac{x^2+y^2}{2\sigma^2}\right], \tag{1.1}$$

where k is a normalization constant usually chosen so that all the nonzero values of G sum to one.

After smoothing, the gradient at each point in the image is computed by taking the partial derivatives in the x and y directions;

$$D = \nabla(G * I) .$$

where * denotes convolution, I is the original image, and D is the gradient of the smoothed image. An edge pixel is defined to be a local maximum of the magnitude of D in the direction of the gradient. The magnitude of D represents the edge strength at any edge point. Thresholding the edge strength is required to reduce false edge points. The smoothing parameter σ is application dependent. Larger values of σ results in better noise insensitivity at the expense of reduced image resolution.

Laplacian of Gaussian

The Laplacian of Gaussian is a rotationally invariant operator for the detection of intensity edges. The operator is of the form

$$\nabla^2 G = k\left[\frac{x^2+y^2}{2\sigma^2} - 1\right]\exp\left[\frac{x^2+y^2}{2\sigma^2}\right] \tag{1.2}$$

where k is a scaling constant.

The edges in an image are detected by convolving the image with the $\nabla^2 G$ operator, and then finding the zero crossings at the output. To reduce false detection , the edge points are often detected by thresholding the slope at the zero crossings.

Facet model approach

The fact model [17] approach to edge detection uses surface fitting techniques to find ideal step edges in an image. It assumes that in each neighborhood of the image, the underlying intensity function f takes on the parametric form of a cubic polynomial in the row and column coordinates;

$$f(r,c) = k_1 + k_2 r + k_3 c + k_4 r^2 + k_5 rc + k_6 c^2 +$$

$$k_7 r^3 + k_8 r^2 c + k_9 rc^2 + k_{10} c^3. \tag{1.3}$$

A pixel is marked as an edge if, based on f, in the pixel's immediate neighborhood there is a zero crossing of the second directional derivative taken in the direction of the gradient. The coefficients k_i of Equation (1.3) are estimated by fitting the intensity data values with discrete orthogonal polynomials. The second directional derivative at point (r,c) on the line in the direction α is given by

$$f'' = 6[k_7 \sin^3\alpha + k_8 \sin^2\alpha\cos\alpha + k_9 \sin\alpha\cos^2\alpha + k_{10}\cos^3\alpha]\rho +$$

$$2[k_4 \sin^2\alpha + k_5 \sin\alpha\cos\alpha + k_6\cos^2\alpha], \tag{1.4}$$

where

$$\rho = \sqrt{r^2 + c^2}.$$

If for some ρ, where the magnitude of ρ is less than the length of the side of a pixel, $f'''(\rho)<0$, $f''(\rho)=0$ and $f'(\rho)\neq 0$, then there is a negatively sloped zero crossing, and the center pixel of the neighborhood is marked as an edge pixel. To reduce false detection, the edge pixels are detected only if the slope exceeds a certain threshold.

Sequential Edge Linking

Eichel and Delp [23] proposed a sequential edge detection scheme called Sequential Edge Linking to find intensity edges in an image. The algorithm constructs a sequence of nodes (pixels) **m** called a path, where

$$\mathbf{m} = m_0, m_1, \ldots, m_n .$$

This path is a candidate edge path in the image. The path is assumed to be modeled as a Kth order Markov chain. That is, if we let

$$s_i = m_i, m_{i-1}, \ldots, m_{i-(K-1)} ,$$

then assuming m_0 is given,

$$\begin{aligned} Pr(\mathbf{m}) &= Pr(m_1, m_2, \ldots, m_n) \\ &= Pr(s_n/s_{n-1})Pr(s_{n-1}/s_{n-2})\ldots Pr(s_1/s_0). \end{aligned}$$

The image is modeled as a two-dimensional random field. At each node x, the conditional probability under the hypothesis that it corresponds to an edge pixel is

$$p_1(f_x = y) = Pr(f_x = y / \text{ x is an edge node}).$$

Similarly, the conditional probability under the null hypothesis is

$$p_0(f_x = y) = Pr(f_x = y / \text{ x is a random node}).$$

The algorithm searches for the paths that correspond to edges in the image based on a derived path metric of the form

$$\Gamma(\mathbf{m},\mathbf{f}) = \sum_{i=1}^{n} \left[\ln \frac{p_1(f_{m_i})}{p_0(f_{m_i})} + \ln Pr(s_i/s_{i-1}) \right] . \qquad (1.5)$$

The first component of the path metric is a function of the image data. It is usually estimated from the output of gradient operators on the original image. The second component is a measure of the a-priori probability that the edge path proceeds in the given path direction. Using a sequential tree searching algorithm, the edges are detected by finding the paths that have high path metrics.

Despite the tremendous amount of research that has been done in edge detection, finding the edges in an image that correspond to true physical boundaries remain a difficult problem. Part of the reason lies in the fact that

we really do not know explicitly what we are looking for in searching for edges. Although an edge has often been modeled as a unit step, it is a simple fact that ideal step edges hardly ever occur in real images. Furthermore, such a narrow concept of an edge ultimately restricts the applicability of the detection algorithm. For instance a detection algorithm which assumes that edges are ideal steps are invariably ineffective in finding roof edges or texture edges.

A second difficulty in many detection techniques is that the decision on the presence (or absence) of an edge is made without considering the local edge structure in the neighborhood of the pixel. This is particularly true of non-sequential detection algorithms. This is a drawback since there is definitely some information from the neighborhood edges that can be exploited in the decision process. For instance, noise in an image causes many detection algorithms to produce fragmented edges; an algorithm that exploits local edge continuity information will be able to use this to reduce the amount of fragmentation. Two other problems often encounted are the detection of thick edges and the detection of false sporadic edges caused by noise. Here again, local edge information can be used to reduce false detection and find only thin edges.

Sequential techniques have been effective in countering the problem of fragmentation and thick edges. However, computation time may be an inhibiting factor because of the sequential nature of the processing. Furthermore, they are applicable mostly in contour tracing tasks where the scene does not contain an excessive number of edges. Since they are usually based on the output of feature enhancement operators, their performance is very much dependent on the operator used.

1.3 Edge Detection by Cost Minimization

We cast edge detection as a problem in cost minimization. We define a cost function over the domain of all possible edge configurations on a square lattice. The edges are detected by finding the configuration that minimizes this function. While most of the other detection techniques previously mentioned can also be viewed as some form of cost minimization, this approach is unique in the way the cost function is defined. The function not only uses information from image data, but it also exploits information from local edge structure. It takes the form of a linear combination of weighted cost factors. These cost factors capture the desirable characteristics of good edges such as edge thinness, continuity and well localization. By appropriately adjusting the weights of the cost factors, we can selectively emphasize the relative

importance of the different edge characteristics in the detection process.

There has been little attempt to formulate the problem of edge detection as one of cost minimization where the function is dependent on edge structure. By this we mean that the function takes into account not only the pointwise presence of edges in an image, but also the local shape and continuity aspects of the edge. Two major difficulties arise in such an approach to edge detection. The first is in the difficulty of defining a suitable cost function for edges. The second is that the minimization of such a function inevitably results in one that belongs to the class of non-deterministic polynomial time complete (NP-complete) problems. The search space for the minimum cost solution is extremely large as the number of possible solutions is equal to 2^K, where K is the number of pixels in the image.

There are a number of advantages of using the cost minimization approach described in this report. The first is that it assumes no preconceived concept of an edge except that it is a boundary separating dissimilar regions. Hence the approach is flexible in terms of being able to detect various types of edges. Second, it uses edge structure information such as edge continuity and thinness, and consequently the algorithm is more capable of detecting edges that are well localized, continuous and thin. Also, it will be seen that the algorithm has edge linking capabilities. Third, unlike sequential techniques, the detection algorithm can be implemented largely in parallel.

In Chapter 2, we present the first cost minimization approach to detect edges based on a comparative cost function. This function is a heuristic cost function for evaluating edges. In Chapter 3, we present a second approach based on an absolute cost function. This a well defined function over the set of all possible edge configurations for an image. We will present a mathematical description of edges and analyze the characteristics of edges that will be produced by minimizing this function. In Chapter 4, we will describe Simulated Annealing and show how it can be used to find low cost edge configurations for an image. In Chapter 5, we present experimental results of the application of both the comparative cost function and absolute cost function approaches to edge detection. Finally, in Chapter 6, we conclude by listing several potential areas of further research.

CHAPTER 2
A COMPARATIVE COST FUNCTION APPROACH TO EDGE DETECTION

2.1 Introduction

The main objective of this work is to formulate edge detection as a problem in cost minimization. We will present two approaches to the formulation. The first approach uses a comparative cost function to evaluate the relative quality of pairs of similar edge configurations. It is heuristic in nature and the function can only be applied to edge configurations that are almost identical. In contrast to this, the second approach uses an absolute cost function which can evaluate the relative quality of any pair of different edge configurations. In this chapter, we will present the comparative cost approach and describe an iterative algorithm to find edges in an image. We will also discuss the similarities and dissimilarities of this algorithm with relaxation techniques.

Central to both approaches is the formulation of a cost function to evaluate edges. In order to accomplish this, we first have to specify what we mean by an edge. Unfortunately, the concept of an edge is a difficult one to define precisely; in the next section, we will present our concept of an edge in order to establish common ground for discussing edge detection.

2.2 Concept of An Edge

A precise notion of an edge is crucial to the formulation of a cost function for evaluating edges. However, it is a difficult task to explicitly define what constitutes an edge in an image. The perception of edges by the human visual system is an extremely complex process that is strongly influenced by prior knowledge. There are a number of visual paradoxes in which an edge is clearly perceived when none physically exists (see for instance [12] p. 51). Every individual has an intuitive notion of what edges are, but this notion varies from person to person. Indeed, if two individuals are given identical images and asked to find the edges, they may well produce similar looking but non-

identical edges. Consequently, no absolute definition of an edge exists, and the performance of edge detection algorithms are only as good as their inherent assumption of what edges are.

Edges in an image can generally be divided into two categories; intensity edges and texture edges. Intensity edges are those edges that arise from abrupt changes in the intensity profile of the image. Examples of these are the step, roof and ramp edges as shown in Figure 1.1. Texture edges are boundaries of texture regions that are invariant to lighting conditions. A number of texture edges are usually defined relative to image models [25]. A number of detection algorithms adopt a narrow concept of edges and are devoted to finding only specific kinds of edges in an image. A weakness of such algorithms is that they are invariably ineffective in detecting edges outside of their scope.

For our purpose, we will define an edge in a general sense so as to include a wide variety of edge types. However, we will restrict our attention to those edges that are evident from the image data itself and not from higher level human cognitive processes. With this in mind, we define an edge to be a boundary in an image that separates two regions that have significantly dissimilar characteristics; the regions are assumed to lie on either sides of the edge. The cause of the dissimilarity may be due to a combination of several factors, such as the geometry of the object, surface reflectance characteristics, viewpoint and illumination. The term "dissimilarity" is used in its broadest sense to include any form of difference in the structure of the intensity values that is evident in the image. Clearly, this definition includes both intensity and texture edges. For instance, the well known step edge is a boundary separating two regions that are dissimilar in the sense that they have different constant intensity values. In the same vein, texture edges are boundaries separating regions having different textural properties.

In addition to the fundamental property that edges separate nonhomogeneous regions, our concept of edges is also governed by certain structural characteristics that edges should possess. These characteristics determine the shape and position of the edges in an image. We list four desirable characteristics that edges should have.

(1) Accurate localization

It is desirable that an edge should lie in a spatially accurate position, partitioning the dissimilar regions in the best possible way. In many real images, the position of an edge may be ambiguous. This is often the case when

a collection of closely adjacent boundaries will separate the same pair of dissimilar regions. Since each boundary in the collection has a unique spatial location, the degree of dissimilarity between the regions on either sides of the boundary will vary for each boundary in consideration. We say that an edge is accurately localized when it coincides with the boundary that results in the maximum degree of dissimilarity.

(2) Thiness

Since edges are boundaries, it is desirable that they form thin lines in the image. Ideally, they should be only one pixel wide in the direction that is perpendicular to the edge direction.

(3) Continuity

Edges should exhibit a continuity that reflects the nature of the boundary in the physical environment. Most physical boundaries of interest are continuous in nature. It is desirable that correct edges should also possess this property. However, we do not constrain edges to form closed boundaries in an image. We will use the term fragmentation to describe edges that are sporadically discontinuous.

(4) Length

Noise and fine texture may cause the appearance of short scattered edges of one or two pixels in length. We will omit from our consideration such short edges and restrict our concept of edges to those that are at least 3 pixels long.

In practice, there is often a tradeoff between the different desirable characteristics of an edge. Due to conflicting edge requirements, there are many situations where it is not possible to simultaneously achieve two or more characteristics. For instance, requiring every edge in an image to be long and continuous may result in poor localization and the appearance of false boundaries. Hence, it is appropriate to associate a measure of importance with each desirable edge characteristic so that situations involving conflicting edge requirements may be resolved. It will be seen in the formulation of the comparative cost function that the importance of each characteristic is emphasized by attaching a weight to its associated cost factor.

2.3 A COMPARATIVE COST FUNCTION

The goal of edge detection is to find the pixels in an image that satisfy the concept of an edge as described in the previous section. The edges should be detected with minimum error, where the error corresponds either to missing edge pixels, or edge pixels that do not satisfy the edge criteria. To find the edges, it is of crucial importance to use information from both local and global edge structure in the detection process. The reason for this is that the criteria for an edge includes characteristics such as thinness, continuity and length which are based solely on the structural nature of the edge. These structural properties are not evident from the image data itself; they have to be determined by examining the structure of the edge configuration. Hence, an important key to good detection is to incorporate edge structure information in the detection process. As an example, consider the case of a fragmented edge that is the result of noise in the image. A detection algorithm that uses information from local edge structure will be able to improve the edge continuity by linking together locally disconnected edge segments. Similarly, thick edges can be made thin by the removal of excess edge pixels. It will be seen that the comparative cost function approach to edge detection uses edge structure information in the detection process.

The comparative cost function approach to edge detection is essentially an iterative algorithm that makes pointwise (pixel by pixel) decisions on the presence of edges in the image. The heart of the decision making process is the comparative cost function. The function mathematically captures the intuitive concept of an edge. It compares two edge configurations by considering their edge structure and the image data. The decision process consists of choosing the better edge configuration and iterating the procedure.

We now introduce some notation which will be used in the definition of the comparative cost function. An image G is a two-dimensional array of pixels g(m,n), $1 \leq m \leq m_{max}$, $1 \leq n \leq n_{max}$, where each pixel g(m,n) has gray level in the range $1 \leq g(m,n) \leq 255$. For simplicity, we will assume that the images are square with $m_{max} = n_{max} = N$. Similarly, we define an edge configuration S_i to be a two dimensional array of pixels $s_i(m,n)$,$1 \leq m,n \leq N$, where each pixel takes on a binary value 0 or 1. If $s_i(m,n) = 1$, the pixel $s_i(m,n)$ is called an edge pixel; otherwise it is a non-edge pixel. We denote as **S**, the set of all possible edge configurations on an $N \times N$ square lattice. Since every site in the lattice can have one of two possible edge labelings, the number of elements in **S** is equal to 2^{N^2}. Even for extremely small images, this number

is so large that it is impossible to implement any exhaustive search algorithm to find the best edge configuration. The comparative cost function and search procedure is a heuristic technique for finding edge configurations according to the edge criteria.

2.3.1 Valid Edge Structures

In order to define the cost function, we have to first specify what is meant by valid edge structures. Using an 8-neighbor representation, every edge pixel has a maximum of 8 neighboring edge pixels in a 3×3 neighborhood. Valid edge structures are defined as follows. An edge pixel that has 0 or 1 other neighboring edge pixel is a valid edge structure. An edge pixel that has 2 other neighboring edge pixels is a valid edge structure if the pixels are arranged such that the resulting edge structure is continuous and does not turn by more than 45 degrees. We call this a valid 2-neighbor edge structure. Figure 2.1 shows 4 valid 2-neighbor edge structures. Figure 2.2 is an invalid edge structure since the edge makes a 90 degree turn to the right. Taking into account rotations of the edges in Figure 2.1, there is a total of 12 possible valid 2-neighbor edge structures. An edge pixel that has 3 other neighboring edge pixels is a valid (3-neighbor) edge structure if the edge pixels form one of the 8 structures shown in Figure 2.3. Although there are 56 different structures involving an edge pixel with 3 neighbors, only the 8 in Figure 2.3 allow for the possibility that each of the neighboring edge pixels can form valid edge structures with other pixels in its neighborhood. An example of this is shown in Figure 2.4. Edges with 4 or more neighboring edge pixels are defined to be invalid structures.

2.3.2 Region Dissimilarity

In order to find edges (or boundaries) that separate regions that are dissimilar, we need to specify the regions of interest on either sides of an edge. This is done by first defining the position of an ideal edge with respect to a given object. The position of this edge must be correctly defined so as to accurately reflect the geometry and size of the object. This is important when high precision measurements are required. Figure 2.5 shows a square object with a corresponding ideal edge. In this case, the position of the ideal edge is poorly defined as it does not accurately depict the relative size of an object. We illustrate this fact by looking at the image of a pair of embedded boxes as shown in Figure 2.6(a). Consider the spacing between the vertical portions of the edges; this figure indicates that the distance between the edges corresponding to the vertical sides of the smaller square is 5 units, while the

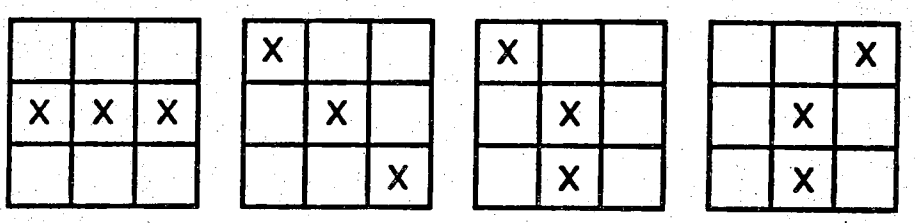


Figure 2.1. Valid 2-neighbor edge structures.

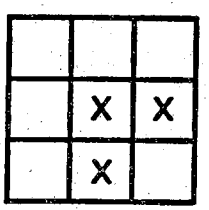


Figure 2.2. An invalid edge structure.

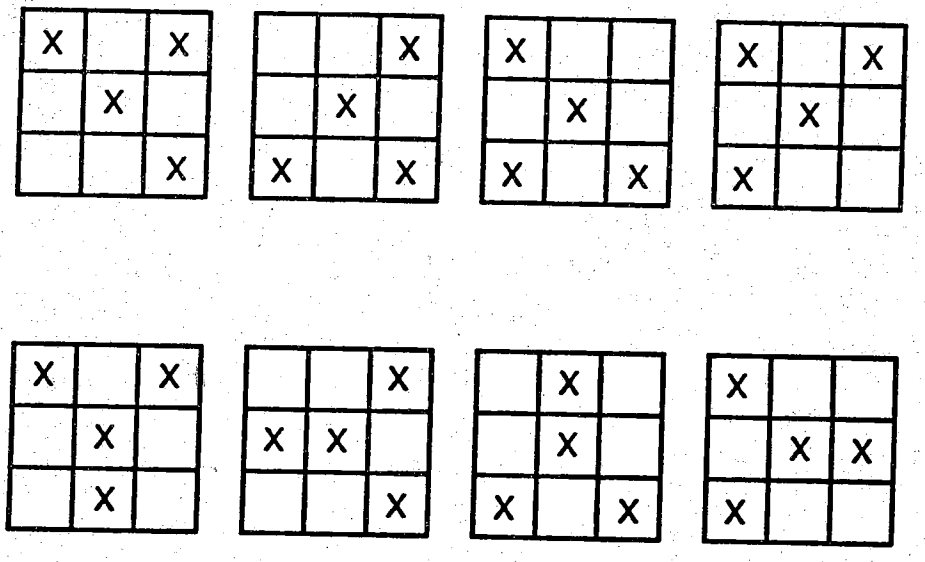


Figure 2.3. The 8 valid 3-neighbor edge structures.

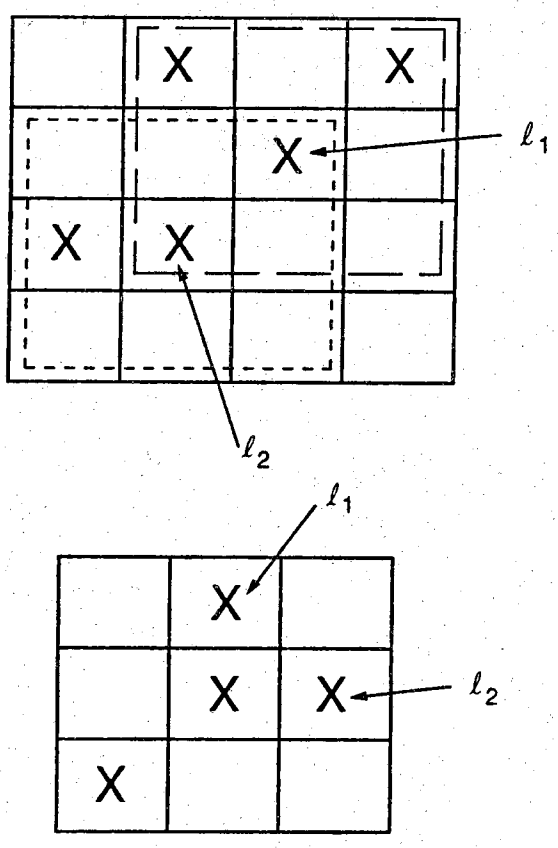


Figure 2.4. Examples of edge structures. (a) An example of a valid 3-neighbor edge structure. Notice that pixel l_1 is part of a valid 3-neighbor structure in a 3×3 window neighborhood indicated by the dotted lines. Pixel l_2 which is a neighbor of l_1 also forms a valid edge structure with its neighbors. (b) An example of an invalid 3-neighbor edge structure. Notice in this structure that it is not possible for the pixel at l_1 or the pixel at l_2 to form a valid edge structure with its neighboring edge pixels because of the invalid 2-neighbor structure in its neighborhood.

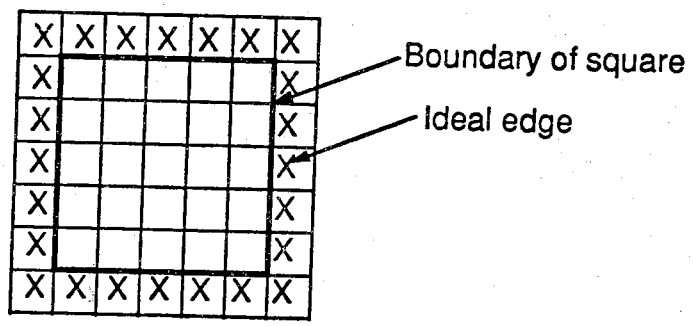
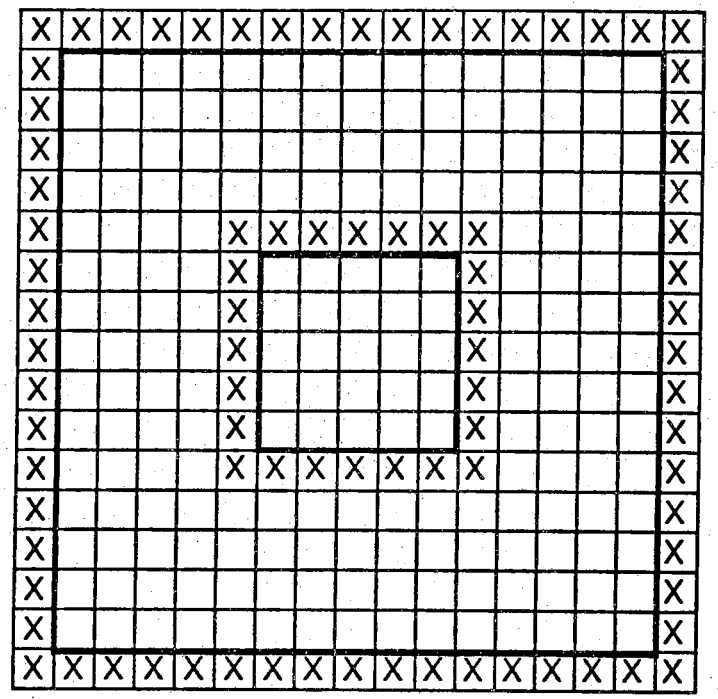


Figure 2.5. An example of a poorly defined ideal edge for a square.

(a)



(b)

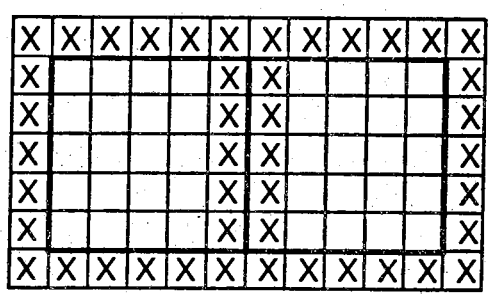


Figure 2.6. Edges resulting from the definition of the ideal edge in Figure 2.5. (a) Edges of a pair of embedded boxes. (b) Edges of a pair of adjacent squares. Notice that the edge positions are either ambiguous or the relative distance between the edges is incorrect.

distance from the edge on one side of the smaller square to the corresponding edge of the larger square is only 4 units. This is of course incorrect as both the measurements should be 5 units. Figure 2.6(b) illustrates another difficulty with the above definition of the ideal edge for a square. Although the dividing line between the adjacent squares is clearly defined in the image, the position of the ideal edge is ambiguous for the vertical edge in the center. Besides these difficulties, the defined edge is also undesirable because it contains invalid edge structures at the corner regions of the square.

A better definition of the ideal edges for a square and hexagon is shown in Figure 2.7. These are thin edges that satisfy our concept of an edge. Figure 2.8 shows the corresponding edges for the embedded boxes and the pair of adjacent boxes of Figure 2.6. Notice that these edges do not suffer from the difficulties of the previous example in Figure 2.6. Based on Figure 2.7, we define for each valid 2-neighbor edge structure, a pair of regions on either sides of the edge. The regions are chosen with the intuitive notion that edges separate regions which are non-intersecting, and that these regions lie in a close vicinity to the edge. These regions, which shall be labeled R1 and R2 for each edge structure, are the regions of interest on which a dissimilarity measure will be applied. The 12 valid 2-neighbor edge structures, the (circled) edge pixel which they are centered around, and their associated regions are shown in Figure 2.9. Depending on the application and the specific measure of dissimilarity used, larger (or smaller) regions for R1 and R2 could be defined. For example, the regions of interest could be extended as shown in Figure 2.10.

2.3.3 The Comparative Cost Function

Given a pair of nearly identical candidate edge configurations S_i and S_j that differ only at one pixel location $l = (m,n)$, we define the comparative cost function $\mathbf{C}(S_i,S_j)$ as:

$$\mathbf{C}(S_i,S_j) = \sum_{k=1}^{5} w_k \left[C_k(S_j,l) - C_k(S_i,l) \right] \tag{2.1}$$

$$= \sum_{k=1}^{5} w_k \Delta C_k(S_i,S_j) \tag{2.2}$$

where $w_k \geq 0$ and $0 \leq C_k \leq 1$.

The function is a weighted sum of the difference of 5 cost factors. Each of the weight values are given by w_k. It should be noted that l is any location within the square array of pixels. For ease of notation, we will write $\mathbf{C}(S_i,S_j)$ as

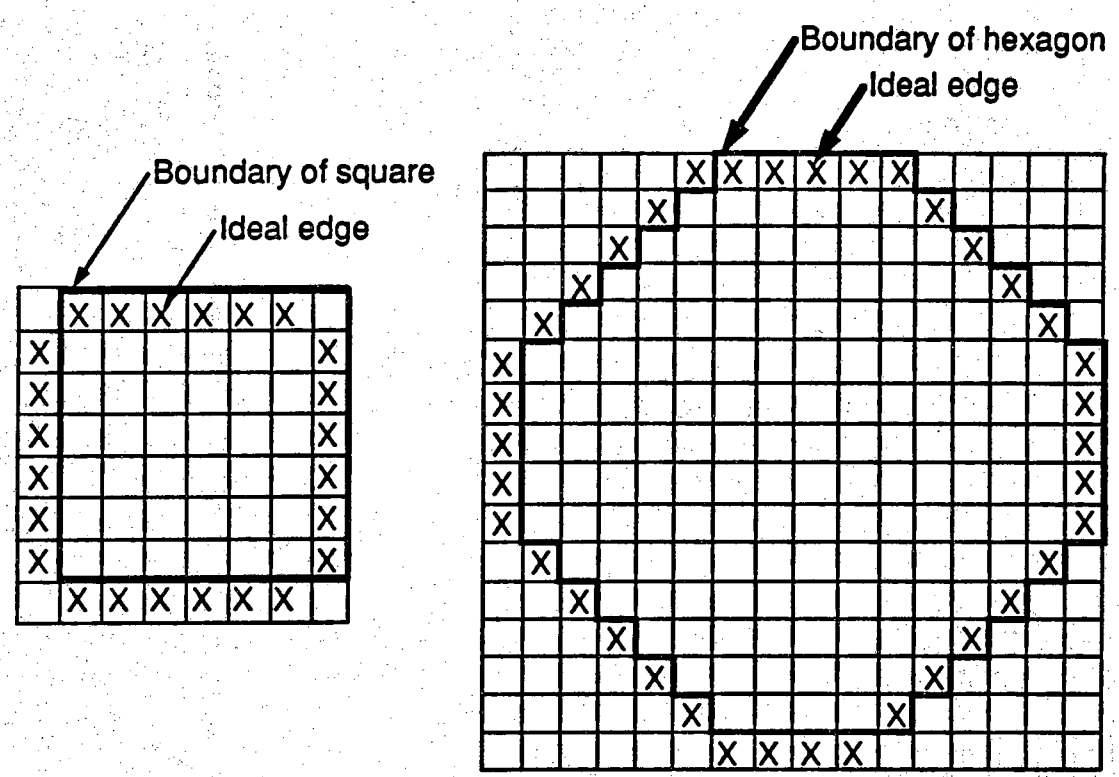


Figure 2.7. Ideal edges for a square and a hexagon.

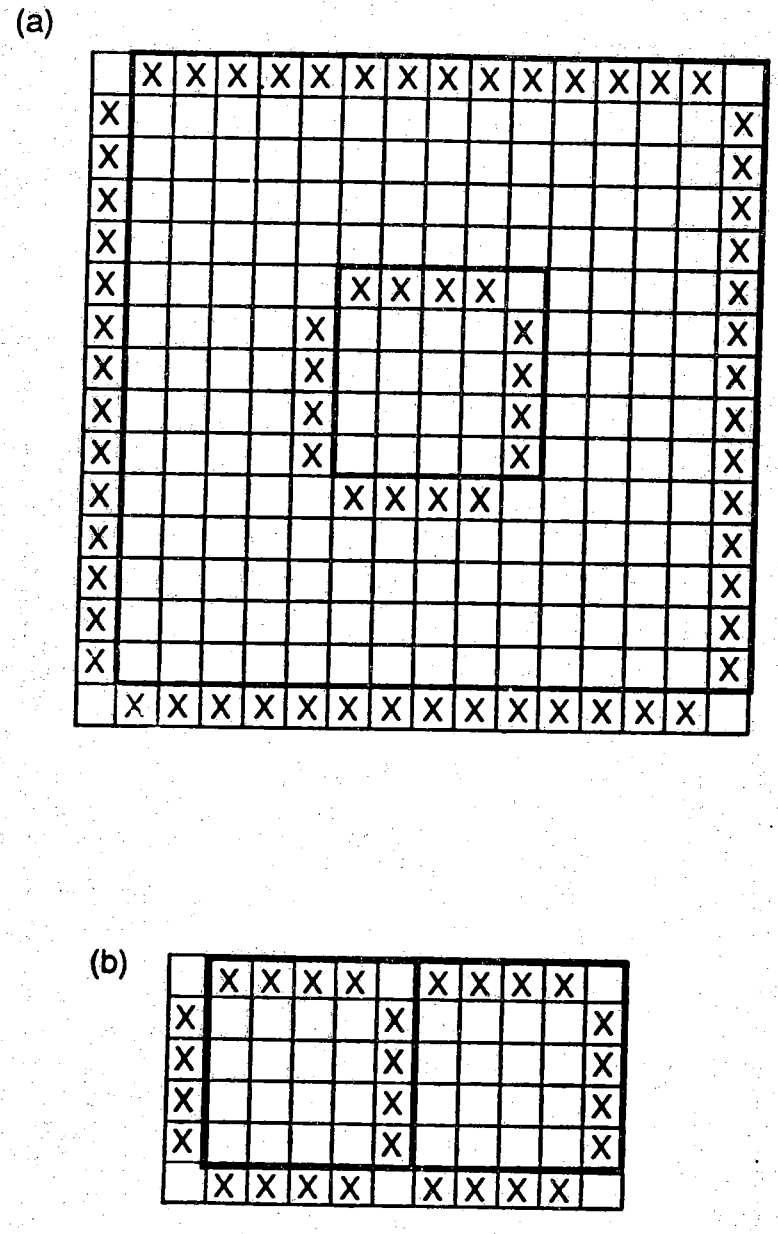


Figure 2.8. Edges resulting from the definition of the ideal edge in Figure 2.7. (a) Edges of a pair of embedded boxes. (b) Edges of a pair of adjacent squares. Notice that the edges do not suffer from the difficulties of the previous example in Figure 2.6.

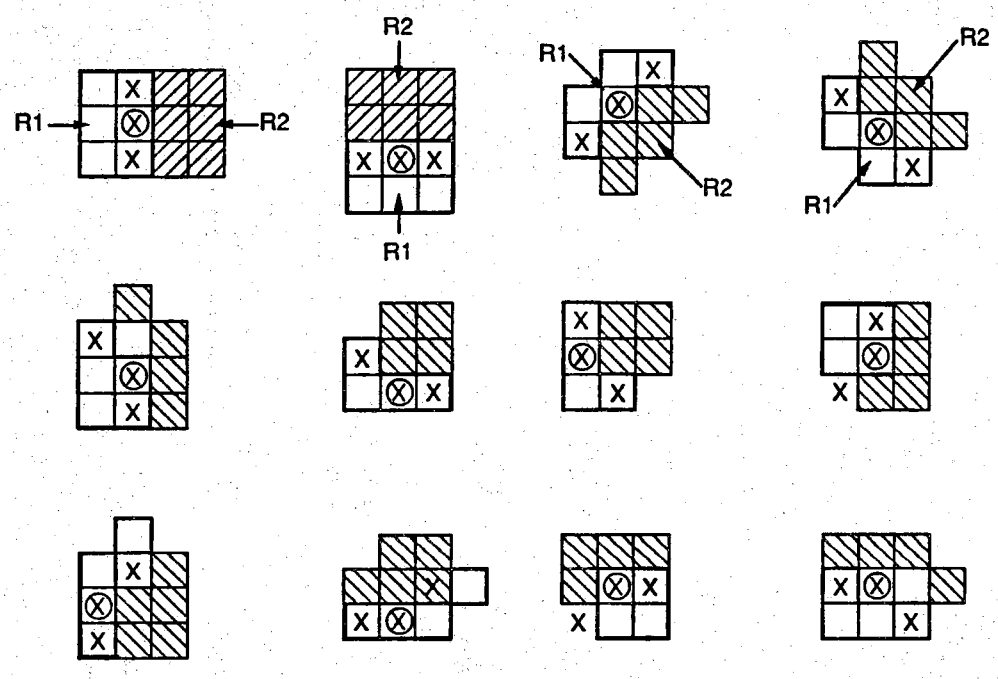


Figure 2.9. The 12 valid 2-neighbor edge structures, the (circled) edge pixel which they are centered around, and their associated regions of interest on each side of the edge.

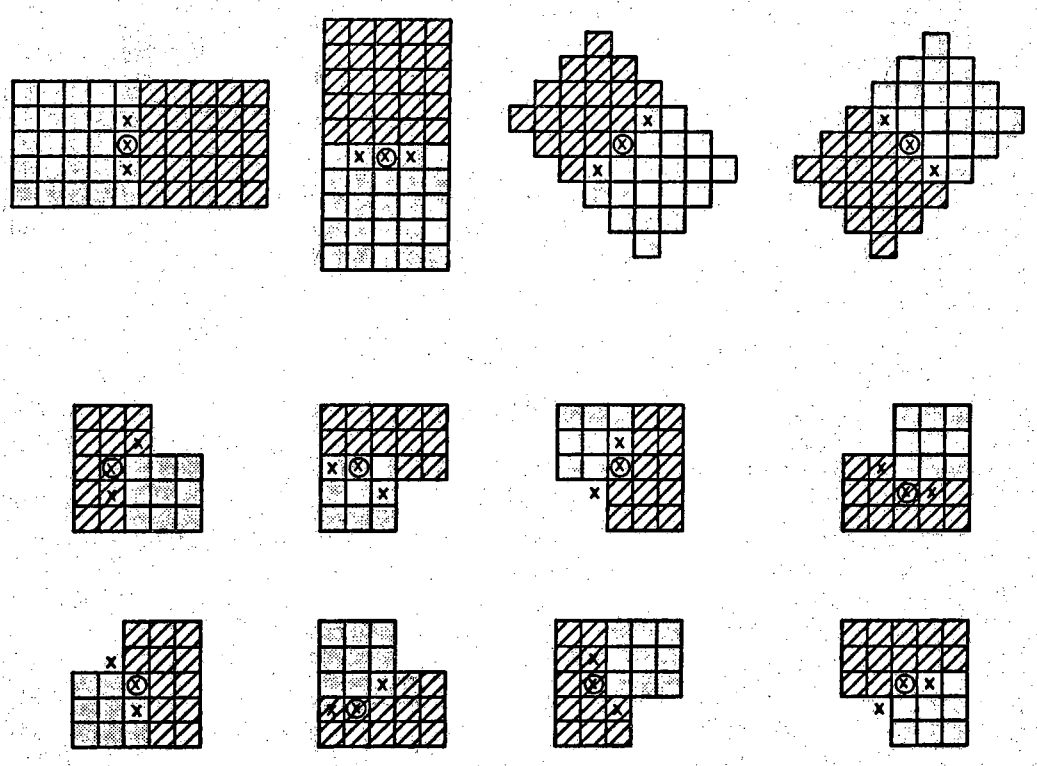


Figure 2.10. Extended regions of interest on each side of the edge.

C_{ij} or, when no confusion occurs, simply as C. Also, we shall refer to the pixel at location *l*, simply as pixel *l*. Now, we specify that

$$C_{ij} \begin{cases} \geq 0 \Rightarrow S_i \text{ is a better configuration} \\ < 0 \Rightarrow S_j \text{ is a better configuration} \end{cases} \tag{2.3}$$

This implies that we try to minimize the sum of the weighted cost factors C_k.

The motivation for making C_{ij} a weighted sum of cost factors C_k is that each of the factors should, in some way, capture a desirable characteristic of edges. Ideally, each cost factor should affect one and only one characteristic so that the relative importance of each can be appropriately emphasized by its corresponding weight w_k. In practice, this is difficult to achieve as the different characteristics often exhibit some form of dependency on each other. For instance, minimizing fragmentation may well result in poor localization and the appearance of false boundaries.

2.3.4 The Cost Factors

The square grid of an edge configuration is visualized as an overlay on the image; the cost factors are computed by examining the local structure of the edge configuration about pixel *l*, and the underlying image data. In the following paragraphs, we define the value of each cost factor C_k which is used in Equation (2.1).

1) C_d: Cost for region dissimilarity.

The cost for region dissimilarity is based on a function f_c(R1,R2) that measures how different region R1 is from R2. Large values of f_c(R1,R2) correspond to large dissimilarity. This measure could be a simple difference of gray level averages in R1 and R2, or it could be a more complicated measure based on other properties of the gray levels. Depending on the application and the features of interest in an image, there are numerous possibilities for the definition of f_c(R1,R2). As previously mentioned, to find the ideal step edges in an image, we could define the dissimilarity measure to be the difference of constant gray levels in the regions R1 and R2. For detecting texture edges, we could define f_c(R1,R2) based on statistical or structural properties of the gray levels in the different regions. It is clear that there is great flexibility in such an approach to edge detection as we do not restrict the nature of the dissimilarity between the nonhomogeneous regions. This is in contrast to many detection algorithms that assume some specific nature of edges and are devoted to

finding only such edges.

Non-maximal suppression is important in ensuring the accurate localization of an edge point in an image. In practically all real images, the dissimilarity measure has the tendency to enhance the points in the vicinity of the true boundary in addition to enhancing the boundary itself. This is undesirable as a large number of false boundary points are enhanced. One approach to mitigate this tendency is to employ non-maximal suppression when computing the dissimilarity. However, an undesirable side effect that results from using non-maximal suppression is that some true boundary points may also be suppressed together with the false points. This may increase the amount of fragmentation in the boundary. It will be seen that the cost factor for continuity will compensate for this effect by linking together locally disconnected edges.

In our implementation, $f_c(R1,R2)$ is computed as follows. Let d be the magnitude of the difference of gray level averages in R1 and R2, i.e.,

$$d = \left| \frac{1}{|R1|} \sum_{(i,j)\in R1} g(i,j) - \frac{1}{|R2|} \sum_{(i,j)\in R2} g(i,j) \right| \tag{2.4}$$

where $|R1|$, $|R2|$ denotes the number of pixels in R1 and R2 respectively. Note that $0 \leq d \leq 255$. Let m(d) be a piecewise linear function that maps d onto the unit interval [0,1]. We use m(d) as our measure of region dissimilarity, i.e.,

$$f_c(R1,R2) = m(d).$$

Suppose

$$m(d) = \begin{cases} \dfrac{d}{2t_r}, & 0 \leq d \leq 2t_r \\ 1, & \text{otherwise} . \end{cases} \tag{2.5}$$

This is a piecewise linear monotonic function that is comprised of a ramp followed by a flat region of constant value equal to one, as shown in Figure 2.11. The parameter t_r, which we shall call the threshold, is application dependent. It determines the slope of the ramp. The flat region causes undesirable effects when non-maximal suppression is applied to the value of $f_c(R1,R2)$. Since values of d greater than $2t_r$ are mapped to the same value, rank order information that is useful in the suppression process is lost. To avoid this, we choose the strictly monotonic mapping function shown in Figure

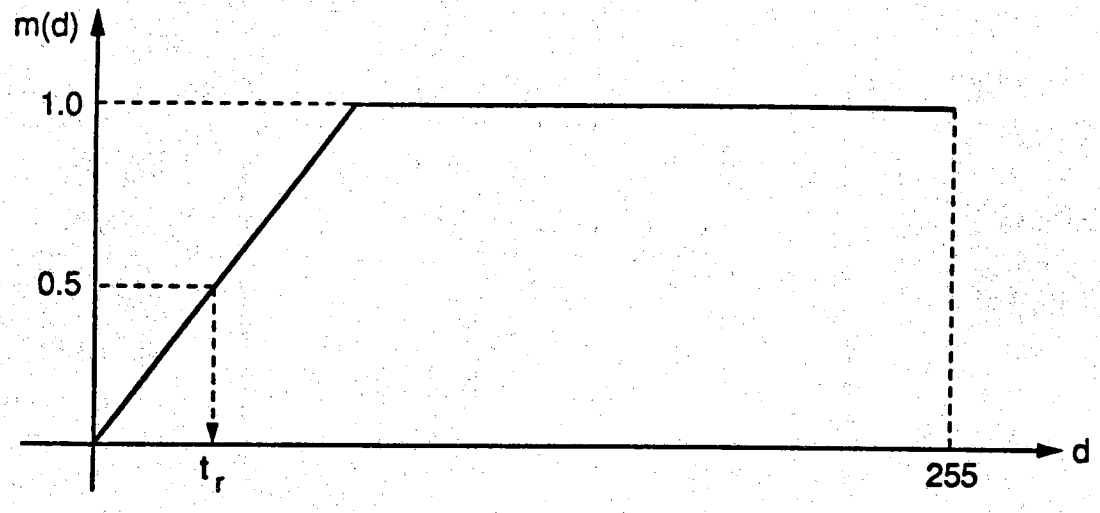


Figure 2.11. A monotonic mapping function.

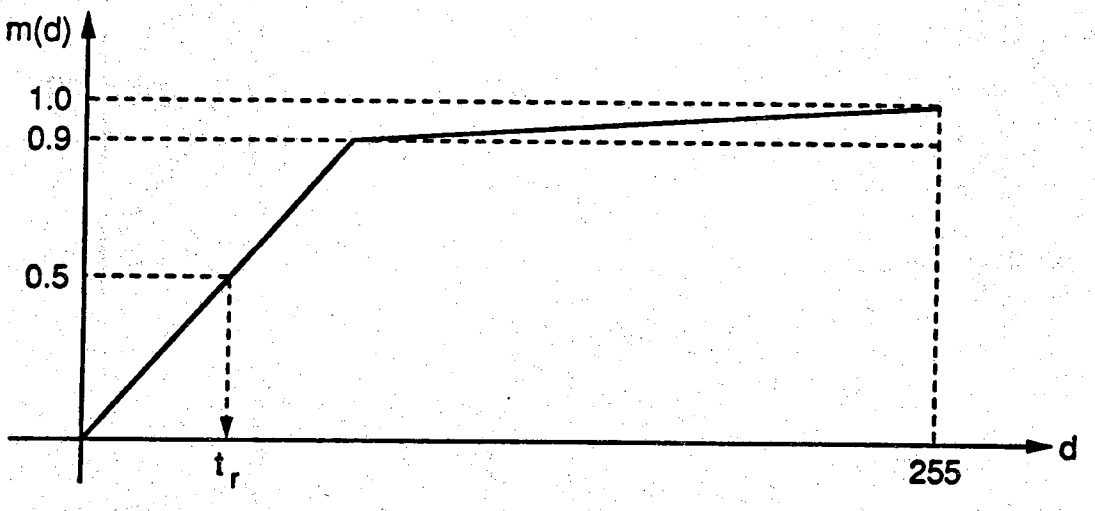


Figure 2.12. A strictly monotonic mapping function.

2.12. It is formed by the concatenation of 2 ramps; the first ramp rises to a maximum value of 0.9 while the second rises from 0.9 to 1. The function is specified by the following equation:

$$m(d) = \begin{cases} \dfrac{d}{2t_r}, & 0 \leq d \leq 1.8t_r \\ 0.9 + (d - 1.8t_r)\left(\dfrac{0.1}{255 - 1.8t_r}\right), & \text{otherwise.} \end{cases} \tag{2.6}$$

The cost for region dissimilarity $C_d(S_i,l)$ penalizes non-edge pixels by assigning to them a cost value that is proportional to the dissimilarity at the pixel location. If l is an edge pixel, no penalty for dissimilarity is made; this is achieved by assigning to edge points a dissimilarity cost value of zero. The cost for region dissimilarity is computed by first examining the edge structure of S_i in a local 3×3 window neighborhood centered at pixel l. If the pixel at l is an edge pixel, we set $C_d(S_i,l) = 0$. If the pixel at l is not an edge pixel, we proceed as follows. Observe that there are 12 possible valid 2-neighbor edge structures that could fit in a 3×3 window region centered at l. The best fitting edge structure is chosen according to the following cases:

Case 1: There are exactly 2 neighboring edge pixels which will form a valid 2-neighbor edge structure with an edge pixel at l. This valid structure is the best fitting edge structure.

Case 2: There are more than 2 neighboring edge pixels, one or more pairs of which will form valid 2-neighbor edge structures with an edge pixel at l. Amongst these valid edge structures, the one which results in the maximum value of $f_c(R1,R2)$ is chosen as the best fitting edge structure.

Case 3: If the local edge structure does not satisfy cases 1 or 2 above, then amongst the 12 possible valid 2-neighbor edge structures that could fit in a 3×3 window region centered at l, the one which results in the maximum value of $f_c(R1,R2)$ is chosen as the best fitting edge structure.

Next, we perform non-maximal suppression by shifting the location of the best fitting edge structure in a direction determined by the edge structure. For straight vertical, horizontal and diagonal edge structures, the shifting is performed by moving the edge location by one pixel in each of the opposite directions perpendicular to the edge. For all other edge structures, the shifting

is done by moving the edge one location in each of the four directions: up, down, left and right. Figure 2.13 shows how the edges are shifted for three edge types. If the maximum value of $f_c(R1,R2)$ over the shifted edge structures is greater than the value of $f_c(R1,R2)$ for the unshifted edge structure, we set $C_d(S_i,l)=0$; otherwise $C_d(S_i,l)=f_c(R1,R2)$ for the unshifted edge structure.

2) C_t: Cost for edge thickness.

Using an 8-neighbor representation of the edge, we define a thick edge to be an edge structure that has multiple links between 2 or more of its edge pixels. A thin edge is an edge that is not thick. A thick edge pixel is defined to be an edge pixel whose presence causes multiple links between its neighboring pixels. The cost for edge thickness is determined by considering pixel l in edge configuration S_i. If l is a thick edge pixel, then $C_t(S_i,l) = 1$; otherwise $C_t(S_i,l) = 0$. Examples of thick edges are shown in Figure 2.14. The edge in Figure 2.14(a) is thick because there are multiple links between several of the edge pixels. For instance, pixel X_1 is connected to pixel X_5 by two links; the first is through pixels X_2 and X_4, and the second is through pixel X_3. The edge in Figure 2.14(b) is also a thick edge because there two links between pixels X_1 and X_3; the first is a direct link between the two, and the second is through pixel X_2.

3) C_c: Cost for edge continuity.

This cost factor reduces the occurrence of single missing edge pixels that result in a disconnected edge. $C_c(S_i,l)$ is computed by examining S_i in a local 5×5 window neighborhood centered at pixel l. If pixel l is not an edge pixel, and there are 2 short edges less than 3 pixels each that could be connected by pixel l to form a thin edge that is at least 4 pixels long, we set $C_c(S_i,l) = 1$; otherwise $C_c(S_i,l) = 0$. Examples of cost assignment for edge continuity is shown in Figure 2.15.

4) C_l: Cost for edge length.

This cost factor reduces the occurrence of short edge pixels that are less than 3 pixels long. If pixel l is part of an edge that is less than 3 pixels long, we set $C_l(S_i,l) = 1$; otherwise $C_l(S_i,l) = 0$.

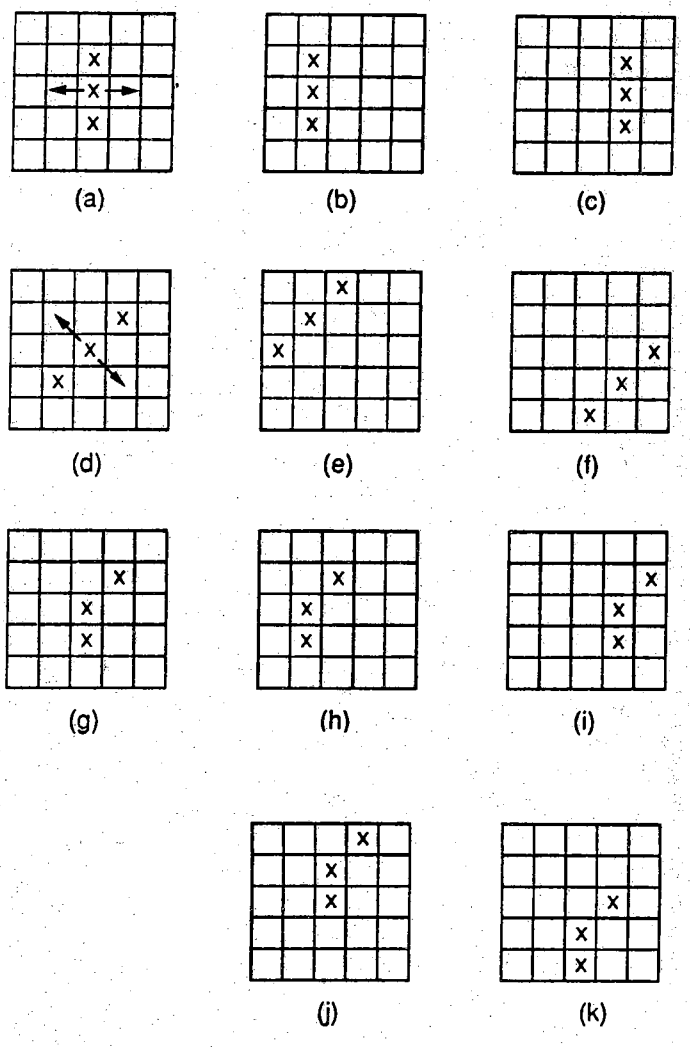


Figure 2.13. Shifting edge positions for non-maximal suppression. (a) Vertical edge and shifting directions. (b) and (c) are the shifted edge positions of the edge in (a). (d) Diagonal edge and shifting directions. (e) and (f) are the shifted edge positions of the edge in (d). (g) An edge that turns by 45 degrees. (h) to (k) are the four shifted edge positions.

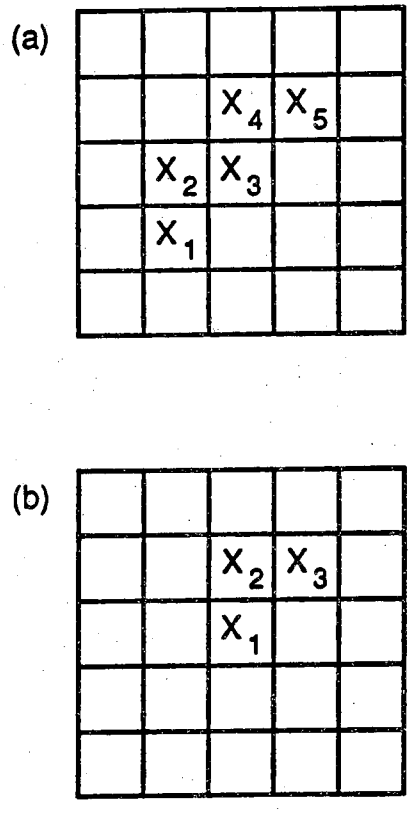


Figure 2.14. Thick edges. (a) Thick edge of 5 pixels. (b) Thick edge of 3 pixels.

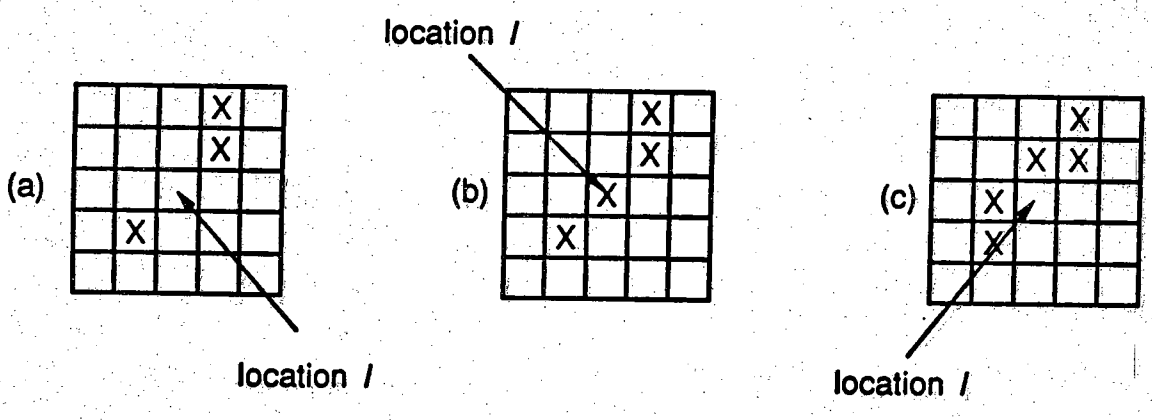


Figure 2.15. Cost assignment for edge continuity. (a) $C_c(S_i,l)=1$. (b) $C_c(S_i,l)=0$. (c) $C_c(S_i,l)=0$.

5) C_e: Cost for number of edge pixels.

The cost factor C_d for region dissimilarity will favor the placement of edge pixels at all points where the measure of dissimilarity $f_c(R1,R2)$ is non-zero. This causes an excessive number of edges to be detected. To suppress this, we assign a cost for each additional edge pixel detected. If pixel l is an edge pixel, we set $C_e(S_i,l) = 1$; otherwise $C_e(S_i,l) = 0$.

The comparative cost function is a weighted sum of the above cost factors. It should be noted that the cost factor C_d uses both image data and edge structure information, while C_c, C_e, C_l and C_t uses only edge structure information. The utilization of this function to detect edges is based on a heuristic search procedure which is the subject of the following section.

2.4 A HEURISTIC SEARCH ALGORITHM

In this section, we describe an iterative algorithm that uses the comparative cost function to find a good edge configuration for the image. As previously described in Section 2.3.3, the cost function compares two very similar edge configurations and produces a value that indicates which of the configurations is better. To use this function, we will need some means of generating new configurations. The method of generating a new configuration is to take the previous best configuration and complement the edge label of one of its $N \times N$ pixels. Clearly, there are a possible of N^2 new configurations that can be generated from the previous best configuration. Basically, the algorithm begins by selecting any arbitrary edge configuration and calling it the best. It then recursively generates new configurations that are compared with the previous best by means of the cost function. The algorithm is as follows:

(1) Begin by selecting any arbitrary edge configuration S_i and any location l $=(m,n)$, where $1 \leq m,n \leq N$.

(2) Define a new edge configuration S_j such that it is identical to S_i except at pixel l (where it is the complement).

(3) Compute C_{ij} and select the better of the two configurations according to:

$$C_{ij} \begin{cases} \geq 0 \Rightarrow S_i \text{ is a better configuration} \\ < 0 \Rightarrow S_j \text{ is a better configuration} \end{cases}$$

Label the selected configuration S_i.

(4) Pick a new location l =(m,n), where $1 \leq m,n \leq N$

(5) If stopping criterion is not satisfied, Repeat from step (2).

The algorithm terminates either when no better configuration can be found after every possible new configuration has been tried, or when a suitable stopping criterion is satisfied. A simple stopping criterion is based on the number of better configurations found after K iterations. If this number does not exceed a certain minimum, the algorithm stops. Each new location l may be selected either in a deterministic or random manner; comparisons have been made and experimentally it has been found to have little effect on the final result. However, it is essential that every possible pixel location be selected at least once. Consequently, it has been found to be computationally more efficient to choose new values of l by sequentially stepping through the image in a raster scan fashion. When this is done, typically 3 to 5 iterations through the image is sufficient for the algorithm to converge according to the stopping criterion.

The algorithm described above begins with a random edge configuration and attempts to change the edge labeling at every pixel in a sequential manner. The comparative cost function is used to decide if the change is successful. When viewed in this way, the algorithm is a sequential pointwise edge detection process that uses information from image data, information from local edge structure, and information from past decisions at neighboring pixels.

2.4.1 Selecting the weights

Many edge detection algorithms do not use local edge structure information in the detection process. Those that do can usually be classified as some type of curve or boundary tracing technique. The comparative cost function approach to finding edges is unique in the way it attempts to incorporate edge structure information in the detection process; the edge information is captured in the cost factors. By altering the weights w_k associated with the cost factors, we can change the amount of emphasis placed on each factor. Consider the situation where all the weights are zero except for w_d and w_e. The edge detection process then becomes similar to the straightforward thresholding approach to edge detection; information about local edge structure, such as thinness, continuity and length is not used. It should be noted that scaling all the weights by a constant will produce the same results as using the unscaled weights.

For our implementation, we used the values $w_d = 2.0$, $w_t = 1.1$, $w_c = 1.1$, $w_e = 1.0$, and $w_l = 1.1$. First, let us just consider the cost factors for region dissimilarity and number of edge pixels. When we use the weight values of 2.0 and 1.0 for w_d and w_e respectively, edge pixels will be detected at all points where $C_d \geq 0.5$. However, when we take into consideration all 5 cost factors and their associated weights, the interaction of the different factors will result in several constraints on the detection process. First, thick edges will be disallowed; even if C_d=1 for a thick edge pixel at location l, the weight values of 1.1 and 1.0 for w_t and w_e, respectively, will always favor the removal of the edge pixel. Similarly, fragmentation will be reduced as edges that are separated by only one pixel will be connected together by the weight value of 1.1 for w_c. Since the cost factor C_l removes short edges, the weight value of 1.1 for w_l will ensure that edges that are less than 3 pixels long will not be detected. When large values (greater than 0.5 approximately) for w_l are used, it is necessary to set w_l initially to zero for the first several iterations, and then to its correct value for the remaining iterations. This is to avoid certain undesirable local minimum states that are possible. For instance, if the initial state contains no edge pixels, then a weight combination of 2.0, 1.0, 1.1 for w_d, w_e, w_l respectively, will produce no edges regardless of how many iterations are made. This is because the combined value of w_e and w_l exceeds that of w_d, preventing transition to any other state from the initial state.

2.5 COMPUTING THE COST

Since the comparative cost function is used repetitively in the detection algorithm, most of the computation is in determining the value of C_{ij}. From a computational standpoint, it is of major importance that this value can be determined in an efficient way. One approach to determine C_{ij} would be to compute each cost factor independently, and then sum the difference as specified in Equation (2.1). However, this is a naive approach that does not take into account the interdependence of the cost factors. For instance, an edge that is a valid 2-neighbor structure is thin, continuous and at least 3 pixels long; the fact that it is a valid structure allows us to determine 3 of the 5 cost factors immediately. A great deal of reduction in computation time can be achieved by pooling together information affecting each of the different factors and organizing it in a form that will allow for efficient computation. This is achieved by a decision tree structure as shown in Figure 2.16. The structure allows for the simultaneous computation of several cost factors by traversing

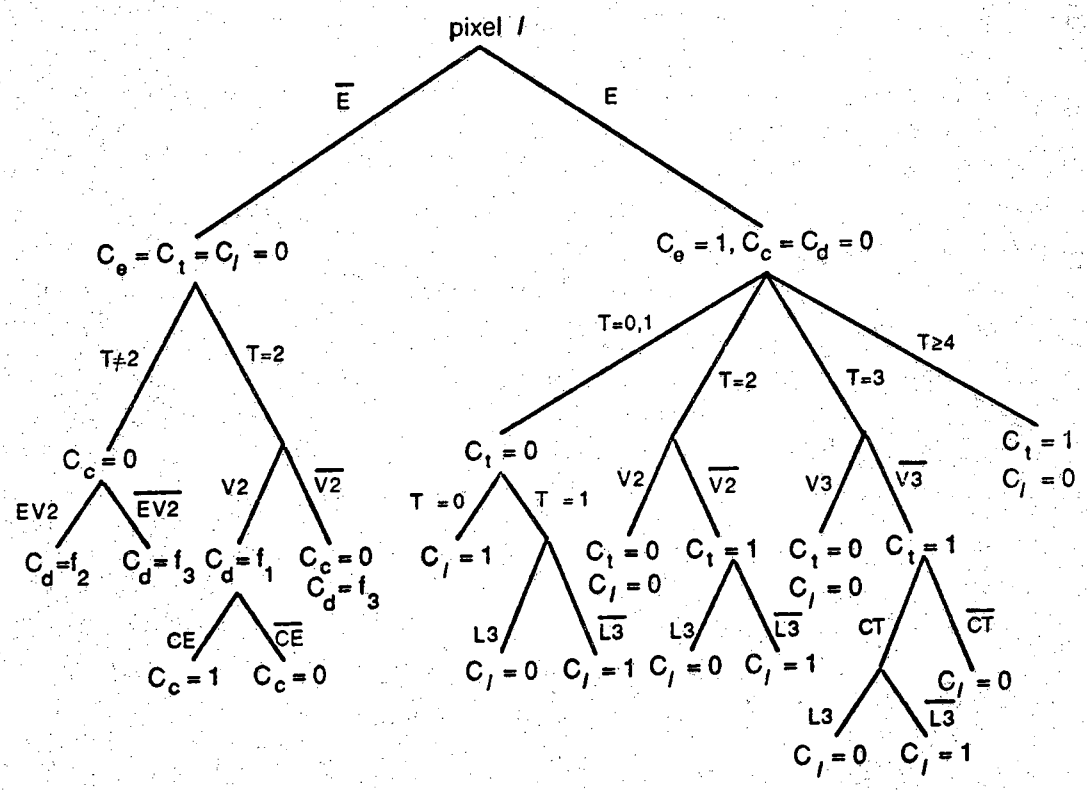


Figure 2.16. Computation of cost factors using a decision tree.

the tree from root to leaf following the relevant path. The tree is 5 levels deep. At each node, the decision as to which branch to take is governed by conditions that are assigned to each branch. These conditions are exhaustive and mutually exclusive; traversal to the next node is made by following the branch where the condition is satisfied. The conditions, which are abbreviated by labels, are summarized as follows:

LABEL	DESCRIPTION
E	The pixel at l is an edge pixel.
$\overline{\text{E}}$	The pixel at l is not an edge pixel.
T=n	The total number of edge pixels T, in a 3×3 neighborhood about l is equal to n. This total does not include the pixel at l.
V2	The two neighboring edge pixels will form a valid 2-neighbor edge structure with an an edge pixel at l.
$\overline{\text{V2}}$	The two neighboring edge pixels will not form a valid 2-neighbor edge structure with an an edge pixel at l.
V3	The three neighboring edge pixels will form a valid 3-neighbor edge structure with an an edge pixel at l.
$\overline{\text{V3}}$	The three neighboring edge pixels will not form a valid 3-neighbor edge structure with an an edge pixel at l.
EV2	Some pairs of neighboring edge pixels will form a valid 2-neighbor edge structure with an edge pixel at l.
$\overline{\text{EV2}}$	No pair of neighboring edge pixels will form a valid 2-neighbor edge structure with an edge pixel at l.
CE	An edge pixel at l will link 2 short segments, each less than 3 pixels, to form a thin continuous edge segment that is at least 4 pixels long.
$\overline{\text{CE}}$	An edge pixel at l will not link 2 short segments to form a thin continuous edge segment that is at least 4 pixels long.
L3	The edge pixel at l is part of an edge that is at least 3 pixels long.
$\overline{\text{L3}}$	The edge pixel at l is not part of an edge that is at least 3 pixels long.
CT	The 3 neighboring edge pixels are either clustered together forming an "L" shaped region in one corner of the 3×3 window, or lined up

straight along one of the 4 straight borders of the window.

$\overline{CT}$ The 3 neighboring edge pixels are neither clustered together forming an "L" shaped region in one corner of the 3×3 window, nor lined up straight along one of the 4 straight borders of the window.

f_n This is the non-maximal suppressed value of $f_c(R1,R2)$; the edge structure used to compute $f_c(R1,R2)$ is the one obtained by using case "n" of the best fitting edge rule (discussed in Section 2.3.4).

A further reduction in computation time (by approximately half) is achieved by observing that configurations S_i and S_j differ only at pixel location l, and consequently that C_{ij} can be determined simply by considering S_i. We need not compute $C_k(S_i,l)$ and subtract it from $C_k(S_j,l)$; we can compute $\Delta C_k(S_i,S_j)$ directly by considering S_i or S_j in the neighborhood of l. The decision tree for this is shown in Figure 2.17. This tree is similar to that shown in Figure 2.16; traversal from one node to the next is governed by the conditions assigned to each branch. The value of C_{ij} is determined by appropriately traversing the tree from root to leaf following the relevant path. This tree assumes that configuration S_i does not have an edge at l while S_j does. If the opposite is true, then C_{ij} is determined by first computing C_{ji} using this tree, and then negating the result. The cost function has the property that $C_{ij} = -C_{ji}$.

It has been previously mentioned in Section 2.4 that the heuristic search algorithm can be viewed as a procedure where we sequentially try to complement the edge labeling at every pixel location in the image. It is important to note that each cost factor $C_k(S_i,l)$ is only dependent on the value of the pixels in a neighborhood that is no larger than a 5×5 window about location l. Consequently, the decision of the edge labeling at pixel l_1 can be made independently of the labeling at l_2, if l_1 and l_2 are 2 or more pixels apart. Hence, although the algorithm is sequential, the processing can be implemented to a large extent in parallel if the pixel locations are chosen such that any pair are at a distance of at least 2 pixels apart. We could, for instance, attempt to change the pixel labeling of every third pixel in a row at every third row. For an $N \times N$ image, there are approximately $N^2/9$ such locations. The processing at each location can be done in parallel and the decisions on the edge labeling can be made simultaneously. This is significant as it results in a reduction of the number of sequential processing steps by a factor of $N^2/9$.

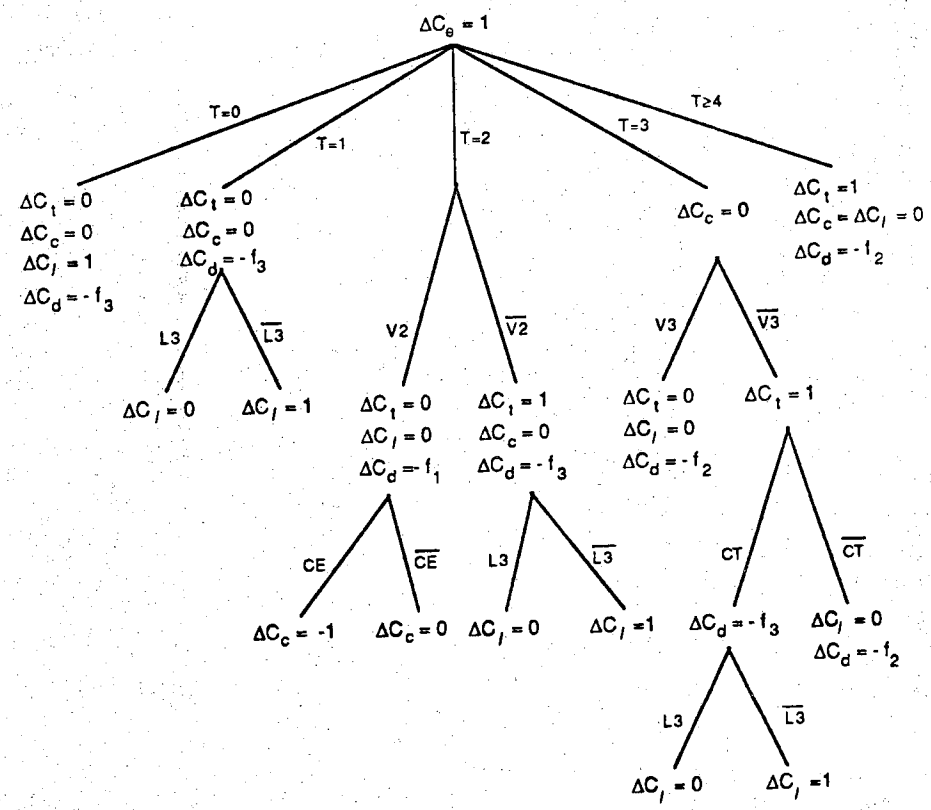


Figure 2.17. Direct computation of $\Delta C_k(S_i,S_j)$ using a decision tree.

2.6 Relaxation Techniques

Relaxation [4, 26-28] is an iterative approach to segmentation that makes "probabilistic decisions" at every point in parallel at each iteration. These decisions are then adjusted at successive iterations based on the decisions made at past iterations. In this section, we will discuss some similarities as well as dissimilarities of the heuristic search algorithm with relaxation techniques.

Consider the task of classifying a set of n objects $A_1,.....A_n$ into m classes $C_1,.....C_m$. The basic approach of probabilistic relaxation is to assign to each object A_i, a vector of probabilities p_{ij}, $1 \leq j \leq m$ where each element of the vector is indicative of the likelihood that object A_i belongs to class C_j. The elements of the vector are assumed to sum to one:

$$\sum_j p_{ij} = 1 .$$

For each pair of class assignments, $A_i \in C_j$ and $A_h \in C_k$, there is a quantitative measure of the compatibility of the pair, denoted by c(i,j; h,k). We assume that c(i,j; h,k) lies in the range [-1,1] with larger values indicating good compatibility and low values indicating poor compatibility; zero represents the "don't care" situation. Based on this compatibility function, the probability vectors are altered in parallel using an iterative scheme. There are no fixed rules as to how the vectors are altered; numerous heuristic methods exist. Intuitively, we would like to increase the probability p_{ij} if the class assignment $A_i \in C_j$ is highly compatible with $A_h \in C_k$, and p_{hk} is large. Conversely, we would like to decrease it if the assignments are incompatible, and p_{hk} is large. If p_{hk} is low, we do not want to alter p_{ij} very much regardless of the value of the compatibility function. One possible method of updating the vector based on this intuition is to use the product

$$c(i,j; h,k) \cdot p_{hk} .$$

The updating process at the (r+1) iteration is given by:

$$p_{ij}^{r+1} = \frac{p_{ij}^r (1 + q_{ij}^r)}{\sum_{j=1}^{m} p_{ij}^r (1 + q_{ij}^r)} , \qquad (2.7)$$

where

$$q_{ij}^{r} = \frac{1}{(n-1)} \sum_{\substack{h=1 \\ h \neq i}}^{n} \left[\sum_{k=1}^{m} c(i,j;\, h,k) \cdot p_{hk}^{r} \right] . \qquad (2.8)$$

Notice that q_{ij}^{r} is simply the average over the sum of all increments due to the product of c(i,j; h,k) and p_{hk}. The denominator in the equation of p_{ij}^{r+1} is just a normalizing constant ensuring that the elements of the vector sum to one. Ideally, the goal is to iterate until every vector converges to the state where only one of its elements is non-zero. Practically, however, this is difficult to achieve and the process is terminated typically after a number of iterations.

Comparing, we see that there is some resemblance of the heuristic search technique with probabilistic relaxation. Both techniques are heuristic iterative processes; at each iteration, new decisions are made based on past decisions. Both are for object classification. Specifically, in the case of edge detection, there are two classes; edge or no edge. However, there are also several distinct differences in the two techniques. First, the heuristic search algorithm is essentially a sequential technique where new decisions are made one object (pixel) at a time. Although it can be implemented to a large extent in parallel, the technique is essentially a sequential process. In contrast, the relaxation technique is a parallel process where all the probability vectors are altered simultaneously at each iteration. Second, the classification process of the heuristic search technique is not probabilistic in nature. At each iteration, firm decisions are made as to whether a pixel is, or is not, an edge. This again is in contrast to relaxation which, for each pixel, assigns a vector of probabilities that is incrementally adjusted at successive iterations. Third, the comparative cost function is not equivalent to the compatibility function. In a sense, the heuristic search algorithm can be viewed as a degenerate form of relaxation where there are only two classification classes, and the elements of the probability vectors are binary valued, 0 or 1. The comparative cost function is then analogous to a complex "compatibility function" of the form

$$c(\, i,j;\, h_1,k_1;\, h_2,k_2;\ldots.\, h_{24},k_{24}\,) ,$$

where each of the 24 objects are the neighboring pixels in a 5×5 window about the object (pixel) A_i. A closer examination will reveal that this function is different not only in form, but also in usage from the usual compatibility functions in relaxation.

We conclude that the heuristic search algorithm is not a relaxation process because of the fundamental differences listed above. It is an iterative process

which can be appropriately viewed as a heuristic cost minimization approach to detect edges. This view will be further justified by the formulation of an absolute cost function which will be described in the next chapter.

2.7 An Absolute Cost Function

The comparative cost function given in Equation (2.1) is defined only for pairs of similar edge configurations; it measures the relative quaity between the configurations. This function can be modified to yield an absolute cost function which is applicable to individual edge configurations. The resulting cost of each configuration is indicative of its quality. One possible definition of an absolute cost function is

$$\mathbf{C}'(S_i) = \sum_{\text{all } l} \left[\sum_{k=1}^{5} w_k C_k(S_i, l) \right], \tag{2.9}$$

where the cost factors C_k's are the same as those of the comparative cost function. In this case, two configurations S_i and S_j can be compared by computing the difference in the cost values. This is given by the difference function

$$\begin{aligned} \Delta\mathbf{C}'(S_i,S_j) &= \mathbf{C}'(S_j) - \mathbf{C}'(S_i) \\ &= \sum_{\text{all } l} \left[\sum_{k=1}^{5} w_k [C_k(S_j, l) - C_k(S_i, l)] \right]. \end{aligned} \tag{2.10}$$

Notice that $\Delta\mathbf{C}'(S_i,S_j) < 0$ if and only if S_j is a lower cost configuration that S_i. The difference function $\Delta\mathbf{C}'(S_i,S_j)$ is similar in form to $\mathbf{C}(S_i,S_j)$ of Equation (2.1).

When used in accordance with Equation (2.1) of the comparative cost function, the cost factors together define a function that mathematically captures the intuitive idea of an edge. However, when the same cost factors are used in Equation (2.9) to define an absolute cost function, the result is a function that is not consistent with our concept of an edge. In other words, lower cost configurations may result in poorer edges. This is particularly evident in the case of edge continuity. An example of this is illustrated in Figure 2.18. The figure shows five hypothetical edge configurations S_0 to S_4. S_0 contains a fragmented edge; there are three missing edge pixels which, if present, would make the edge continuous. Based on Equation (2.9), the total continuity cost for an arbitrary edge configuration S_m is given by:

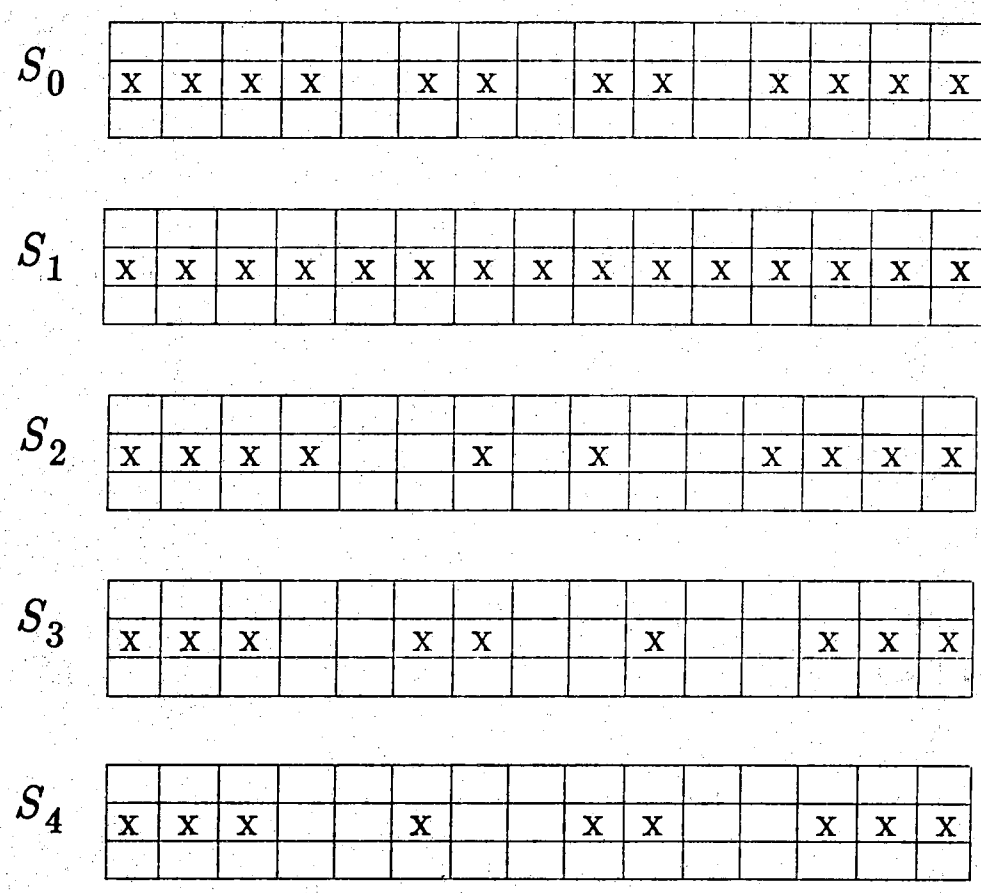


Figure 2.18. Continuity cost for different edge configurations. $C_c(S_0) = 3w_c$. $C_c(S_1) = C_c(S_2) = C_c(S_3) = C_c(S_4) = 0$. Although configurations S_2 to S_4 have a lower value for continuity cost than S_0, it is clearly noticeable that they have a higher degree of fragmentation.

$$C_c(S_m) = \sum_{\text{all } l} \left[w_c C_c(S_m, l) \right]$$

According to the definition of the cost for edge continuity in Section 2.3.4, this implies that

$$C_c(S_0) = 3w_c \, .$$

The continuous version of S_0 is S_1. Clearly, this edge configuration has an associated continuity cost $C_c(S_1) = 0$. An examination of the edge structures in configurations S_2, S_3 and S_4 reveals that they also have zero continuity cost; i.e., $C_c(S_2) = C_c(S_3) = C_c(S_4) = 0$. However, it is clearly noticeable that configurations S_2 to S_4 have a higher degree of fragmentation than S_0. Consequently, we see that the cost for continuity may not reduce fragmentation when used in the manner specified by Equation (2.9). In fact, as seen in the above example, it has a greater tendency to increase than to decrease fragmentation.

A better definition of an absolute cost function will be given in the next chapter. It takes the form of Equation (2.9); the cost factors are appropriately redefined to capture desirable edge characteristics.

2.8 Summary

In this chapter, we have shown how edge detection can be cast as a problem in cost minimization. We first described our concept of an edge which is based on criteria such as accurate localization, thinness, continuity and length. Based on this description, we formulated a comparative cost function that mathematically captures the intuitive ideas of an edge. The function uses information from both image data and local edge structure in evaluating the relative quality of pairs of edge configurations. Computation of the comparative cost function is performed efficiently by organizing the information in the from of a decision tree. Edges are detected using a heuristic search algorithm based on the comparative cost function. The detection process can be implemented largely in parallel. An extension of this approach to detect edges would be to formulate an absolute cost function that assigns an absolute cost value to any given edge configuration. The best edge configuration would be the one that achieves the global minimum of this cost function. The formulation of the absolute cost function is presented in Chapter 3.

CHAPTER 3
AN ABSOLUTE COST FUNCTION APPROACH TO EDGE DETECTION

3.1 Introduction

In the previous chapter, we have presented a comparative cost function that evaluates the relative quality of pairs of very similar edge configurations. Although fairly good results have been achieved using this function, two difficulties arise in its use.

First, the comparative function measures only relative quality. Furthermore, the pairs of configurations that it compares are constrained to be almost identical, differing at only one pixel site. This is rather restrictive because for any given edge configuration, only a relatively small subset of all possible configurations can be used for comparison. A practical consequence of this is that the heuristic search algorithm is sometimes trapped in undesirable local minimum states.

Second, the heuristic iterative search algorithm based on the comparative cost function is difficult to analyze. The goal of analysis is to determine specific properties or characteristics of the edges in the output of the algorithm. For instance, we would like to know if there are any thick edges in the output, the minimum length of each edge, and how well the edges are connected. Except for superficial analysis, it is difficult to track and analyze these characteristics in the comparative cost function approach to edge detection.

A solution to the difficulties mentioned above is to modify the comparative cost function approach to one that uses an absolute cost function which is mathematically well grounded. As the terminology suggests, it is a function that measures absolute instead of relative quality. The function is applicable to individual edge configurations and the resulting cost of each configuration is indicative of its quality; lower cost implies better edges. This chapter deals with the formulation and analysis of the absolute cost function. From here on, we will use the term cost function to refer to the absolute cost

function. Three things are required in formulation of a cost function for evaluating edge quality:

(1) A precise concept of an edge.

(2) A mathematical description of edges and their related properties.

(3) A suitable cost function which captures the above concept of an edge.

The concept of an edge has already been described in detail in Section 2.2 of the previous chapter; we use the same concept as that of the comparative cost function. We draw attention to the fact that it is not the concept of an edge but the approach to edge detection that is different when comparing the absolute and comparative cost function techniques. A mathematical description of edges is essential as it enables us to state a precise description of the intuitive concept of an edge. Its primary purpose is to provide a basis for unambiguous definition and analysis of the cost function. The goal of the formulation is to find a suitable cost function which, when minimized, will yield edges that are consistent with the above concept of an edge. The ultimate test of its validity is in its performance in finding good edges in an image.

3.2 A Mathematical Description of Edges

The intuitive concept of an edge has been described in Section 2.2 of the previous chapter. We now describe in mathematical terms the ideas presented in the concept. Based on this description, we will be able to state the precise definition of a cost function and perform a detailed analysis of edge structures.

We will describe edges in terms similar to graph theoretic terms because of the close analogy between edges and planar graphs [29]. In fact, any edge structure can be considered to be a planar graph where each vertex in the graph corresponds to an edge pixel, and each arc in the graph corresponds to adjacent pixels in the edge structure. An example of this is shown in Figure 3.1. One approach to describe edges using graph terminology is to first transform the edges into their corresponding planar graphs. However, because of the need to keep track of the one to one correspondence between the edge pixels and the vertices, it seems unnecessarily cumbersome to describe edges in terms of planar graphs. In view of the analysis in the following sections, there seems to be no specific advantage in using a description based entirely on graphs. Instead, we will describe edges in their own context, using a number of terms that are similar to those in graph theory. The definition of these terms follow closely to their graph theoretic counterparts, but they apply directly to edge pixels and their corresponding edge structures.

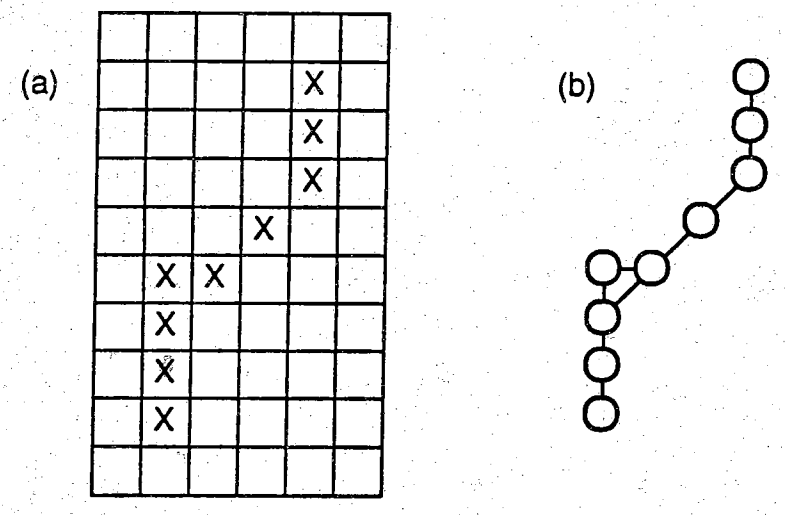


Figure 3.1. An edge with its corresponding planar graph representation. (a) An edge. (b) Planar graph representation of the edge.

3.2.1 Preliminary Definitions

In this section, we begin with some preliminary definitions of images and edge configurations. We will also define some basic terminology that will be frequently used, such as neighborhood, window, connection, path and cycle. Based on these definitions, we will state a proposition about the pixels in an image.

An *image* G is a two-dimensional array of pixels

$$G = \{ g(i, j) ;\ 1 \leq i \leq i_{max}, 1 \leq j \leq j_{max} \},$$

where each pixel g(i,j) is assumed to have gray level in the range $0 \leq g(i, j) \leq 255$. For simplicity, we will also assume that the images are square with $i_{max} = j_{max} = N$. That is, the pixels occupy the sites of an $N \times N$ uniform square lattice.

An *edge configuration* S_m is also a two-dimensional array of pixels

$$S_m = \{ s_m(i, j) ;\ 1 \leq i, j \leq N \} ,$$

where each pixel takes on a binary value 0 or 1. If $s_m(i, j) = 1$, the pixel $s_m(i, j)$ is called an *edge pixel*; otherwise it is a *non-edge pixel.*

An edge configuration can be considered to be a binary image where the gray levels take on values of either 0 or 1. As seen in the definitions, we will always denote images by uppercase letters and their pixels by the corresponding lowercase letters. We shall denote as **S**, the set of all possible edge configurations on an $N \times N$ square lattice. Since each pixel in the lattice can have one of two possible edge labelings, and since there are N^2 pixels in a configuration, the number of elements in S is equal to 2^{N^2}. Sometimes, we will refer to an edge configuration S_m simply as S, with the understanding that we are referring to any arbitrary edge configuration. The pixels of S are denoted by the corresponding lowercase letters s(i, j).

As observed in the definition, each pixel in an image or edge configuration is uniquely specified by the pair of indices (i, j) representing the location of its site in the lattice. We shall denote as L, the set of all pairs of indices for an $N \times N$ lattice of sites:

$$L = \{ (i, j) ;\ 1 \leq i, j \leq N \} .$$

Definition 3.1: The *neighborhood* of a pixel $s(i, j) \in S$ is the set of 8 pixels

specified by:

$$N_{i,j}(S) = \left\{ s(m,n) : \left| m-i \right| \leq 1 \ , \ \left| n-j \right| \leq 1 \ , \text{ and } (m,n) \neq (i,j) \right\} ,$$

where $\left| \cdot \right|$ denotes the absolute value. This is the typical "8-neighborhood" representation of connectivity in images. Notice that $s(i,j) \notin N_{i,j}(S)$; a pixel is not a member of its own neighborhood. If $s(m,n) \in N_{i,j}(S)$, then s(m,n) is a *neighbor* of s(i,j), and s(m,n) is said to be *adjacent* to s(i,j).

It is straightforward to observe that adjacency is a symmetric relation; s(i, j) is adjacent to s(m,n) implies that s(m,n) is adjacent to s(i, j). However, it is not reflexive since a pixel is not a neighbor of itself and hence cannot be adjacent to itself. When the exact location of the edge pixel s(i, j) is not of importance, for ease of notation, we will sometimes denote s(i, j) simply as e_k, for some integer value of k.

Definition 3.2 The *window* $W_{i,j}(S)$ is the set of 9 pixels contained in a 3×3 region centered at pixel s(i, j):

$$W_{i,j}(S) = \left\{ s(m,n) : \left| m-i \right| \leq 1 \text{ and } \left| n-j \right| \leq 1 \right\} .$$

Fact 3.1: $W_{i,j}(S) = N_{i,j}(S) \cup s(i, j)$
This is easily seen from the definition of window and neighborhood, and it is always true that $N_{i,j}(S) \subset W_{i,j}(S)$.

A *walk* is a non-null sequence of edge pixels $W = e_1, e_2, e_3, \ldots e_k$ such that e_i is adjacent to e_{i+1} for all $1 \leq i \leq k-1$. The *ends* of the walk are e_1 and e_k, and W is a (e_1, e_k)—walk. The *origin* of the walk is e_1, the *terminus* is e_k, and the *internal pixels* are $e_2, .., e_{k-1}$. The *length* of the walk is equal to k.

A *path* is a walk in which every edge pixel is distinct. Intuitively, a path is a walk that does not intersect or merge with itself.

Two edge pixels e_h, e_k are *connected* if there is a (e_h, e_k)—path.

Fact 3.2: Connection is an equivalence relation.

(1) e_h is connected to e_k implies that e_k is connected to e_h.

(2) e_h is connected to e_h.

(3) e_h is connected to e_k which is connected to e_l implies that e_h is connected to e_l.

A collection of edge pixels $M = \{e_1, e_2, ... e_m\}$ is *connected* if for any e_h, $e_k \in M$, there is a (e_h, e_k)—path in M.

Let $S_e \subseteq S$ be the set of all the edge pixels of edge configuration S. There is a partition of S_e into non-empty subsets S_e^1, S_e^2, ... S_e^ω such that e_h and e_k are connected if and only if they belong to the same subset. The subsets S_e^i, $1 \leq i \leq \omega$, are the *components* of S (or S_e). Clearly, the components are connected. In Figure 3.2, we show an example of an edge configuration on a 10×10 lattice that contains 4 components. Notice that one of the components contains only one isolated pixel.

Proposition 3.1: In any connected set M, such that $\| M \| > 1$, every edge pixel has at least one other edge pixel in its neighborhood.

Proof: Consider any pixel $e_a \in M$; it is always connected to some other pixel $e_b \in M$ by an (e_a, e_b)—path. If the path has a length that is greater than 2, its first internal pixel is in the neighborhood of e_a. If the length is equal to 2, then e_b is in the neighborhood of e_a.

□

A *cycle* C is a walk such that:

1) the origin and internal pixels are distinct,
2) the origin and terminus are the same,
3) there is at least 1 internal pixel.

The *length* of a cycle is the length of the corresponding walk minus 1.

Let A be any collection of pixels. The *size* of A is the number of distinct edge pixels in A, and is denoted by $\| A \|$.

3.2.2 Definition and Properties of Edges

Most of the definitions in the previous section involve edge pixels and their associated structures. Up to this point, we have not yet specified what edges are, and how they relate to edge pixels. In this section, we will specify what is meant by an edge (of S), and a segment of an edge. The term "thick" edges

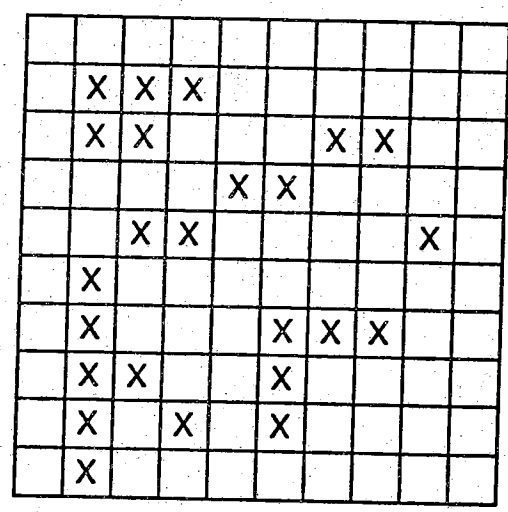


Figure 3.2. An edge configuration on a 10×10 lattice which contains 4 components.

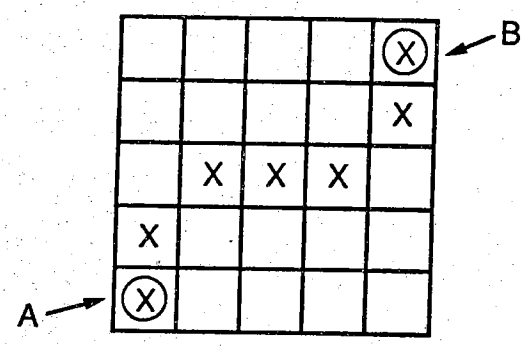


Figure 3.3. An edge which contains a unique path between pixels A and B.

has often been used without clearly defining the meaning of the term thick. We will give explicit definitions of thick and thin edges, and state several propositions concerning the structure of thin edges.

Definition 3.3: An *edge* E is a component of S.

Definition 3.4: A *segment* of an edge E is a subset of E that is connected.

Again referring to the example in Figure 3.2, based on Definition 3.3, there are 4 edges in the configuration. It should be clear from this that when we refer to an edge, we are always referring to a maximal collection of connected edge pixels. However, this collection may be just a single pixel as in the case when a component is comprised of only one isolated edge pixel.

According to the edge concept, the edges in an image should be thin. Intuitively, we know what thinness means, but mathematically, it is a term that is difficult to describe. As in the case of the cost factor for thickness in the comparative cost function, we will use the idea of multiple links to describe edge thickness. Consider an edge E that joins pixel A to pixel B in an image; E contains a path from A to B. We will say that the edge is thin when this path is unique. This is shown in Figure 3.3. However, when the path is not unique, we say that the edge is thick. A path that is not unique implies that there could be a collection of closely adjacent paths in E that would join the same pixels A and B. This collection of closely adjacent paths form what we call multiple links between A and B. An illustration of an edge containing multiple links is shown in Figure 3.4. As multiple adjacent lines form a thick line, so multiple links form a thick edge. We therefore choose to describe thin edges as edges that contain no multiple links between any of its edge pixels.

Based on an 8-neighbor representation of edges, an examination of edge structures reveals that the cycle of length three can be considered to be the basic building block of multiple links. An example of this can be seen in Figure 3.1. Notice that in the planar graph representation of the edge shown in the figure, the middle left portion of the edge contains a triangular region which is a cycle of length three. This cycle is the source of multiple links in the edge. Consequently, in the following definitions, thin edges are those that contain none of these cycles. Figure 3.5 shows a cycle of length three; notice that each pixel of the cycle is multiply linked by 2 paths to the other pixels within the same cycle. For instance, if we represent the cycle as $\{e_1, e_2, e_3, e_1\}$, the first path between e_1 and e_3 is $e_1e_2e_3$, and the second path is e_1e_3. All cycles of

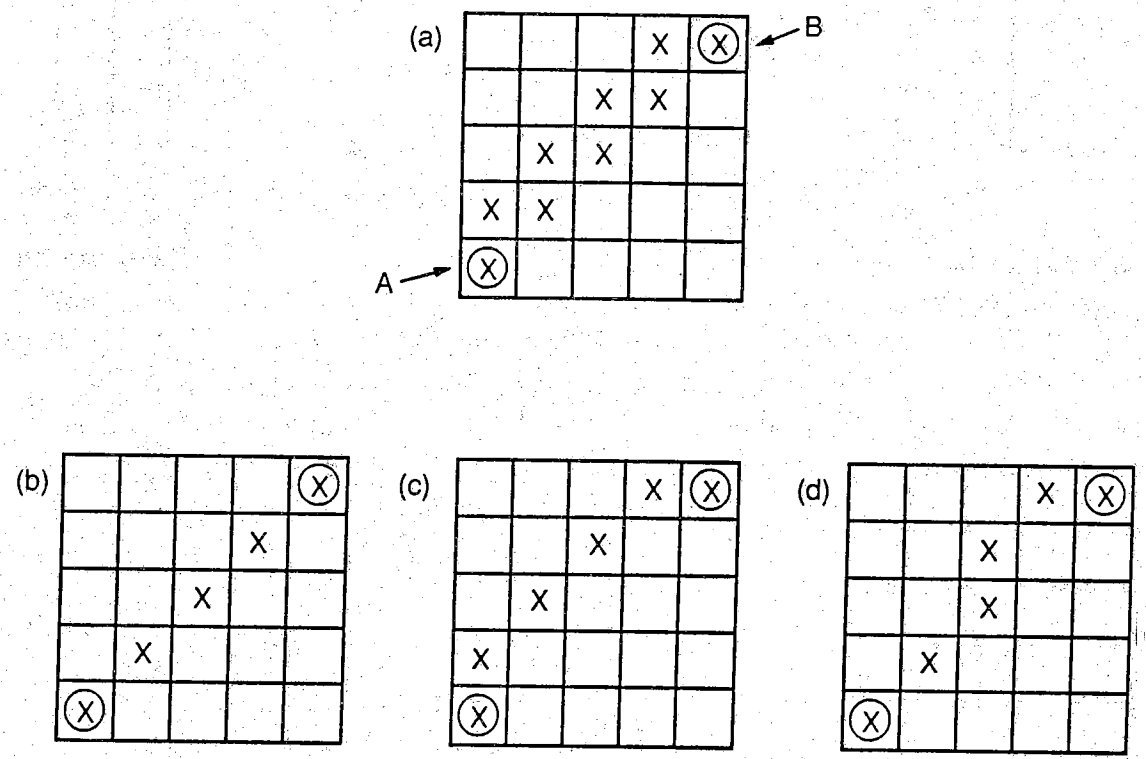


Figure 3.4. An example of an edge which contains multiple links. (a) An edge E joining pixels A and B. (b), (c) and (d) are three possible paths contained in E that join the same pixels.

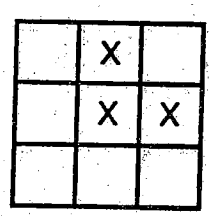


Figure 3.5. A cycle of length three.

length three have a characteristic L shape, differing only in orientation and position.

Definition 3.5: An edge pixel that is not contained in any cycle which has a length equal to three is called a *thin edge pixel*; otherwise, it is called a *thick edge pixel*.

Definition 3.6: An edge that contains only thin edge pixels is called a *thin edge*; otherwise, it is called a *thick edge*.

It is clear that edge pixels are either thick or thin. Since an edge is thick if and only if it contains one or more thick edge pixels, a thick edge can be transformed into a thin edge by the removal of the thick edge pixels. In Figure 3.6, we show several examples of thick edges, and the possible transformations of these edges into thin ones. The definition of thick and thin edges also apply to edge segments; an edge segment is thin if and only if it contains only thin edge pixels.

We now state several facts and propositions concerning cycles and the structure of thin edges which will be frequently used in the analysis of later sections.

Fact 3.3: If $e_k \in E$ is contained in any cycle C of length three, then C is contained in E.
This is a simple yet important observation from the fact that all the pixels in a cycle are connected and must belong to the same component.

Proposition 3.2: E is a thick edge if and only if E contains a cycle of length three.

Proof: If E is a thick edge, then from the definition, it must contain a thick edge pixel which is contained in some cycle of length three. By Fact 3.3, this cycle is contained in E. Conversely, if E contains a cycle of length three, then each pixel of the cycle is a thick edge pixel, and hence by definition, E is a thick edge.

□

Proposition 3.3 If C is a cycle of length three that contains the edge pixel $s_m(i, j) \in S_m$, then C is completely contained in the window $W_{i,j}(S_m)$.

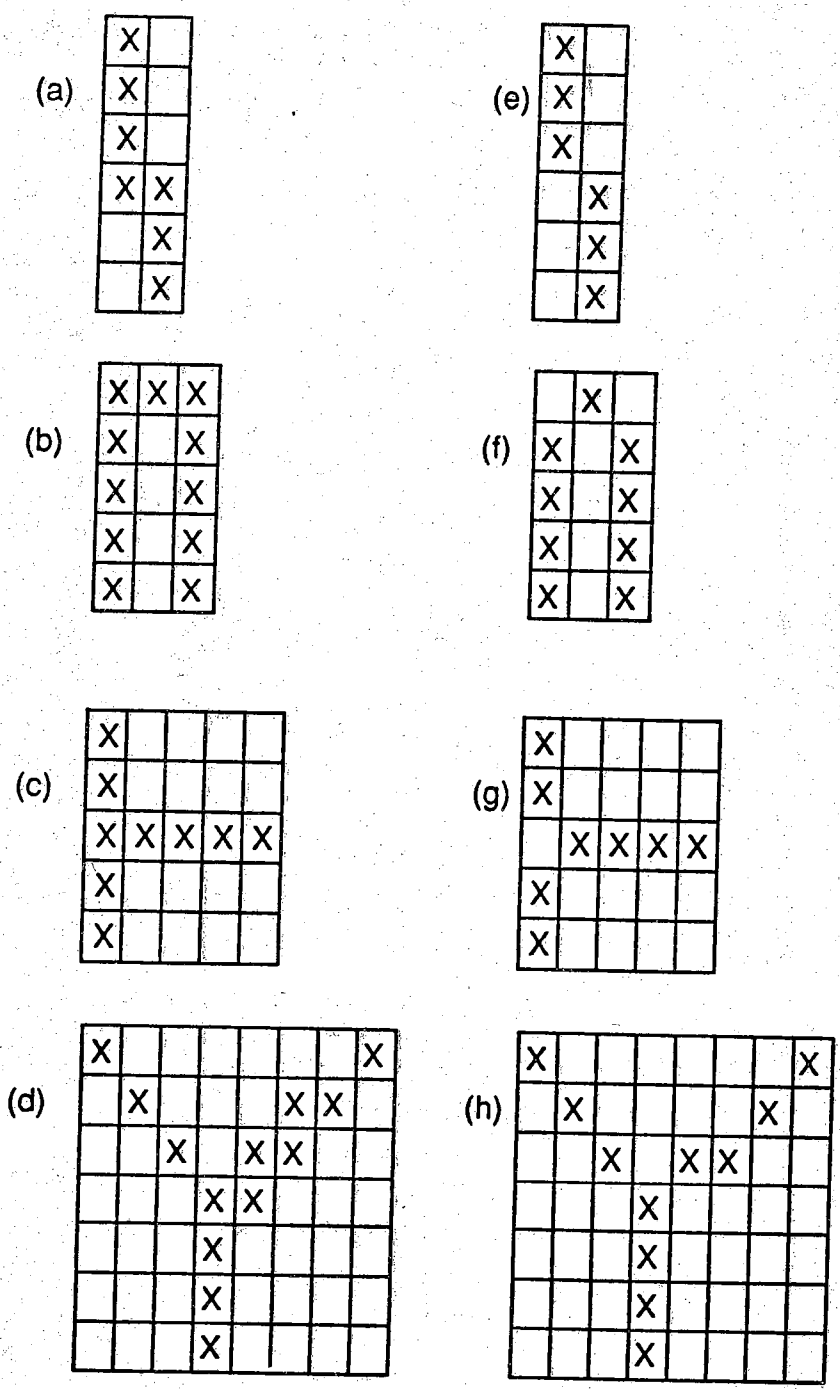


Figure 3.6. Thick and thin edges. The edges on the left, (a) to (d), are thick edges. Those on the right, (e) to (h) are thin edges obtained by the removal of several thick edge pixels from the corresponding edges on the left.

Proof: Let the cycle be represented by $C = s_m(i, j), e_1, e_2, s_m(i, j)$. Since C is a walk, the edge pixels e_1 and e_2 musty be adjacent to $s_m(i, j)$, and consequently, they must be contained in $N_{i,j}(S_m)$. Since $N_{i,j}(S_m) \subset W_{i,j}(S_m)$ (by Fact 3.1), we conclude that $C \subset W_{i,j}(S_m)$.

□

From Proposition 3.2, we conclude that one way to determine if E is a thin edge is to look for a cycle of length three in E. If one cannot be found, then E is a thin edge, otherwise it is a thick edge. Proposition 3.3 tells us that if we wish to determine whether a pixel e_k is thick or thin, we only have to consider the pixels in the window centered about e_k. That is, the pixels outside of the window do not affect the thickness or thinness property of the center pixel.

The following 5 propositions relate to the structure of thin edges in a 3×3 square lattice. They list the different kinds of thin edge structures that can exist in the lattice.

Proposition 3.4: Any edge E such that $\| E \| \leq 2$ is a thin edge.

Proof: The proof is trivial since for E to be a thick edge, it has to contain a cycle of at least 3 distinct edge pixels. This is impossible since E has at most 2 pixels.

□

Proposition 3.5: The only possible thin edge E contained in a 3×3 square lattice such that the center is an edge pixel and $\| E \| = 3$, is one of the 16 structures shown in Figure 3.7.

Proof: By construction and use of Proposition 3.2. Of the 28 structures satisfying the above condition, only these 16 contain no cycle of length three.

□

Proposition 3.6: The only possible thin edge E contained in a 3×3 square lattice such that the center is an edge pixel and $\| E \| = 4$, is one of the 8 structures shown in Figure 3.8.

Proof: By construction and use of Proposition 3.2. Of the 56 structures satisfying the above condition, only these 8 contain no cycle of length three.

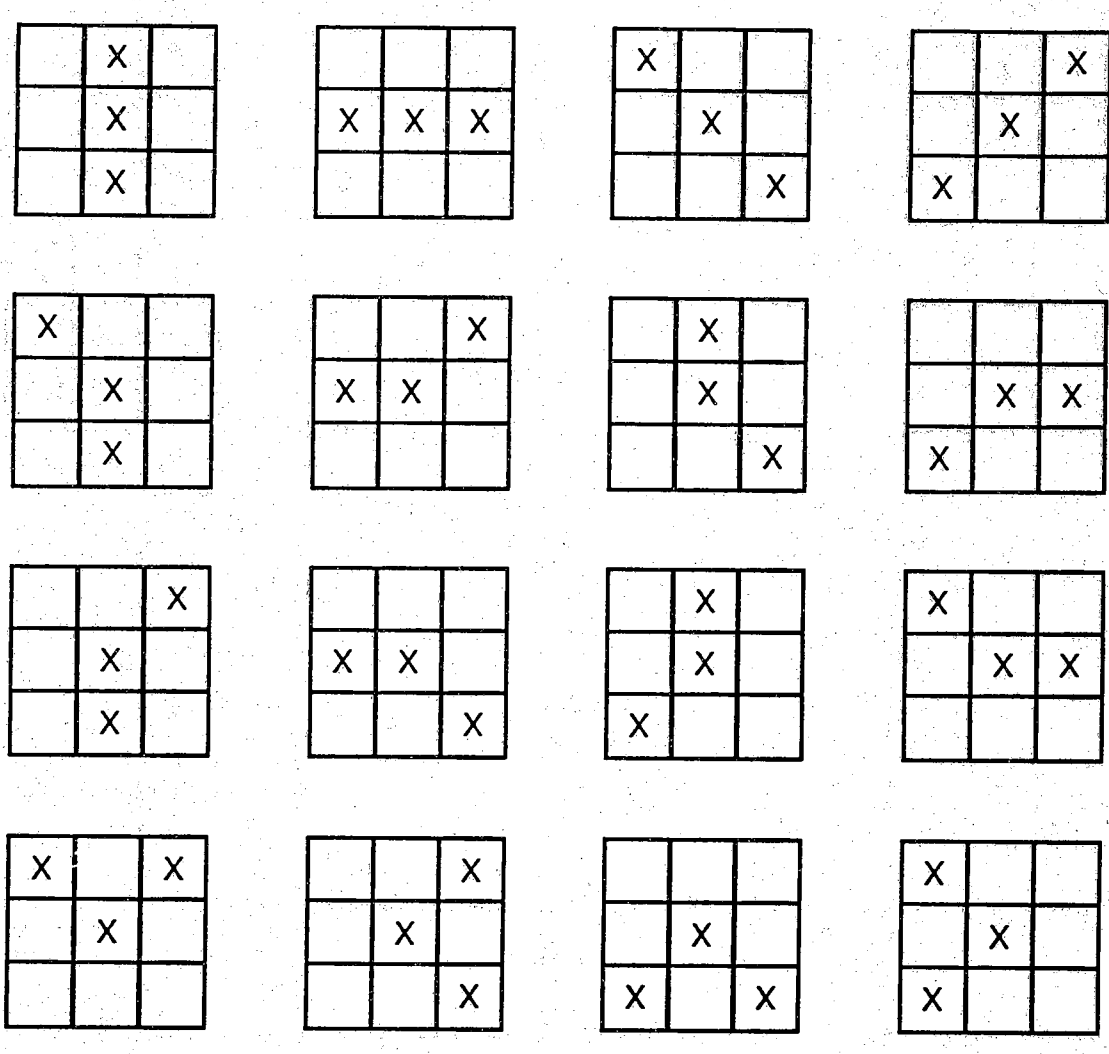


Figure 3.7. The 16 thin edge structures in a 3×3 lattice. Each of the structures has 3 edge pixels.

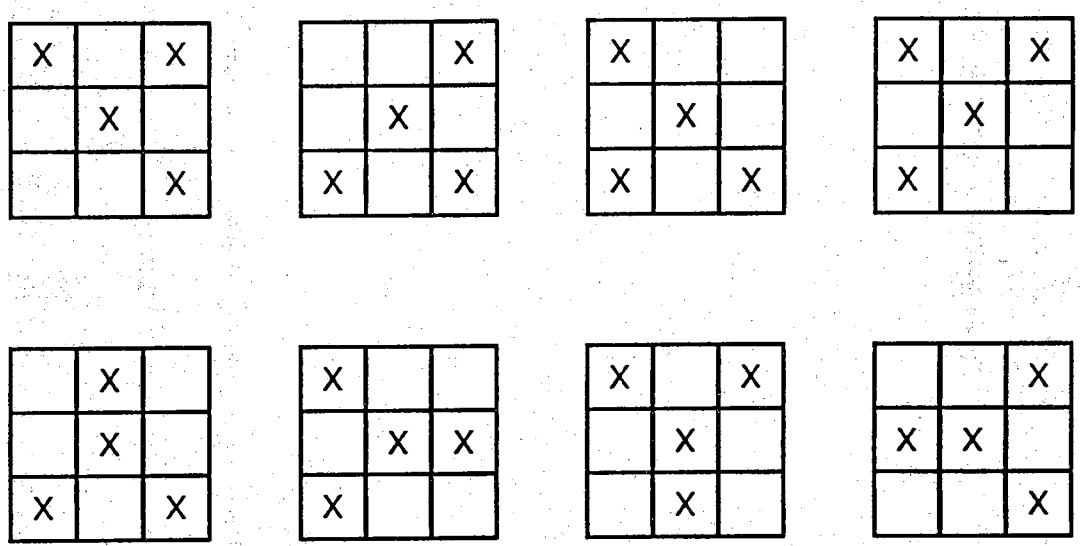


Figure 3.8. The 8 thin edge structures in a 3×3 lattice. Each of the structures has 4 edge pixels.

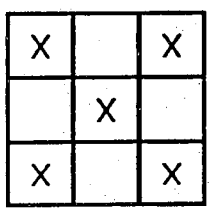


Figure 3.9. The only thin edge structure on a 3×3 lattice which contains five edge pixels.

□

Proposition 3.7: The only thin edge E contained in a 3×3 square lattice such that the center is an edge pixel and $\| E \| = 5$, is the structure shown in Figure 3.9.

Proof: By construction and use of Proposition 3.2. Of the 70 structures satisfying the above condition, only this contains no cycle of length three.

□

Proposition 3.8: Any edge E contained in a 3×3 square lattice such that the center is an edge pixel and $\| E \| > 5$ is a thick edge.

Proof: By construction and use of Proposition 3.2. Each of the 98 edge structures satisfying the above condition contains at least one cycle of length three.

□

The above propositions hold only for small lattices of size 3×3, and may seem irrelevant as they cannot be directly applied to real images of larger size. However, their importance is seen when they are used in conjunction with Proposition 3.3 and the next proposition. These propositions together provide the basis for an alternative method of determining whether an edge pixel is thick or thin.

Proposition 3.9: Let E be an edge contained in the window $W_{i,j}(S)$ such that the center pixel s(i, j) is an edge pixel. Then E is a thick edge if and only if s(i, j) is a thick edge pixel. Similarly, E is a thin edge if and only if s(i, j) is a thin edge pixel.

Proof: If E is a thick edge, then by Proposition 3.2 it must contain a cycle of length three. By construction, every cycle of length three contained in a 3×3 lattice must include the center pixel. Hence, the center pixel must be a thick edge pixel. Conversely, if the center pixel is a thick edge pixel, then it must belong to E. Thus by definition E is a thick edge. The proof of the second statement follows trivially from the first.

□

To determine if s(i, j) is a thin/thick edge pixel, by Proposition 3.3, we simply have to consider the pixels in the window $W_{i,j}(S)$. But by Proposition

3.9, this is the same as determining if the edge E in $W_{i,j}(S)$ is a thin/thick edge. All the possible thin edges in a 3×3 square lattice are given in Propositions 3.4 through 3.8. Hence an alternative method of determining if s(i, j) is a thin/thick edge pixel is to see if the edge structure in $W_{i,j}(S)$ is identical to any one of the thin edge structures listed in Propositions 3.4 to 3.8. If it is, then s(i, j) is a thin edge pixel; otherwise, it is a thick edge pixel. We will make use of this fact in Section 3.3.3 to reduce the amount of computation required to determine the value of the cost function.

3.3 A Cost Function for Evaluating Edges.

Having established the necessary mathematical preliminaries and definition of edges in the previous sections, we now turn our attention to the formulation of a cost function for evaluating edges. As mentioned in the introduction to this chapter, we seek to use an absolute cost function that measures absolute quality of edges instead of relative quality. The function should be applicable to individual edge configurations by assigning a cost value to each configuration. The configuration with the lowest cost corresponds to the best configuration in the sense that it is most consistent with our concept of an edge.

The motivation and approach to the formulation of the absolute cost function is very similar to that of the comparative cost function. In fact, we will employ essentially the same form of the cost function, using a linear combination of weighted cost factors. As in the comparative cost function, each cost factor captures a desirable characteristic of edges. However, it will be seen that the definition of the absolute and comparative and cost functions differ in several important aspects. First, the absolute cost function is defined using only one single edge configuration as its argument, while the comparative cost function uses two configurations. Second, although they have the same form, the definition of four out of five of the cost factors are different for the two functions.

We will first describe the general form of the absolute cost function, and then describe the cost factors. We will also state a number of propositions that will aid us in the computation of the cost.

Again, let $S_m \in \mathbf{S}$ be an edge configuration and L be the set of all pairs of indices for an $N \times N$ lattice of sites: $L = \{ (i, j) ; 1 \leq i, j \leq N \}$.

Definition 3.7: The *point cost* of S_m at site $l = (i, j) \in L$ is defined as the

following linear sum of weighted cost factors:

$$\mathbf{C}_p(S_m, l) = \Big[w_c C_c(S_m, l) + w_d C_d(S_m, l) + w_e C_e(S_m, l) + w_f C_f(S_m, l) + w_t C_t(S_m, l) \Big]$$

$$= \sum_{k=1}^{5} w_k C_k(S_m, l) \tag{3.1}$$

where $w_k \geq 0$ and $0 \leq C_k \leq 1$.

Definition 3.8: The *total cost* of edge configuration S_m is the sum of the point cost at every point in the image:

$$\mathbf{F}(S_m) = \sum_{l \in L} \mathbf{C}_p(S_m, l) \tag{3.2}$$

or equivalently,

$$\mathbf{F}(S_m) = \sum_{l \in L} \left[\sum_{k=1}^{5} w_k C_k(S_m, l) \right]. \tag{3.3}$$

This total cost is the absolute cost function for evaluating edges. We will often omit the term absolute and refer to this simply as the cost function when there is no confusion with that of the comparative cost function. Notice that the cost function is the sum of the point cost at every site in the image, and also takes the form of a linear sum of weighted cost factors.

Definition 3.9: For any pair of edge configurations $S_m, S_n \in \mathbf{S}$, the *incremental cost* from S_m to S_n is given by

$$\Delta\mathbf{F}(S_m, S_n) = \mathbf{F}(S_n) - \mathbf{F}(S_m) \tag{3.4}$$

$$= \sum_{l \in L} \mathbf{C}_p(S_n, l) - \sum_{l \in L} \mathbf{C}_p(S_m, l)$$

$$= \sum_{l \in L} \Big[\mathbf{C}_p(S_n, l) - \mathbf{C}_p(S_m, l) \Big] \tag{3.5}$$

$$= \sum_{l \in L} \left[\sum_{k=1}^{5} w_k C_k(S_n, l) - \sum_{k=1}^{5} w_k C_k(S_m, l) \right]$$

$$\Delta F(S_m, S_n) = \sum_{k=1}^{5} w_k \left\{ \sum_{l \in L} \left[C_k(S_n, l) - C_k(S_m, l) \right] \right\} \tag{3.6}$$

Alternatively, we can write the incremental cost given in Equation (3.6) as a sum of five *incremental cost factors*, ΔC_k.

$$\Delta F(S_m, S_n) = \sum_{k=1}^{5} w_k \Delta C_k(L\ ;\ S_m,\ S_n) \tag{3.7}$$

$$\text{where } \Delta C_k(L\ ;\ S_m,\ S_n) = \sum_{l \in L} \left[C_k(S_n, l) - C_k(S_m, l) \right]$$

For notational simplicity, we will often write $\Delta F(S_m, S_n)$ simply as $\Delta F_{m,n}$. Notice that while Equation (3.4) is the basic definition of the incremental cost, Equation (3.5) expresses it in terms of the point cost, and Equation (3.7) expresses it in terms of the incremental cost factors. A comparison of Equation (3.7) with Equation (2.2) shows that the two equations are very similar in form. However, because of the difference in the definition of the cost factors, the results produced using the two equations are significantly different. The incremental cost $\Delta F_{m,n}$ gives the cost difference between configurations S_m and S_n. If it is negative, then S_n has a lower cost than S_m, and is consequently a better configuration. Conversely, if it is positive, then S_m is better.

Proposition 3.10: Let $\{S_1, S_2, \ldots S_m\} \subset \mathbf{S}$ be any collection of edge configurations. The incremental cost from S_1 to S_m can be written as the sum

$$\Delta F_{1,m} = \sum_{i=1}^{m-1} \Delta F_{i,i+1}$$

Proof:

$$\sum_{i=1}^{m-1} \Delta F_{i,\, i+1} = \Delta F_{1,2} + \Delta F_{2,3} + \Delta F_{3,4} \ldots + \Delta F_{m-1,\, m}$$

$$= F(S_2) - F(S_1) + F(S_3) - F(S_2)$$

$$+ \mathbf{F}(S_4) - \mathbf{F}(S_3) \ \ldots\ + \mathbf{F}(S_m) - \mathbf{F}(S_{m-1})$$

$$= \mathbf{F}(S_m) - \mathbf{F}(S_1)$$

$$= \Delta \mathbf{F}_{1,m}$$

□

This proposition provides an indirect method of computing the incremental cost from S_1 to S_m. This indirect method is very useful, especially when it is difficult or computationally inefficient to determine the incremental cost value directly. In most of our applications, this method of computing the incremental cost will be used. This proposition is also useful in analysis; in Section 3.4, we will use this property of the incremental cost in the proof of a number of other propositions.

Figure 3.10 shows a block diagram of our cost minimization approach to edge detection. The fundamental property of edges is that they separate regions that are dissimilar. The first step in the detection process is to enhance those points in an image that satisfy this fundamental property of edges. These points serve as good candidates for edge points. The enhancement is based on a given dissimilarity measure and a enhancement scale factor. We refer to this processing stage as dissimilarity enhancement. In this stage, we also attempt to ensure that the enhanced points satisfy the desirable edge property of accurate localization. It will be seen that this property will be achieved by using non-maximal suppression for the dissimilarity values.

Instead of using the original image directly, the cost function is defined in terms of the enhanced image. Desirable characteristics of edges such as thinness and continuity that are difficult to capture in the dissimilarity enhancement stage are embedded into the cost function. The edges are detected by finding a suitably low cost solution to the cost function. Simulated Annealing will be employed as a technique of finding low cost solutions. As seen in Equation (3.3), the cost function is a weighted sum of five cost factors. The choice of weights for the cost factors is application dependent, and it determines the nature of the edges which will be detected.

In the following sections, we will elaborate on dissimilarity enhancement and the definition of the cost function. We will also analyze the cost function and provide guidelines on the choice of weights to achieve specific characteristics in the detected edges.

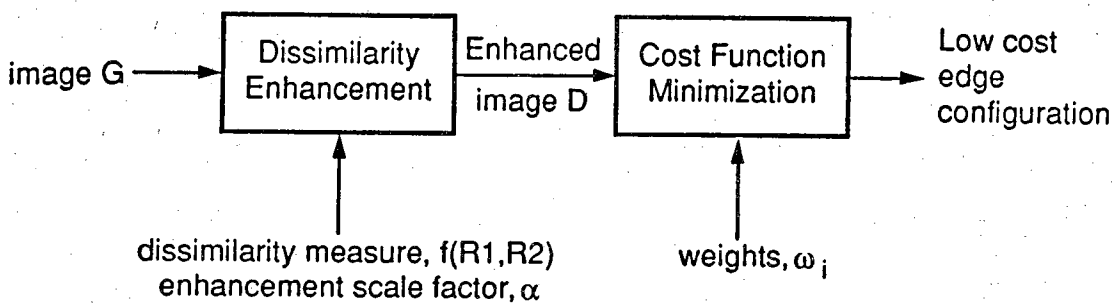


Figure 3.10. A block diagram of the cost minimization approach to edge detection.

3.3.1 Determining Region Dissimilarity

We have mentioned in Section 2.2 that an edge is a boundary in an image that separates regions that are dissimilar. In this section, we focus on the task of enhancing the points in an image that are good candidates for edge points. The enhancement procedure is very much dependent on how dissimilarity is defined in an image. For instance, we could model an edge as an ideal step and define dissimilarity to mean that the region on each side of the edge has different constant gray levels. In this case, a possible method of enhancement is to convolve the image with a gradient operator to obtain the enhanced image.

Dissimilarity based on the ideal step is only one of a myriad of possible region dissimilarities that could exist in an image. Instead of focusing on one specific kind of dissimilarity, we will give a general definition of region dissimilarity in the form of a dissimilarity function. We will describe a procedure that uses this function to enhance the points in an image that have a high degree of dissimilarity in its neighboring regions. These points are good candidates for edge points based on the criterion that edge points separate dissimilar regions. However, for reasons to be stated in the following paragraphs, we emphasize that region dissimilarity itself provides insufficient information for good edge detection.

Referring again to Figure 3.10, the first step in the detection process is to obtain an enhanced image from the original image. The edges are then detected by finding the edge configurations that minimize the cost function. Thresholding the enhanced image can be considered to be the simplest form of cost minimization where the cost function does not take into account edge structure information. The required complexity of the cost function and the subsequent minimization procedure is very much dependent on the performance of the dissimilarity enhancement stage. For instance, if we could have perfect performance at the enhancement stage in the sense that the dominant features in the enhanced image follow closely to our concept of an edge, then the edges could be detected by a simple thresholding operation.

However, in practice, it is impossible to have perfect performance in dissimilarity enhancement so that high quality edges can be obtained by simple thresholding. This is because of two main reasons. First, region dissimilarity based on the original image data often provides insufficient information for edge detection. Good edges are those that exhibit the desirable characteristics of accurate localization, thinness, continuity, and sufficient length. Some of

these characteristics, particularly the last three, are structural characteristics of edges that are difficult to determine directly from the image data. They are embedded in the structure of the edge configuration. Second, dissimilarity enhancement is a process that is usually sensitive to noise. Noise processes will cause many points to be incorrectly enhanced as potential edge points. Except for artificial images, noise is always present in an image.

The above discussion leads us to conclude that it is necessary to exploit information from local or global edge structure to aid in the detection process. Our approach to detect edges is to attempt to achieve the best we can at the enhancement stage. Desirable edge characteristics that are not captured in the enhanced images are embedded into the cost function. The cost minimization procedure will then find the edges which exhibit the characteristics that are consistent with our concept of an edge.

The fundamental property of edges is that they separate dissimilar regions. In dissimilarity enhancement, we concentrate on the following two goals that relate to our concept of an edge.

(1) To signify those points in an image that possess the fundamental property of edges.

(2) To ensure that those enhanced points are accurately localized.

The enhanced image

$$D = \{ d(i, j) ; 1 \leq i, j \leq N \}$$

is a collection of pixels where each pixel value is proportional to the degree of region dissimilarity that exists at that pixel site. The pixel values lie in the range $0 \leq d(i, j) \leq 1$. Pixels with large values close to 1 are good candidates for edge points in an image. Three things are required in order to enhance an image according to the set goals:

(1) Well defined regions of interest on either sides of an edge.

(2) A function that measures dissimilarity between the regions of interest.

(3) Non-maximal suppression as a method of ensuring accurate localization.

The regions of interest are defined with reference to a set of selected edge structures. We call this set of edge structures the basis set. Within the scope of this report, the basis set is constrained to be 3-pixel edge structures contained in a 3×3 window region. In line with our concept of an edge, we also require these structures to be thin. The basis set is thus selected from the 16 edge structures given in Proposition 3.5. In most of our applications, we selected as

our basis set the first 12 of the 16 structures shown in Figure 3.7. The regions of interest on ether sides of each edge of the basis set are defined in the same way as those for the comparative cost function in Section 2.3.2. These regions are again labeled as R1 and R2 for each edge structure in the basis set. Figures 2.9 and 2.10 show an examples of the basis set and the corresponding regions of interest for each edge structure.

The function that measures the dissimilarity between regions R1 and R2 is denoted by $f_a(R1,R2)$. This measure could be a simple difference of gray level averages in R1 and R2, or it could be more complicated measures based on statistical or structural properties in the gray levels. Depending on the application and the features of interest in an image, there are numerous possibilities for the definition of $f_a(R1,R2)$. As previously mentioned, to find step edges in an image, we could define the dissimilarity measure to be the difference of constant gray levels in the regions R1 and R2. It is clear that there is extreme flexibility in such an approach to dissimilarity enhancement as we do not restrict the nature of the dissimilarity. This is in contrast to many detection algorithms that assume some specific nature of edges and are devoted to finding only those edges. At this point, we do not need to know the explicit definition of the dissimilarity measure $f_a(R1,R2)$ which will be used; we simply assume that one exists.

Non-maximal suppression is important in ensuring the accurate localization of an edge point in an image. In practically all real images, the dissimilarity measure has the tendency to enhance the points in the vicinity of the true boundary in addition to enhancing the boundary itself. This is undesirable as a large number of false boundary points are enhanced. One approach to mitigate this tendency is to employ non-maximal suppression in dissimilarity enhancement. However, an undesirable side effect that results in using non-maximal suppression is that some true boundary points may also be suppressed together with the false points. This may increase the amount of fragmentation in the boundary. It will be seen that the cost factor for fragmentation will compensate for this effect by linking together locally disconnected edges.

We now describe a procedure to obtain an enhanced image D from the original image G. It performs non-maximal suppression by shifting the edge structure in a direction perpendicular to the edge direction. The procedure is as follows:

(1) Initially, all the pixels d(i, j) are set equal to zero.

(2) At each pixel site (i,j), we perform steps A and B.

A. Each of the edge structures of the basis set is fitted onto the site by centering it on the location (i,j) in G. The corresponding paired regions R1 and R2 in G are determined for each structure, and the value of f_a(R1,R2) is computed. The structure that results in the maximum value of f_a(R1,R2) is chosen as the best fitted edge structure.

Note that each edge structure of the basis set contains exactly three edge pixels; we will denote the sites of the three edge pixels of the best fitted edge structure in G as (i, j), (i_1, j_1), and (i_2, j_2).

B. Next, we perform non-maximal suppression by shifting the location of the chosen best fitted edge in a direction determined by the edge structure. For vertical, horizontal and diagonal edge structures, the shifting is performed by moving the edge location by one pixel in each of the opposite directions perpendicular to the edge. For all other edge structures, the shifting is done by moving the edge one location in each of the the four directions: up, down, left and right. Figure 2.13 shows how the edges are shifted for three edge types. For each shifted edge, we determine the new regions for R1 and R2, and compute the corresponding value of f_a(R1,R2).

One of the following two cases results:

(i) If no larger value of f_a(R1,R2) results from shifting the best fitted edge structure, we set

$$\delta = \alpha \cdot \frac{f_a(R1,R2)}{3},$$

where f_a(R1,R2) is determined using the best fitted edge. The factor α is called the *enhancement scale factor*. We then increment the value of each of the pixels d(i, j), d(i_1, j_1), and d(i_2, j_2) by δ.

(ii) If there is a larger value of f_a(R1,R2) from one of the shifted edge structures, we do not alter any pixel value.

(3) Finally, the values of the pixels d(i, j) at all sites are truncated to a maximum of 1.

Step (3) is performed essentially to ensure that the dissimilarity values lie in the assumed range $0 \leq d(i, j) \leq 1$. The value of the enhancement scale

factor α is application dependent. It serves as a selection parameter in determining the number of edge points that will be detected. Section 3.4.3 gives guidelines to selecting the value of α.

3.3.2 Defining the Cost Factors

The general form of the cost function has been given in Equation (3.3); it is a linear combination of five weighted cost factors. Specifically, the five factors are as follows.

(1) Cost for curvature

(2) Cost for region dissimilarity

(3) Cost for number of edge points

(4) Cost for fragmentation

(5) Cost for edge thickness

In this section, we will define each of these cost factors and discuss their relevance to edge evaluation. Each of these factors affect a desirable characteristic of edges. It will be seen that the cost for region dissimilarity is the only one that is based on information from the image data; the others are based on information from local edge structure. Ideally, each cost factor should affect one and only one desirable characteristic so that the relative importance of each characteristic can be appropriately emphasized by their corresponding weight. In practice however, this is difficult to achieve as the different characteristics often exhibit some form of dependency on each other.

The cost factors together give an objective measure of how well a given edge configuration fits our concept of an edge. These factors are defined based on the assumption that lower cost configurations are better edges. Consequently, the best configuration is the one that achieves the global minimum of the cost function. The ultimate test of the validity of the cost function is in its performance in detecting edges. In Chapter 5, we will show experimental results of detecting edges using this cost function.

In order to define the cost factors, we have to first specify what is meant by a straight edge and an endpoint. These are given in the following two definitions.

Definition 3.10: An edge E (or segment of an edge) is *straight* if all its edge pixels lie on a single horizontal, vertical or diagonal line of the lattice on which it is defined.

Definition 3.11: An *endpoint* is an edge pixel that has at most one other edge pixel in its neighborhood.

Using the definition of straight edges and endpoints, we will now specify what is meant by the *angle of turn* at a point. If e_k is an edge pixel that is not an endpoint, then it can be considered to be the connection point (or common point) of at least one pair of straight edge segments. The direction of each straight edge segment is uniquely specified by the straight line joining the site of e_k with the site of any other pixel of the segment. This is illustrated in Figure 3.11. Let n be the maximum number of different pairs of straight edge segments connected at e_k, each pair being denoted by the label p_i, $1 \leq i \leq n$. Let ϕ_i be the larger of the two angles between the edge segments of p_i. The *angle of turn* between the pair of edge segments in p_i is given by

$$\theta_i = \phi_i - 180.$$

In Figure 3.12, we show an example of an edge pixel that is the connection point of 3 pairs of straight edge segments.

Definition 3.12: The *curvature* $\theta(l)$ at any site $l \in L$ of configuration S is defined as follows:

(1) If $s(l)$ is a non-edge pixel or an endpoint, then the curvature is equal to zero.

(2) If $s(l)$ is an edge pixel that is not an endpoint, then the curvature is the maximum angle of turn at that point:

$$\theta(l) = \max_i \left\{ \theta_i(l) \right\} .$$

Assuming that the image lattice is uniformly spaced, the curvature at any site can take on one of four possible values; $\theta \in \{ 0, 45, 90, 135 \}$. In the case of the example in Figure 3.12, the curvature is 135 degrees.

Cost for curvature

The cost for curvature assigns a cost to each point in the edge configuration according to the value of the curvature at that point. As previously mentioned, the curvature at any point can take on any one of the possible values of 0, 45, 90, or 135 degrees. At site l of configuration S_m, the curvature cost $C_c(S_m, l)$ is given in Table 3.1.

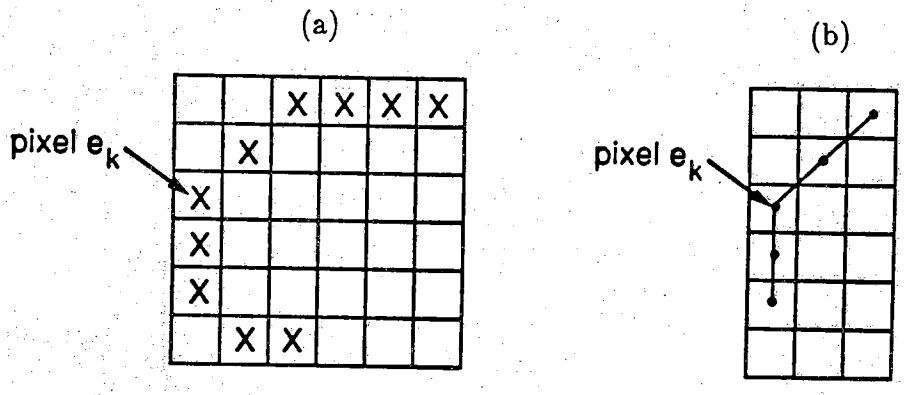


Figure 3.11. The angle of turn at a point. (a) The pixel e_k is a connection point of a pair of straight edge segments. (b) The pair of straight edge segments and the resulting angle of turn. In this case, the angle of turn is 45 degrees.

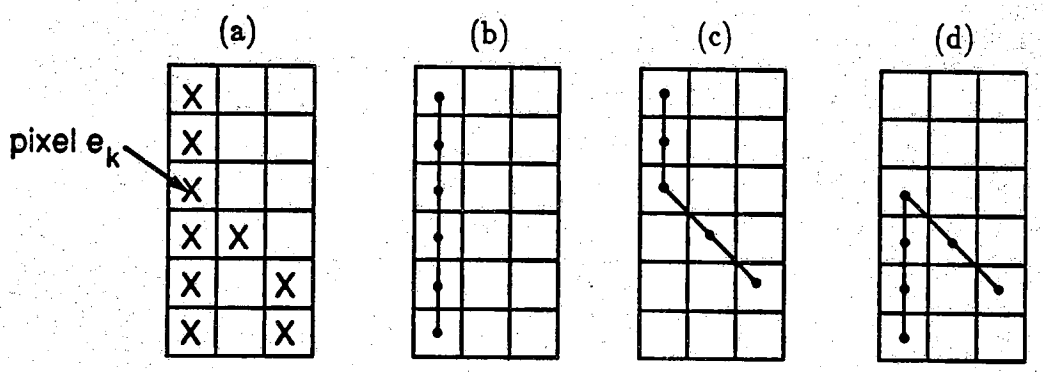


Figure 3.12. An edge pixel that is the connection point of 3 pairs of straight edge segments. (a) The pixel e_k as part of the edge. (b) A pair of segments with θ=0. (c) A pair of segments with θ=45degrees. (d) A pair of segments with θ=135 degrees.

Table 3.1. Curvature cost at pixel l

Curvature $\theta(l)$	Cost $C_c(S_m, l)$
0	0
45	0.5
90	1.0
135	1.0

The above assignment causes edges that have many turns to have a higher curvature cost than those with relatively few turns. By appropriately choosing the weight of the curvature cost, we can avoid excessive meandering and turning of edges. This is particularly useful when, for instance, we know a priori that the edges of interest are straight. Such edges often occur when we are dealing with polygonal objects. This factor is also useful in the suppression of noise effects. Noise in an image often stimulates the formation of jagged edges which have high curvature cost. A sufficiently large weight for curvature will tend to smooth out such edges.

Cost for region dissimilarity

This cost factor is based on the enhanced image D. It assigns a cost to non-edge pixels that is proportional to the degree of dissimilarity at that point. In other words, a site that contains a non-edge pixel but has a high degree of dissimilarity will have a high cost. On the other hand, if it has a low degree of dissimilarity, then the cost is low. This factor is intended to be used in conjunction with the cost for number of edge points. It will favor the placement of edge pixels at points of high region dissimilarity. The definition of the cost factor is as follows:

$$C_d(S_m, l) = \begin{cases} 0, & \text{if } s_m(l) = 1 \\ d(l), & \text{if } s_m(l) = 0. \end{cases}$$

Cost for number of edge points

When used by itself, the cost for region dissimilarity will favor the placement of edge pixels at all points in an image that have non-zero dissimilarity values. This will result in an excessive number of edge pixels being detected. To compensate for this, we assign a cost to each additional edge pixel as follows:

$$C_e(S_m, l) = \begin{cases} 0, & \text{if } s_m(l) = 0 \\ 1, & \text{if } s_m(l) = 1. \end{cases}$$

Cost for fragmentation

This cost factor reduces fragmentation by assigning a cost to the endpoints of an edge. It is based on the intuition that fragmentation causes the formation of surplus endpoints. For example, a straight continuous edge contains two endpoints; the same edge fragmented in two places will contain six endpoints.

There are two kinds of endpoints. The first is an endpoint that is the terminus of some segment or path. The second is an isolated endpoint. An isolated endpoint can be considered to be a path that has shrunk in length to a single point. In the process, the former two endpoints of the path are merged into one single point. Hence, as will be seen in the definition, the cost of an isolated endpoint is twice that of a path endpoint. By assigning a cost to endpoints, fragmentation will be reduced. This is because adjacent endpoints which represents locally disconnected edges will be removed by linking the edges together. The cost is defined in the following way:

Let T be the number of edge pixels in the neighborhood of pixel $s_m(l)$ in configuration S_m.

$$C_f(S_m, l) = \begin{cases} 0, & \text{if } s_m(l) \text{ is not an endpoint} \\ 0.5, & \text{if } s_m(l) \text{ is an endpoint and } T = 1 \\ 1.0, & \text{if } s_m(l) \text{ is an endpoint and } T = 0. \end{cases}$$

Our concept of an edge includes the property of minimum length; edges should be at least 3 pixels long. Although it is not obvious in the definition, it will be seen later that the cost for fragmentation will guarantee that detected edges are of a certain minimum length. Hence, unlike the comparative cost approach, we do not need a separate cost factor for edge length to ensure that edges are of a given minimum length. The minimum length property is inherently embedded in the cost factor for fragmentation.

Cost for edge thickness

Since thinness is a desirable edge property, we foster the formation of thin edges by assigning a cost to thick edge pixels. This is achieved by the following cost factor for edge thickness.

$$C_t(S_m, l) = \begin{cases} 0, & \text{if } s_m(l) \text{ is not a thick edge pixel} \\ 1, & \text{if } s_m(l) \text{ is a thick edge pixel.} \end{cases}$$

In Section 3.4, the practical consequence of the above definitions of the cost factors will be examined. We will also consider the choice of weights to achieve specific characteristics in the detected edges. In the next section, we concern ourselves with the question of how to compute the cost factors efficiently.

3.3.3 Computing the Cost

The cost function is used to evaluate the quality of an edge configuration. It will be seen in the minimization procedure that this function will be used repeatedly in the search for low cost configurations. Hence, from a computational standpoint, it is of major importance that this function can be computed in an efficient way. By taking into account the interdependence of the cost factors, a great deal of computation time can saved. We will now state the first of several propositions that will aid us in finding an efficient procedure to compute the cost.

Proposition 3.11: The point cost $C_p(S_m, l)$ is dependent only on the dissimilarity value $d(l)$ and on the pixels in the window $W_l(S_m)$.

Proof: Since the point cost is the sum of 5 cost factors,

$$C_p(S_m, l) = \sum_{k=1}^{5} w_k C_k(S_m, l)$$

it suffices to show that each of the 5 factors is dependent only on $W_l(S_m)$ and $d(l)$.

(1) $C_c(S_m, l)$

Case 1: $s_m(l)$ is not an edge pixel. It follows trivially from the definition that C_c is dependent only on $s_m(l) \in W_l(S_m)$.

Case 2: $s_m(l)$ is an edge pixel that is an endpoint. Since an endpoint is determined by considering a pixel and its neighborhood, it is seen easily for this case that C_c is dependent only on $s_m(l) \cup N_l(S_m) = W_l(S_m)$.

Case 3: $s_m(l)$ is an edge pixel that is not an endpoint; it is the connection point of n pairs of straight edge segments. Since the direction of each straight edge segment is uniquely determined by the straight line connecting $s_m(l)$ with any other pixel of the segment, we can choose the other pixel to be the one in its neighborhood, $N_l(S_m)$. This is always possible by Proposition 3.1. Hence, the direction of each segment is uniquely determined by the pixels in $N_l(S_m)$. The curvature at l, the endpoint property (see Case 2), and consequently C_c, are dependent only on the pixels in $N_l(S_m) \cup s_m(l) = W_l(S_m)$.

(2) $C_d(S_m, l)$

From the definition, it is trivially seen that C_d is dependent only on $s_m(l) \in W_l(S_m)$ and $d(l)$.

(3) $C_e(S_m, l)$

Again, from the definition, it is trivially seen that C_e is dependent only on $s_m(l) \in W_l(S_m)$.

(4) $C_f(S_m, l)$

This factor assigns a cost to endpoints. Whether a pixel is an endpoint is determined solely by the pixel itself and its neighborhood $N_l(S_m)$. Hence, this factor depends only on $W_l(S_m)$.

(5) $C_t(S_m, l)$

This factor assigns a non-zero cost to thick edge edge pixels. An edge pixel is thick if and only if it is contained in a cycle of length 3. According to Proposition 3.3, this cycle, if it exists, is completely contained in $W_l(S_m)$. Hence the factor is dependent only on the pixels in $W_l(S_m)$.

Since each of the 5 factors are dependent only on $W_l(S_m)$ and $d(l)$, so the linear combination of them is also dependent only on these pixels.

□

Computing the point cost

From the above proposition, we only have to consider the pixels in the 3×3 window about a site to compute the point cost at that site. A straightforward method of computing the point cost is to determine the values of each of the cost factors independently. However, this is computationally inefficient as we do not take into account the inter-dependence of the cost factors. Our method of computing the cost function is based on the decision tree show in Figure 3.13. This tree is obtained by pooling together all the information affecting the different cost factors. It represents a compact description of the cost factors, and it allows for the simultaneous computation of several cost factors by traversing the tree from root to leaf.

As mentioned in Section 3.2.2, Proposition 3.9 gives an alternative method of determining if a given pixel s(i, j) is a thin/thick edge pixel. All that is needed is to see if the edge structure contained in $W_{i,j}(S)$ is identical to any of the thin edge structures in Propositions 3.4 through 3.8. If it is, then s(i, j) is a thin edge pixel; otherwise it is a thick edge pixel. By using this method, we avoid the need to trace an edge pixel to see if it belongs to a cycle of length three. Contour tracing is time consuming compared to the alternative method we have just described.

It is important to note that the decision tree in Figure 3.13 gives an equivalent definition for each of the cost factors we have defined in the previous section. The validity of this tree in representing the cost factors is hinged on Propositions 3.3 to 3.9, and the following two propositions.

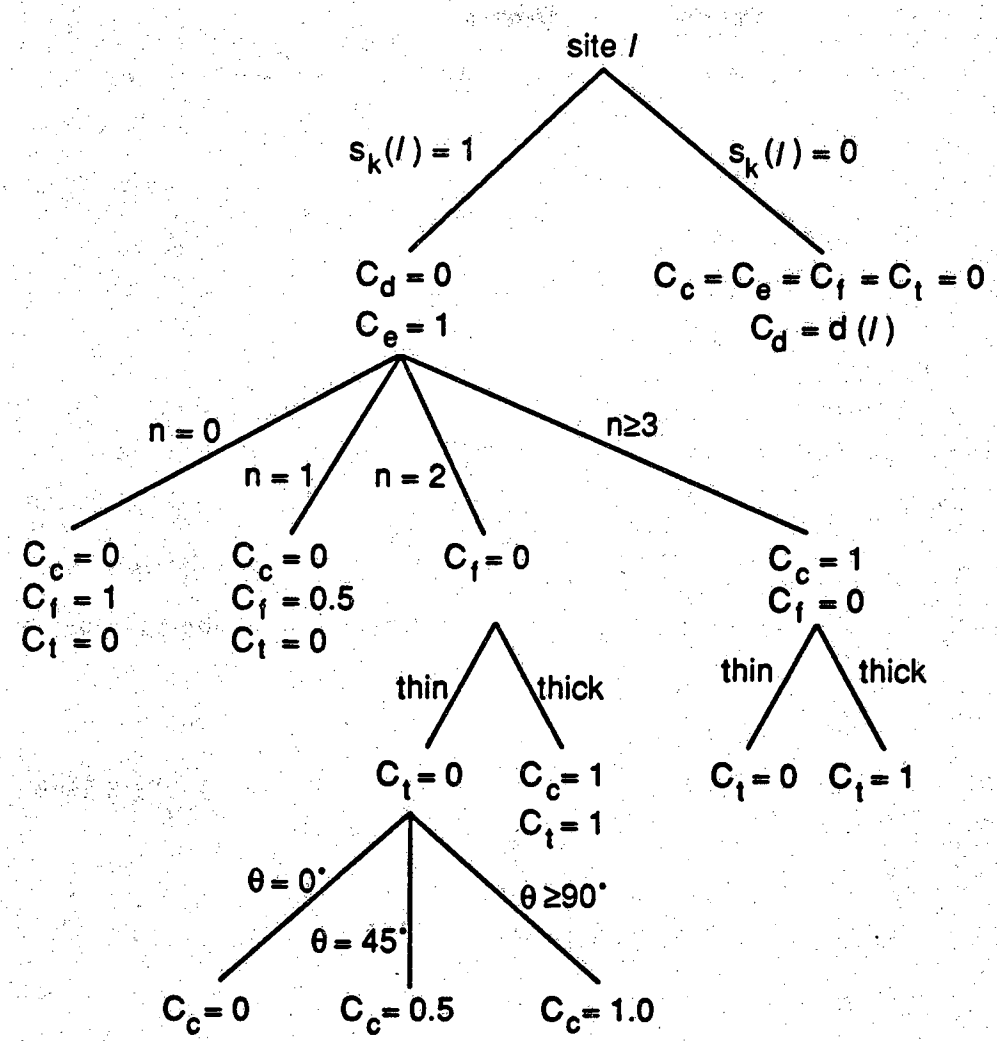
Proposition 3.12: Every thick edge pixel has a corresponding curvature greater than or equal to 90 degrees.

Proof: Every cycle of length three has a characteristic L shape. Hence at each pixel of the cycle, there is a pair of straight edge segments that form either a 90 or 135 degree angle of turn. Since a thick edge pixel belongs to a cycle of length three, it must have a curvature of at least 90 degrees.

□

Proposition 3.13: Every edge pixel with three or more neighboring edge pixels has a corresponding curvature greater than or equal to 90 degrees.

Proof: It is sufficient to show that the above is true for the case of three



$n = \| N_l(S_k) \|$

thin: The edge contained in $W_l(S_k)$ is a thin edge.

thick: The edge contained in $W_l(S_k)$ is a thick edge.

Figure 3.13. Computation of point cost $C_p(S_k,l)$ using a decision tree.

neighboring pixels. The simplest proof is by construction; each of the 56 edge structures which has 3 neighbors in a 3×3 window has a curvature of at least 90 degrees.

□

Computing the incremental cost

According to Equation (3.5), the incremental cost from S_m to S_n is

$$\Delta F_{m,n} = \sum_{l \in L} \Big[C_p(S_n, l) - C_p(S_m, l) \Big]$$

Since there are N^2 sites in L, this represents a total of $2N^2$ times the point cost has to be computed. This is a tremendous amount of computation, and even for small images of size 128×128, the value of $2N^2$ is equal to 32,768. We will show that by appropriately restricting the choice of S_n, the incremental cost can be reduced to a summation over a small subset of L, i.e.

$$\Delta F_{m,n} = \sum_{l \in R} \Big[C_p(S_n, l) - C_p(S_m, l) \Big] ,$$

where R is a small subset of L, containing only 9 sites. We essentially reduce the summation of N^2 terms to that of 9 terms. This is given in Proposition 3.14. Before stating it, we give several preliminary definitions and lemmas.

Definition 3.13: Let $\tilde{W}_l(S)$ be the set of pixels of S whose windows contain the pixel $s(l)$. That is, for l, $q \in L$,

$$\tilde{W}_l(S) = \{ s(q) : \; s(l) \in W_q(S) \}$$

Lemma 3.1: For any l, $q \in L$,

$$s(q) \in W_l(S) \text{ if and only if } s(l) \in W_q(S) \; .$$

Proof: Let $l = (i, j)$ and $q = (m,n)$. Then, from the definition of a window, the pixel $s(q)$ is a member of $W_l(S)$ implies that $|i-m| \leq 1$ and $|j-n| \leq 1$. Since i is interchangeable with m, and j is interchangeable with n within the absolute value signs, we conclude that $s(q) \in W_l(S)$ implies that $s(l) \in W_q(S)$. In the same way, a simple change of variables will show that $s(l) \in W_q(S)$ implies that $s(q) \in W_l(S)$.

□

Lemma 3.2: For any $l \in L$, $W_l(S) = \tilde{W}_l(S)$.

Proof: We will show that the following holds for any $l \in L$; $W_l(S) \subseteq \tilde{W}_l(S)$ and $\tilde{W}_l(S) \subseteq W_l(S)$. Let $s(q) \in W_l(S)$, $q \in L$. By Lemma 3.1, we have $s(l) \in W_q(S)$. From the definition of $\tilde{W}_l(S)$, we see that $s(q) \in \tilde{W}_l(S)$, which is true for every $s(q) \in W_l(S)$. Hence $W_l(S) \subseteq \tilde{W}_l(S)$)
Now, let $s(q) \in \tilde{W}_l(S)$, then from the definition of $\tilde{W}_l$, we know that $s(l) \in W_q(S)$. Again, by Lemma 3.1, we have $s(q) \in W_l(S)$, which is true for every $s(q) \in \tilde{W}_l(S)$. Hence $\tilde{W}_l(S) \subseteq W_l(S)$.

□

Definition 3.14: The *index set* of A, I(A) is the collection of the pairs of indices of the pixels in A. For example, if

$$A = \{ s(i_1,j_1), s(i_2,j_2), \ldots. s(i_m,j_m) \} ,$$

then the index set of A is given by:

$$I(A) = \{ (i, j) : s(i, j) \in A \}$$

$$= \{ (i_1,j_1), (i_2,j_2), \ldots. (i_m,j_m) \} .$$

Note that I(S) is the set of all possible indices of the edge configuration S, and is equal to L. For notational purposes, we will write $I(W_{i,j}(S))$ as $W_{i,j}$. That is, when the window is used without specifying its argument, we are referring to the indices of the pixels in the window.

Proposition 3.14: If $S_m, S_n \in S$ are edge configurations that have identical edge labelings at every pixel site, except at site $x=(y,z) \in L$, where they are complementary, then

$$\Delta F_{m,n} = \sum_{l \in W_x} \Big[C_p(S_n, l) - C_p(S_m, l) \Big] . \tag{3.8}$$

Proof: From the definition of incremental cost in Equation (3.5), we essentially have to show that

$$\sum_{l \in L} \Big[C_p(S_n, l) - C_p(S_m, l) \Big] = \sum_{l \in W_x} \Big[C_p(S_n, l) - C_p(S_m, l) \Big]$$

Let $\tilde{W}_x = I (\tilde{W}_x(S))$, and partition L into disjoint sets:

$$L = (L - \tilde{W}_x) \cup \tilde{W}_x \ .$$

We can write the incremental cost according to the definition in Equation (3.5) as

$$\Delta F_{m,n} = \sum_{l \in (L-\tilde{W}_x)} \Big[C_p(S_n, l) - C_p(S_m, l) \Big] + \sum_{l \in \tilde{W}_x} \Big[C_p(S_n, l) - C_p(S_m, l) \Big]$$

For each $l \in (L-\tilde{W}_x)$, it is easy to deduce that $x \notin W_l$. Since all the pixels of S_m are identical to those in S_n except only at site x, therefore, the corresponding pixels in the windows $W_l(S_n)$ and $W_l(S_m)$ have identical edge labelings. Using this fact and Proposition 3.11, we conclude that

$$C_p(S_n, l) = C_p(S_m, l) \ \text{ for all } l \in (L-\tilde{W}_x) \ .$$

Hence, the partial sum

$$\sum_{l \in (L-\tilde{W}_x)} \Big[C_p(S_n, l) - C_p(S_m, l) \Big] = 0 \ .$$

By Lemma 3.2,

$$\tilde{W}_x = I (\tilde{W}_x(S)) = I (W_x(S)) = W_x \ .$$

Thus, the expression for the incremental cost becomes

$$\Delta F_{m,n} = \sum_{l \in \tilde{W}_x} \Big[C_p(S_n, l) - C_p(S_m, l) \Big] = \sum_{l \in W_x} \Big[C_p(S_n, l) - C_p(S_m, l) \Big] \ .$$

□

Fact 3.4: An equivalent expression for Equation (3.8) using incremental cost factors is

$$\Delta F_{m,n} = \sum_{k=1}^{5} w_k \Delta C_k(W_x \, ; S_m, S_n) \ , \tag{3.9}$$

where $\Delta C_k(W_x \, ; S_m, S_n) = \sum_{l \in W_x} \Big[C_k(S_n, l) - C_k(S_m, l) \Big]$.

From the above proposition, we see that by restricting S_n to be an edge configuration that differs from S_m at only one site x, the incremental cost can be reduced to the form shown in Equation (3.8). This proposition in itself is not very useful because of the restriction on S_n. However, it can be used in conjunction with Proposition 3.10 to provide a very efficient method of computing the incremental cost between any pair of configurations S_m and S_n. For example, let S_m and S_n be configurations that differ at K sites. It is possible to find a sequence of configurations $\{S_0, S_1, \ldots S_K\}$ such that $S_0 = S_m$, $S_K = S_n$, and any consecutive pair of configurations differ at only one site. Then Proposition 3.10 can be used to express $\Delta F_{m,n}$ in terms of consecutive pairs of configurations, and Proposition 3.14 can be applied directly to each of these pairs. This is the indirect method of computing the incremental cost. It is particularly efficient for values of K much less than N^2. We will be using this method of computing the incremental cost in our search of low cost configurations.

3.4 Analysis of Minimum Cost Configurations

In Sections 3.1 to 3.3, we have provided the necessary mathematical preliminaries and presented a cost function for evaluating edges. This cost function is a linear sum of weighted cost factors which mathematically captures our intuitive concept of an edge. The validity of this cost function for evaluating edges is ultimately determined by its performance in detecting edges that fit our edge concept.

The cost function has been formulated with the inherent assumption that lower cost configurations are better configurations according to our concept of an edge. The best configuration is the one that achieves the global minimum of the cost function. Two important issues have to be addressed in using the cost function for edge detection. First, we need to address the issue of how to find low cost edge configurations. Second, we need to know the nature of the edges in the low cost configurations. The method of finding low cost configurations will be discussed in chapter 4. We will use a stochastic optimization technique known as Simulated Annealing to find suitably low cost configurations.

In this section, we focus on the second issue mentioned above, which is analyzing the nature of edges in low cost configurations. The goal of analysis is to determine specific properties or characteristics of the edges that will be produced. For instance, we are interested in knowing whether there are any thick edges in the low cost configurations, the minimum length of each edge,

and how well the edges are connected. Some of the results obtained from this analysis will be used in the next chapter to prove certain bounds on the depth of the cost function.

The nature of the edges in low cost configurations is necessarily related to the set of weights chosen for the cost factors. For instance, if we set the weight for thick edges to be very large, then low cost configurations will probably not contain any thick edges. However, large values for the weights will tend to cause the cost function to have many deep local minimums which are highly undesirable. Deep local minimums are potential hazardous points which will trap many algorithms in search of the global minimum. Hence, a judicious choice of weights is essential in ensuring good performance for the cost function. Our choice of weights will be based on several propositions which will be presented in this section.

We will begin by formally stating the definitions of local minimum, global minimum and neighborhood of a configuration. Based on this, we will state a proposition which gives a sufficient condition on the choice of weights to ensure that the detected edges will be thin. Also, we will analyze edges for other properties such as their minimum length and certain characteristics of the endpoints. Hypothetical examples will be given in the later part to provide a better understanding of the nature of the edges in low cost configurations.

The cost minimization procedure which will be described in the next chapter is based on Markov chains. Each state in the chain corresponds to a possible solution to the minimization problem. For this reason, an edge configuration is considered to be a state in the chain. In the following sections, we will use the terms "state" and "edge configuration" interchangeably to mean the same thing.

Again, let **S** represent the collection of all possible edge configurations on an $N \times N$ square lattice.

Definition 3.15: The *neighborhood of a state* S_m is a subset of **S** defined by a neighborhood function $\mathbf{H}(S_m)$. If $S_n \in \mathbf{H}(S_m)$, then S_n is a neighbor of S_m.

Definition 3.16: A state S_G is a *global minimum* if it has the following property:

$$\mathbf{F}(S_G) \leq \mathbf{F}(S_k) \quad \text{for all } S_k \in \mathbf{S} \ .$$

Notice that the global minimum may not be unique; there may exist a set of

different states with the same minimum cost value.

Definition 3.17: A state S_L is a *local minimum* if it has the following property:

$$\mathbf{F}(S_L) \leq \mathbf{F}(S_k) \quad \text{for all } S_k \in \mathbf{H}(S_L) \ .$$

Definition 3.18: The neighborhood function $\mathbf{H}_1(S)$ is the subset of $\mathbf{S}$ such that for each state $S_k \in \mathbf{H}_1(S)$, the edge labeling at every site is identical to that of S, except at a single site $l_k \in L$, where it is the complement. That is,

$$S_k = \left\{ \begin{array}{l} s_k(l) : \ s_k(l) = s(l) \quad \text{for all } l \neq l_k,\ l \in L \\ \qquad\quad s_k(l) = \overline{s}(l) \quad \text{for } l = l_k \end{array} \right\} ,$$

where $\overline{s}(l)$ denotes the complement of $s(l)$.

Since there are N^2 different sites that could be specified by l_k, so there are N^2 different states in $\mathbf{H}_1(S)$.

3.4.1 Formation of thin edges

An important aspect of our edge concept is that edges should be thin. By a proper choice of weights, we can ensure that all the edges in any local or global minimum state are thin. We do this essentially by placing a sufficiently large weight for the cost of thick edge pixels. The following is a sufficient condition for the formation of thin edges.

Proposition 3.15: Assume that the neighborhood function is $\mathbf{H}_1(S)$. If $w_t > (2w_f + w_d - w_e - w_c)$, then there are no thick edges in any local or global minimum state.

Proof: It is necessary and sufficient to show that there are no thick edge pixels in any minimum state. Let S_o be any state that contains a thick edge pixel. If the condition holds, we will show that we can always find a lower cost neighboring state that has at least one less thick edge pixel than S_o.

Since S_o has a thick edge, it contains a cycle of length 3; $C = s_o(l_1)s_o(l_2)s_o(l_3)s_o(l_1)$. Let S_n be the edge configuration that has identical edge labelings as S_o except only at site l_1, where it is a non-edge pixel. Notice that this state is a neighbor of S_o and has one less thick edge pixel. From Proposition 3.14 and Fact 3.4 the incremental cost can be written as

$$\Delta F_{o,n} = \sum_{k=1}^{5} w_k \Delta C_k(W_x ; S_o, S_n)$$

$$= w_c \Delta C_c + w_d \Delta C_d + w_e \Delta C_e + w_f \Delta C_f + w_t \Delta C_t .$$

By taking into consideration the various edge structures in a 5×5 region about site l_1, we obtain bounds for each of the above incremental cost factors. The incremental cost for curvature lies in the range $-7 \leq \Delta C_c \leq -1$. An example of the edge structure for each of the limiting cases is shown in Figure 3.14. The incremental cost for region dissimilarity lies in the range $0 \leq \Delta C_d \leq 1$. The incremental cost for the number of edge points, $\Delta C_e = -1$. The incremental cost for fragmentation lies in the range $0 \leq \Delta C_f \leq 2$. Two examples of the edge structure for the upper limit is shown in Figure 3.15. Similar to curvature, the incremental cost for edge thickness lies in the range $-7 \leq \Delta C_t \leq -1$. From the above equation, we have

$$\max\left[\Delta F_{o,n} \right] \leq \sum_{k=1}^{5} w_k \left(\max\left[\Delta C_k \right] \right)$$

$$= w_c(-1) + w_d(1) + w_e(-1) + w_f(2) + w_t(-1)$$

$$= 2w_f + w_d - w_c - w_e - w_t$$

Assuming the condition of the proposition holds,

$$2w_f + w_d - w_c - w_e - w_t < 0 \ ,$$

we conclude that

$$\max\left[\Delta F_{o,n} \right] < 0.$$

This implies that S_n is a lower cost state than S_o. Hence, we have shown that for any state which contains a thick edge pixel, we can always find a neighboring lower cost state by relabeling that pixel as non-edge. Therefore, in any minimum state, there cannot be any thick edge pixels.

□

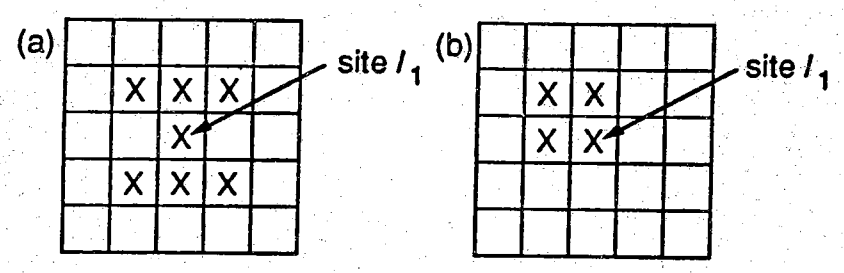


Figure 3.14. Computation of ΔC_c. (a) Removal of edge pixel at l_1 results in ΔC_c=−7. (b) Removal of edge pixel at l_1 results in ΔC_c=−1.

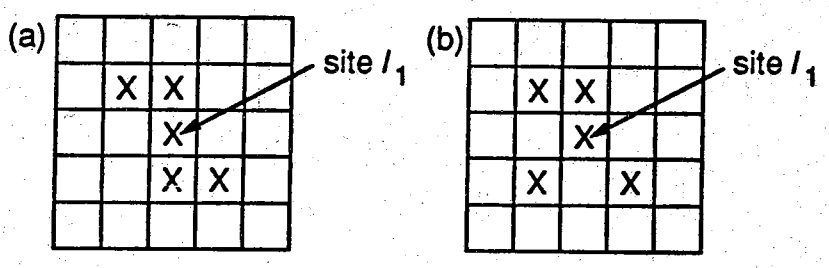


Figure 3.15. Computation of ΔC_f. Removal of edge pixel at l_1 of either (a) or (b) results in ΔC_f=2.

3.4.2 Minimum length of edges

The cost for fragmentation increases the continuity of edges by assigning a cost to the endpoints of an edge. Although it is not obvious from the definition, this cost also guarantees that the detected edges are of a certain minimum length. This length is dependent on the choice of the weights w_f, w_d and w_e. By appropriately selecting the weights, we can ensure that the detected edges are of an arbitrary minimum length.

Before stating the proposition relating the minimum length of edges to the weights, we first state two lemmas and a related proposition.

Lemma 3.3: Let $s_m(l)$ be any edge pixel of configuration S_m, and let S_n be the configuration that has identical edge labelings as S_m at every site, except at l where it is a non-edge pixel. Then, the incremental cost factor for curvature $\Delta C_c(L\ ;\ S_m,S_n)$ is always less than or equal to zero.

Proof: By using Proposition 3.14 and Fact 3.4, and setting all the weights to zero except w_c, it is straightforward to see that

$$\Delta C_c(L\ ;\ S_m,S_n) = \Delta C_c(W_l\ ;\ S_m,S_n)\ .$$

Using Fact 3.1, we can rewrite the incremental cost as

$$\Delta C_c(\ W_l\ ;\ S_m,S_n) = \Delta C_c(\ l\ ;\ S_m,S_n) + \Delta C_c(N_l\ ;\ S_m,S_n)\ .$$

It is sufficient to show that $\Delta C_c(\ l\ ;\ S_m,S_n) \leq 0$ and $\Delta C_c(N_l\ ;\ S_m,S_n) \leq 0$. From the definition of the curvature cost, it is easily deduced that the first inequality is always true. For the second inequality, we observe that for each $x \in N_l$,

$$\|\ N_x(S_n)\ \| = \|\ N_x(S_m)\ \| - 1\ .$$

Consequently, from the decision tree for computing the point cost shown in Figure 3.13, we see that for each of the cases of the number of edge pixels in the neighborhood of x,

$$C_c(S_n,x) \leq C_c(S_m,x)\ .$$

Hence, the sum

$$\sum_{x \in N_l} \Big[C_c(S_n,x) - C_c(S_m,x) \Big] \leq 0\ .$$

□

Lemma 3.4: Let $s_m(l)$ be any edge pixel of configuration S_m, and let S_n be the configuration that has identical edge labelings as S_m at every site, except at l where it is a non-edge pixel. Then, the incremental cost factor for thick edges $\Delta C_t(L; S_m,S_n)$ is always less than or equal to zero.

Proof: Let A_m be the union of the pixels in all possible cycles of length 3 in S_m, and similarly, let A_n be the union for the cycles in S_n. The cost for edge thickness assigns a cost value of one to each distinct edge pixel belonging to a cycle of length 3, and hence,

$$\sum_{l \in L} C_t(S_n, l) = \| A_n \| , \quad \text{and} \quad \sum_{l \in L} C_t(S_m, l) = \| A_m \| .$$

Since S_n is identical to S_m except that it has 1 less edge pixel, then every cycle $C = s_n(l_1)s_n(l_2)s_n(l_3)s_n(l_1)$ contained in S_n must have a corresponding cycle $C' = s_m(l_1)s_m(l_2)s_m(l_3)s_m(l_1)$ in S_m. Consequently, the size of A_n must be less than or equal to that of A_m, and so

$$\sum_{l \in L} C_t(S_n, l) - \sum_{l \in L} C_t(S_m, l) = \| A_n \| - \| A_m \| \leq 0 .$$

□

Using the above lemmas, we now state a proposition that relates how the cost for edge thickness and the cost for curvature change when edge pixels are removed.

Proposition 3.16: Let M be any collection of edge pixels in S_m, and let S_n be the edge configuration that has identical edge labelings as S_m at every site, except at the sites of the pixels in M, where they are labeled as non-edge pixels. Then, the incremental cost factors for curvature and thick edges, $\Delta C_c(L; S_m,S_n)$ and $\Delta C_t(L; S_m,S_n)$, are always less than or equal to zero. Conversely, the factors $\Delta C_c(L; S_n,S_m)$ and $\Delta C_t(L; S_n,S_m)$ are always greater than or equal to zero.

Proof: Consider any collection of edge configurations $\{ S_1, S_2, ... S_k \}$. Using Proposition 3.10 and Equation (3.7), and setting all the weights except w_c equal to zero, we see that

$$\Delta C_c(L ; S_1,S_k) = \sum_{i=1}^{k-1} \Delta C_c(L ; S_i,S_{i+1})$$

By letting $k = \| M \| + 1$, we can construct a sequence of configurations

beginning with the initial configuration $S_1 = S_m$, and ending with $S_k = S_n$, such that each consecutive configuration contains one less edge pixel of M. That is, S_{i+1} is identical to S_i except at a single site of M, where it is non-edge. By Lemma 3.3, each of the terms $\Delta C_c(L\ ;\ S_i, S_{i+1})$ is less than or equal to zero, and hence

$$\Delta C_c(L\ ;\ S_m, S_n) = \sum_{i=0}^{k-1} \Delta C_c(\ L\ ;\ S_i, S_{i+1}) \leq 0\ .$$

From Equation (3.7),

$$\Delta C_c(L\ ;\ S_m,\ S_n) = \sum_{l \in L} \Big[C_c(S_n,\ l) - C_c(S_m,\ l) \Big] \leq 0\ ,$$

it is easily concluded that

$$\Delta C_c(L\ ;\ S_n,\ S_m) = \sum_{l \in L} \Big[C_c(S_m,\ l) - C_c(S_n,\ l) \Big] \geq\ 0\ .$$

The proof for the incremental cost for thick edges follows the same procedure as for curvature, except that Lemma 3.4 is used instead of Lemma 3.3 above.

□

Intuitively, the proposition tells us that when edge pixels are removed from a configuration, the cost for curvature and the cost for thickness never increase. Conversely, when edge pixels are added, the two cost factors never decrease. This proposition is important as it gives us an intuitive idea of how the cost factors affect the edges.

We now state an important proposition which gives the minimum length of any edge in a global minimum state.

Proposition 3.17: Assume that there are no thick edges in the global minimum states. In a global minimum state, any edge E that contains at least two endpoints has size

$$\| E \| \ \geq\ \left\lceil \frac{w_f}{w_d - w_e} \right\rceil\ ,$$

where $\lceil x \rceil$ denotes the smallest integer greater than or equal to x.
If E contains less than two endpoints, then $\| E \| \geq 4$.

Proof: Let E be an edge in a global minimum state S_G, and let S_n be the state that has identical edge labelings as S_G at every site except at the sites of E

where they are labeled as non edge pixels. Using Equation (3.7), the incremental cost can be written as

$$\Delta F_{n,G} = \sum_{k=1}^{5} w_k \Delta C_k(L\ ;\ S_n,\ S_G)$$

$$= w_c\Delta C_c + w_d\Delta C_d + w_e\Delta C_e + w_f\Delta C_f + w_t\Delta C_t$$

The incremental cost factors have the following values:

$$\Delta C_t = 0\ ,\quad \Delta C_e = \|\ E\ \|\ ,\ \text{and}\ \ \Delta C_d = \sum_{l\in I(E)} -d(l)\ .$$

Hence,

$$\Delta F_{n,G} = w_c\Delta C_c - \sum_{l\in I(E)} w_d d(l) + w_e\|\ E\ \| + w_f\Delta C_f$$

$$= w_c\Delta C_c + \sum_{l\in I(E)} \Big[w_e - w_d d(l)\Big] + w_f\Delta C_f\ .$$

Since $0 \le d(l) \le 1$,

$$\Delta F_{n,G} \ge\ w_c\Delta C_c + \sum_{l\in I(E)} \Big[w_e - w_d\Big] + w_f\Delta C_f$$

$$= w_c\Delta C_c + \|\ E\ \|\ (w_e - w_d) + w_f\Delta C_f\ .$$

From the fact that S_G is a global minimum state, we have $\Delta F_{n,G} \le 0$, which implies that

$$\|\ E\ \|\ (w_d - w_e) \ge\ w_c\Delta C_c + w_f\Delta C_f\ .$$

If E contains at least 2 endpoints, $\Delta C_f \ge\ 1$. Using Proposition 3.16 we have $\Delta C_c \ge\ 0$. Therefore, taking the minimum of the factors to the right of the above inequality,

$$\|\ E\ \| \ \ge\ \ \frac{w_f}{w_d - w_e}\ .$$

The size of E is an integer, and using the ceiling notation, we have

$$\|\ E\ \| \ \ge\ \ \left\lceil \frac{w_f}{w_d - w_e} \right\rceil\ .$$

If E contains less than 2 endpoints, it must contain a cycle. Again, by Proposition 3.15, the cycle must be thin. By construction, the smallest cycle

that is thin must have at least 4 distinct edge pixels as shown in Figure 3.16. □

3.4.3 Dissimilarity values at the endpoints

Dissimilarity enhancement signifies with large values those pixels which are good candidates for edge points. The following proposition gives a lower bound on the value of region dissimilarity at the endpoints of an edge. It is useful in estimating the value of the enhancement scale factor α used in dissimilarity enhancement.

Proposition 3.18: Assume that the neighborhood function is $\mathbf{H}_1(S)$. Let E be a thin edge that is a path in a local or global minimum state, such that $\| E \| \geq 2$. Then, the dissimilarity value at each endpoint of E located at site $l \in \{l_1, l_2\}$ must satisfy

$$d(l) \geq \frac{w_e}{w_d} . \tag{3.10}$$

Proof: Let S_o be a minimum state, and let S_n be a member of $\mathbf{H}(S_o)$ such that S_o and S_n differ only at site l_1; pixel $s_n(l_1)$ is a non-edge pixel. From Proposition 3.14 and Fact 3.4 the incremental cost can be written as

$$\Delta F_{n,o} = \sum_{k=1}^{5} w_k \Delta C_k(W_x ; S_n, S_o)$$

$$= w_c \Delta C_c + w_d \Delta C_d + w_e \Delta C_e + w_f \Delta C_f + w_t \Delta C_t .$$

Since E is thin and $s_o(l_1)$ is an endpoint of a path, the incremental cost factors must take on the values:

$$\Delta C_f = 0 , \ \Delta C_t = 0 , \ \Delta C_e = 1 , \text{ and } \Delta C_d = -d(l_1) .$$

Hence,

$$\Delta F_{n,o} = w_c \Delta C_c - w_d d(l_1) + w_e .$$

Since S_o is a minimum state, $\Delta F_{n,o} \leq 0$. This implies that

$$d(l_1) \geq \frac{w_e}{w_d} + \frac{w_c}{w_d} \Delta C_c .$$

By Proposition 3.16, $\Delta C_c \geq 0$, and hence

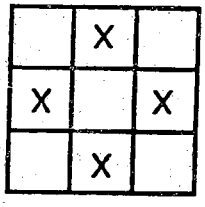


Figure 3.16. A cycle of 4 distinct edge pixels.

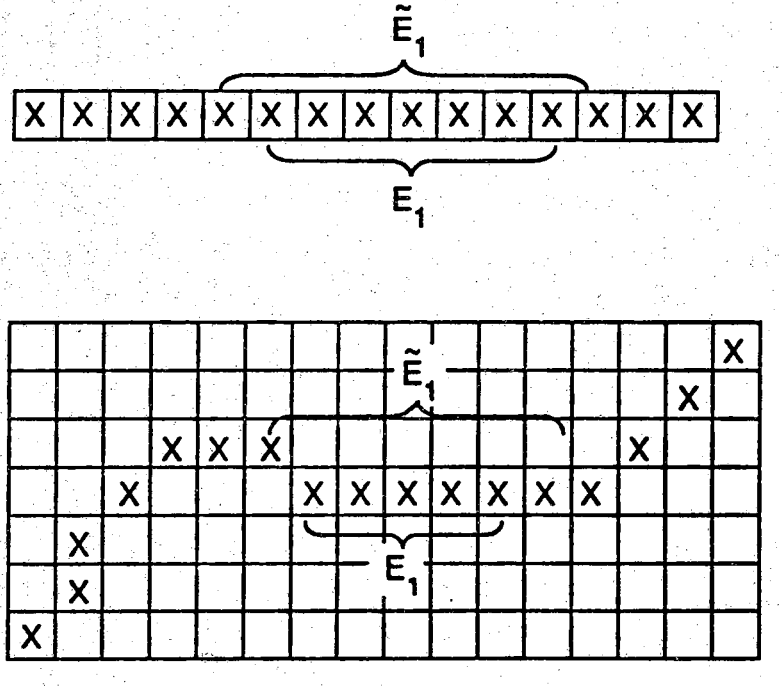


Figure 3.17. Two examples of extended edge segments.

$$d(l_1) \geq \frac{w_e}{w_d} .$$

□

Since $d(l)$ is linearly dependent on the enhancement scale factor α (up to a maximum of 1.0), the above proposition provides a guideline to selecting the value of α. Essentially, edges can be extended by increasing α so that more points satisfy Equation (3.10). The above weight ratio will be given a special term called dissimilarity threshold which will be discussed in the next section.

3.4.4 General Considerations in Selecting the Weights

Selection of weights is of major importance as it determines the nature of the edges and the number of edge points detected. In the previous sections, we have given a sufficient condition for the weights to ensure that only thin edges will be detected. Also, we have analyzed the minimum length of edges in terms of the weights, and gave a lower bound on the values of the enhanced image at the endpoints of an edge. Further insight is gained into the choice of weights by considering the minimization of the cost function from the standpoint of thresholding and edge linking. To do so, we have to first state an important property of thin edges in a global minimum state. This property is given in the following proposition. Several hypothetical edge structures will be used as examples to provide additional insight into the weight selection process.

If E is a thin edge in any state S, we can partition E into non-empty disjoint segments $E_1, E_2, \ldots E_m$ such that $E = \bigcup_{i=1}^{m} E_i$. We define the extended segments $\tilde{E}_i$ as the set of edge pixels of E contained in the union of the windows of the pixels in E_i. Two examples of extended segments are shown in Figure 3.17. Note that if $m = 1$, $E_1 = E = \tilde{E}_1$, and if $m \geq 2$, $E_i \subset \tilde{E}_i$.

Proposition 3.19: Let E be a thin edge that is a path or cycle in a global minimum state S_G. If $E_1, E_2, \ldots E_m$ are non-empty disjoint edge segments such that $E = \bigcup_{i=1}^{m} E_i$, and $\tilde{E}_i$ are the corresponding extended segments, then for each segment, the following inequality holds:

$$\| E_i \| w_e + w_c \sum_{l \in I(\tilde{E}_i)} C_c(S_G, l) - w_d \sum_{l \in I(E_i)} d(l) \leq w_f . \qquad (3.11)$$

Furthermore, if m=1, or if E_i contains an endpoint of E, then

$$|| E_i || w_e + w_c \sum_{l \in I(\tilde{E}_i)} C_c(S_G, l) - w_d \sum_{l \in I(E_i)} d(l) \leq 0 . \qquad (3.12)$$

Proof:

Case 1: $m \geq 2$.

Let S_n be the state that has identical edge labelings as S_G at every site, except at the sites of E_i, where they are labeled as non-edge pixels. The incremental cost from S_n to S_G is

$$\Delta F_{n,G} = \sum_{k=1}^{5} w_k \Delta C_k(L ; S_n, S_G)$$

$$= w_c \Delta C_c + w_d \Delta C_d + w_e \Delta C_e + w_f \Delta C_f + w_t \Delta C_t .$$

Since E is a thin edge, $\Delta C_t = 0$, and $\Delta C_e = || E_i ||$. Therefore,

$$\Delta F_{n,G} = w_c \Delta C_c - w_d \sum_{l \in I(E_i)} d(l) + w_e || E_i || + w_f \Delta C_f .$$

The state S_G is a global minimum, and so $\Delta F_{n,G} \leq 0$. This implies that

$$w_c \Delta C_c - w_d \sum_{l \in I(E_i)} d(l) + w_e || E_i || \leq -w_f \Delta C_f .$$

The incremental cost for curvature is $\Delta C_c = \sum_{l \in I(\tilde{E}_i)} C_c(S_G, l)$.

We will refer to the segments containing the endpoints of a path as the end segments. Segments that do not contain any end point are referred to as interior segments. The incremental cost factor for fragmentation takes on one of two possible values.

$$\Delta C_f = \begin{cases} 0, & \text{if E is a path and } E_i \text{ is an end segment} \\ -1, & \text{if E is a path and } E_i \text{ is an interior segment} \\ -1, & \text{if E is a cycle} \end{cases}$$

Substituting these into the above equation and taking the upper bound of the factor to the right of the inequality,

$$w_c \sum_{l \in I(\tilde{E}_i)} C_c(S_G, l) - w_d \sum_{l \in I(E_i)} d(l) + w_e || E_i || \leq w_f .$$

It is easily seen that if E_i is an end segment, then $\Delta C_f = 0$, and the inequality becomes

$$w_c \sum_{l \in I(\tilde{E}_i)} C_c(S_G, l) - w_d \sum_{l \in I(E_i)} d(l) + w_e || E_i || \leq 0 .$$

Case 2: m = 1.

We have $E_1 = E = \tilde{E}_1$. Again, let S_n be the state that has identical edge labelings as S_G except with the corresponding pixels in E_1 labeled as non-edge pixels. Following the same procedure as in case 1, we have

$$w_c \sum_{l \in I(\tilde{E}_1)} C_c(S_G, l) - w_d \sum_{l \in I(E_1)} d(l) + w_e || E_1 || \leq -w_f \Delta C_f .$$

The incremental cost factor for fragmentation takes on one of the 2 values:

$$\Delta C_f = \begin{cases} 0, & \text{if E is a cycle} \\ +1, & \text{if E is a path} \end{cases}$$

Substituting this into the above equation and taking the upper bound of the factor on the right of the inequality,

$$w_c \sum_{l \in I(\tilde{E}_i)} C_c(S_G, l) - w_d \sum_{l \in I(E_i)} d(l) + w_e || E_i || \leq 0 .$$

□

3.4.4.1 Thresholding

Thresholding is the simplest form of edge detection based on an enhanced image. It can also be considered to be a trivial form of cost minimization where the cost function does not take into account edge structure information. For the cost function which we defined, the edge structure information is contained in the cost factors for curvature, fragmentation and thickness. By setting their respective weights to zero, the cost minimization procedure becomes a simple thresholding operation. From Equation (3.12) of Proposition 3.19, any edge E must satisfy

$$w_d \sum_{l \in I(E)} d(l) \geq || E || w_e + w_c \sum_{l \in I(E)} C_c(S_G, l) . \qquad (3.13)$$

If E is a single edge pixel at l, then the following must hold:

$$d(l) \geq \frac{w_e}{w_d} .$$

We call the above ratio the *dissimilarity threshold*, and denote it by

$$\varsigma = \frac{w_e}{w_d} \quad . \tag{3.14}$$

Using a thresholding approach, in the minimum cost state, dissimilarity values that are greater than or equal to ς will be labeled as edge pixels, while those that are less than ς will be non-edge pixels. The dissimilarity values are linearly dependent on the enhancement scale factor α, up to a maximum value of one. Hence, the number of edge pixels that are detected can be adjusted by varying the value of α. It is seen that a cost function comprising only of the cost factors for dissimilarity and number of edge points, C_d and C_e, represents the general class of edge detection by pointwise thresholding algorithms. However, for reasons that we have already mentioned in Section 3.3.1, pointwise thresholding algorithms do not perform well in finding edges that suit our edge concept.

Consider a hypothetical minimum cost configuration S_1 containing a single thin edge E as shown in Figure 3.18. Let S_2 be the edge configuration that contains no edge pixels. If the cost function uses only the cost factors C_d and C_e, then by considering $\Delta F_{1,2}$, S_1 is a lower cost state if and only if the following inequality holds:

$$w_d \sum_{l \in I(E)} d(l) \geq \| E \| w_e \quad . \tag{3.15}$$

However, when the cost factors for fragmentation and curvature, C_f and C_c, are included, the inequality becomes:

$$w_d \sum_{l \in I(E)} d(l) \geq \| E \| w_e + w_f + w_c \quad . \tag{3.16}$$

A comparison of Equation (3.15) with Equation (3.16) shows that the sum of the dissimilarity values for the latter equation has to be larger than that of the former. In this case, when the cost factors C_f and C_c are included into the cost function, the lower bound of the sum of the dissimilarity values is increased by $w_f + w_c$. Consequently, the edges that are detected for the case using Equation (3.15) may not be detected for the case using Equation (3.16). This is an example where we observe the influence of C_f and C_c in suppressing short edges.

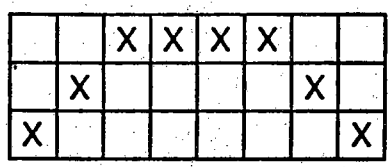


Figure 3.18. A minimum cost configuration containing a single thin edge.

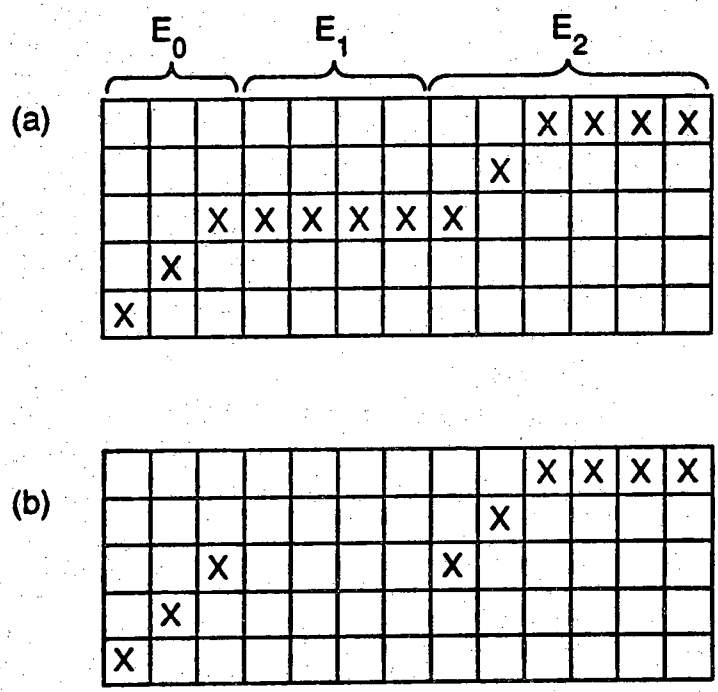


Figure 3.19. An example of edge linking. (a) Configuration S_1 which contains an edge E comprised of the segments E_0, E_1, and E_2. (b) Configuration S_2 which contains fragmented version of edge E by the removal of segment E_1.

3.4.4.2 Edge linking

We will now consider how the cost for fragmentation promotes edge linking. Consider a minimum cost configuration S_1 that contains a single thin edge E as shown in Figure 3.19(a). Let E be partitioned into 3 disjoint segments E_0, E_1 and E_2. Now let S_2 be the configuration that contains E but with the edge pixels of the center segment E_1 relabeled as non-edge. This is shown in Figure 3.19(b). Notice that the edges in S_2 is actually a fragmented version of edge E. The size of the fragmentation, or the *fragmentation length*, is equal to $\| E_1 \|$. By considering $\Delta F_{1,2}$, S_1 is a lower cost state if and only if the following relationship holds:

$$w_d \sum_{l \in I(E_1)} d(l) + w_f \geq \| E_1 \| w_e + w_c \ . \tag{3.17}$$

Notice in this case that if

$$w_f \geq \| E_1 \| w_e + w_c \ ,$$

then S_1 will have a lower cost than S_2 regardless of $d(l)$; the continuous edge E has a lower cost than the fragmented version of the edge. Assuming that the weight for curvature w_c is comparatively small, we can approximate the above inequality by

$$w_f \geq \| E_1 \| w_e \ . \tag{3.18}$$

If we let

$$\kappa = \left\lfloor \frac{w_f}{w_e} \right\rfloor \ , \tag{3.19}$$

where $\lfloor \ \rfloor$ is the floor function, we see that thin edges with a fragmentation length of less than or equal to κ pixels will have a lower cost when they are linked together. Stated in another way, endpoints which are less than or equal to κ pixels apart have a lower cost when they are linked together. For this reason, we will call κ the *fragmentation linkage length*.

In arriving at the value of κ given in Equation (3.19), we have not taken into consideration the dissimilarity values; they were assumed to be zero. When these values are taken into account, edges that have fragmentation lengths larger than κ can also be linked together. This is illustrated in the following two examples.

In the first example shown in Figure 3.20, we show a straight edge and the dissimilarity values along the edge. Assume that the dissimilarity values are

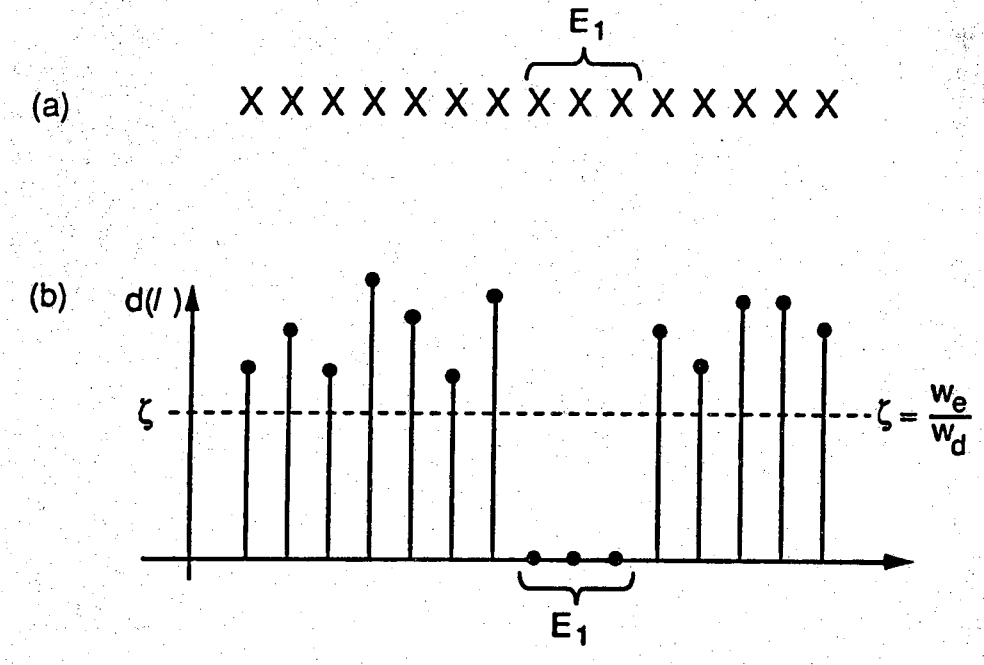


Figure 3.20. An example of edge linking across a region where the dissimilarity values are equal to 0. (a) A straight edge. (b) The dissimilarity values along the edge.

zero at all other points. Notice that the edge contains a segment E_1 where all the sites have dissimilarity values equal to 0. If the cost for fragmentation is not included, only those points with dissimilarity greater than the dissimilarity threshold ς will be detected. This will result in a fragmented edge as the center portion E_1 has dissimilarity values equal to 0. However, with the inclusion of the fragmentation cost, fragmentation will occur if and only if the length of 0s in the dissimilarity values exceeds the fragmentation length κ. That is, $||E_1|| > \kappa$.

In the second example shown in Figure 3.21, we show another straight edge and the dissimilarity values along the edge. Here again, fragmentation will occur in the center region if the fragmentation cost is not included. In this case however, since the dissimilarity values at the sites of E_1 are non-zero, fragmentation may not occur even if the size of E_1 exceeds κ. In fact, fragmentation will occur if and only if the following relation based on Equation (3.17) holds:

$$w_d \sum_{l \in I(E_1)} d(l) + w_f < || E_1 || w_e \ .$$

If we let

$$\tilde{\kappa} = \left\lfloor \frac{1}{w_e} \left(w_f + w_d \sum_{l \in I(E_1)} d(l) \right) \right\rfloor \ , \tag{3.20}$$

then fragmentation will occur if and only if

$$||E_1|| > \tilde{\kappa} \ .$$

Using the property

$$\lfloor A + B \rfloor \geq \lfloor A \rfloor + \lfloor B \rfloor \ ,$$

it can be deduced that

$$\tilde{\kappa} \geq \kappa + \left\lfloor \frac{w_d}{w_e} \sum_{l \in I(E_1)} d(l) \right\rfloor \ . \tag{3.21}$$

Note that $\tilde{\kappa}$ is always greater than or equal to κ. Hence edges with fragmentation length greater than κ can be linked together. However, this value of $\tilde{\kappa}$ only holds for straight edges. For general paths or other thin edge structures, Equation (3.11) will have to be used to account for curvature costs. From the above two examples, we can view the effect of the fragmentation cost from another standpoint; for some regions that have dissimilarity values below

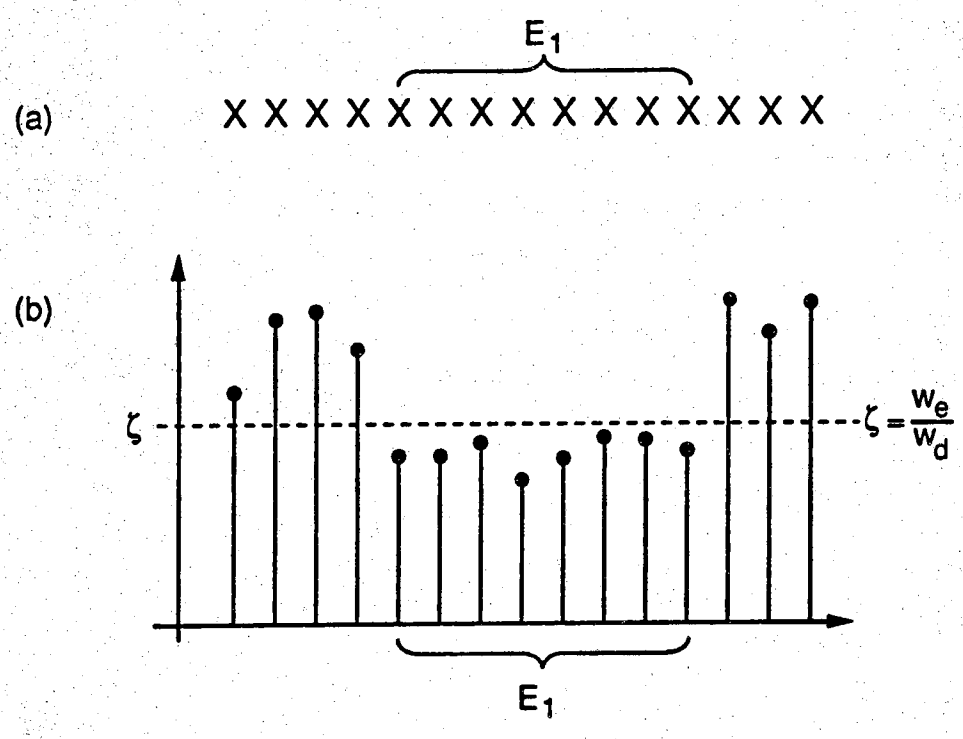


Figure 3.21. An example of edge linking across a region where the dissimilarity values are non-zero. (a) A straight edge. (b) The dissimilarity values along the edge.

the threshold ς, the fragmentation cost lifts them above the threshold.

3.5 Summary

In this chapter, we have given a mathematical definition of edges using terms that are similar to those in graph theory. Based on this definition, we accomplished two things. First, we analyzed certain properties of edges and stated several propositions governing the structure of edges in a 3×3 lattice. Second, we formulated an absolute cost function that measures edge quality. As the term suggests, this cost function is different from the comparative cost function of the previous chapter in that it measures the absolute quality of an edge configuration instead of the relative quality between configurations.

The absolute cost function is a linear sum of five weighted cost factors. The cost factors are curvature, dissimilarity, fragmentation, thickness and the number of edge points. Each of the cost factors captures a desirable characteristic of edges. We have provided efficient methods of computing both the cost of a configuration, and the incremental cost between configurations. We have analyzed the cost function in terms of the nature of the edges that will be detected. Based on this analysis, we have stated a number of propositions which provide guidelines on the choice of weights to achieve certain desirable characteristics in the detected edges.

CHAPTER 4
SIMULATED ANNEALING

4.1 Introduction

The general problem of combinatorial optimization which we are concerned with can be briefly stated as follows. Given a large but finite set of states **S**, each state $S_i \in \mathbf{S}$ having an associated cost defined by the cost function $C(S_i)$, it is required to find the state with the minimum cost. Depending on the specific nature of the problem, a variety of techniques [30] exist for minimizing the cost. In this chapter, we focus on the use of Simulated Annealing as a method of combinatorial optimization. In particular, we will show how it can be applied to edge detection by minimizing the cost function for edges which has been described in the previous chapter.

Simulated annealing is a stochastic optimization algorithm derived from Monte Carlo methods [31] in statistical mechanics. Metroplois et al. [32] originally proposed the algorithm as a simulation method for investigating the behavior of substances consisting of interacting molecules. One of its many later applications is in the study of properties of magnetic materials based on the Ising model [33-35]. The Metropolis algorithm has been used extensively to simulate the behavior of substances in thermal equilibrium as the temperature was slowly decreased to the point of crystallization; hence the term "Simulated Annealing". The goal of the annealing process is to find the ground states of a substance which corresponds to the configurations of low energy in its molecular structure.

Kirkpatrick et al. [36] and Cerny [37] independently observed that the search for the low energy configurations in the annealing process could be likened to the search for the low cost solutions in a combinatorial optimization problem. The many different states that a system can exist in corresponds to the many possible solutions of the optimization problem. The energy of a particular state corresponds to the cost of a particular solution, and the ground state corresponds to the lowest cost solution. A direct correspondence between statistical mechanics and combinatorial optimization was thus drawn in the

following way.

Statistical Mechanics	**Combinatorial Optimization**
1) States (of system)	Solutions (to problem)
2) Energy (of state)	Cost (of solution)
3) Ground State	Optimal Solution

Based on the above correspondence, the annealing algorithm of Metropolis et al. was first applied to the optimization of wire routing in integrated circuits [36], and the traveling salesman problem [37]. Good results comparable to present heuristic algorithms were reported. Since then, the algorithm has been successfully applied to a number of diverse optimization problems such as the traveling salesman problem[38], wire routing[39], coding[40], speech recognition[41], image processing[42, 43], and logic optimization[44].

4.1.1 Markov Chains

The Metropolis algorithm is based on stationary Markov chains. The definition and theory of such chains can be found extensively in the literature on stochastic processes, such as [45-48]. We will give a brief description of Markov chains concentrating only on those aspects that are relevant to Simulated Annealing. We will state several properties and a theorem concerning the limiting behavior of such chains.

Let Ω be a sample space and P be a probability measure on it. Let $X = \{X_n; n \in N\}$ be a stochastic process with a countable state space **E**. That is, for each $n \in N = \{0,1,...\}$ and $\omega \in \Omega$, $X_n(\omega)$ is an element of the countable set **E**. We will say that "the process is in state j at time n" to mean $X_n = j$.

Definition 4.1: The stochastic process $X = \{X_n; n \in N\}$ is called a Markov chain provided that

$$P\{X_{n+1}=j/X_0, \ldots, X_n\} = P\{X_{n+1}=j/X_n\}$$

for all $j \in \mathbf{E}$ and $n \in N$.

A Markov chain is thus a sequence of random variables such that for any n, X_{n+1} is conditionally independent of $X_0, \ldots, X_{n-1}$ given X_n. That is, the next state X_{n+1} is independent of the past states $X_0, \ldots, X_{n-1}$ provided that the present state X_n is known. If the conditional probability

$$P\{X_{n+1} = j/X_n = i\} = P(i, j)$$

is independent of n, the process is a time-homogeneous or stationary Markov

chain; otherwise, if it is dependent on n, it is a time-inhomogeneous or nonstationary chain. The probabilities P(i, j) are called the transition probabilities. They can be arranged in a square array resulting in a transition matrix of the form:

$$P = \begin{bmatrix} P(0,0) & P(0,1) & P(0,2) & .... \\ P(1,0) & P(1,1) & P(1,2) & .... \\ P(2,0) & P(2,1) & P(2,2) & .... \\ . & . & . & .... \\ . & . & . & .... \\ . & . & . & .... \end{bmatrix} .$$

If P(i, j) is not equal to zero, we say that state j is reachable from state i. A set of states is closed if no state outside of it can be reached from any state in it. A Markov chain is irreducible if its only closed set is the set of all states. A criterion for irreducibility is that a Markov chain is irreducible if and only if all states can be reached from each other. The proof for this can be found in [45].

A state j is said to be recurrent if and only if starting at j, the probability of returning to j is one. Beginning from a recurrent state j at time n=0, let n=R be the time of the first return to state j. Assuming that $\delta \geq 2$ is the largest integer for which the probability that R is some integer multiple of δ is equal to one, then state j is said to be periodic with period δ. If no such δ exists, then j is aperiodic. It can be shown [45] that for irreducible chains, either all states are aperiodic, or all states are periodic with the same period δ. It follows from this that if the chain is irreducible and if there exists some state i for which $P(i, i) \neq 0$, then the chain is aperiodic.

Limiting distribution

We now state a well know property of Markov chains relating to the limiting distribution of a chain.

Theorem 4.1: If X is an irreducible aperiodic Markov chain with finitely many states, then the system of linear equations

$$\pi(j) = \sum_{i \in E} \pi(i) P(i, j), \quad j \in E , \tag{4.1}$$

$$\sum_{j \in E} \pi(j) = 1 \tag{4.2}$$

has a unique solution that is strictly positive.

Proof: Refer to the text by Cinlar [45].

The probability distribution π which satisfies Equations (4.1) and (4.2) is called the invariant distribution of the Markov chain X. For simplicity in notation, we will write $\pi(j)$ simply as π_j.

4.1.2 The Metropolis Algorithm

The Metropolis algorithm, first proposed in 1953, was a method of simulating the behavior of substances at thermal equilibrium. The description and analysis of the algorithm given here follows closely to that given by Hammersley and Handscomb [31]. Let $\mathbf{S} = \{S_1, S_2, \ldots S_K\}$ be the finite set of all possible states of a physical system. Each state S_i has a corresponding positive energy denoted by $E(S_i)$. In statistical mechanics, it is often desirable to simulate the behavior of the system at thermal equilibrium at temperature T. To do so, it is necessary to be able to sample the states with the following probability density:

$$P(S_i) = \frac{\exp\left(\frac{-E(S_i)}{\alpha T}\right)}{Z_T}, \quad 1 \leq i \leq K, \tag{4.3}$$

where α is a positive scalar constant, and

$$Z_T = \sum_{S_i \in S} \exp\left(\frac{-E(S_i)}{\alpha T}\right)$$

is a normalization factor which ensures that the sum of $P(S_i)$ over all possible states is equal to one.

The denominator Z_T of Equation (4.3) is unknown and cannot be computed because the number of states, although finite, is very large. Hence, although $E(S_i)$ is known, the probability $P(S_i)$ of Equation (4.3) cannot be determined. As a result, it is not possible to generate the states according to the given distribution using direct sampling methods. The Metropolis algorithm achieves the above sampling requirement by constructing a finite stationary Markov chain that has an invariant distribution which is identical to that of Equation (4.3). That is, the chain has the set of all possible states of

the system as its state space, and its invariant distribution is given by:

$$\pi(j) = P(S_j) . \tag{4.4}$$

The method of generating the Markov chain is as follows. Consider any arbitrary chain with a symmetric matrix P^* of transition probabilities; the elements of this matrix must satisfy

$$p^*_{ij} \geq 0 , \quad \sum_j p^*_{ij} = 1 , \quad p^*_{ij} = p^*_{ji} . \tag{4.5}$$

We now define a new set of transition probabilities p_{ij} using the known quantities $\frac{\pi(i)}{\pi(j)}$.

If $i \neq j$ we define

$$p_{ij} = \begin{cases} p^*_{ij} \, \pi_j/\pi_i & \text{if } \pi_j/\pi_i < 1 \\ p^*_{ij} & \text{if } \pi_j/\pi_i \geq 1. \end{cases} \tag{4.6}$$

If $i = j$ we define

$$p_{ii} = p^*_{ii} + \sum_j{}' p^*_{ij}(1 - \pi_j/\pi_i) , \tag{4.7}$$

where $\sum'$ is taken over all values of j such that $\pi_j/\pi_i < 1$.

We will next prove that these p_{ij} are elements of a stochastic matrix and that the Markov chain defined by these transition probabilities has π as its invariant distribution.

Proof:
We will denote as $\sum''$ the summation over all values of j such that $j \neq i$ and $\pi_j/\pi_i \geq 1$.

(1) From Equation (4.3), each $\pi_j \geq 0$, and hence by Equations (4.5), (4.6) and (4.7), all the p_{ij} satisfy

$$p_{ij} \geq 0 . \tag{4.8}$$

(2) The summation over all j of the quantities p_{ij} can be written as

$$\sum_j p_{ij} = p^*_{ii} + \sum_j{}' p^*_{ij}(1-\pi_j/\pi_i) + \sum_j{}' p^*_{ij}\pi_j/\pi_i + \sum_j{}'' p^*_{ij}$$

$$= p^*_{ii} + \sum_j{}' p^*_{ij} + \sum_j{}'' p^*_{ij}$$

$$= p^*_{ii} + \sum_{j\neq i} p^*_{ij}$$

$$= \sum_j p^*_{ij} = 1 \,. \tag{4.9}$$

Thus by Equations (4.8) and (4.9), the p_{ij} are elements of a stochastic matrix.

(3) To show that the chain has an invariant distribution equal to π, we essentially have to show according to Equation (4.1) that

$$\pi_j = \sum_i \pi_i p_{ij} \,.$$

First we observe that for any pair i and j such that $\pi_i = \pi_j$, we have by Equations (4.5) and (4.6)

$$p_{ij} = p^*_{ij} = p^*_{ji} = p_{ji} \,,$$

and therefore, since $\pi_i = \pi_j$,

$$\pi_i p_{ij} = \pi_j p_{ji} \,. \tag{4.10}$$

Next, if $\pi_j < \pi_i$, we have

$$p_{ij} = p^*_{ij}\pi_j/\pi_i = p^*_{ji}\pi_j/\pi_i = p_{ji}\pi_j/\pi_i \,,$$

which again gives Equation (4.10). Similarly, it can be shown that the same equation still holds for $\pi_i < \pi_j$. Consequently, it holds for all values of i and j. Finally, we see that

$$\sum_i \pi_i p_{ij} = \sum_i \pi_j p_{ji} = \pi_j \sum_i p_{ji} = \pi_j \,,$$

which completes the proof that the invariant distribution is π.

□

Notice that by Equations (4.3) and (4.4), and the assumption that the energy is finite, $\pi_j > 0$ for all j. Consequently the matrix $P = [p_{ij}]$ represents an irreducible aperiodic Markov chain whenever $P^* = [p^*_{ij}]$ does. Hence by Theorem 4.1, a unique solution exists and is of the form defined in Equation (4.3).

We now consider the implementation of Equations (4.6) and (4.7), which essentially is the Metropolis algorithm. The algorithm begins by defining an arbitrary transition matrix P over the state space. The restriction on P is that it must be symmetric, aperiodic and irreducible, which are the assumptions made in Equation (4.5). For a fixed temperature T, the algorithm proceeds as follows.

Algorithm (Metropolis):

(1) Pick a random initial state, and set k=0.

(2) Call this the present state S_p.

(3) Based on transition matrix P, randomly select another state S_n.

(4) If ($E(S_n) < E(S_p)$) then
transition to state S_n
else
transition to state S_n with probability $\exp\left(\frac{-[E(S_n)-E(S_p)]}{\alpha T}\right)$.

(5) Increment k, then go to step (2).

The algorithm loops from steps (2) through (5), and as the number of repetitions k becomes very large, the process approaches it invariant distribution. We have shown that the invariant distribution takes on the form given in Equation (4.3). Similar results relating to the invariant distribution are given in [49-51].

By making a simple substitution of the energy of a state $E(S_i)$ with the cost $C(S_i)$ of a solution, the above algorithm generates a Markov chain with an invariant distribution of the same form as that of Equation (4.3):

$$\pi(S_i) = \frac{\exp\left(\frac{-C(S_i)}{\alpha T}\right)}{Z_T}, \quad 1 \leq i \leq K, \tag{4.11}$$

where K is the total number of all possible solutions, and α is a positive scalar constant. Z_T is again a normalization constant. Assuming that $C(S_i)$ is non-negative for all i, Equation (4.11) specifies that lower cost solutions will have higher probability of occurrence. Notice that if T is small, the distribution will be concentrated about the low cost solutions. That is, when the Markov chain achieves its invariant distribution at low temperatures, there is a high

probability that it is in a state corresponding to a low cost solution. As T tends to zero, the distribution will be concentrated at the minimum cost states. Hence the algorithm can be used in combinatorial optimization to find the minimum cost solutions.

It is of interest to note that the Metropolis algorithm presented above allows for uphill state transitions so that it does not get stuck in a local minimum of the cost function. The temperature T can be interpreted as a control parameter, and if it is set equal to zero, the algorithm is similar to the steepest descent search method and will usually terminate in a local minimum.

4.2 Temperature Variation and Simulated Annealing

In using the Metroplois algorithm for optimization, two related issues have to be resolved. The first is in estimating the number of repetitions or transitions sufficient for the Markov chain to reach its invariant distribution. The second is in the devise of a sequence of temperature decrements to bring the system to the states of minimal cost. This is known as the temperature schedule. The schedule has to be efficient in the sense that it ensures that the lowest cost states are reached rapidly.

In order to ensure convergence to the global minimum states, temperature variation has been incorporated into the Metropolis algorithm by changing the temperature parameter T in step (4) so that it becomes time dependent. The process generated by such such an algorithm is a non-stationary Markov chain. A number of researchers [52-58] have proved the asymptotic convergence properties of the chain and estimated various rates of convergence. The general form of the Simulated Annealing algorithm for cost minimization is as follows.

Algorithm (Simulated Annealing):

(1) Pick a random initial state, and set $k=0$.

(2) Call this the present state S_p.

(3) Based on a transition matrix, randomly select another state S_n.

(4) If ($C(S_n) < C(S_p)$) then
transition to state S_n
else
transition to state S_n with probability $\exp\left(\frac{-[C(S_n)-C(S_p)]}{T_k}\right)$.

(5) Increment k, then go to step (2).

In devising a temperature schedule, we will focus our attention on the work of Hajek [58] where he states a theorem which gives a necessary and sufficient condition on the temperature schedule for the convergence of the annealing algorithm to the set of global minimum states. We will also present some related results of Geman and Geman [52], and Mitra et al. [54] in this area.

Before stating Hajek's theorem, some preliminary definitions are in order. The problem is to minimize a function C defined on some finite set **S**. The set of states in **S** at which C attains the minimum is denoted by S^*. Assume that for each state S_i in **S** there is a neighborhood set $\mathbf{H}(S_i)$ contained in **S**. In addition, there is a transition probability matrix R over **S** such that $R(S_i,S_j) > 0$ if and only if S_j is in $\mathbf{H}(S_i)$. A state i is reachable from state j if there is a sequence of states $j=i_0, i_1, \ldots, i_p=i$ such that $R(i_k, i_{k+1}) > 0$ for $0 \leq k < p$. **(S, H)** is irreducible when for any pair of states i and j, i is reachable from j.

A state i is reachable at height E from state j if there is a sequence of states $j=i_0, i_1, \ldots, i_p=i$ such that

$$R(i_k, i_{k+1}) > 0 \text{ for } 0 \leq k < p$$

and

$$C(i_k) \leq E \text{ for } 0 \leq k < p \ .$$

Property 4.1 (Weak reversibility): For any real number E and any two states i and j, i is reachable at height E from j if and only if j is reachable at height E from i.

A state i is said to be a local minimum if no state j with $C(j) < C(i)$ is reachable from i at height C(i). The depth of a local minimum i is plus infinity if i is a global minimum. Otherwise, the depth of i is the smallest number E, $E > 0$, such that some state j with $C(j) < C(i)$ can be reached from i at height $C(i) + E$.

Let the temperature schedule $T_1, T_2, \ldots$ be a sequence of strictly positively numbers such that

$$T_1 \geq T_2 \geq \ldots \tag{4.12}$$

and

$$\lim_{k \to \infty} T_k = 0 \quad . \tag{4.13}$$

Suppose that a discrete time non-stationary Markov chain $X_0, X_1, \ldots$ on the state space **S** is generated using the Simulated Annealing algorithm described above. The convergence in probability of the chain to the set of globally minimum cost states is given by the following theorem.

Theorem (Hajek): Assume that the Simulated Annealing based on (**S**, **H**, C) is irreducible and satisfies weak reversibility, and that the temperature schedule satisfies Equations (4.12) and (4.13). Then

$$\lim_{k \to \infty} P[X_k \in S^*] = 1 \tag{4.14}$$

if and only if

$$\sum_{k=1}^{\infty} \exp\left(\frac{-d^*}{T_k}\right) = +\infty \quad , \tag{4.15}$$

where d^* is the maximum of the depths of all states which are local but not global minima.

Proof: Refer to the paper by Hajek [58].

Remark: If T_k takes on the parametric form

$$T_k = \frac{c}{\log(k+1)} \quad , \tag{4.16}$$

then Equation (4.15) and hence Equation (4.14) holds if and only if $c \geq d^*$.

Mitra et al. [54] showed that convergence can be achieved by a temperature schedule of the form

$$T_k = \frac{\gamma}{\log(k + k_0 + 1)} \quad , \tag{4.17}$$

where k_0 is any parameter satisfying $0 \leq k_0 \leq \infty$, and

$$\gamma \geq r\rho \, ,$$

where r and ρ are defined below. Let S' be the set of all local minima, then the radius r is defined as

$$r = \min_{i \in (S-S')} \max_{j \in S} d(i, j) , \tag{4.18}$$

where d(i, j) is the minimum number of transitions from i to j. The parameter ρ is the maximum change in cost across any transition, and is defined by

$$\rho = \max_{i \in S} \max_{j \in H(i)} | C(j) - C(i) | . \tag{4.19}$$

Geman and Geman [52] applied the annealing algorithm to Markov random fields and image restoration, and proved its convergence based on a temperature schedule of the form

$$T_k = \frac{M\Delta}{\log(1 + k)} .$$

The parameter Δ corresponds to the maximum difference in the cost for any pair of states in S, and M is the number of pixels in the image. However, the above schedule is not useful because the number of iterations k required to reach a typical temperature of T_k=0.5 is far too large for any practical implementation. For example, if M=20,000 and Δ=1, it would take $k = \exp(40{,}000)$ iterations to reach a temperature of 0.5. In their implementation, Geman and Geman concluded that the bound $M\Delta$ is far from optimal and used an empirical value of 3.0 in place of the value $M\Delta$.

To summarize, three things are required in the use of Simulated Annealing for general problem solving:

(1) a cost function defined over the state space of all possible solutions,

(2) a method of generating next states (i.e. a suitable transition matrix), and

(3) an efficient temperature schedule.

4.3 Edge Detection Using Simulated Annealing

In Chapter 3 we have presented a cost function that evaluates the quality of an edge configuration. This function mathematically captures the intuitive ideas of an edge and serves as an objective measure of how well an edge configuration fits a given image. It has been shown that when this function is minimized, a number of desirable characteristics of good edges are achieved. The goal is to find the configurations that achieve the global minimum of the cost function. Since there are 2^{N^2} possible edge configurations, it is not possible to implement any exhaustive search approach because of the large number of configurations to be considered.

We will use Simulated Annealing as a tool to find relatively low cost solutions to the cost function. Although asymptotic convergence to the global minima is guaranteed with the use of a suitable temperature schedule, the finite time behavior of the annealing algorithm will often yield solutions that are not global minima. However, they are local minimum of relatively low cost. In the context of edge detection, we find that it is not necessary to achieve the global minimum states; very satisfactory results are obtained from these relatively low cost solutions. This is particularly evident from the fact that many of the desirable characteristics of edges are achieved in low cost locally minimum states which may not correspond to global minima.

Since we have already formulated a suitable cost function for the annealing process, we proceed to discuss the remaining two requirements of Simulated Annealing mentioned in the previous section. These are the method of generating next states and the temperature schedule.

4.3.1 Method of Generating Next States

The state space of the annealing process is **S** which is the set of all possible edge configurations on an $N \times N$ square lattice. For each configuration $S \in \mathbf{S}$, the cost of the configuration is given by $\mathbf{F}(S)$. Each configuration corresponds to a state in the Markov chain generated by the annealing algorithm; the terms "configuration" and "state" will be used interchangeably to mean the same thing. At any state S_m, the potential next state S_n is generated according to a transition matrix. Conceptually, the next state is selected according to the probability distribution defined by the matrix. Practically however, it is unnecessary to explicitly define a transition matrix for the selection of potential new states; all that is needed is a method of generating next states such that certain conditions on irreducibility and reversibility are satisfied.

Our method of generating the next state is based on a combination of five possible strategies. The first strategy generates the next state by complementing a single pixel labeling in the present state. The second strategy complements two pixel labelings in the present state. The third and fourth strategies generate next states by shifting or perturbating the location of the edges in the present state. The fifth strategy involves changing an arbitrary number of pixel labelings in a window region. We now give the details of each of these strategies and the method of combining them together. Again, we let L denote the set of all pairs of indices of the pixels in a configuration. In each

case, we assume that $l \in L$ is a given parameter; the method of selecting l will be discussed later.

Strategy 1: Single pixel change

$$S_n = M_1(S_p, l) .$$

In this strategy we generate the next state S_n by complementing the edge labeling at l of the present state S_p. That is, for every pixel $s_n(x) \in S_n$, and $s_p(x) \in S_p$ such that $x \neq l$,

$$s_n(x) = s_p(x) ,$$

and for $x = l$,

$$s_n(l) = \bar{s}_p(l) ,$$

where the bar notation denotes the usual binary complementation.

Strategy 2: Double pixel change

$$S_n = M_2(S_p, l) .$$

This is the same as the strategy M_1 except that we change the labeling of two pixels in the window $W_l(S_p)$. We first randomly select a neighboring pixel of l, $s_p(r) \in N_l(S_p)$. Then the new configuration is the state that is specified by

$$s_n(x) = \begin{cases} s_p(x) , & x \in L, \ x \notin \{l, r\} \\ \bar{s}_p(x) , & x \in \{l, r\} . \end{cases}$$

Strategy 3: Single pixel shift

$$S_n = M_3(S_p, l) .$$

This strategy of generating a new state is based on locally perturbating the edge structure in the window $W_l(S_p)$. The next state S_n has identical edge labeling as S_p at every site except for the pixels in $W_l(S_p)$. The pixels in $W_l(S_n)$ are labeled according to the transformation of the edge structure in $W_l(S_p)$ shown in Figure 4.1. If the edge structure in $W_l(S_p)$ is one of the fourteen edge structures shown in the figure, the edge structure in $W_l(S_n)$ is the corresponding structure shown on the right. Where there are two structures possible for the transformation on the right, either of them are selected on an equally likely basis. If the edge structure in $W_l(S_p)$ does not correspond to one

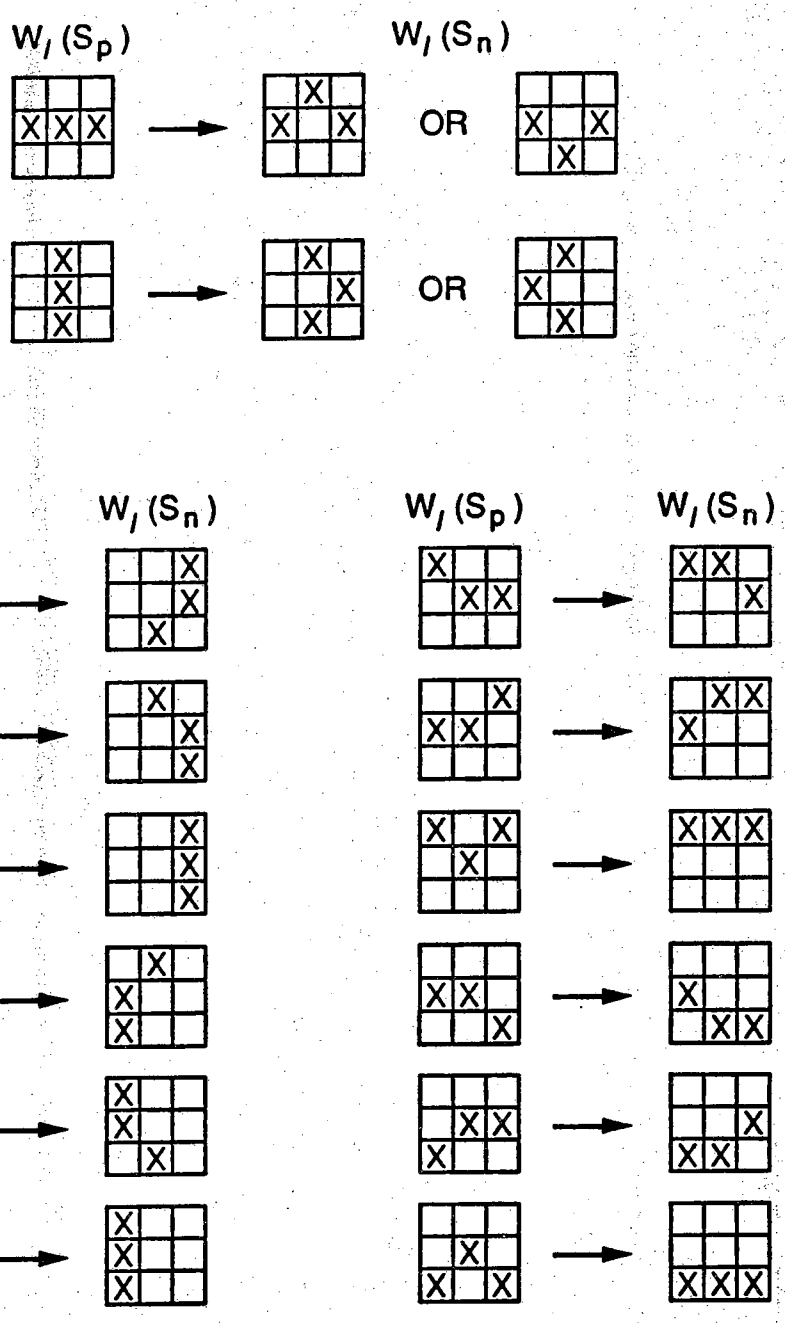


Figure 4.1. The fourteen edge structures in $W_l(S_p)$ and their corresponding transformations in $W_l(S_n)$ using strategy M_3.

of the structures shown in the figure, then the structure in $W_l(S_n)$ is made identical to that in $W_l(S_p)$. In doing so, we are actually setting $S_n = S_p$.

Strategy 4: Multiple pixel shift

$$S_n = M_4(S_p, l) .$$

This strategy of generating a new state is again based on locally perturbating the edge structure in the window $W_l(S_p)$. It is very similar to the strategy of M_3 except that the perturbation is more significant. The next state S_n has identical edge labeling as S_p at every site except for the pixels in $W_l(S_p)$. The pixels of $W_l(S_n)$ are labeled according to the transformation of the edge structure in $W_l(S_p)$ shown in Figure 4.2. If the edge structure in $W_l(S_p)$ is one of the ten edge structures shown in the figure, the edge structure in $W_l(S_n)$ is one of the two corresponding structures shown on the right; either of the two are selected on an equally likely basis. If the edge structure in $W_l(S_p)$ does not correspond to one of the structures shown in the figure, then the structure in $W_l(S_n)$ is made identical to that in $W_l(S_p)$. In doing so, we are again setting $S_n = S_p$ as in the case of Strategy 3.

Strategy 5: Window region change

$$S_n = M_5(S_p, l) .$$

In this strategy, the next state is generated by arbitrarily changing all the pixel labelings in the window $W_l(S_p)$. That is, for all $s_n(x) \in S_n$ such that $s_n(x) \notin W_l(S_n)$, $s_n(x) = s_p(x)$, and for each $s_n \in W_l(S_n)$, the pixels are labeled randomly; each pixel in the window has equal likelihood of being an edge or non-edge pixel. This strategy allows for as many as nine changes in the edge labelings of S_p when generating the new state S_n. In fact, the edge labeling of S_n and S_p are identical at every site except for a random number of K sites in W_l, where $0 \leq K \leq 9$. When K=0, S_n and S_p are identical.

The method of selecting the next state is a combination of the five strategies mentioned above. Given l, we randomly choose from one of the five strategies to generate the next state. Mathematically, the selection process can be expressed in the form

$$S_n = M_X(S_p, l) , \qquad (4.20)$$

where X is a discrete random variable taking on values in the set $\{1,2,...,5\}$.

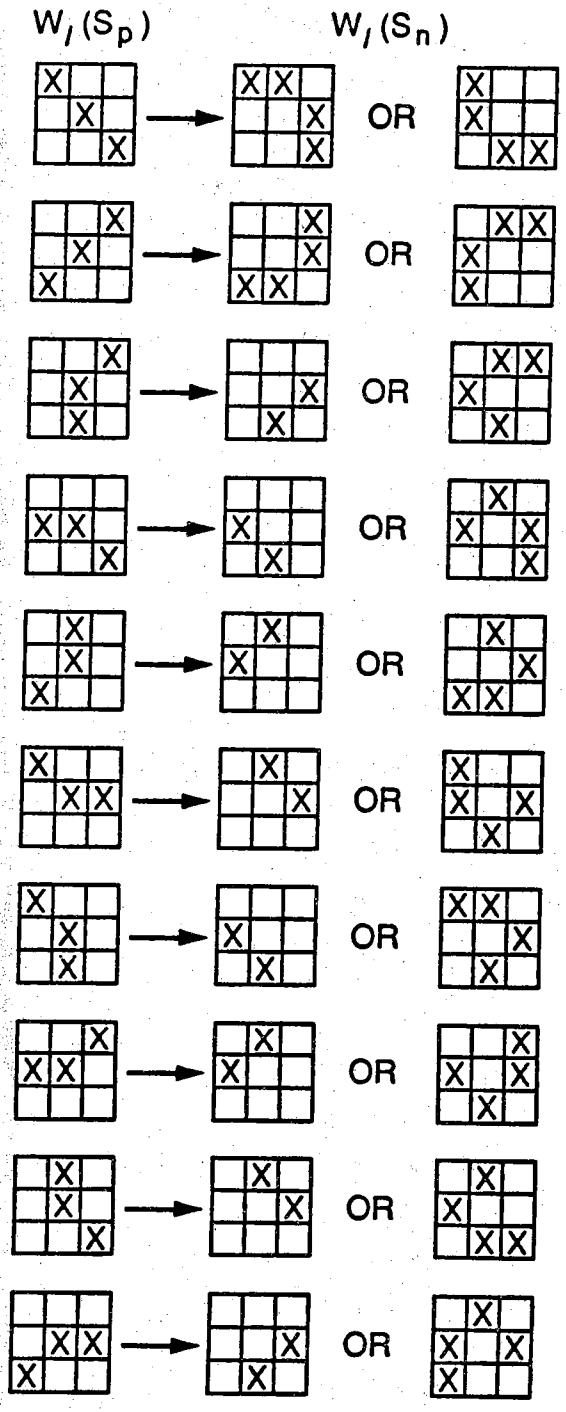


Figure 4.2. The ten edge structures in $W_l(S_p)$ and their corresponding transformations in $W_l(S_n)$ using strategy M_4.

The probability distribution of X is given by

$$P(X = i) = p_i, \quad i = 1,2,...,5 \tag{4.21}$$

where

$$\sum_{i=1}^{5} p_i = 1.$$

The specific values of p_i are application dependent; they determine the frequency that each strategy will be used. Notice that given l and S_p, each new state that can be generated using M_1 or M_3 can also be generated using M_2. That is, the set of states that can be generated using M_1 and M_3 is a subset of the states that can be generated using M_2. Similarly, the set of states that can be generated using M_1, M_2, M_3 and M_4 is a subset of the states that can be generated using M_5. Consequently, the total collection of states that can be generated from a given state using the five different strategies is determined by M_5. Given l and S_p, there are 256 possibilities of generating the next state. Given S_p only, there are approximately $256N^2/9$ different possibilities for the next state. Figure 4.3 shows two examples of the various possible transitions using the five different strategies.

The strategies M_1 and M_5 have a reversible property in the sense that given l, if S_a can be generated from S_b, then S_b can also be generated from S_a. Since M_5 will generate all states possible with the other four strategies, it can be deduced that the method of generating new states using Equation (4.20) also has this reversible property, provided that p_5 is non-zero. This property will be useful in the proof of the weak reversibility (Property 4.1) of the annealing process.

At each iteration through the annealing algorithm, the value of l can be chosen either in a random or deterministic manner. An example of a random approach would be to select the l on an equally likely basis from the set L. An example of a deterministic approach of selecting l is to to sequentially step through each pixel site in the image in a raster scan manner. One guideline that is used for the selection of l is that at low temperatures, every site should be selected at least once before the termination of the annealing process. When this is achieved, we have found experimentally that the results of cost minimization obtained by both approaches are fairly similar. It should be noted that using the above criterion that each pixel site should be selected at least once, a random approach in selecting the value of l would require a

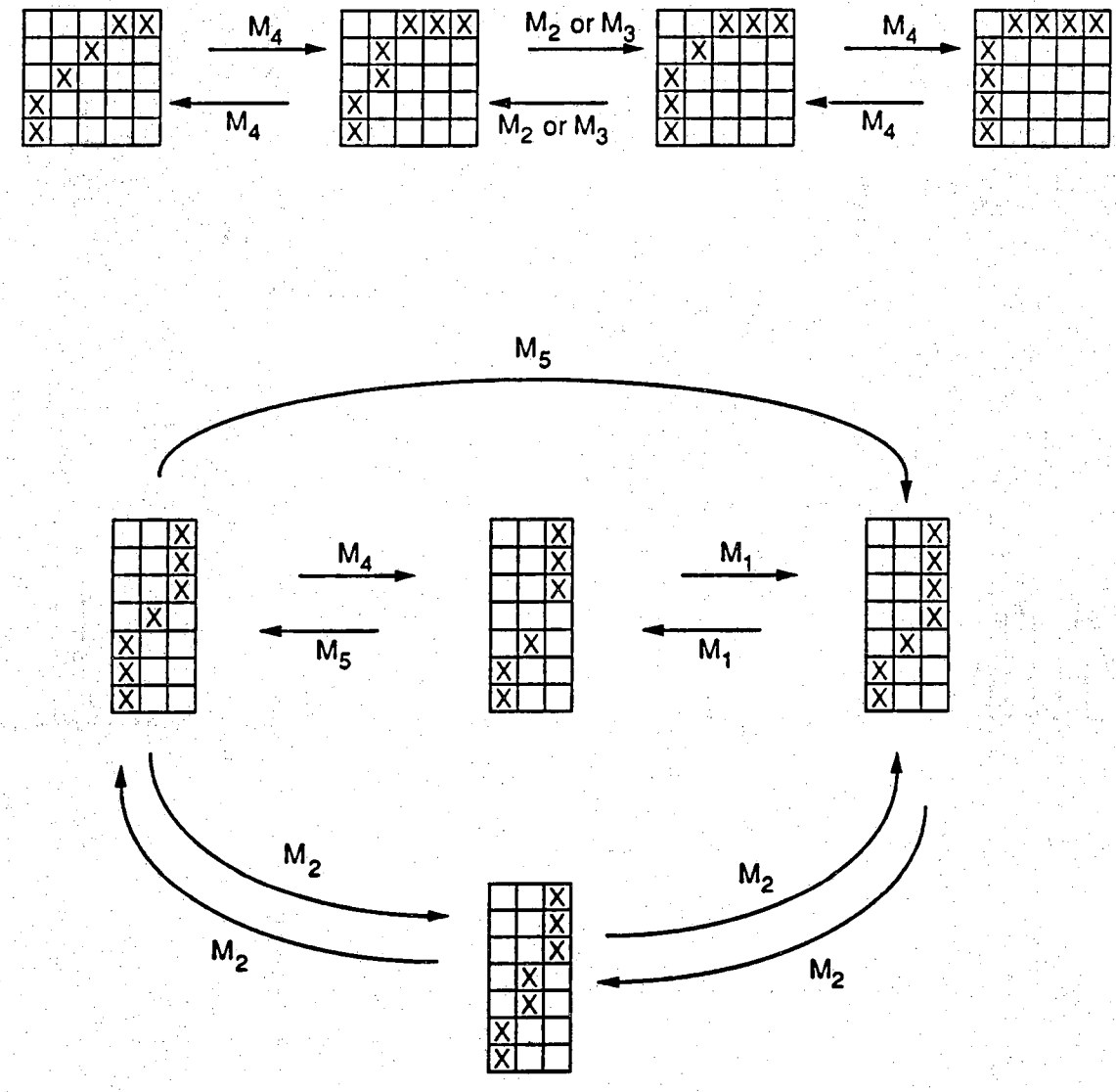


Figure 4.3. **Examples of possible transitions using the five different strategies of generating next states.**

significantly larger number of iterations through the image than the deterministic approach. For example, if the sites are selected on an equally likely and independent basis, after K iterations, the probability that a given site has not been selected,

$$P(\text{a given site has not been selected}) = \left(\frac{N^2 - 1}{N^2}\right)^K,$$

where N^2 is the number of pixels in the image.
Consequently, the probability that it has been selected at least once,

$$P(\text{a given site has been selected}) = 1 - \left(\frac{N^2 - 1}{N^2}\right)^K.$$

Hence, the probability that every site has been selected after K iterations is

$$P(\text{every site has been selected}) = \left[1 - \left(\frac{N^2 - 1}{N^2}\right)^K\right]^{N^2}. \qquad (4.22)$$

For a 128×128 image, a value of $K = 10N^2$ iterations would yield a probability of 0.475 that every site has been visited at least once; a value of $K = 13N^2$ would yield a probability of 0.964. For a 256×256 image, it would require $K = 14N^2$ iterations to yield a probability of 0.947 that every site has been visited at least once. A deterministic raster scan method of selecting l requires only one iteration through the image to ensure that every site has been selected. Hence, from a computational standpoint, it is far more efficient to use a deterministic approach rather than a random approach in selecting l.

We will now show that the method of selecting the next state involving the use of the five different strategies as given in Equation (4.20) results in a Markov chain that is irreducible and has the property of weak reversibility. It does not matter whether the above mentioned random or deterministic approach in selecting l is used; both will result in irreducible and reversible chains. We will assume that the value of p_5 in Equation (4.21) is non-zero.

First we observe that using either the random or deterministic methods of selecting l described above, if state S_m can be generated from S_n, then S_n can be generated from S_m. Consequently, for any sequence of next states $S_0, S_1, \ldots, S_m$, there is a non-zero probability of generating another sequence of states by backtracking the original sequence, which yields the sequence $S_m, S_{m-1}, \ldots, S_0$. Hence the process has the property of weak reversibility as given in Property 4.1.

Let S_m and S_n be any pair of states that have different edge labelings at k sites contained in the set $M=\{l_1, \ldots, l_k\}$; $0 < k \leq N^2$. Assume that the method of selecting l is the deterministic raster scan approach described above. That is, l is selected by sequentially iterating through each site in the image. Beginning from S_m, there is a non-zero probability of generating a sequence of next states such that

(1) if $l \notin M$, then the next state is the same as the present state, and

(2) if $l \in M$, then the next state is generated using M_1.

Each next state generated using M_1 has one less different edge labeling from S_n. At most k intermediate next states are needed to arrive at state S_n. Hence every state is reachable from any other state, and the chain is irreducible.

If the method of selecting l is random and equally likely in L, then it is straightforward to observe that there is a non-zero probability that every member of M will be selected. Hence, as in the previous deterministic case, it is possible to generate a sequence of next states from S_m to S_n. This again results in an irreducible chain.

4.3.2 Temperature Variation

The selection of a suitable temperature schedule is important in the annealing process because it governs in part the rate of convergence to the set of global minimum states. The other governing factor in convergence is the method of generating next states; we could conceively have a very "intelligent" method of generating next states so that the minimums states would be approached rapidly along a path of least cost. We will now focus on the use of Hajek's theorem in the device of a temperature schedule. In particular, we will use a schedule of the form given in Equation (4.16). For practical purposes, the parameter c in the equation has to be kept as small as possible so that the number of iterations can be held within a reasonable limit. For instance, if $c = 10$, then to decrease the temperature to a typical value of 0.3 using Equation (4.16) would require $k = 300 \times 10^{12}$ iterations. However, if $c = 5$, then it would require only $k = 17.3 \times 10^6$ iterations. Since convergence is guaranteed if and only if $c \geq d^*$, it is crucial to be able to find a relatively tight upper bound on d^*. The remainder of this section deals with the analysis and estimation of an upper bound on the value of d^*.

We will estimate the upper bound of d^* by first stating a theorem on the maximum cost ascent necessary to reach the global minimum from a given state S_o. Based on this theorem, we will then give an estimate of the maximum

cost ascent necessary to reach the global minimum from any state. The theorem is as follows.

Theorem 4.2: Let $E = s_G(l_1)s_G(l_2)\ldots. s_G(l_K)$ be any thin edge that is a path or cycle in a global minimum state S_G. Let M be the sites of the pixels contained in the union of all the windows of each pixel site in E;

$$M = \bigcup_{s(l)\in E} W_l \ .$$

Define a sequence of states $S_0, S_1, \ldots, S_K$ such that

(1) $S_0 = \{s_0: l \in L\}$ is any state with $s_0(l) = 0$ for all $l \in M$,

(2) $S_i = \begin{cases} s_i(l) = s_{i-1}(l) \ ; \ l \neq l_i \\ s_i(l) = 1 \ ; \ l = l_i \ , \end{cases}$

for i = 1,2,...K.
Then for all $0 \leq m < n \leq K$,

$$\Delta F_{m,n} \leq \begin{cases} w_f & \text{if E is a path} \\ 2w_f & \text{if E is a cycle} \ , \end{cases}$$

and

$$\Delta F_{0,K} \leq 0 \ .$$

Notice that the construction of the sequence of states is such that each consecutive state S_i differ from the previous state S_{i-1} in that it contains one additional edge pixel of E. The proof of the theorem follows.

Proof:
(A) Assume that E is s thin path. We will first show that $\Delta F_{0,n} \leq w_f$ for $0 < n \leq K$. Next we will show that $\Delta F_{m,n} \leq w_f$ for all $1 \leq m < n \leq K$. Based on these we can conclude that $\Delta F_{m,n} \leq w_f$ for all $0 \leq m < n \leq K$.

(1) $\Delta F_{0,n} \leq w_f$, $0 < n \leq K$.
Let $E_n = \{s_G(l_1),\ldots., s_G(l_n)\}$ be a segment of E, and $\tilde{E}_n$ be the corresponding extended edge segment. From the construction of S_0, it can be deduced that for any state S_n, $0 < n \leq K$, the incremental cost between S_0 and S_n can be written as

$$\Delta F_{0,n} = w_e||E_n|| + w_c \sum_{l \in I(E_n)} C_c(S_G, l) - w_d \sum_{l \in I(E_n)} d(l) + w_f + w_t \Delta C_t .$$

Since E is thin $\Delta C_t = 0$. For each edge segment, $E_n \subset \tilde{E}_n$. Using Proposition 3.19 and the fact that E_n contains an endpoint of E, it is straightforward to conclude that

$$\Delta F_{0,n} \leq w_f .$$

(2) $\Delta F_{m,n} \leq w_f$, $1 \leq m < n \leq K$.
Let $E_{mn} = \{s_G(l_{m+1}), \ldots, s_G(l_n)\}$ be a segment of E, and $\tilde{E}_{mn}$ be the corresponding extended edge segment. Let

$$E'_{mn} = E_{mn} \cup s_G(l_m) .$$

For any state S_n, $1 \leq m < n \leq K$, the incremental cost between S_m and S_n be written as

$$\Delta F_{m,n} = w_e||E_{mn}|| + w_c \sum_{l \in I(E'_{mn})} C_c(S_G, l) - w_d \sum_{l \in I(E_{mn})} d(l) + w_t \Delta C_t .$$

Again, since E is thin $\Delta C_t = 0$. For each edge segment $E'_{mn} \subset \tilde{E}_{mn}$. Using Proposition 3.19, we again conclude that

$$\Delta F_{m,n} \leq w_f .$$

Combining the results of steps (1) and (2) above, we see that

$$\Delta F_{m,n} \leq w_f \quad \text{for } 0 \leq m < n \leq K .$$

(B) Now assume that E is a cycle. Following the same procedure as in (A) above, we can conclude that

$$(1)\ \Delta F_{0,n} \leq 2w_f ;\ 0 < n \leq K, \text{ and}$$

$$(2)\ \Delta F_{m,n} \leq w_f ;\ 1 \leq m < n \leq K .$$

Hence,

$$\Delta F_{m,n} \leq 2w_f ;\ 0 \leq m < n \leq K .$$

(C) We will now show that $\Delta F_{0,K} \leq 0$. Let S_n be the state that is identical to S_G except with the edge E removed; that is,

$$S_n = \begin{cases} s_n(l) = s_G(l) & s_G(l) \notin E \\ s_n(l) = 0 & s_G(l) \in E. \end{cases}$$

By the construction of S_0, S_K, and the use of Proposition 3.11, it can be deduced that the incremental cost between S_0 to S_K is the same as that from S_n to S_G;

$$\Delta F_{0,K} = \Delta F_{n,G}.$$

Since S_G is the global minimum, $\Delta F_{n,G} \leq 0$, and hence

$$\Delta F_{0,K} \leq 0.$$

□

4.3.2.1 An Additional Cost Factor

For the purpose of estimating a tight upper bound on d^*, we define an additional cost factor in order to restrict the edges of minimum cost configurations to be either paths or closed cycles; edge pixels that connect three or more edge segments are disallowed. This is necessary to limit the numerous possible edge structures that need to be taken into consideration. It is achieved simply by assigning a cost to edge pixels that have three or more neighboring edge pixels. For typical images, this restriction affects the final output of the edges only in a very minor way. In most images of interest, the number of points where three or more edge points are connected are few. Furthermore, it will be seen that at such points, the use of the additional cost factor will result in a local discontinuity of usually only one or two pixels; if necessary, this can be easily corrected by a post detection process.

The cost factor is labeled as C_n and is called the cost for number of neighboring edge pixels. It has the following definition:

$$C_n(S_m, l) = \begin{cases} 0, & s_m(l)=0 \\ 0, & \|N_l(S_m)\| < 3 \text{ and } s_m(l)=1 \\ 1, & \|N_l(S_m)\| \geq 3 \text{ and } s_m(l)=1. \end{cases} \tag{4.23}$$

The cost function is now a linear combination of six cost factors instead of the previous five. That is,

$$\mathbf{F}(S_m) = \sum_{l \in L} \left[\sum_{k=1}^{6} w_k C_k(S_m, l) \right] \tag{4.24}$$

$$= w_c C_c + w_d C_d + w_e C_e + w_f C_f + w_t C_t + w_n C_n ,$$

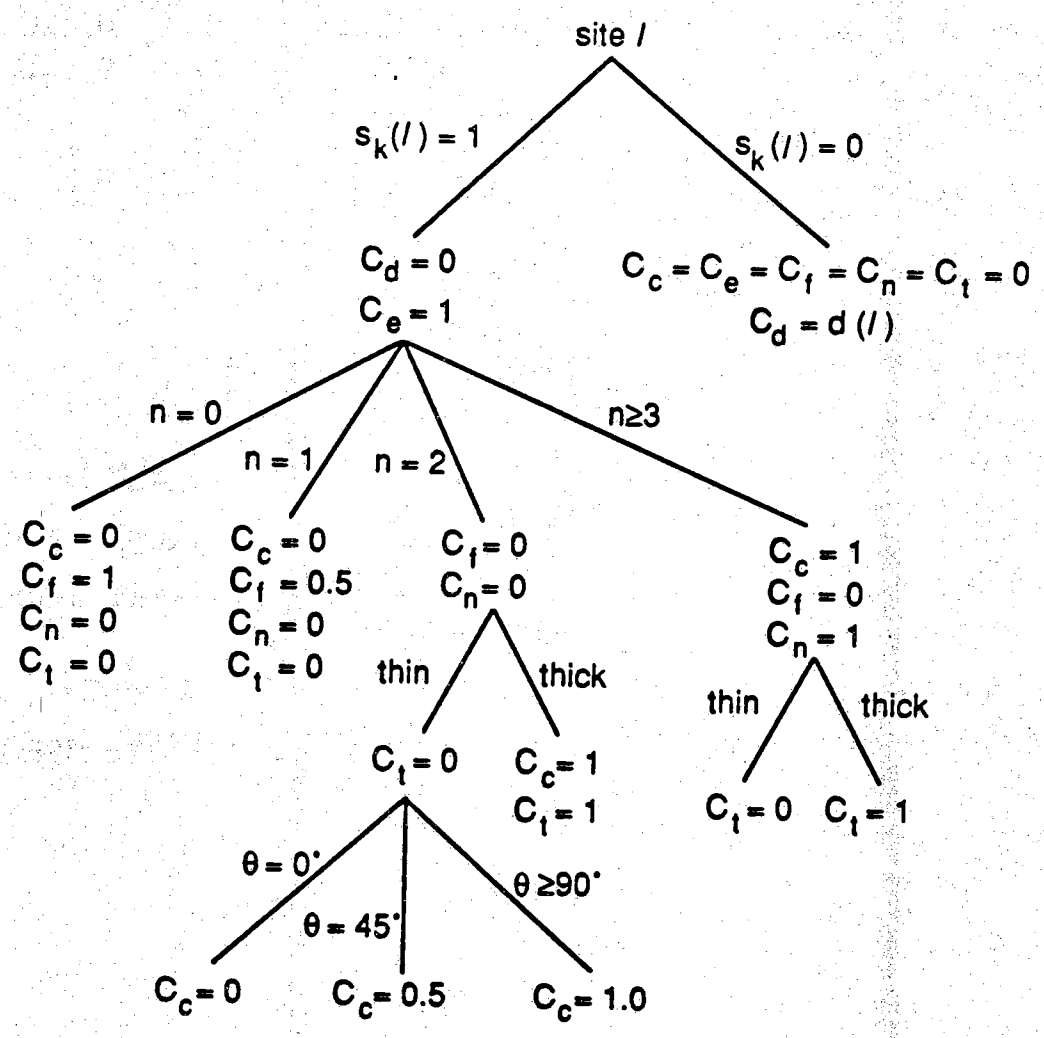
where each of the other five cost factors have been previously defined in Chapter 3. By a trivial modification of the cost tree of Figure 3.16, we obtain the new cost tree shown in Figure 4.4 which includes the cost factor C_n. It is clear that the factor C_n also depends only on the pixels in $W_l(S)$, and consequently, Proposition 3.11 also holds for the new cost function. By simple modification of the proofs to include C_n, it can be shown that Propositions 3.14, 3,16, 3.17, 3.18, 3.19 and Theorem 4.2 also holds for the cost function with 6 cost factors. The two functions are essentially the same except that in the new cost function, we place a cost which tends to disallow edge pixels from having more that two neighbors. We will now state two propositions relating to this new cost function which governs how the weights w_n and w_t are to be chosen to achieve certain characteristics in the detected edges.

Proposition 4.1 Assume that the neighborhood function is $\mathbf{H}_1$. If $w_n > 2w_f + w_d - w_e$, then in any local or global minimum state, every edge pixel has at most two other neighboring edge pixels.

Proof: The proof is by contradiction. We will assume that there exists a local or global minimum state S_0 that has three or more neighboring edge pixels. Assuming that the condition of the proposition holds, we will then show that there exists a neighboring state that has a lower cost; this contradicts the assumption that the initial state is a local minimum.

Assume that S_0 is a local or global minimum with $s_0(l)=1$ for some l, and that $\| N_l(S_0) \| = k$, where k is greater than or equal to three. Let S_n be the state that is identical to S_0 at every site except at l where it is the complement. Clearly, $S_n \in \mathbf{H}_1(S_0)$. The incremental cost can be written as

$$\Delta \mathbf{F}_{0,n} = \sum_{k=1}^{6} w_k \Delta C_k(W_x ; S_0, S_n)$$



$n = \| N_l(S_k) \|$

thin: The edge contained in $W_l(S_k)$ is a thin edge.

thick: The edge contained in $W_l(S_k)$ is a thick edge.

Figure 4.4. Decision tree for computing the six different cost factors.

$$= w_c\Delta C_c + w_d\Delta C_d + w_e\Delta C_e + w_f\Delta C_f + w_t\Delta C_t + w_n\Delta C_n .$$

The factor $\Delta C_e = -1$, and $\Delta C_d = d(l)$. Hence,

$$\Delta F_{0,n} = w_c\Delta C_c + w_d d(l) - w_e + w_f\Delta C_f + w_t\Delta C_t + w_n\Delta C_n .$$

Using the fact that the removal of a single edge point can result in at most four additional endpoints we have $\Delta C_f \leq 2$. By Proposition 3.16, $\Delta C_c \leq 0$, and $\Delta C_t \leq 0$. Since $d(l) \leq 1$, we can conclude that

$$\Delta F_{0,n} \leq w_d - w_e + 2w_f + w_n\Delta C_n .$$

It is clear from the definition of C_n that $\Delta C_n \leq -1$. Therefore

$$\Delta F_{0,n} \leq w_d - w_e + 2w_f - w_n .$$

Assuming that the condition of the proposition holds,

$$w_n > w_d - w_e + 2w_f ,$$

it is straightforward to conclude that $\Delta F_{0,n} < 0$. This implies that S_n is a state of lower cost; we have a contradiction of the assumption that S_0 is a local or global minimum state. Hence, if the condition holds and S_0 is a minimum state, every edge pixel in S_0 can have at most two other neighboring edge pixels.

□

Proposition 4.2: Assume the neighborhood function is H_1 and $w_n > w_d - w_e + 2w_f$. Let S_L be a local or global minimum state. If E is a thick edge, then $||E|| = 3$.

Furthermore, if $w_t > \frac{1}{3}[w_f + w_d - w_e - 3w_c]$, then there are no thick edge pixels in S_L.

Proof:

(1) We will prove by contradiction that $||E|| = 3$. Assume that E is thick and $||E|| \geq 4$. Since E is thick, there must exist a cycle of length three comprising of the pixels $C = \{e_1, e_2, e_3\} \subset E$. Since $|\,|E|\,| \geq 4$, there exists a pixel $e_x \in E$ such that $e_x \notin C$, and e_x is adjacent to one of the pixels in C. Refer to this pixel in C as e_1. Now e_1 has three neighbors; e_2, e_3 and e_x. This contradicts the fact that any edge pixel has at most two other neighboring edge pixels according to Proposition 4.1. Hence $||E|| < 4$. Clearly, every thick edge

must have at least three distinct edge pixels and so $||E|| = 3$.

(2) We will show that if the conditions hold and S_o contains a thick edge pixel, we can always find a neighboring state S_n which does not contain that edge pixel and $F(S_n) < F(S_o)$. Consequently, any local or global minimum state cannot contain a thick edge pixel.

Assume $w_t > \frac{1}{3}[w_f + w_d - w_e - 3w_c]$, and S_o contains a cycle of length three comprising of the pixels $C = \{s_o(l_1), s_o(l_2), s_o(l_3)\}$. Let S_n be the configuration that is identical to S_o at every site except at $l = l_1$ where it is non-edge. Clearly S_n is a neighbor of S_o based on the H_1 neighborhood function. From part (1), we know that C must be an isolated cycle of length three. Consequently the pair of edge pixels $\{s_n(l_2), s_n(l_3)\}$ must also be isolated. The incremental cost can be written as

$$\Delta F_{o,n} = \sum_{k=1}^{6} w_k \Delta C_k(W_x ; S_o, S_n)$$

$$= w_c\Delta C_c + w_d\Delta C_d + w_e\Delta C_e + w_f\Delta C_f + w_t\Delta C_t + w_n\Delta C_n .$$

It is easily deduced that $\Delta C_c = -3$, $\Delta C_d = d(l_1)$, $\Delta C_e = -1$, $\Delta C_f = 1$, $\Delta C_t = -3$, and $\Delta C_n = 0$. This implies that

$$\Delta F_{o,n} = -3w_c + w_d d(l_1) - w_e + w_f - 3w_t .$$

Since $w_t > \frac{1}{3}[w_f + w_d - w_e - 3w_c]$, we have

$$\Delta F_{o,n} < w_d[d(l_1) - 1] ,$$

and since $d(l) \leq 1$, the incremental cost

$$\Delta F_{o,n} < 0.$$

Therefore, S_n is a state of lower cost than S_o.

□

4.3.2.2 Estimating the Upper Bound of d^*.

The parameter d^* in Equation (4.15) is by definition the maximum cup depth of all states which are local but not global minima. We will now discuss a method of estimating an upper bound on the value of this parameter. It is important that this estimate should be fairly tight as it governs the rate at

which we can decrease the the temperature of the annealing process. This ultimately affects the rate of convergence to the set of globally minimum states. Our approach in estimating the upper bound of d^* is to show by construction that we can transition from one local minimum to another local minimum of lower cost without having to encounter a maximum cost accent greater than δ. Except for the global minimum, the maximum depth of each local minimum is thus bounded to a maximum value of δ and hence

$$d^* \leq \delta .$$

Because of the complex nature of the interaction between the different cost factors, and because of the large number of possible edge structures that have to be taken into consideration, we are unable at this time to give a precise theoretical upper bound on d^* which is tight. Instead, we estimate the value of δ based on Proposition 3.19, Theorem 4.2, and an heuristic argument on edge formation. Our approach is to first estimate δ for simple edge structures and their resulting local minimum states that are not global minimum. We progressively move from trivial to more complex forms of local minimum states. We will show heuristically by construction that even in extreme cases, it is possible to transition from one local minimum to another local minimum of lower cost without having to undergo a maximum cost climb exceeding δ, where $\delta = 2w_f + w_d - w_e$.

In the following paragraphs, we will discuss six different cases of edge structures and the corresponding estimates of δ for each case. We will denote δ for each case as δ_i where i denotes the case number. In each case, the corresponding figures depict an edge as a thin continuous line. The position of the edge that corresponds to the global minimum state is represented by a dotted line. We will refer to an edge that exists in the global minimum state as an "optimal" edge. We will assume for each of the cases that the weights of the cost factors are chosen such that in any local minimum there are no thick edges, and every edge pixel has at most two other neighboring edge pixels.

Case 1

The edge corresponding to the global minimum state is a path extending from the top right to the bottom left region of the square lattice, as shown by the dotted line of Figure 4.5. This is the "optimal" edge position. In this case, the local minimum shown is a configuration that contains no edge pixel. We estimate the value of δ_4 using Theorem 4.2; we can construct a sequence of states where each consecutive state contains one additional pixel of the optimal

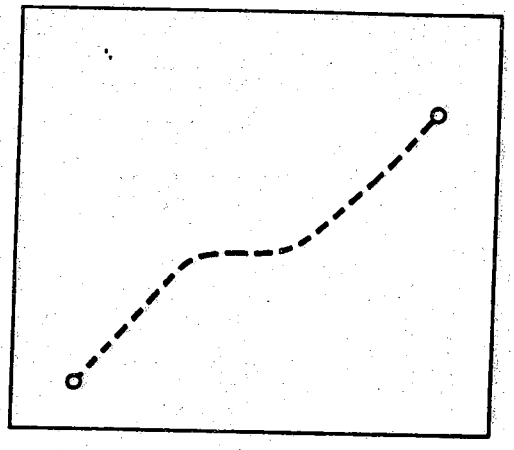


Figure 4.5. An edge configuration that contains no edge pixels. The dotted line indicates the optimum edge position.

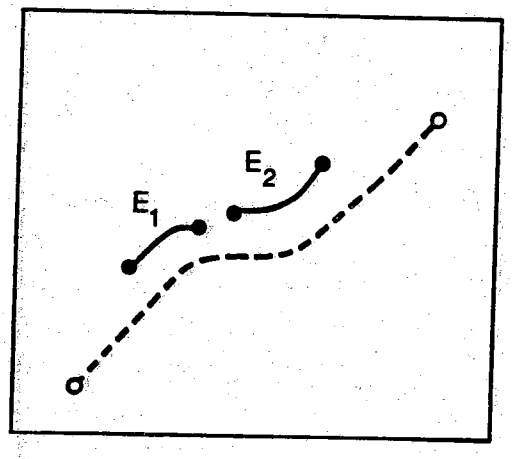


Figure 4.6. An edge configuration that contains two short false edges.

edge, with the final state corresponding to the global minimum. The theorem specifies that the maximum cost ascent is no larger than $2w_f$, and hence

$$\delta_1 \leq 2w_f . \tag{4.25}$$

Case 2

The local minimum in this case is a configuration that contains two edges E_1 and E_2 that are displaced a distance away from the optimal edge position. This situation is shown in Figure 4.6. Since neither E_1 nor any segment of it exists in the global minimum, it is straightforward to deduce that for any segment $E_s \subseteq E_1$,

$$w_d \sum_{l \in E_s} d(l) - w_e ||E_s|| - w_c \sum_{l \in \tilde{E}_s} C_c(S_L, l) \leq w_f .$$

The same can be said for segments of E_2. Consequently, beginning from an endpoint, we can sequentially remove each pixel of E_1 or E_2 without exceeding a cost climb greater than w_f and arrive at lower cost states. Hence

$$\delta_2 \leq w_f . \tag{4.26}$$

Case 3

The local minimum in this case contains an edge that spans only a portion of the optimum edge, as shown in Figure 4.7. We can construct a sequence of states by extending the edge in this local minimum one pixel at a time along the position of the optimal edge. Using Proposition 3.19, it can be concluded that the maximum total cost ascent will not exceed w_f. Hence for this case

$$\delta_3 \leq w_f . \tag{4.27}$$

Case 4

In Figure 4.8, we show a continuous edge of a local minimum in which part of the edge is just slightly displaced from the position of the optimal edge. This is possibly the most common local minimum that will be encounted in the minimization process. It is possible to generate a sequence of states in which the edge pixels are sequentially locally shifted into the position of the optimal edge without breaking the continuity of the edge structure. Consequentially, if there are cost ascents, they will be dominated by curvature costs caused by perturbation of the the edge position. The ascent will not exceed $2w_c$ as the

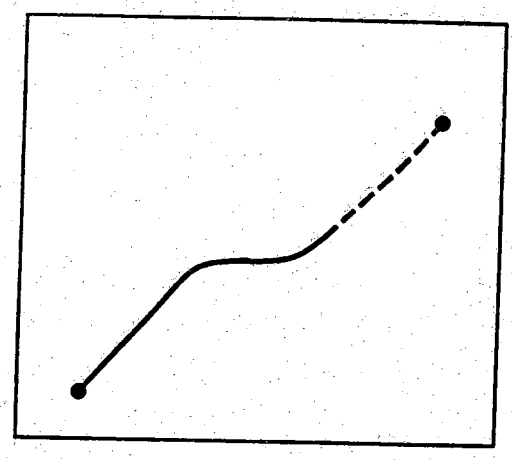


Figure 4.7. An edge that spans only a portion of the optimal edge position.

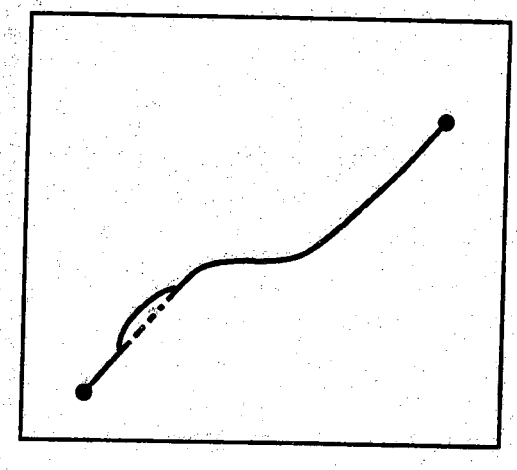


Figure 4.8. An edge that is just slightly displaced from the optimal edge position.

pixels are shifted into the positions of the optimal edge. Thus

$$\delta_4 \leq 2w_c \,. \tag{4.28}$$

Typically, the value of w_c is small compared to the other weights.

Case 5

In Figure 4.9, we show a continuous edge of a local minimum S_L in which part of the edge is displaced some distance away from the position of the optimal edge. We denote the sites of the missing edge pixels of the optimal edge as R_o and the sites of the displaced edge as R_d. It is possible to generate a sequence of states in the following way. First we generate S_1 by introducing a local discontinuity in the edge as shown in the same figure. The resulting incremental cost is bounded by

$$\Delta F_{L,1} \leq w_f + w_d - w_e \,. \tag{4.29}$$

Next, we generate a sequence of states from S_1 to S_2 by sequentially adding edge pixels in the positions of the optimal edge. By Proposition 3.19, the maximum cost ascent required for the transitions given by this sequence is no greater than w_f. In addition, since we are constructing the optimal edge in S_2, it is safe to assume that $F(S_2) < F(S_1)$. As a result, the maximum cost ascent so far from S_L is still given by Equation (4.29). From the results Case 2, we deduce that it is possible to transition from S_2 to the global minimum S_3 without encountering a maximum cost ascent greater than w_f. This is done by sequentially removing the edge pixels in R_d. An estimate for an extreme case of the total cost ascent is given by taking the sum of the maximum ascents of $\Delta F_{L,1}$ and $\Delta F_{2,3}$:

$$\delta_5 = 2w_f + w_d - w_e \,. \tag{4.30}$$

Case 6

The example shown in Figure 4.10 shows a local minimum S_0 that is a combination of several of the five cases discussed above. Each consecutive state, S_1, S_2, S_3 is a lower cost state with S_3 corresponding to the global minimum. The maximum cost ascent required to reach the global minimum from S_L is the maximum of the δ for the five cases above, and is given by

$$\delta_6 \leq 2w_f + w_d - w_e \,. \tag{4.31}$$

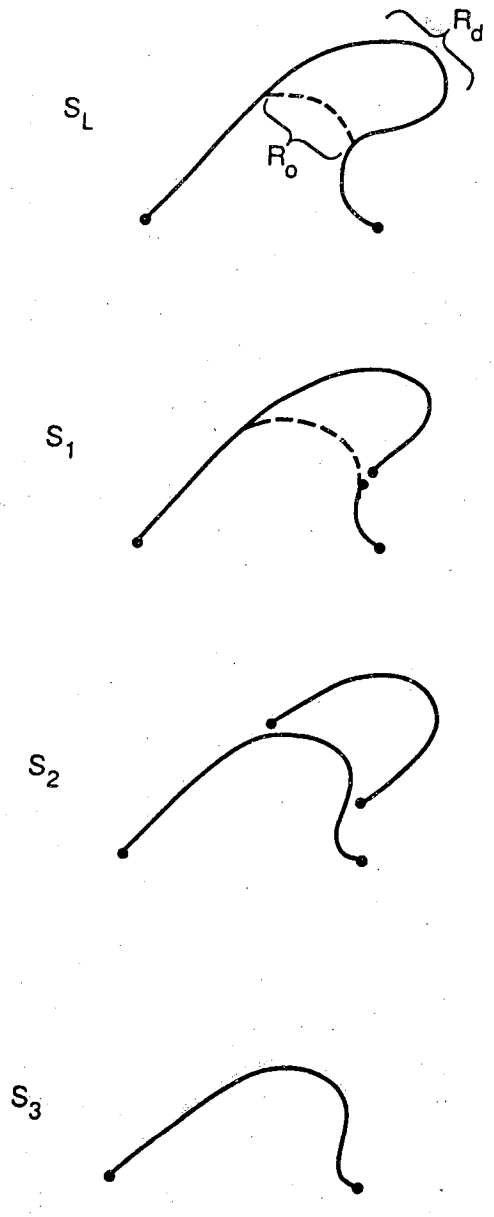


Figure 4.9. Displaced edge. S_L is a continuous edge that has a portion that is displaced some distance away from the position of the optimal edge. Each consecutive state can be reached by a sequence of transitions from the previous state. S_3 corresponds to the global minimum state.

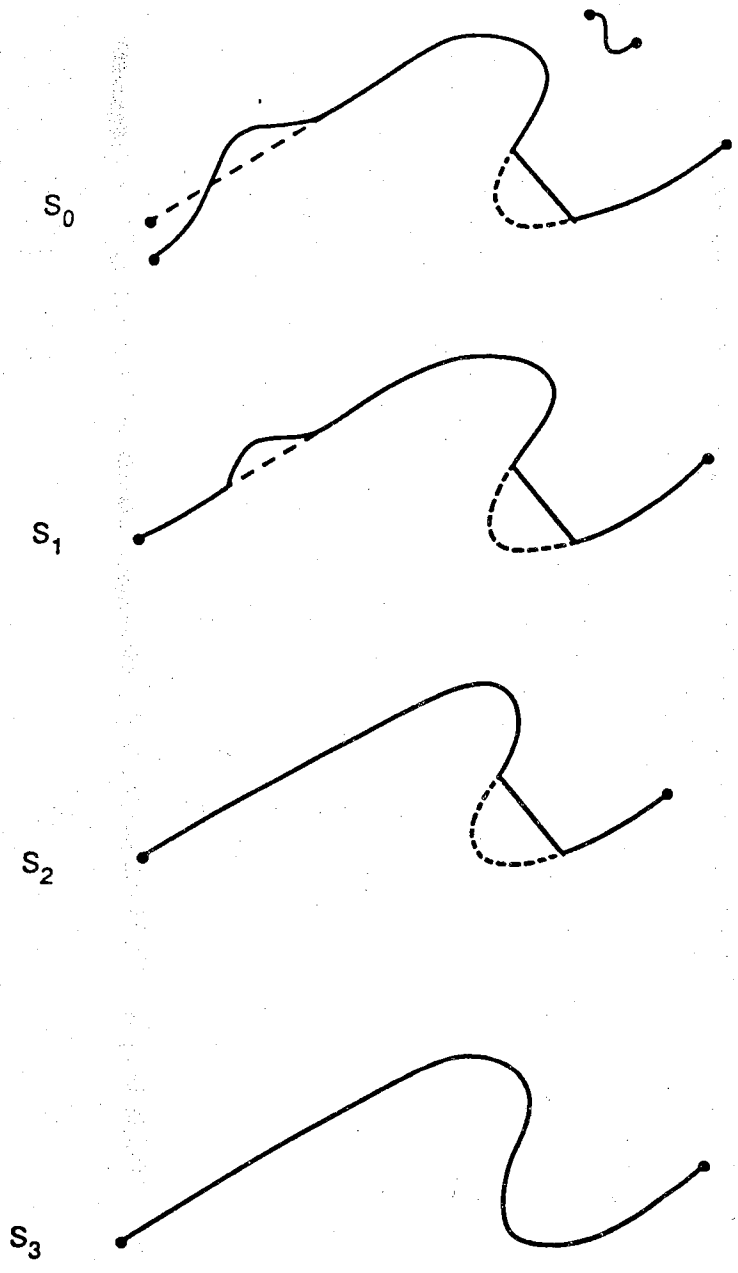


Figure 4.10. A sequence of states of lower cost. S_0 is a local minimum. Each consecutive state has a lower cost and S_3 corresponds to the global minimum state.

General case

In the preceding discussion, we have estimated by construction that for minimum cost configurations containing a solitary edge, the maximum depth of any local minimum is given by Equation (4.31). It should be noted that the estimate is quite conservative as the method of constructing a sequence of low cost transitions is based on assumptions corresponding to fairly extreme cases of edge structure. It is conceivable that for typical images, the maximum depth is much smaller than the given bound. Given a specific image and a local minimum state, it is very likely that one could construct a sequence of transitions to the global minimum with a total maximum cost ascent much less than the bound of Equation (4.31). However, when the specific image is not known, we are unable to devise a general method of constructing a sequence of low cost transitions that has a lower maximum cost ascent. This is due mainly to the complexity of the interaction of the cost factors, the vast number of possible transitions, and the uncertainty in the values of the pixels of interest in the enhanced image.

As mentioned before, we have assumed that the weights of the cost factors are chosen so that in a local or global minimum state, there are no thick edges and every edge pixel had at most two other neighboring edge pixels. Consequently, for a general image, the corresponding global minimum cost configuration is simply a collection of non-intersecting edges which are isolated paths or cycles. Each of these edges corresponds to an optimum edge. In cases 1 to 6 discussed above, we have dealt only with images that have one single optimum edge. The estimation of δ for images containing more than one optimum edge is similar to that for images with only one edge. The intuitive notion is that using the techniques described in the different cases above, we can sequentially construct one edge at a time by appropriately chosen transitions. This is repeated until we arrive at the global minimum configuration containing all the desired edges. Based on this notion, we anticipate that the maximum depth of any local minimum is again no larger that that given in Equation (4.31). That is,

$$\delta \leq 2w_f + w_d - w_e \,. \tag{4.32}$$

4.3.2.3 Temperature schedule

The method of generating next states in the annealing algorithm is given by Equation (4.20). The image is assumed to be of size $N \times N$. The temperature schedule used is the the following:

$$T_k = \frac{c}{\log(nk_s + 2)}\ , \tag{4.33}$$

where k_s is a scaling constant, and n is defined using the floor function:

$$n = \left\lfloor \frac{k}{N^2} \right\rfloor \cdot N^2\ .$$

Note that the temperature is monotone decreasing and is changed only after every N^2 iterations through the annealing algorithm. It can be easily verified that for any finite and strictly positive value of k_s greater than or equal to 2, the temperature schedule satisfies Equation (4.15) if $c \geq d^*$. This implies that convergence to the set of global minimum states is guaranteed if $c \geq d^*$. Our estimate of the upper bound on d^* is based on Equation (4.32). Hence, in our implementation, we set $c = \delta = 2w_f + w_d - w_e$ for the temperature schedule given in Equation (4.33). Hence, to ensure that the process will converge asymptotically, the value of c must be no less than δ. That is,

$$c \geq 2w_f + w_d - w_e\ . \tag{4.34}$$

It is interesting to note that if we attempt to use the temperature schedule of Mitra et al. given in Equation (4.17), the value of r is approximately $N^2/9$, and a very conservative lower bound of ρ is 1. This implies that the value of γ which is analogous to c in the above equation is given by

$$\gamma \geq N^2/9\ .$$

The schedule is impractical from an implementation standpoint because it would take far too many iterations to span even a small range of low temperatures. As an example, consider a 128×128 image. Assume that k is small. For the temperature to be in the proximity of 0.3, the value of k_0 will have to be set approximately to

$$k_0 = \exp(6000)\ .$$

From an implementation standpoint, this number is far too large for computer representation. Even if representation is possible, the temperature schedule according to Equation (4.17) would then remain constant for any practical

range of the values of k.

4.3.3 Parallel Implementation

The Simulated Annealing algorithm described in Section 4.2 is essentially a sequential algorithm. We will now discuss a method of generating next states which will allow the algorithm to be to be implemented to a large extent in parallel. In fact, we will show that the number of sequential computations can be reduced by a factor of $N^2/9$. That is, up to $N^2/9$ computations can be made simultaneously in parallel. We will assume that given a present state S_p and a site $l \in L$, the method of generating the next state is given by Equation (4.20);

$$S_n = M_X(S_p, l) .$$

Up to this point we have interpreted the above equation as a method of generating next states, and the Simulated Annealing algorithm as a method of transitioning from one state to another. We now present another interpretation of Equation (4.20) and the annealing process in the context of detecting edges by cost minimization. The above equation can be viewed as a method of altering the local edge structure in a window region centered at site l. The transition rules of the annealing algorithm correspond to a method of deciding if the alteration is to be accepted based on the change in cost caused by the alteration. The annealing process is thus a procedure where we repeatedly attempt to alter the local edge structure at each site in an image according to the rules of annealing. Since the annealing process is guaranteed to converge, the eventual result of the repeated changes is that the edges will take the form of a minimum cost edge configuration.

As mentioned in Section 4.3.1, the value of l can be selected either in a random or deterministic manner. We will now present a deterministic method of selecting l which allows for parallel computation. If the raster scan approach mentioned in Section 4.3.1 is used, the annealing process can be viewed as a procedure where we sequentially attempt to change the edge structure in a window region as the window is shifted through each pixel in the image. Clearly, this is a strictly sequential process as each decision on accepting a change is dependent on the immediate past decisions. Such a method of selecting l does not allow for parallel execution.

By using Proposition 3.11, it is easy to deduce that if an edge structure is altered at a single site l according to Equation (4.20), then the resulting change in cost is dependent at most on the pixels in a 5×5 window region about l.

Consequently, if l_1 and l_2 are two sites that are at least two pixels horizontally or vertically apart (i.e. two pixels between them), the decisions to accept any alterations of the edge structure in W_{l_1} and W_{l_2} can be made independently. In fact, for any set of sites $\{l_1, \ldots, l_M\}$ in which every pair is at least two pixels apart, the decisions to accept alterations in the edge structure can be made independently of each other.

The set of all sites in the lattice denoted by L can be partitioned into k disjoint subsets where any pair of sites in the same subset are at least two pixels apart;

$$L = L_1 \cup L_2 \cup .... L_k . \tag{4.35}$$

It is easy to deduce that at most 9 subsets are required to partition L in this manner. This holds for images of any size. An example of this is shown in Figure 4.11. If alterations in the local edge structure are made at any number of sites belonging to the same subset, the decisions to accept each of the alterations can be made independently. Consequently, it is always possible to make 1 iteration through each pixel in the image in 9 sequential processing steps, where each step requires approximately $N^2/9$ parallel computations. Instead of making N^2 sequential decisions in altering the local edge structure at each site in the image, the same can be achieved by simultaneous decisions in 9 sequential steps. This of course is significantly more efficient in terms of the total required computation time.

Assuming that the method of selecting the sites is such that every site will be repeatedly selected in the annealing algorithm, it can be shown that the corresponding annealing process which allows for parallel computation has the property of irreducibility and weak reversibility. Hence this method of selecting l will result in asymptotic convergence to the global minimum.

4.3.4 State Space Reduction

Simulated Annealing is a computationally intensive algorithm suitable for minimizing complex optimization problems. In the context of edge detection, the amount of computation time can be significantly decreased by reducing the state space of the annealing process. This is achieved by introducing a preliminary processing stage which we call "low resolution detection". The output of low resolution detection is a binary image indicating where edge pixels can and cannot lie; ones indicate the possible positions of edge pixels, and zeros indicate the positions where edge pixels cannot lie. By using this, we effectively reduce the set of all possible edge configurations by placing a

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24
25	26	27	28	29	30
31	32	33	34	35	36

$$L = L_1 \cup L_2 \cup \cdots L_9$$

$L_1 = \{1, 4, 19, 22\}$
$L_2 = \{2, 5, 20, 23\}$
$L_3 = \{3, 6, 21, 24\}$
.
.
.
$L_9 = \{15, 18, 33, 36\}$

Figure 4.11 Example of partitioning L into disjoint subsets. Any pair of pixels in the same subset is at least 2 pixels apart. The pixels of the above 6×6 image are labeled 1 through 36. The image is first divided into blocks of 3×3 pixels. Partitioning is then achieved by selecting corresponding pixels of different blocks.

constraint on the configurations that are taken into account. In other words, the cost function is minimized subject to the constraint that the edge configurations can contain edge pixels only in the regions specified by the low resolution output.

There are a number of ways of performing low resolution detection. We chose to first threshold the enhanced image. Next we performed the morphological operation [59] "dilation" on the thresholded image using a square 3×3 or 5×5 structuring element. Examples of the output of low resolution detection for state space reduction is shown in Chapter 5, Section 5.2.1.

4.4 Summary

In this chapter, we have presented Simulated Annealing as a technique in cost minimization. It has been shown that the annealing algorithm is a stochastic optimization technique based on non-stationary Markov chains; the chain will converge in probability to the set of global minimum states of the cost function. We have described the asymptotic convergence properties of the algorithm and discussed the use of various temperature schedules suitable for convergence.

We used the Simulated Annealing algorithm to find low cost solutions to the cost function for edges described in Chapter 3. First, we showed how to generate next states in the annealing process based on a set of five strategies for changing the edge structure in a given configuration. Second, we devised a suitable temperature schedule by estimating a relatively tight upper bound on the maximum depth of all local minimum states which do not correspond to the global minimum. Third, we showed that although the annealing process is sequential in nature, it can be implemented largely in parallel by a proper choice of next states. Finally, we proposed the use of state space reduction to reduce the computation time for the annealing process.

CHAPTER 5
EXPERIMENTAL RESULTS

5.1 Introduction

In this chapter we present experimental results of detecting edges using the comparative cost function (CCF) and absolute cost function (ACF) techniques described in the previous chapters. The ultimate test of any detection technique is in its ability to find edges that correspond to true boundaries in an image. Comparison of the detection performance is made with four other recent techniques mentioned in Section 1.2; derivative of Gaussian (∇G), Laplacian of Gaussian ($\nabla^2 G$), facet model approach, and Sequential Edge Linking (SEL). It should be noted that the ∇G and facet model are techniques which are optimized for the detection of step edges. Non-maximal suppression for the ∇G technique was performed by quantizing the edge direction of the ∇G operator output into one of eight possible directions and suppressing the non-maximum magnitude values in a direction perpendicular to the edge direction. The SEL technique used the ∇G operator (without non-maximal suppression) as the edge enhancement operator.

As described in Section 2.4.1, the CCF used the weight values w_e=1.0, w_d=2.0, w_t=1.1, w_l=1.1 and w_c=1.1. For the ACF, we first assigned values for the weights w_c, w_d, w_e and w_f according to the desired emphasis on each cost factor. Then, to avoid the detection of thick edges, w_n and w_t were chosen based on Propositions 4.1 and 4.2. In all examples using Simulated Annealing, the value of d^* (in Hajek's Theorem) was estimated using Equation (4.32). Except for the examples in Section 5.4 and parts of Section 5.6, the measures of dissimilarity, f_c(R1,R2) and f_a(R1,R2), were based on the difference of gray level averages in R1 and R2. That is, f_c=m(d) as specified in Equation (2.6), and f_a=d where d is as defined in Equation (2.4).

For both the heuristic search technique and Simulated Annealing, it is necessary to generate new edge configurations by iterating through each pixel location in the image. Assuming that the image is of size $N \times N$, a single iteration through the image represents the generation of N^2 new edge

configurations. Typically, the CCF approach required 3 to 5 iterations through an image. The ACF approach required between 50 to 200 iterations. In all examples, the probabilities p_i in Equation (4.21) were:

$$p_1=\frac{200}{1024},$$

$$p_2=\frac{300}{1024},$$

$$p_3=\frac{200}{1024},$$

$$p_4=\frac{200}{1024}, \text{ and}$$

$$p_5=\frac{124}{1024}.$$

The temperature schedule for the annealing process was based on Equation (4.33):

$$T_k = \frac{c}{\log(nk_s + 2)},$$

where

$$n = \left\lfloor \frac{k}{N^2} \right\rfloor \cdot N^2 .$$

The value of k_s was selected based on the criterion that T_k should be approximately 0.3 at the final iterations through the image. This value of 0.3 was chosen empirically based on the observation that as the temperature decreased toward 0.3, the processes approached a point of "freezing" where very few uphill climbs were made. In the final 2 iterations, the process was quickly "frozen" by dropping the temperature suddenly towards zero. This was achieved by setting the temperature to a value of 0.01, and allowing for transitions based only on strategies M_1 and M_3 (described in Section 4.3.1).

A thorough experimental analysis and comparison of different edge detection techniques would require taking into account a number of different factors. Some of these are: (1) the test images used, (2) the characteristics of the detected edges (in terms of continuity, thinness, and well localization), (3) the operator size, (5) computation time, (6) the difficulty of implementation, and (7) the flexibility of the detection algorithm in detecting various edge

types. There is a trade-off between the different factors; for instance, one usually has to sacrifice computation time for improvement in the characteristics of the detected edges. We will examine the performance of the detection algorithms with respect to several of these factors.

5.2 Experiments with Artificial Images

We compare the performance of the different techniques by first showing examples of the detected edges for artificial images. Evaluation of the detection performance is based on the accuracy in localization of the detected edges, and the form of the edges in terms of thinness and continuity. However, it is difficult to define a performance measure that correctly evaluates the detection performance for all cases of the detected edges. A method of evaluating edge detection performance is the Pratt figure of merit [60] which is denoted by the symbol P:

$$P = \frac{1}{I_M} \sum_{i=1}^{I_D} \frac{1}{1 + \alpha l_i^2} \cdot 100$$

where

I_M = max (I_D,I_I),

I_I = number of ideal edge points,

I_D = number of detected edge points,

I_i = displacement of the ith detected edge point from the ideal edge, and

α = scaling factor.

The value of P ranges from 0 to 100 with higher values indicating better detection performance. The value of 0.1 was used for α which is approximately the same as that used in [61]. This figure of merit is usually applied to artificial images where the ideal edge positions are known. It penalizes edge pixels which are displaced from the ideal edge position according to the displacement distance and the value of α. It also penalizes missing edge pixels or an excessive number of detected edges. However, it does not take into account local edge coherence information such as continuity and edge thickness. A discussion of the shortcomings of this figure of merit is given in[62]. When using this figure of merit, it is important to bear in mind its inherent inadequacy in using local edge coherence information. We use the Pratt figure of merit as a rough indicator of the performance of the different

detection techniques. Two ideal step images which are shown in Figure 5.1 were used; they were the vertical step image and the rings image. The vertical step image had a size of 256×256 pixels and was comprised of two tones of constant gray levels of values 110 and 140. The rings image had a size of 128×128 and was made up of concentric circles of gray levels 115 and 140, constructed in the manner described in [62]. The step heights of the ideal vertical edge and the rings image were consequently 30 and 25 respectively. The images were corrupted with additive zero mean independent identically distributed (i.i.d.) Gaussian noise. The signal to noise ratio of the corrupted images is defined as

$$\text{SNR} = \left(\frac{h}{\sigma_n}\right)^2, \tag{5.1}$$

where h is the ideal step height and σ_n is the standard deviation of the Gaussian noise. The noise corrupted images are also shown in Figure 5.1.

The Pratt figure of merit is often applied to the vertical step image shown in Figure 5.1. In Figure 5.2, we show an example of the difficulties that could arise in the use of this figure of merit. The ∇G operator without the use of non-maximal suppression was applied to the noisy vertical step image. The detected edges obtained by thresholding the output of the ∇G operator at 53 and 35 have corresponding performance values P=78.2 and P=52.3 respectively. It can be seen that if edge continuity and recovery of the complete boundary is of importance, the edge which corresponds to thresholding at 35 is better. Hence, when using this figure of merit, it is important to bear in mind its inherent inadequacy in using local edge coherence information.

Noise smoothing

It is advantageous to preprocess a noise corrupted image by filtering prior to edge detection [7]. We used a Gaussian function to smooth the noise corrupted images. The function is the same as that in Equation (1.1). This smoothing prior to detection was performed only for the facet model, comparative cost function, and absolute cost function techniques; the ∇G and $\nabla^2 G$ operators have Gaussian smoothing inherently incorporated in them. Except for the case of the house image, the standard deviation (σ_G) of each Gaussian function was independently chosen for the different cases so as to optimize the performance of the various detection techniques. The value of σ_G was constrained to be some integer multiple of 0.5. Figure 5.3 shows experimental results of the improvement in detection performance of the

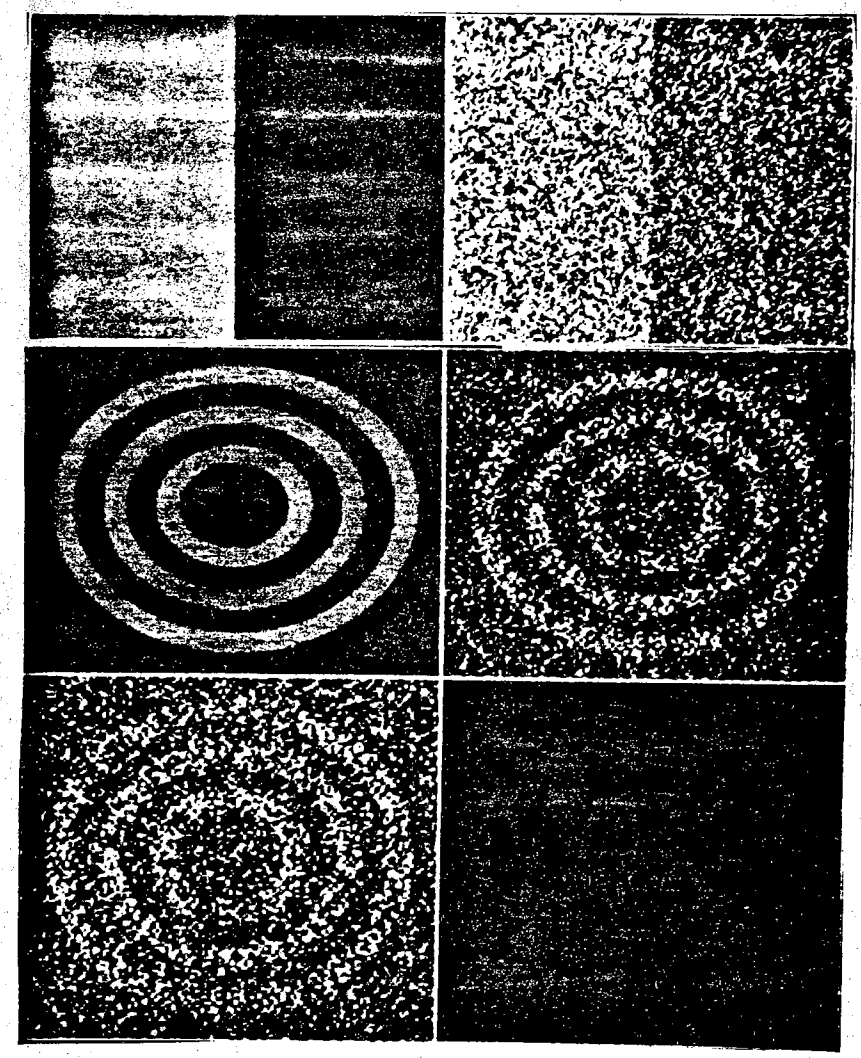


Figure 5.1. Step images. Top left: Vertical step edge. Top right: Noisy step edge with SNR = 0.25. Middle left: Rings image. Middle right: Noisy rings image with SNR=1.0. Bottom: Noisy rings image with SNR=0.574.

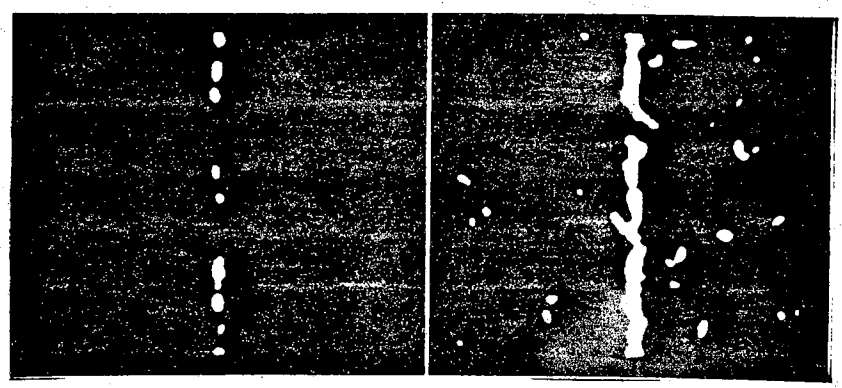


Figure 5.2. Edges of noisy step image detected using the thresholded ∇G operator without non-maximal suppression. Left: Threshold at 53 resulting in P=78.2. Right: Threshold at 35 resulting in P=52.3.

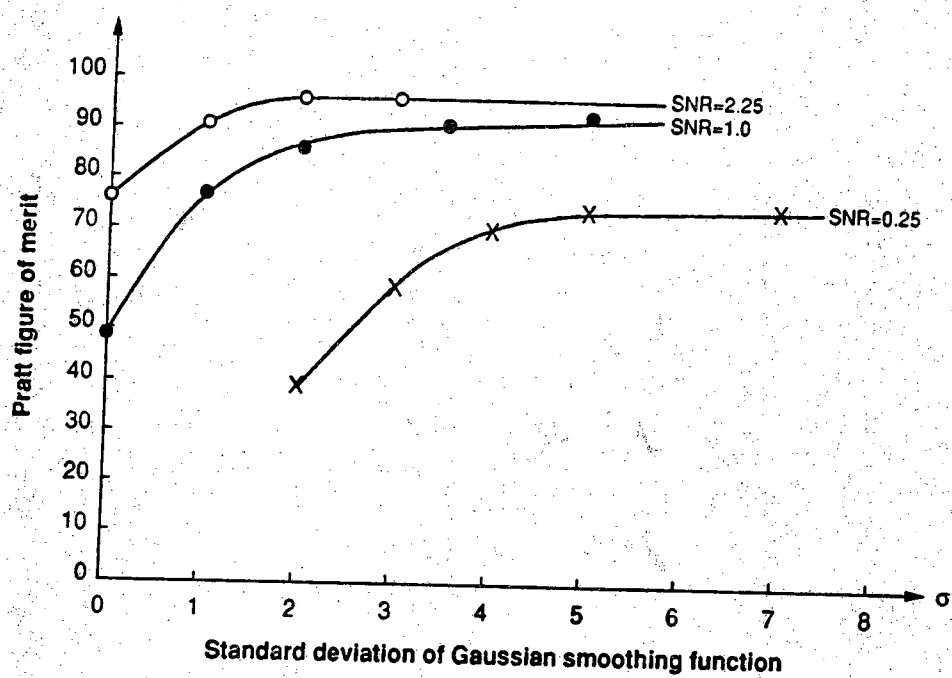


Figure 5.3 Improvement in detection performance by preprocessing noisy raw image with Gaussian smoothing prior to edge detection. The CCF technique was used on vertical step images with different SNR values.

comparative cost function technique by preprocessing the image with Gaussian smoothing prior to edge detection. The detection algorithm was the CCF technique, and the test image used was the vertical step image at various signal to noise ratios. Using this test image, the results indicate that detection performance increases with the standard deviation of the smoothing function. However, it should be noted that this image contains a single isolated edge; if the image contains several adjacent edges, then large values for the standard deviation could cause the edges to be merged together resulting in degraded performance.

5.2.1 Vertical Step Image

The noisy vertical step edge with SNR=0.25 was used to compare the output of different edge detector techniques. The results are shown in Figure 5.4. The figure shows the best edges (based on performance measure P) detected under the constraint that approximately 90% of the ideal edge should be detected. The ∇G technique used a value of 5.5 for the standard deviation of the Gaussian function. The $\nabla^2 G$ operator used a standard deviation of 10.0. SEL was based on the output of the ∇G operator with a standard deviation of 4.0. For the CCF and facet model techniques, we preprocessed the image with a Gaussian smoothing operator with standard deviation 5.0. For the ACF approach, the image was pre-filtered with a Gaussian function of standard deviation of 5.5. The weights of the cost factors of the ACF were: w_c=0.75, w_d=2.0, w_e=1.0, w_f=3.0, w_n=7.01, and w_t=0.583. A total of 200 iterations through the image were made. State space reduction was used to reduce computation time. The ACF implementation as described above required 1.28 hours of CPU computation time on the VAX 11/780. Table 5.1 shows the corresponding performance of the various detection algorithms.

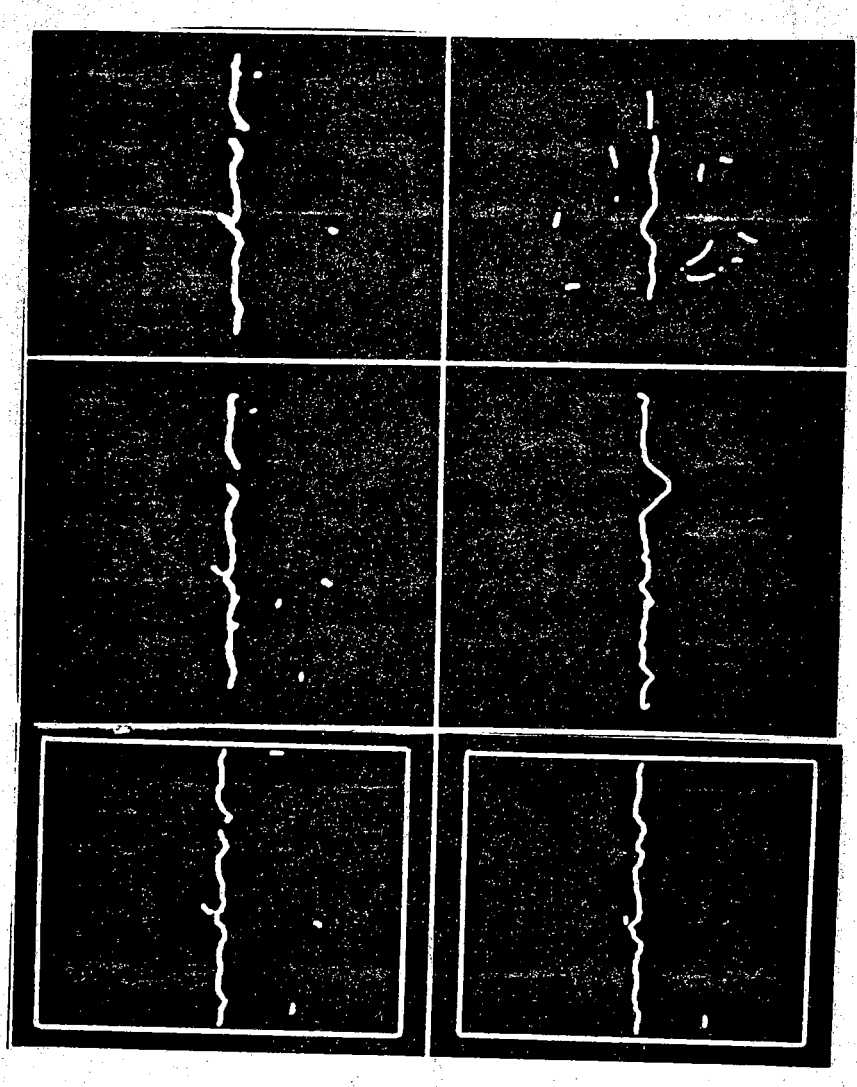


Figure 5.4 Comparison of edge detector performance using vertical step edge with SNR=0.25. Top left: ∇G, P=73.1. Top right: $\nabla^2 G$, P=44.5. Middle left: Facet model technique, P=71.1. Middle right: SEL, P=65.4. Bottom left: CCF approach, P=73.7. Bottom right: ACF approach, P=78.4.

Table 5.1. Detection performance of various detection techniques.

Detection technique	Pratt figure of merit P
∇G	73.1
$\nabla^2 G$	44.5
Facet model	71.1
SEL	65.4
CCF	73.7
ACF	78.4

A comparison of the performance based on P shows that except for $\nabla^2 G$, the different techniques yield approximately the same performance. We extracted the detected edges and placed them alongside each other for more detailed comparison. This is shown in Figure 5.5. A visual examination shows that the ∇G and the facet model techniques produced edges which are thick along many portions of the edge. The CCF and ACF techniques produced edges which are thin. The best performance in terms of continuity and edge thinness is achieved by the ACF technique.

In Figure 5.6, we show the effect of using only the cost factors C_d and C_e of the ACF; the other cost factors were discarded by setting their corresponding weights equal to zero. This method corresponds to a simple thresholding approach to detect the edges. By altering the value of the dissimilarity threshold $\frac{w_e}{w_d}$ (see Section 3.4.4.1), we can arbitrarily select the total number of edge points to be detected. Several important observations can be made from comparing the detected edges shown in Figure 5.4 and Figure 5.6 using the ACF approach. First, based on a cost function that uses only C_d and C_e, it is not possible to detect a thin continuous edge for the noisy step image. Second, there are no thick edge pixels when the cost factor C_t was included, and the corresponding weight w_t was appropriately chosen according to Proposition 4.2. Third, the inclusion of the fragmentation cost C_f forces adjacent edges to be continuous. At the same time, C_f also suppresses short sporadic edges which are visible in Figure 5.6, but not in Figure 5.4.

In Figure 5.7, we demonstrate the effect of changing the weights of the curvature and fragmentation costs in the absolute cost function. For the detected edge in Figure 5.7(a), the weights of the cost factors were w_c=0.2,

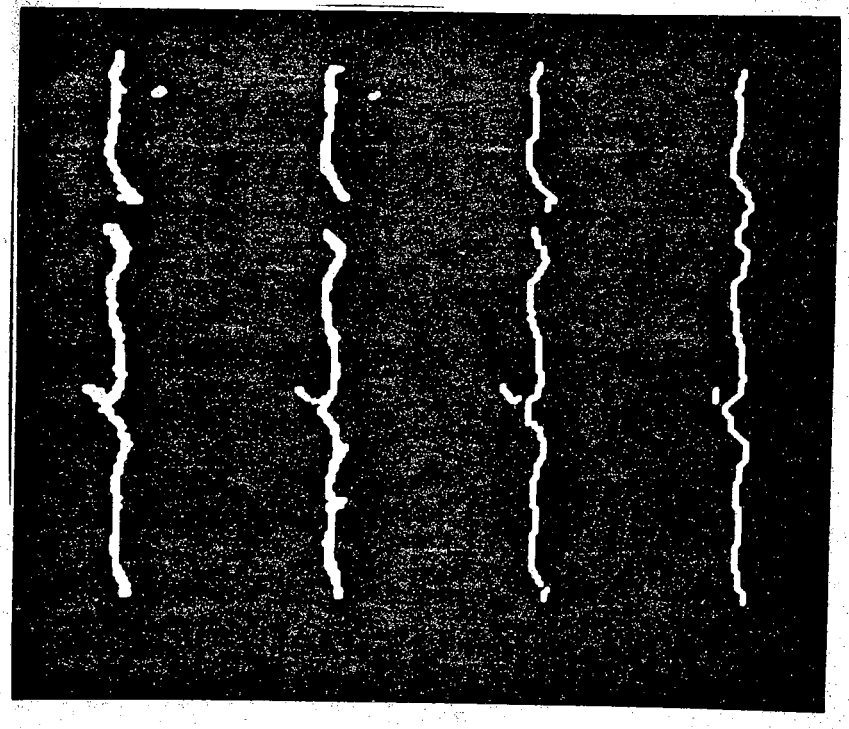


Figure 5.5 Comparison of edge characteristics for noisy vertical step image. Extreme left: ∇G. Center left: Facet model approach. Center right: CCF. Extreme right: ACF.

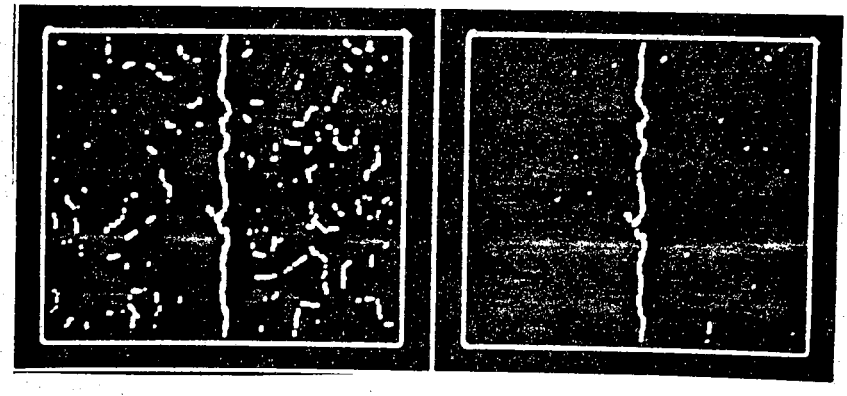


Figure 5.6 Edges detected using only C_d and C_e of the ACF. Left: Low dissimilarity threshold. Right: High dissimilarity threshold.

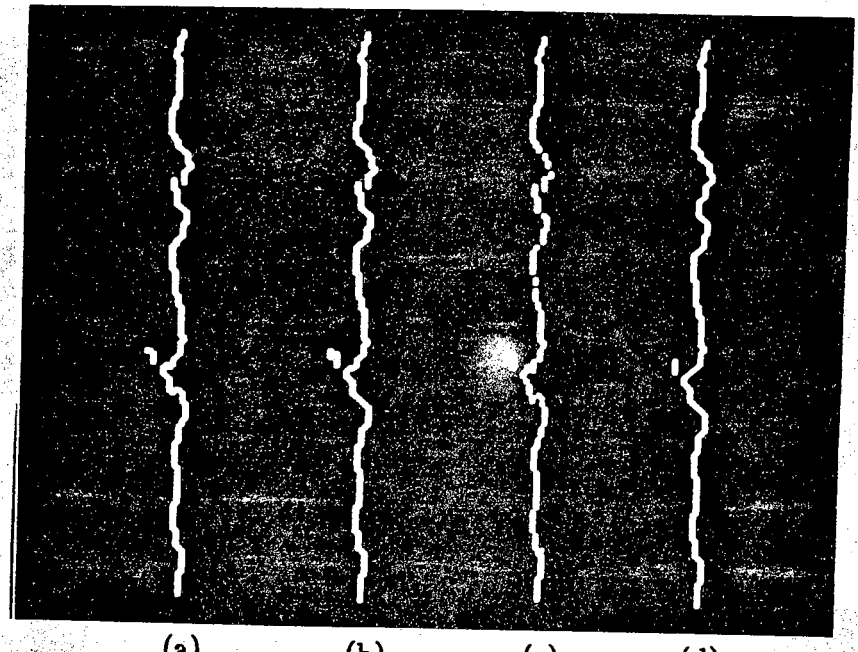


Figure 5.7 Effect of changing the weights for curvature and fragmentation. (a) w_c=0.2, w_f=2.0. (b) w_c=0.5, w_f=2.0. (c) w_c=0.75, w_f=0. (d) w_c=0.75, w_f=3.0.

w_d=2.0, w_e=1.0, w_f=2.0, w_n=5.01, and w_t=0.81. For the detected edge in Figure 5.7(b), the weight for curvature was altered so that w_c=0.5. The remaining weights were kept the same, except for w_n and w_t which were altered according to Proposition 4.2 to ensure that all edges remained thin. The resulting weight values were: w_c=0.5, w_d=2.0, w_e=1.0, w_f=2.0, w_n=5.01, and w_t=0.51. Notice that because of the increase in the weight of the curvature cost, the detected edge has a smoother boundary than in the previous case. This is particularly evident when comparing the portions of the edges slightly below the mid-section. For the edge in Figure 5.7(c), the cost for fragmentation was removed by setting $w_f = 0$; the weight values were: w_c=0.75, w_d=2.0, w_e=1.0, w_f=0.0, w_n=1.01, and w_t=0.01. Fragmentation is clearly visible in this case. In Figure 5.7(d), the cost for fragmentation was increased to 3.0. The weights were: w_c=0.75, w_d=2.0, w_e=1.0, w_f=3.0, w_n=7.01, and w_t=0.583. Notice that because of the increase in w_f, the fragmented edge in the upper region has been made continuous. In Figure 5.8, we show the cost minimization process using Simulated Annealing for the case of the detected edges shown in Figure 5.7(d). The plot was obtained by sampling the annealing process after every 10 iterations through the image. Assuming the the image has size $N \times N$, each iteration represents N^2 attempts in transitioning to new states based on the annealing algorithm.

In Figure 5.9, we show examples of the use of state space reduction (SSR) which has been described in Section 4.3.4. Edges were constrained to lie only in the bright regions. The regions were obtained by thresholding the dissimilarity values and dilating the image with square 3×3 and 5×5 structuring elements using mathematical morphology [59].

5.2.2 Rings Image

We show examples of the detected edges for the rings image shown in Figure 5.1. The image was corrupted with additive zero mean i.i.d. Gaussian noise with signal to noise ratio as defined in Equation (5.1). Figure 5.10 shows the detected edges for the noisy rings image with SNR=1.0. For the ∇G, $\nabla^2 G$ and SEL techniques, the standard deviation of the Gaussian function was 4.0, 4.5 and 3.0 respectively. For the facet model, CCF and ACF techniques, the image was pre-filtered using a Gaussian smoothing function with a standard deviation of 3.5. The ACF technique used state space reduction and the following set of weights: w_c=0.5, w_d=2.0, w_e=1.0, w_f=3.0, w_n=7.01, and w_t=0.833. A total of 200 iterations through the image was performed.

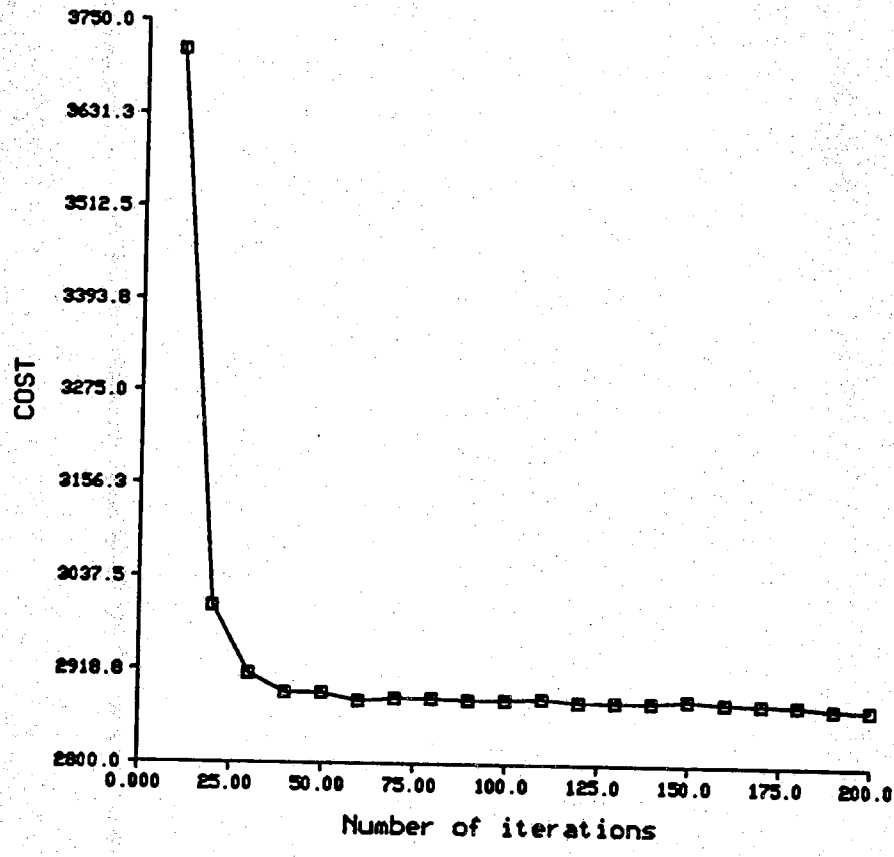


Figure 5.8 Cost minimization process for vertical step image using Simulated Annealing. Plot obtained by sampling annealing process at every 10 iterations through image.

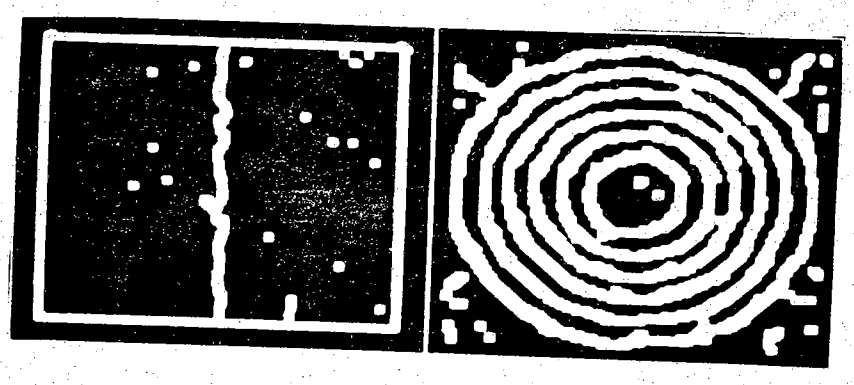


Figure 5.9 Examples of state space reduction. The bright regions were obtained by thresholding the enhanced images and performing the morphological operation "dilation" on each binary image. Edges were restricted to lie only in the bright regions. Left: State space reduction for noisy vertical step image. Right: State space reduction for noisy rings image (SNR=1.0).

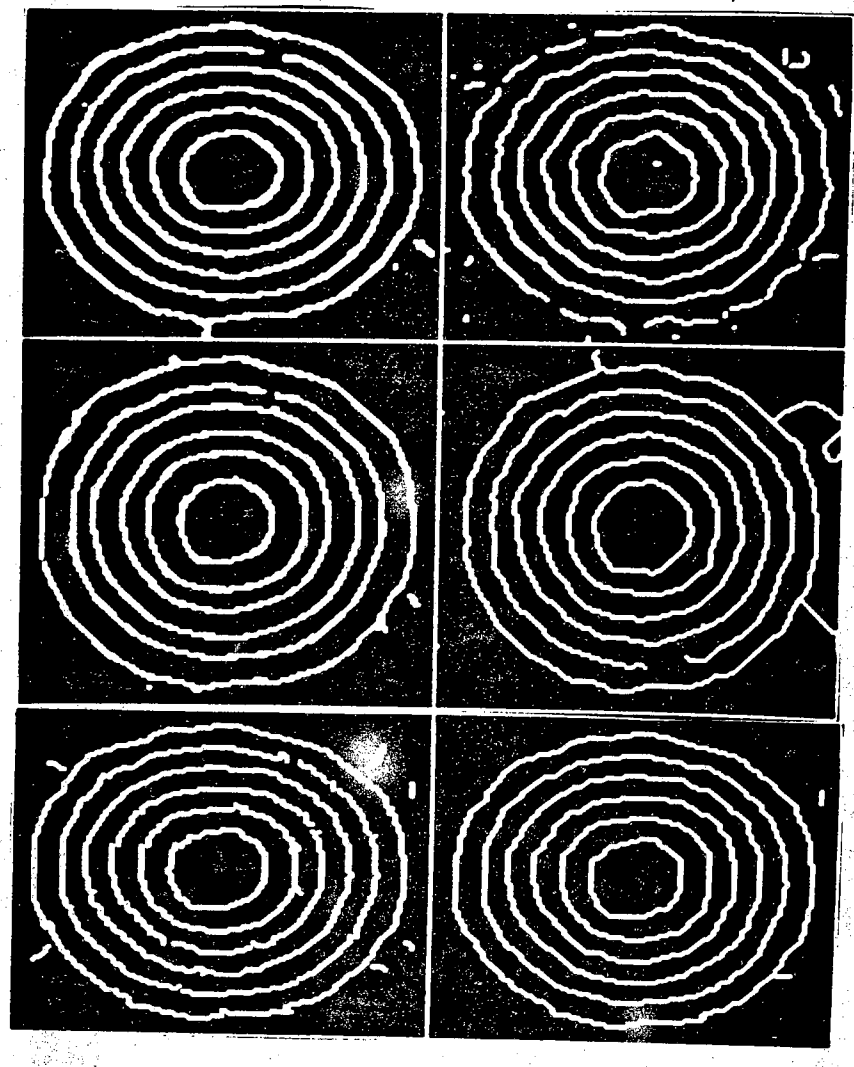


Figure 5.10 Comparison of edge detection performance using noisy rings image with SNR=1.0. Top left: ∇G. Top right: $\nabla^2 G$. Middle left: Facet model approach. Middle right: SEL. Bottom left: CCF. Bottom right: ACF.

Computation required 1.24 hours of CPU time on the VAX 11/780 minicomputer. A subjective evaluation of the different detection techniques shows that in terms of edge continuity, the ACF, the facet model, and the ∇G techniques produced the best results. In terms of edge thinness, the ACF, CCF and SEL techniques produced the best results. It is observed that the contour tracing nature of SEL produces some false boundaries. Figure 5.11 shows the edges detected for a slightly noiser image with SNR=0.574. For the ∇G, $\nabla^2 G$ and SEL techniques, the standard deviation of the Gaussian function was 4.0, 5.0 and 3.5 respectively. For the facet model, CCF and ACF techniques, the image was pre-filtered using a Gaussian smoothing function with a standard deviation of 4.0. The ACF technique used the same set of weights as in the previous case of the noisy image with SNR=1.0. The subjective evaluation of the detected edges is similar to the previous case, except that there is a slight increase in false boundaries.

5.2.3 Temperature Variation and Parallel Implementation

Simulated Annealing is a minimization algorithm that allows for uphill cost climbs while searching for the minimum cost states. The amount of "hill climbing" activity is controlled by the temperature T_k, where k denotes the kth iteration through the algorithm. If the temperature is set equal to zero, no hill climbing is allowed and the algorithm corresponds to a steepest descent search algorithm. This approach usually causes the algorithm to terminate in an undesirable local minimum that is of relatively high cost. A physical analogy of such an annealing process is the rapid cooling of a system, causing it to freeze in a meta-stable state. In Figure 5.12, we show an example of the use of rapid cooling in Simulated Annealing. The test image used was the rings image with SNR=1.0. The lower curve shows the cost minimization process using the logarithmic temperature schedule given in Equation (4.33). The upper curve shows the results for a temperature schedule which remains constant at a value of 0.01 throughout the annealing process. For both temperature schedules, the ACF technique used state space reduction and an identical set of weights: w_c=0.5, w_d=2.0, w_e=1.0, w_f=3.0, w_n=7.01, and w_t=0.833. In each case, 200 iterations through the image was performed. These parameters are exactly the same as those of the rings image example in Section 5.2.2. Two important observations can be made. First the process based on the logarithmic schedule converges to the set of low cost states much more quickly than that based on the constant temperature schedule. Second, the final state for the logarithmic schedule has a much lower cost than the final

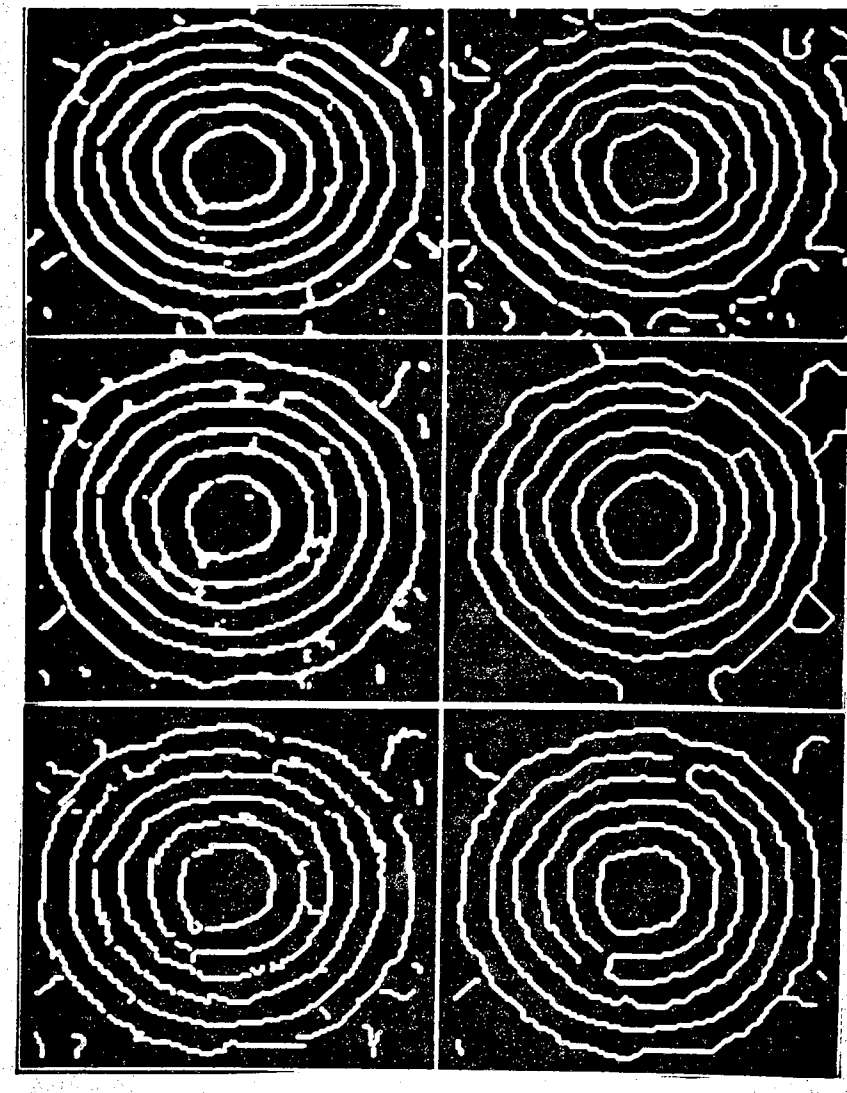


Figure 5.11 Comparison of edge detection performance using noisy rings image with SNR=0.574. Top left: ∇G. Top right: $\nabla^2 G$. Middle left: Facet model approach. Middle right: SEL. Bottom left: CCF. Bottom right: ACF.

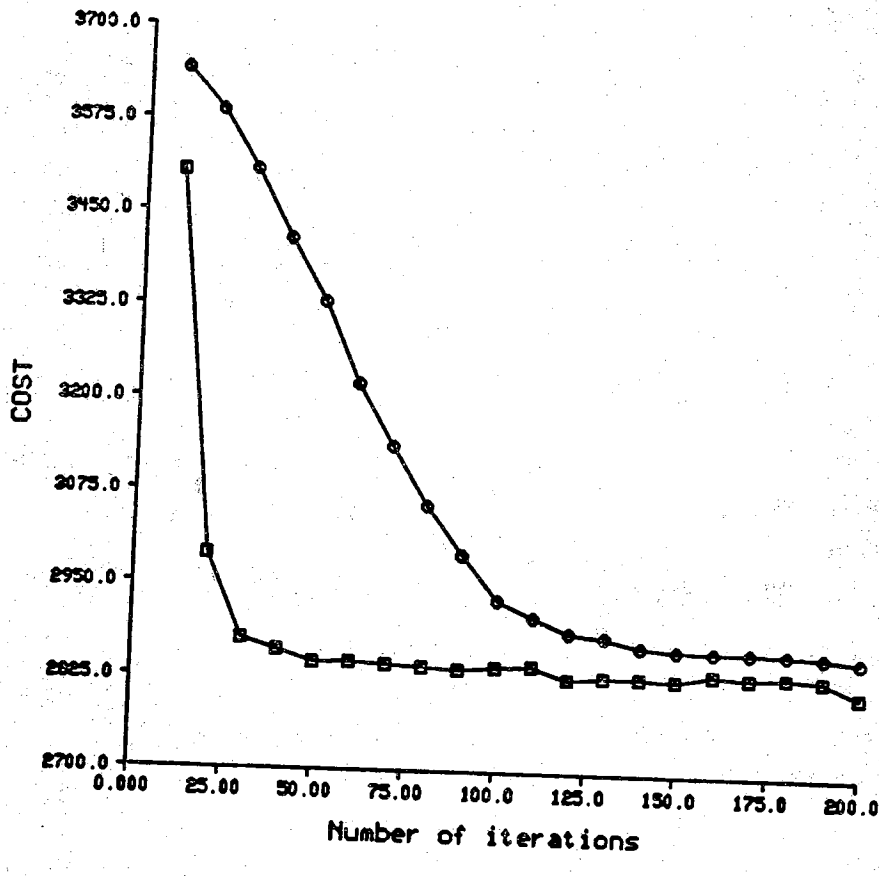


Figure 5.12 Rapid cooling in Simulated Annealing. Squares denote the data points for the annealing process which uses a logarithmic temperature decrement schedule. Circles denote the data points for the annealing process which uses a constant temperature schedule, with T_k=0.01 for all k.

state based on rapid cooling. In this case, the cost of the final states for the logarithmic and constant temperature schedules were 2815 and 2859 respectively.

Figure 5.13 shows the intermediate edge configurations of the annealing process which used the logarithmic temperature schedule. The test image was the noisy rings image (SNR=1.0), and 200 iterations through the image were made. No SSR was used. As mentioned in the introduction, in the last 2 iterations, the process was "frozen" by dropping the temperature to zero. Iterations 198 and 200 in the figure correspond to the states of the system just before and after freezing. It can be seen that after approximately 50 to 100 iterations, comparatively good edges were obtained. For most applications, it has been found that about 100 iterations are sufficient to produce edges which are thin and well localized. For the purpose of standardization and comparison, we used 200 iterations in all except one of the examples contained in this chapter.

In Section 4.3.1, we mentioned that in minimizing the ACF, there are a number of methods of generating next states. One method is based on selecting l by sequentially stepping through each pixel location in a raster scan manner. This method does not allow the annealing process to be implemented in parallel. However, in Section 4.3.3, we have shown there is a method of selecting l that would allow the Simulated Annealing algorithm to be implemented largely in parallel. Using the same test image and the exact same parameters for the ACF as in the example of Figure 5.10, we implemented the algorithm using the method that would allow for parallel execution. The results are shown in Figure 5.14. In this figure, we both the the cost curves for the annealing process that can be implemented only sequentially, and the process that can be implemented in parallel. The curves are very close to each other and intersect at a number of points. The tail ends of the curves are almost merged together implying that in the final iterations, both the processes arrived at states that have approximately the same cost values. The results indicate that in terms of cost minimization, both the methods gave approximately the same performance. When parallel processing is available, it is clearly more advantageous in terms of computation time to implement the algorithm that allows for parallel implementation. Figure 5.15 shows the detected edges for the noisy rings image (SNR=1.0) using the three different methods of implementing Simulated Annealing; the method of rapid cooling, sequential implementation, and parallel implementation.

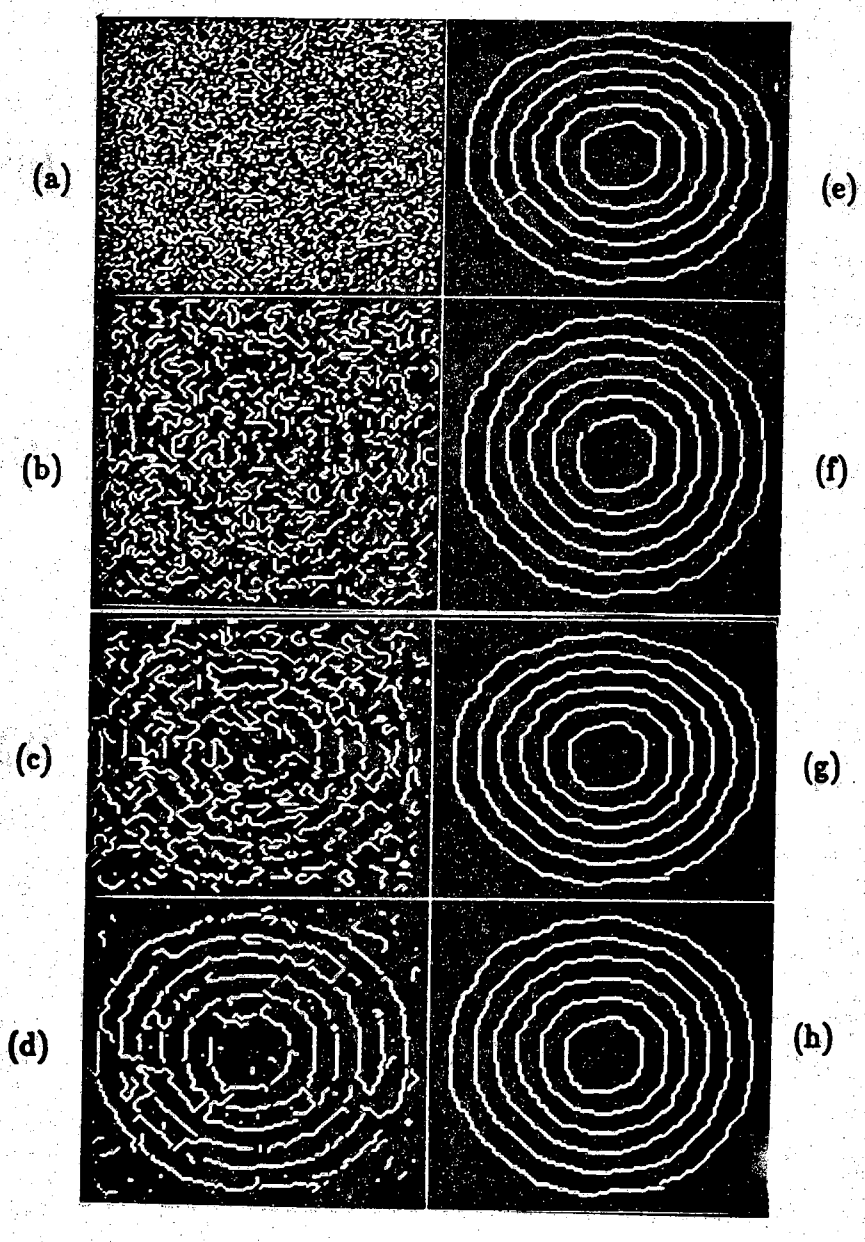
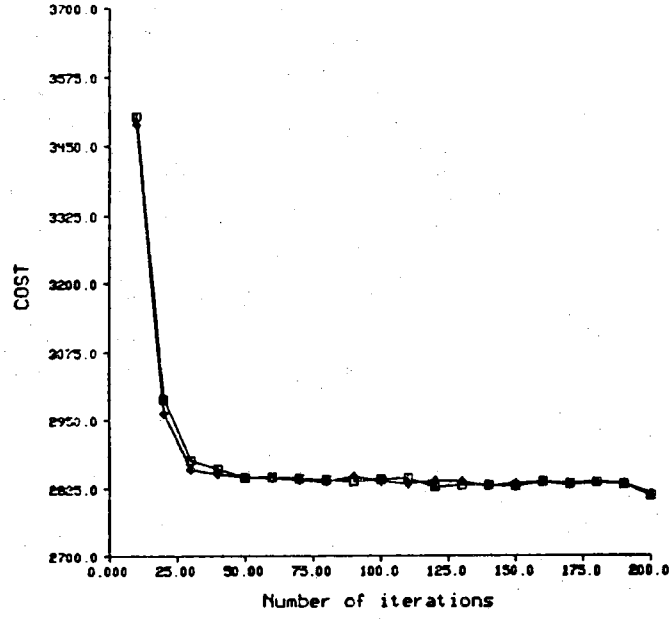


Figure 5.13 Intermediate edge configurations in annealing process.
(a) Iteration 1, Cost=16190. (b) Iteration 5, Cost=6954. (c) Iteration 10, Cost=5217. (d) Iteration 20, Cost=3714. (e) Iteration 50, Cost=2864. (f) Iteration 100, Cost=2833. (g) Iteration 198, Cost=2827. (h) Iteration 200, Cost=2810.

(a)



(b)

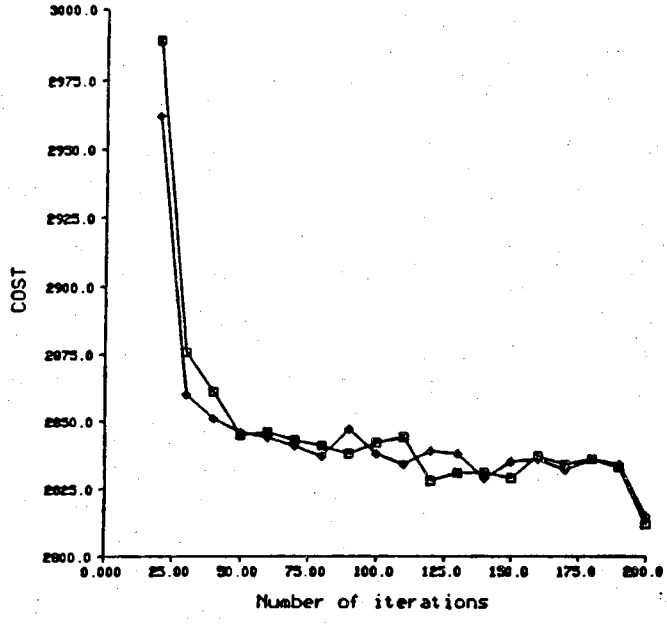


Figure 5.14 Comparison of parallel and sequential implementations. Squares denote the data points for the annealing process that can only be implemented sequentially. Diamonds denote the data points for the annealing process that can be implemented in parallel. (a) Plot of cost vs the number of iterations through image. (b) Same plot on expanded scale.

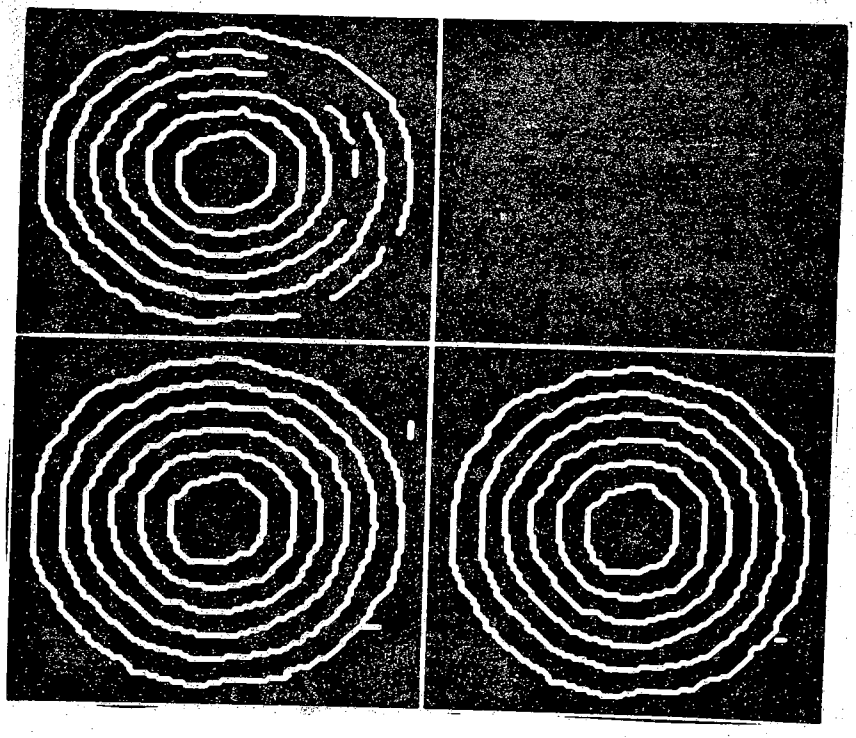


Figure 5.15 Edges obtained using three different methods of implementing Simulated Annealing. Test image used was the noisy rings image with SNR=1.0. Top : Edges detected by rapid cooling (see Figure 5.12). Bottom left: Edges detected using annealing process that can only be implemented sequentially (see Figure 5.14). Bottom right: Edges detected using annealing process that can be implemented in parallel (see Figure 5.14).

5.3 Experiments with Real Images

In this section we show two examples of the detected edges for general scenes. Both images were of size 256×256.

House image

In this example, we show the detected edges for both the original and noisy image of a general outdoor scene. The house image is shown in Figure 5.16. The image was corrupted with additive zero-mean i.i.d. Gaussian noise of standard deviation 35. In each case of the detected edges, the detection parameters were selected so that the different techniques produced approximately the same number of edge points, and the edges were visibly similar. The choice of parameters is quite subjective as it is difficult to quantify edge quality for general scenes. After determining the necessary parameters for the noiseless image, the same parameters were then used to detect edges in the noisy image. In all cases except one, the standard deviation of the Gaussian function was set at 2.0. The $\nabla^2 G$ operator used a standard deviation of 2.5. For the ACF technique, the weights of the cost factors were: w_c=0.75, w_d=2.0, w_e=1.0, w_f=3.0, w_n=7.01, and w_t=0.583. No state space reduction was used and 100 iterations through the image were made. The detected edges are shown in Figure 5.17.

Airport image

In Figure 5.18 we show the detected edges for an airport image using the ACF technique. The weights of the cost factors were: w_c=0.5, w_d=2.0, w_e=1.0, w_f=3.0, w_n=7.01, and w_t=0.833. A total of 200 iterations through the image were made.

5.4 Other Dissimilarity Measures

In the previous examples, we have detected edges using dissimilarity functions f_c and f_a which measure the difference of gray level averages of the regions on either sides of the edge. In this section, we will show examples of detected edges using other forms of dissimilarity measures. In the first example, we show how a priori information can be incorporated into the measure so as to detect specific kinds of edges. In the second example, we show how the measure can be defined to find texture edges based on second order statistical properties of the regions of interest.

Figure 5.16 House image. Top: Original house image. Bottom: House image corrupted with additive zero-mean i.i.d. Gaussian noise of standard deviation 35.

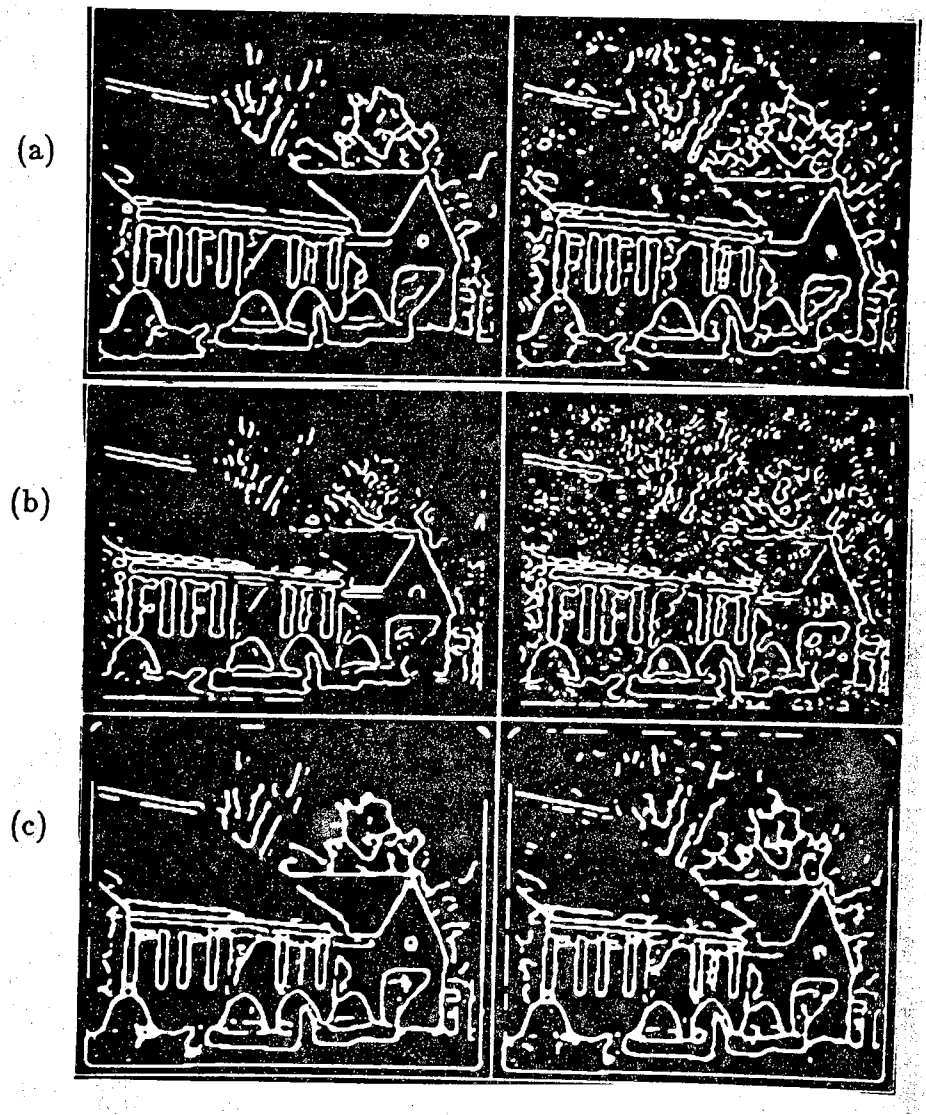


Figure 5.17 Comparison of various detection techniques on house image. In each case, the figure on the left shows the detected edges for the noiseless house image while the figure on the right shows the edges for the noisy image. (a) ∇G. (b) $\nabla^2 G$. (c) Facet model. (d) SEL. (e) CCF. (f) ACF.

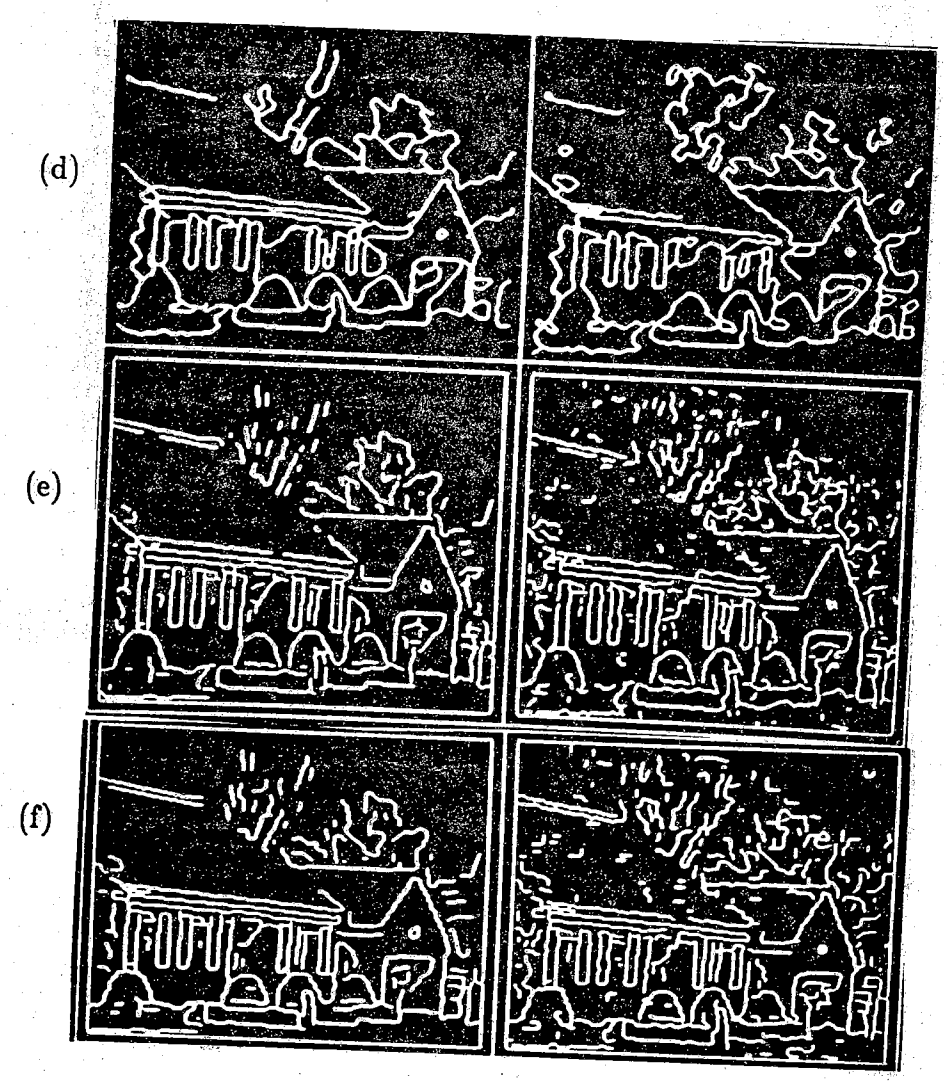


Figure 5.17, continued

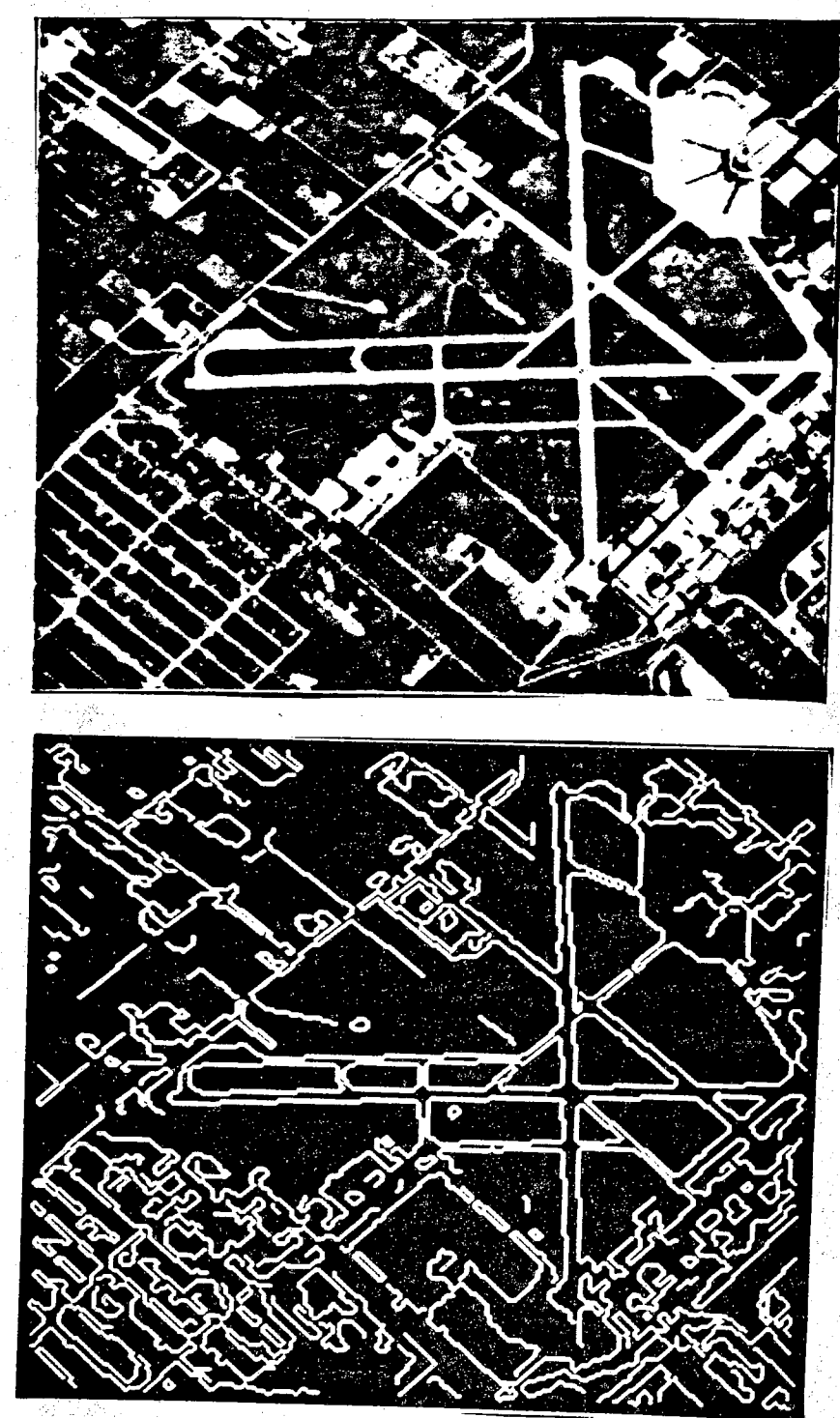


Figure 5.18 Airport image. Intensity image and edges detected using ACF approach.

Airplanes image

Figure 5.19 shows an image containing ten airplanes. The important features of interest in this image are the airplanes, the two large buildings on the left part of the image, and portions of the tarmac. The edges detected using the CCF, ACF, the facet model, and the ∇G operator techniques are also shown in the figure. For the CCF and ACF techniques, the image was smoothed by a Gaussian function of standard deviation 1.0 prior to edge detection. The ∇G technique also used a value of 1.0 for the standard deviation. In each case, the thresholds and relevant parameters were chosen so as to recover as much of the boundaries of the airplanes and the large buildings as possible, without introducing an excessive number of false edges. For all four techniques, it was found that selecting a threshold low enough to recover the boundaries of the large buildings resulted in a high degree of false edges being detected. Hence, by thresholding alone, it is not possible to obtain a good set of edges representing the features of interest.

It can be seen that the important features of interest in the image generally have a lighter shade than the background, corresponding to higher image intensity values. We use this a priori information by incorporating it into the dissimilarity measures f_c and f_a. We specify that regions have high dissimilarity when two conditions hold: (1) the average intensity values are significantly different, and (2) the average intensity value for one of the regions is sufficiently high. This is different from the previous definition of dissimilarity where we do not include the latter condition. For the comparative cost function, this new definition of dissimilarity is mathematically captured simply by specifying the dissimilarity measure to be $f_c(R1,R2)=\tilde{m}(d,\beta)$. The function $\tilde{m}$ is defined as:

$$\tilde{m}(d,\beta) = m(d) \cdot g(\beta) ,$$

where $m(d)$ is as defined in Equation (2.6), β is the larger of the average intensity values of the two regions, and g is the piecewise linear function shown in Figure 5.20. For the absolute cost function, this definition is captured by specifying the measure to be

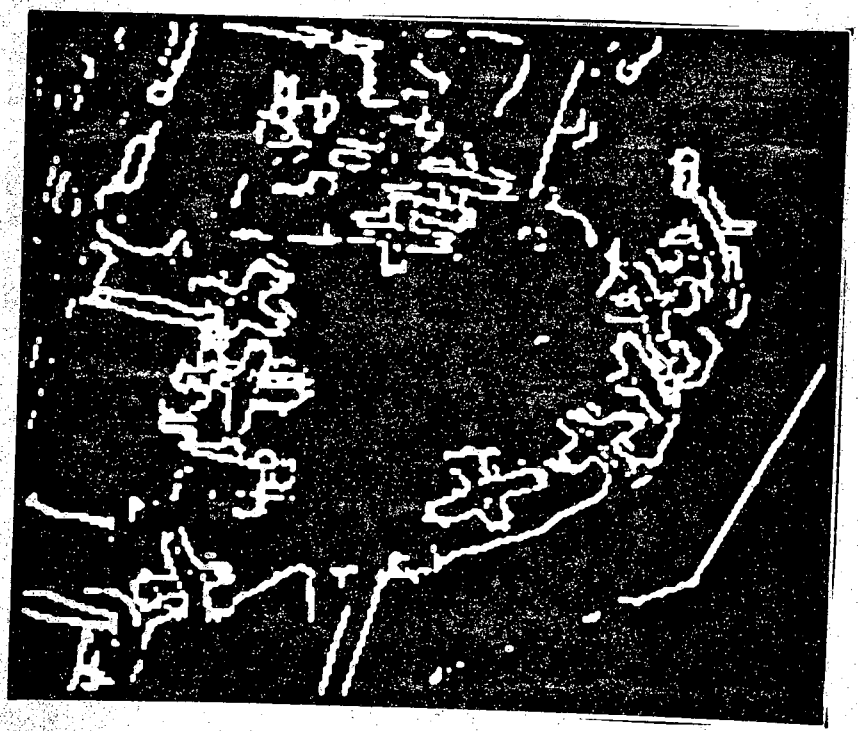
$$f_a(R1,R2) = d \cdot g(\beta) ,$$

where d is as given in Equation (2.4), and $g(\beta)$ is the same function defined above. The weights of the ACF were: w_c=0.2, w_d=2.0, w_e=1.0, w_f=2.0, w_n=5.01, and w_t=0.81. The annealing process made 200 iterations through the image. Using these new definitions of region dissimilarity, the detected edges

(a)

Figure 5.19 Airplanes image. The important features of interest are the airplanes, the two large buildings on the left, and portions of the tarmac. (a) Intensity image. (b) Facet model. (c) ∇G. (d) CCF. (e) ACF.

(b)



(c)

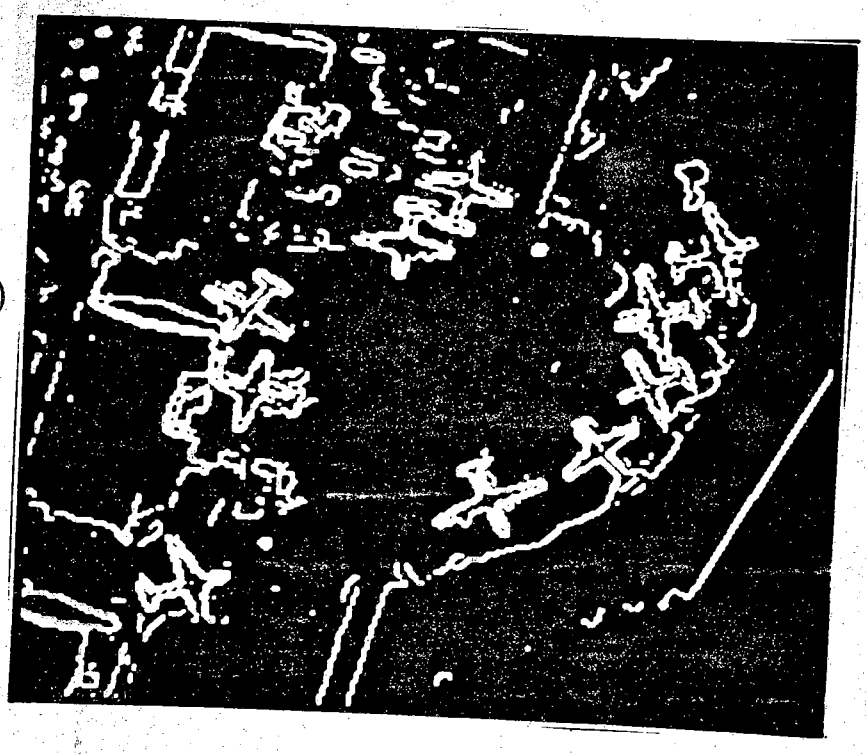
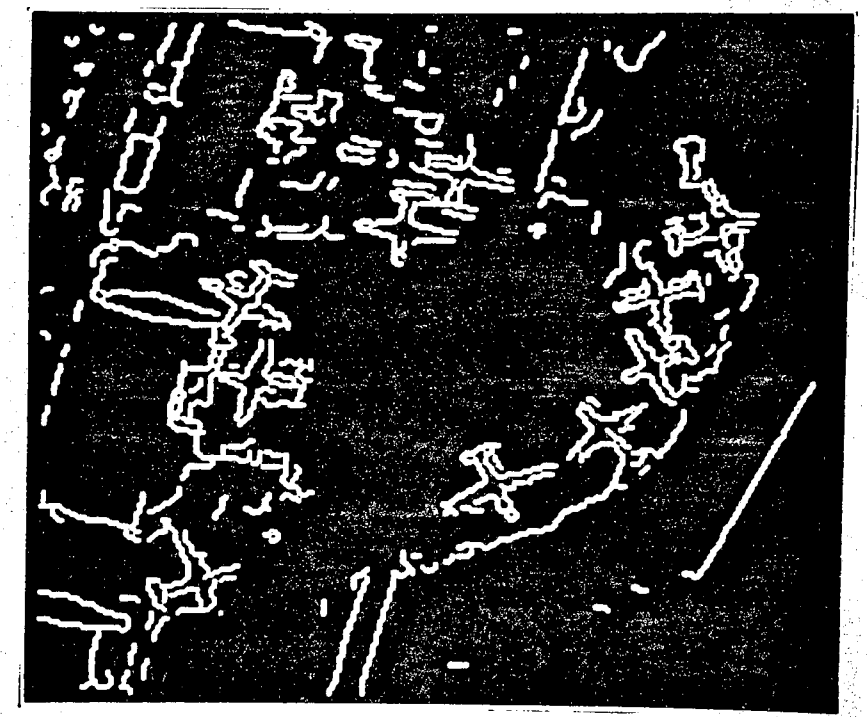


Figure 5.19, continued

(d)



(e)

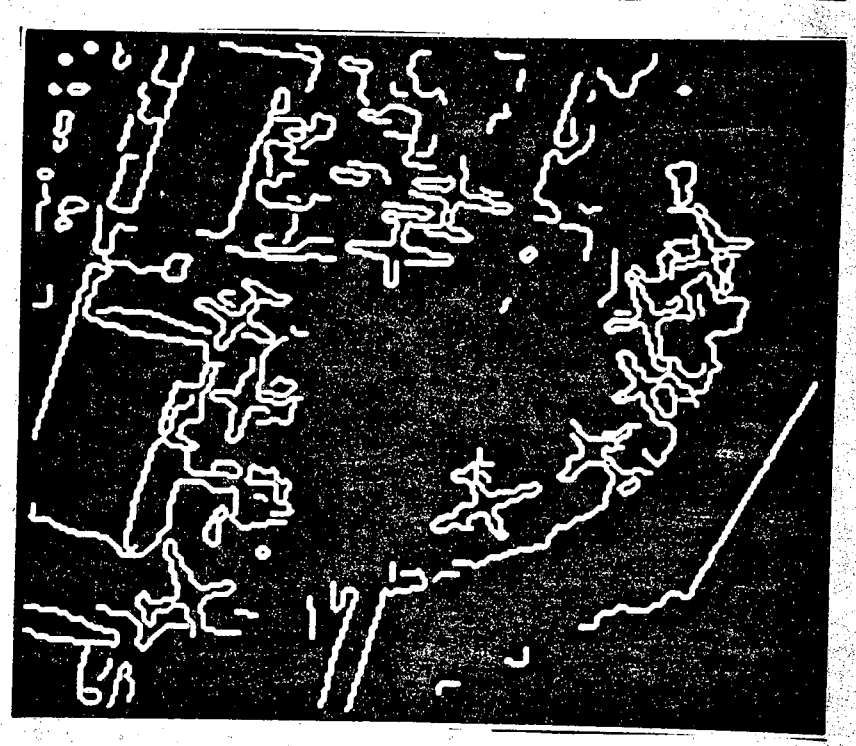


Figure 5.19, continued

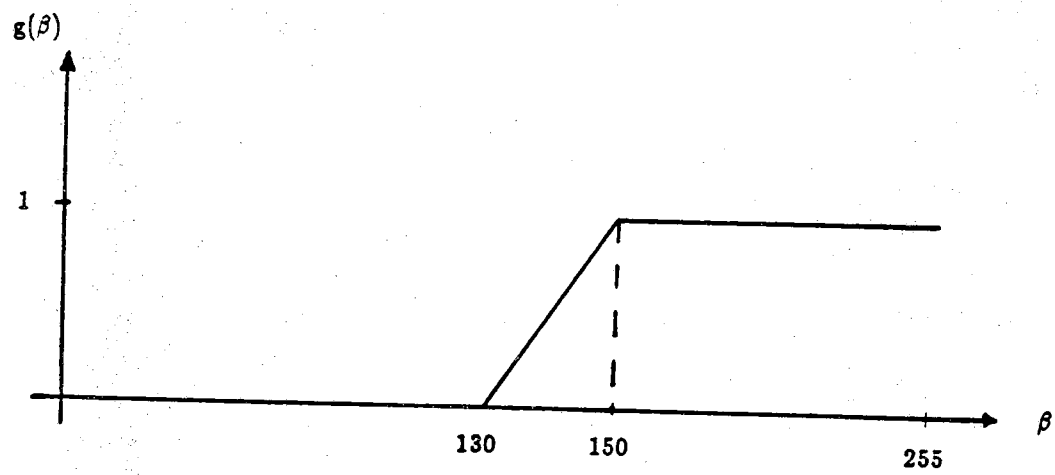


Figure 5.20 Piecewise linear function $g(\beta)$ used in the definition of $\tilde{m}(d,\beta)$.

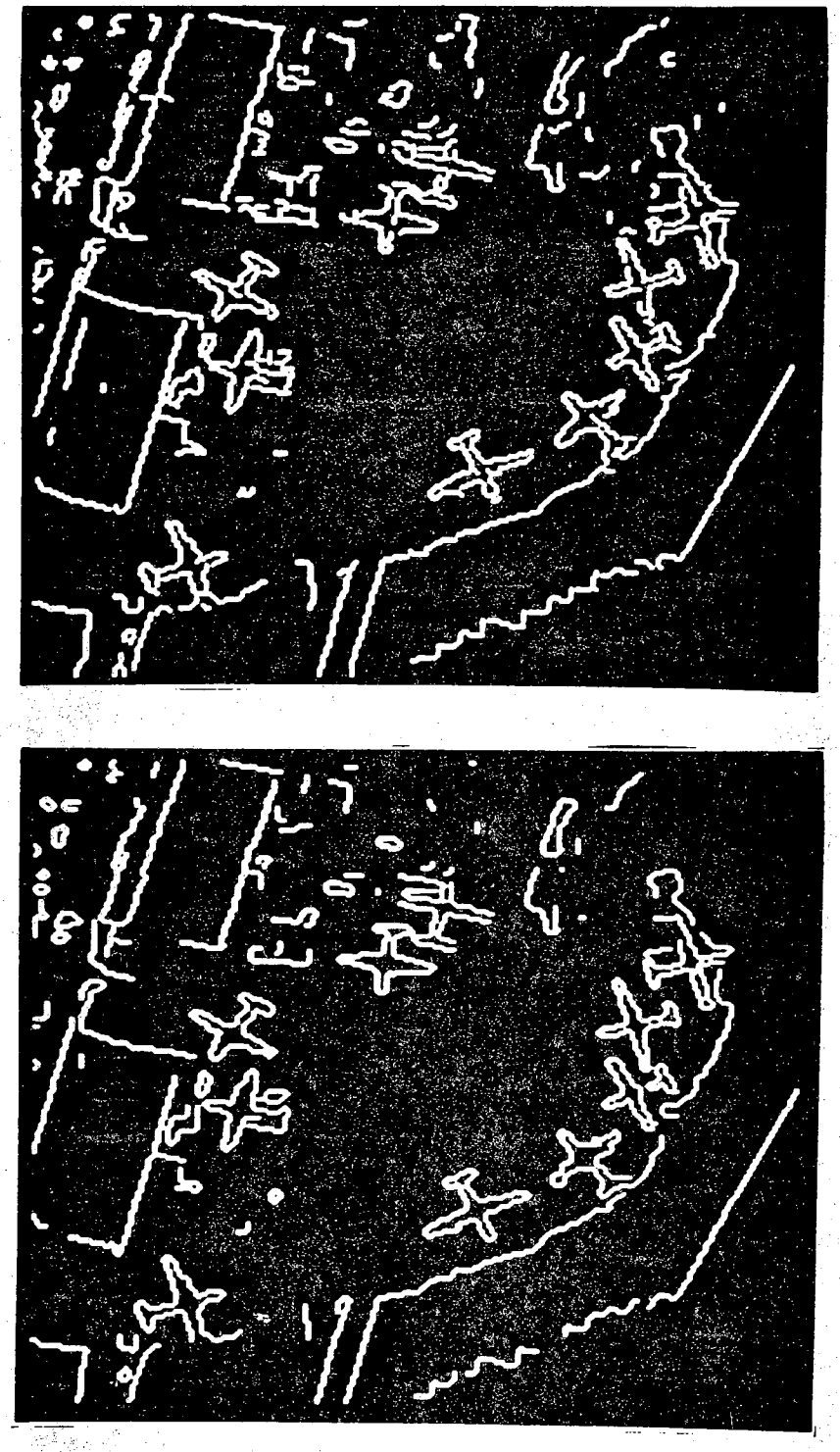


Figure 5.21 Edges of airplanes image detected using a priori information about the features of interest. Top: CCF approach. Bottom: ACF approach.

are shown in Figure 5.21. Notice that the edges of the large buildings, the airplanes, and the boundary region of the tarmac on the lower right portion of the image are clearly visible.

Box texture image

Figure 5.22 shows an image of size 128×128 containing two texture regions. The average intensity value was approximately the same throughout the image. However, the variance of the intensity values within the box region was higher than the variance of the background. This image was constructed by adding zero-mean i.i.d. Gaussian random noise to an image of constant gray level equal to 128. Within a 64×64 box region, the noise standard deviation was 30; outside of the box region, the noise standard deviation was 10. Since the boundary of the box does not correspond to a step or a ramp, it is not possible to use the ∇G or facet model methods to detect the edges of the box.

The cost function approach can be used to detect the boundary of such texture regions by the use of an appropriate measure for region dissimilarity. In this example, a suitable measure of dissimilarity is the difference of the standard deviation of the pixels in the regions of interest. We show an example of this using the absolute cost function. Let m1 and m2 be the gray level averages of the pixels in R1 and R2 respectively. The dissimilarity measure is defined as:

$$f_a(R1,R2) = | \sigma_1 - \sigma_2 | ,$$

where

$$\sigma_1 = \left\{ \frac{1}{|R1|} \sum_{(i,j) \in R1} [g(i,j) - m1]^2 \right\}^{\frac{1}{2}} ,$$

and σ_2 is similarly defined. Figure 5.22 shows the detected edges using the following weight values: w_c=0.75, w_d=2.0, w_e=1.0, w_f=4.0, w_n=9.01, and w_t=0.917. The annealing process made 200 iterations through the image.

5.5 Computation Time and Final Costs

In this section, we summarize some of the results of using the ACF approach in terms of the computation time required and the cost of the final edge configurations achieved by the annealing process. In Table 5.2, we tabulate the computation time required to detect the edges of the different test

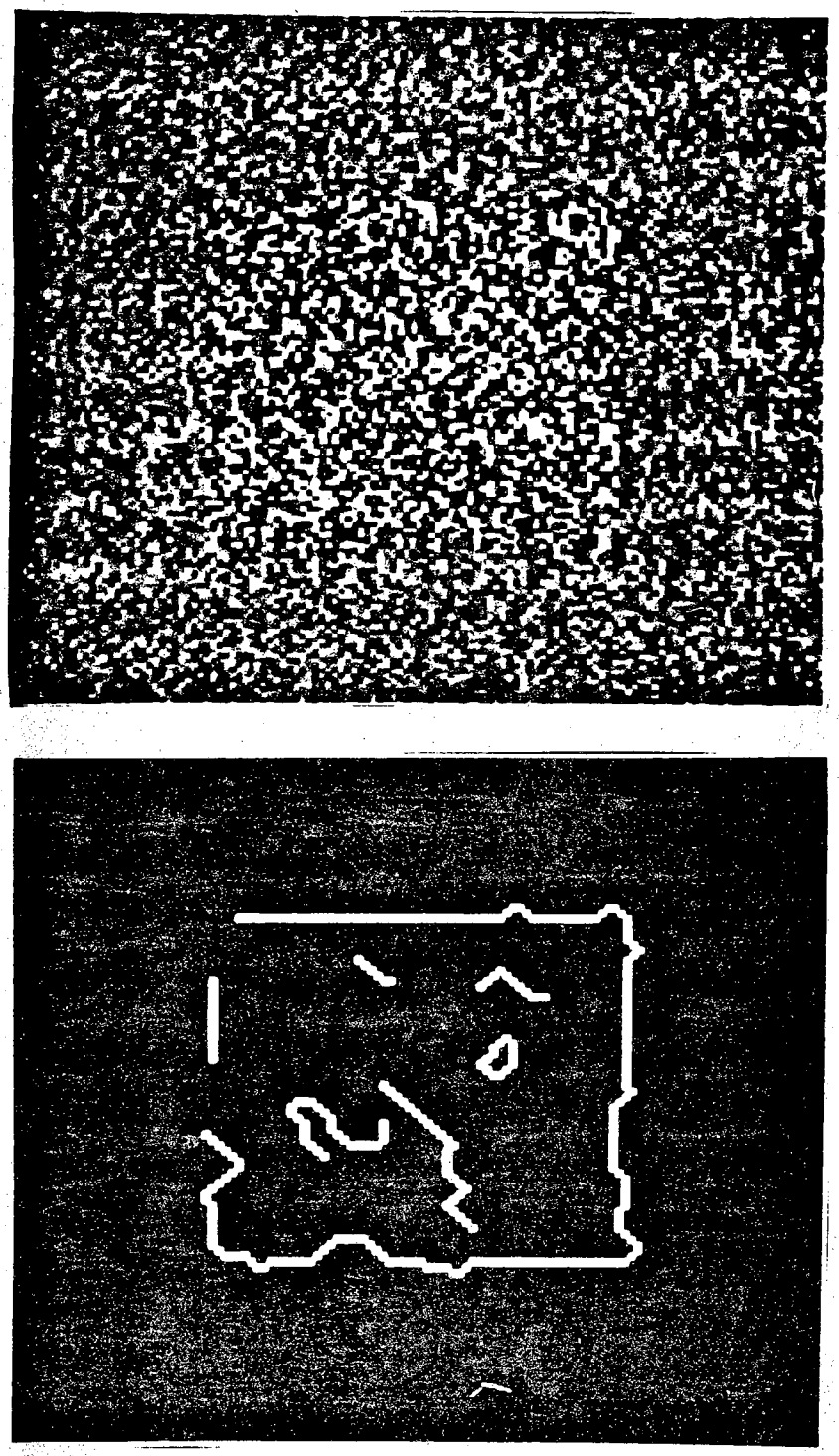


Figure 5.22 Texture edge detection. Box image and detected edges using ACF approach.

Table 5.2. Computation time for minimizing the ACF using Simulated Annealing.

Image	Size	SSR	Iterations	CPU time (hr)
Vertical step	256×256		200	10.09
Vertical step	256×256	Yes	200	1.28
Rings, SNR=1.0	128×128		200	2.70
Rings, SNR=1.0	128×128	Yes	200	1.24
House	256×256		100	5.32
Airport	256×256	Yes	200	7.77
Airplanes	256×256	Yes	200	3.59
Texture box	128×128		200	2.42

images. The cost minimization process using Simulated Annealing was implemented by sequential processing on the VAX 11/780. For purposes of comparison and standardization, the annealing process was implemented using 200 iterations through each image, except for the house image. Typically, for general images, about 100 iterations is sufficient to bring the annealing process to a suitably low cost state. Table 5.2 shows a comparison of the computation time required for each image with and without the use of state space reduction (SSR). The results indicate that, depending on the scene content, computation time for the annealing process can be reduced by a factor of about 1 to 8 times through the use of SSR. Generally, images that have smaller number of edges achieve greater reduction in computation time.

Table 5.3 shows a comparison of the cost of the final states for different cases of the annealing process. Two important observations can be made. First, the use of SSR results in edge configurations that have approximately the same cost as those configurations produced without it. Second, the annealing algorithm which can be executed in parallel produces edge configurations which have approximately the same cost as those configurations produced by the algorithm which can be implemented only sequentially. Based on these observations, we deduce that the most efficient method of producing edge configurations of low cost is to use SSR and parallel implementation.

5.6 Use of 5 Cost Factors

In all the previous examples of the absolute cost function approach, we used 6 cost factors in the definition of the cost function. The cost factor C_n described in Section 4.3.2.1 was included to constrain all edges to be either isolated paths or cycles; multiple edge segments linked at a single point were disallowed. The main reason for including this factor was to enable us to derive a tight estimate of the upper bound on the parameter d^*, as given in Equation (4.32).

In this section, we show the experimental results of minimizing a cost function that does not contain the cost factor C_n. That is, the cost function is a a weighted sum of only 5 cost factors: C_c, C_d, C_e, C_f and C_t. Figures 5.23 through 5.29 show the detected edges for the previous test images using 5 cost factors for the ACF. The cost minimization annealing process is also shown. In each case, all the parameters except w_t were chosen to be the same as those of the corresponding previous examples which used 6 cost factors. To avoid thick edges, the weight w_t was chosen based on Proposition 3.15. We used the same

Table 5.3. Cost of the final state in the annealing process.

Image	SSR	Parallel	Iterations	Cost
Rings, SNR=1.0			200	2810
Rings, SNR=1.0	Yes		200	2812
Rings, SNR=1.0	Yes	Yes	200	2815
Rings, SNR=1.0 (Rapid cooling)	Yes		200	2859
Rings, SNR=0.574			200	3161
Rings, SNR=0.574	Yes		200	3160
Rings, SNR=0.574	Yes	Yes	200	3161
Vertical step			200	2876
Vertical step	Yes		200	2873
Vertical step	Yes	Yes	200	2878
House			100	9253
House		Yes	100	9244
Airport	Yes		200	21771
Airport	Yes	Yes	200	21765
Airplanes	Yes		200	7572
Airplanes	Yes	Yes	200	7567

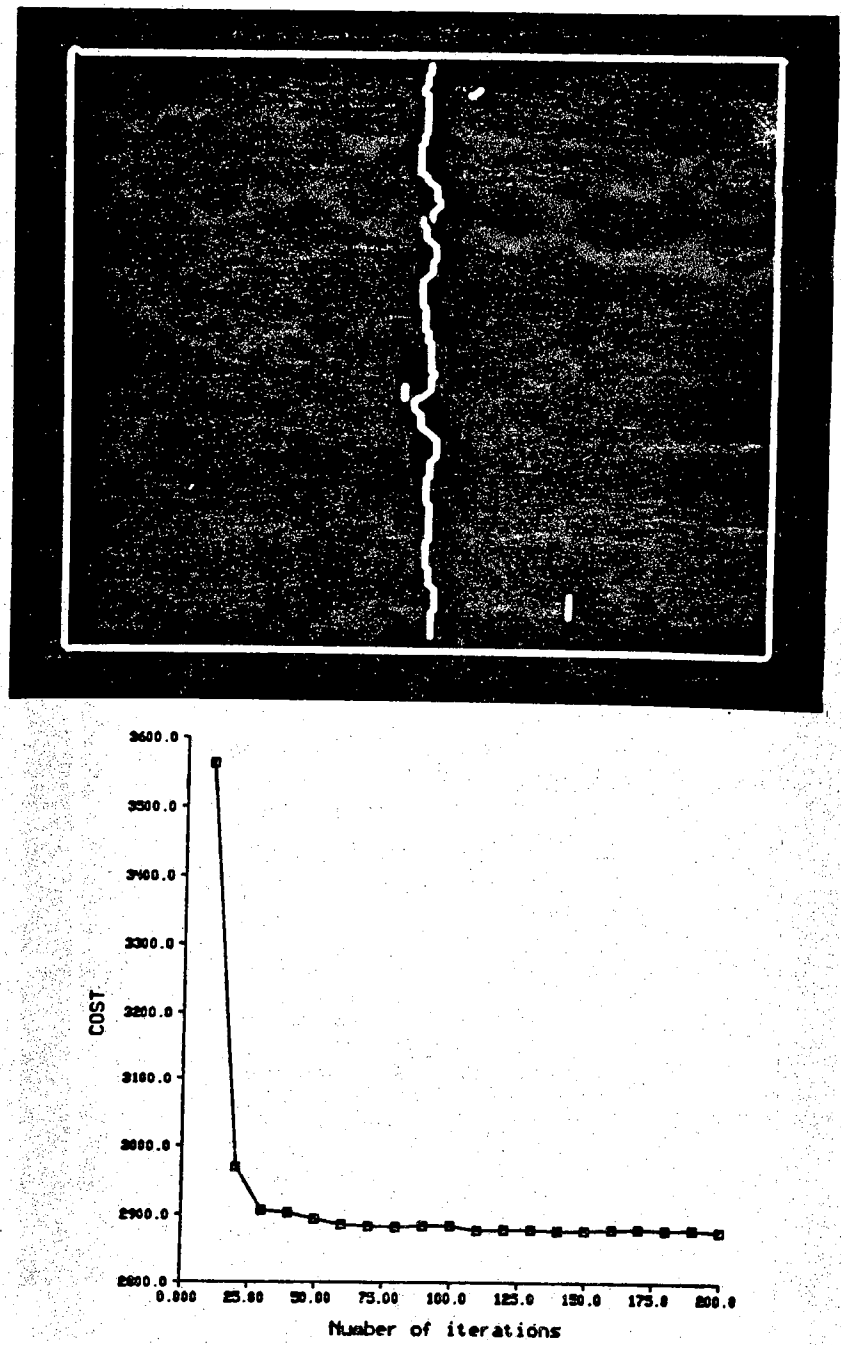
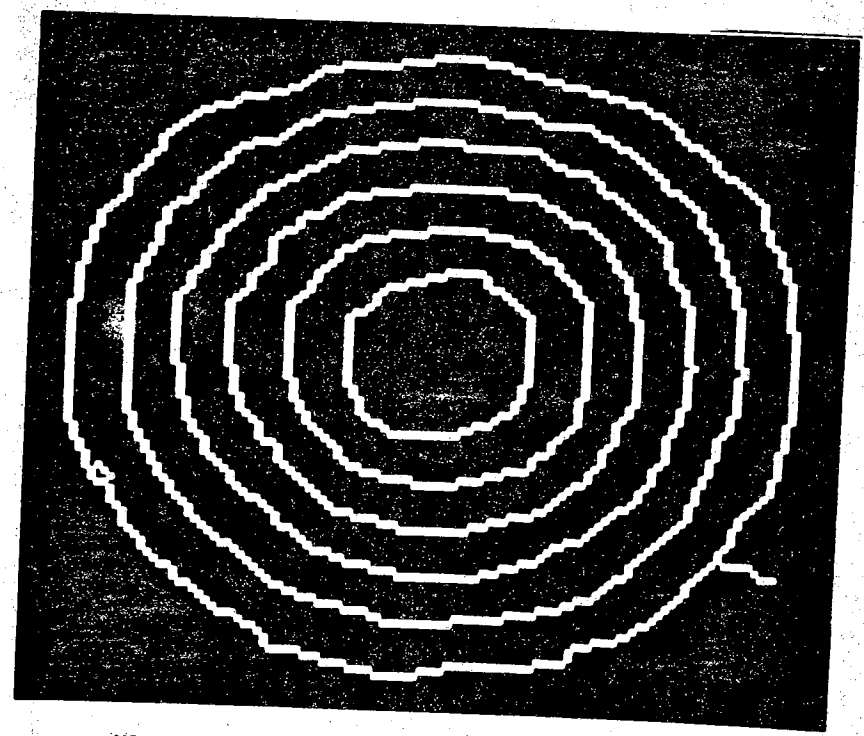


Figure 5.23 Detected edges and annealing process of vertical step image using 5 cost factors of the ACF. The weights were: w_c=0.75, w_d=2.0, w_e=1.0, w_f=3.0, and w_t=6.25. (Figure 5.4 shows the results using 6 cost factors).



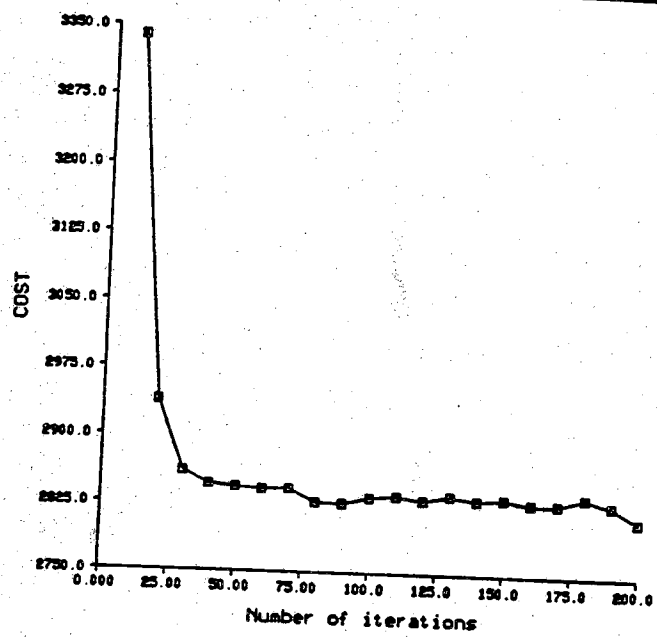


Figure 5.24 Detected edges and annealing process of rings image (SNR=1.0) using 5 cost factors of the ACF. The weights were: w_c=0.5, w_d=2.0, w_e=1.0, w_f=3.0, and w_t=6.5. (Figure 5.10 shows the results using 6 cost factors).

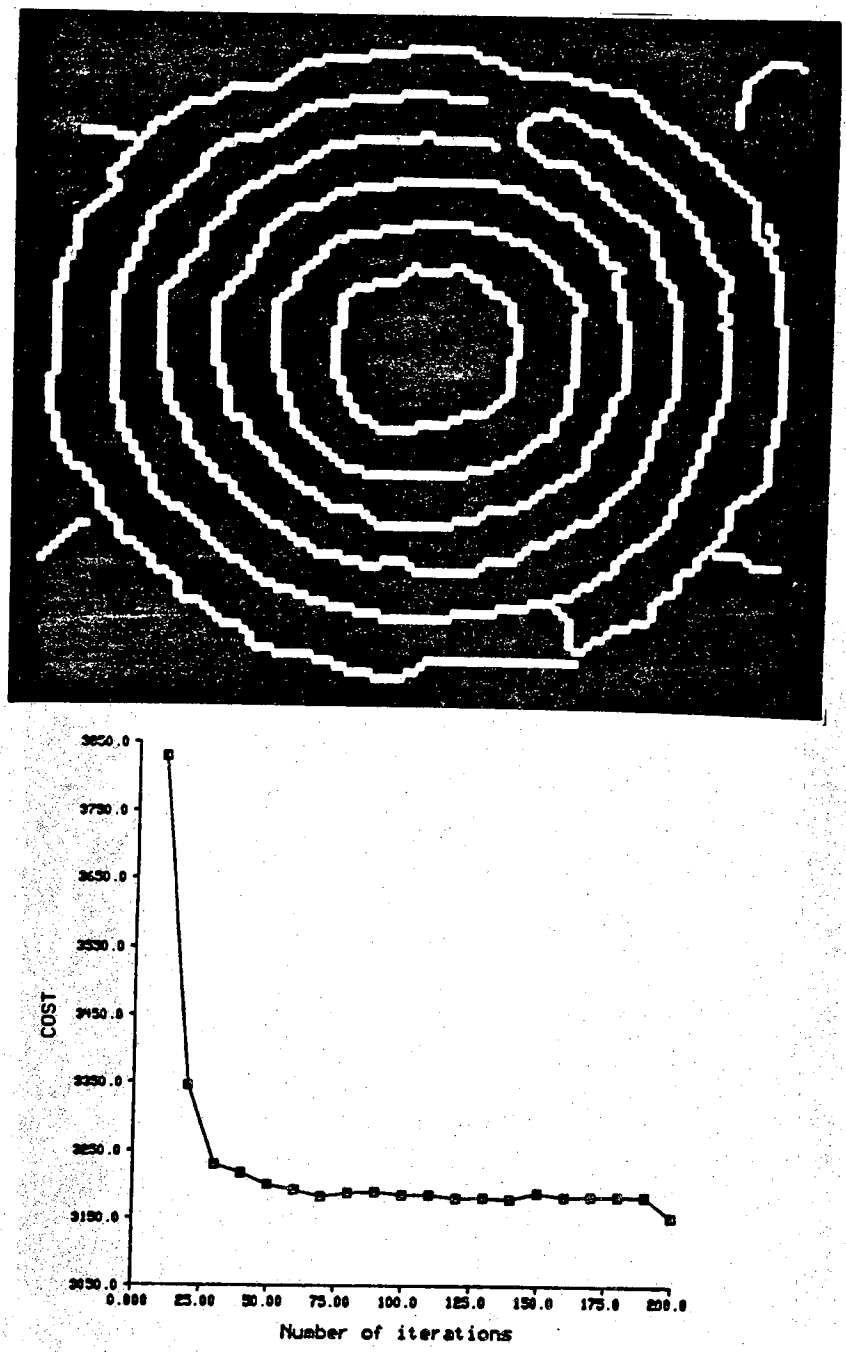


Figure 5.25 Detected edges and annealing process of rings image (SNR=0.574) using 5 cost factors of the ACF. The weights were: w_c=0.5, w_d=2.0, w_e=1.0, w_f=3.0, and w_t=6.5. (Figure 5.11 shows the results using 6 cost factors).

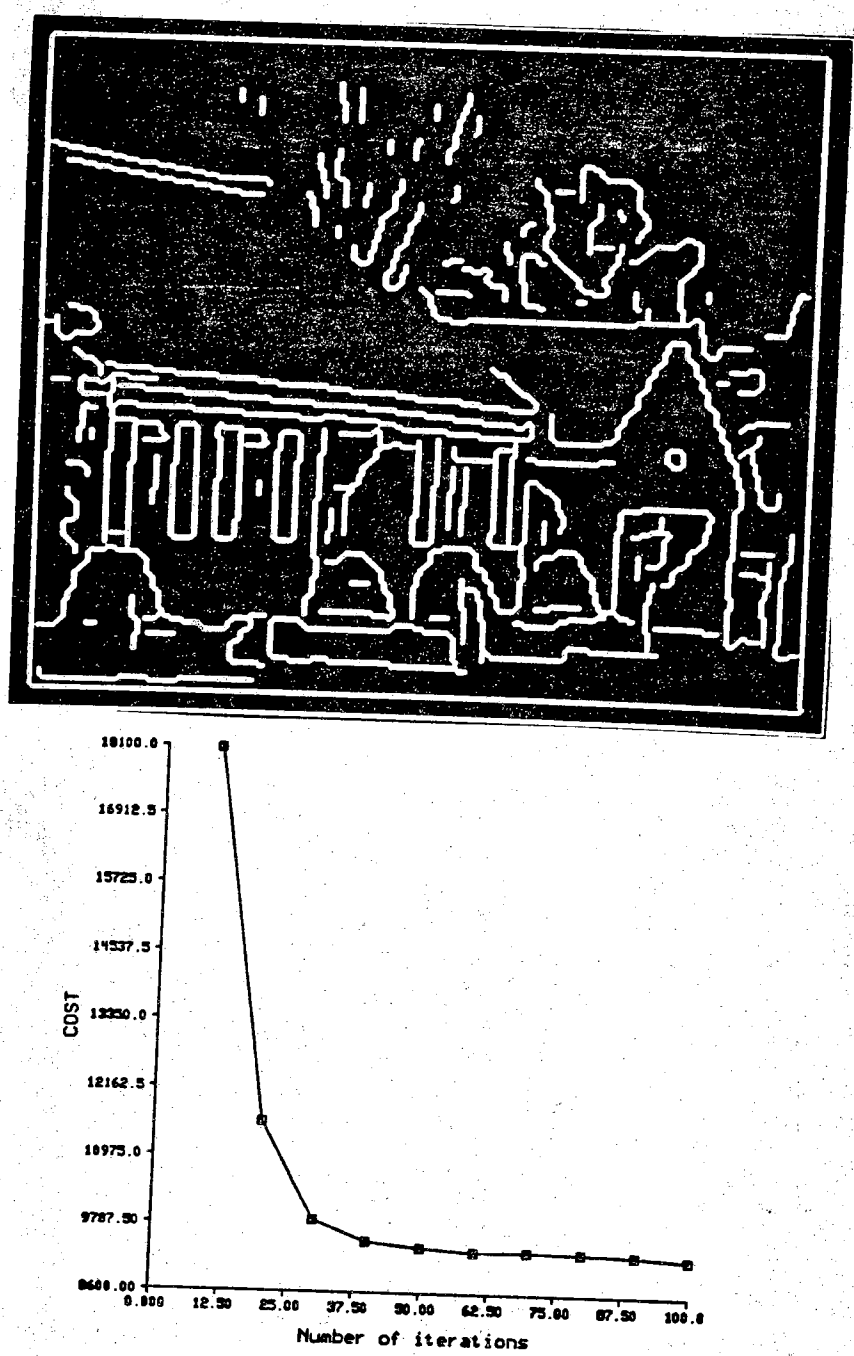
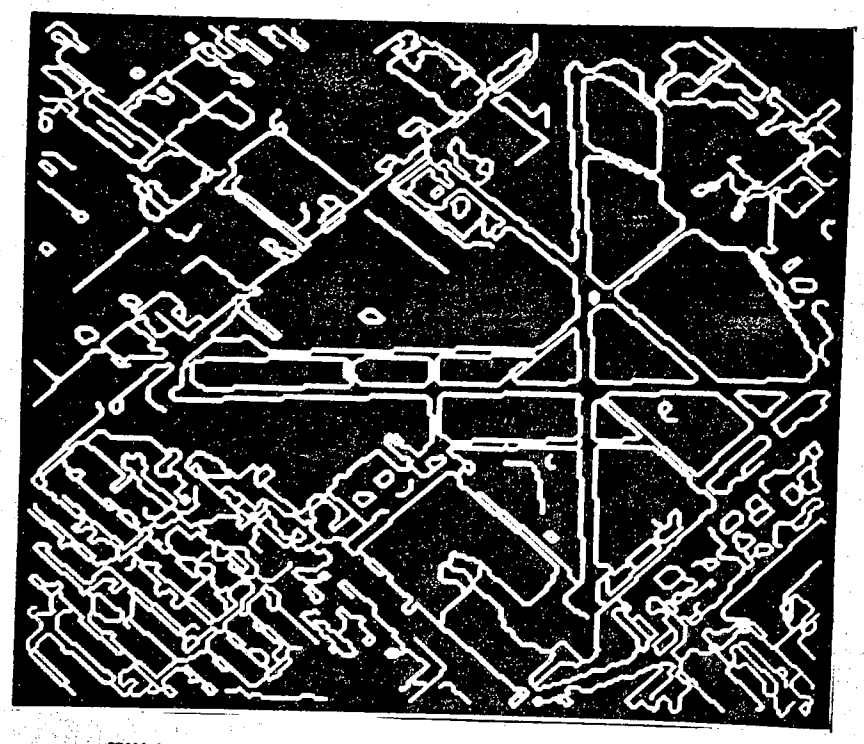


Figure 5.26 Detected edges and annealing process of house image using 5 cost factors of the ACF. The weights were: w_c=0.75, w_d=2.0, w_e=1.0, w_f=3.0, and w_t=6.25. (Figure 5.17 shows the results using 6 cost factors).



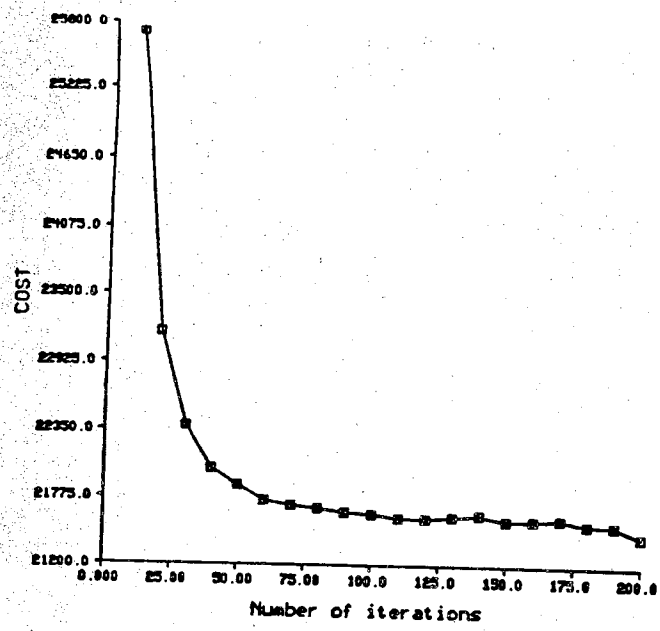
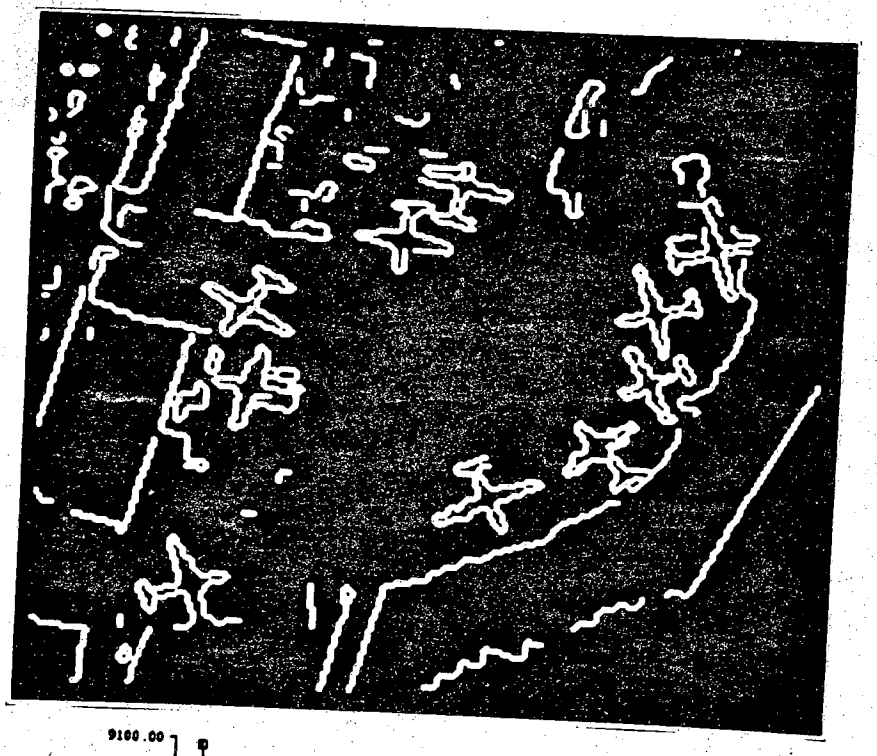


Figure 5.27 Detected edges and annealing process of airport image using 5 cost factors of the ACF. The weights were: w_c=0.5, w_d=2.0, w_e=1.0, w_f=3.0, and w_t=6.5. (Figure 5.18 shows the results using 6 cost factors).



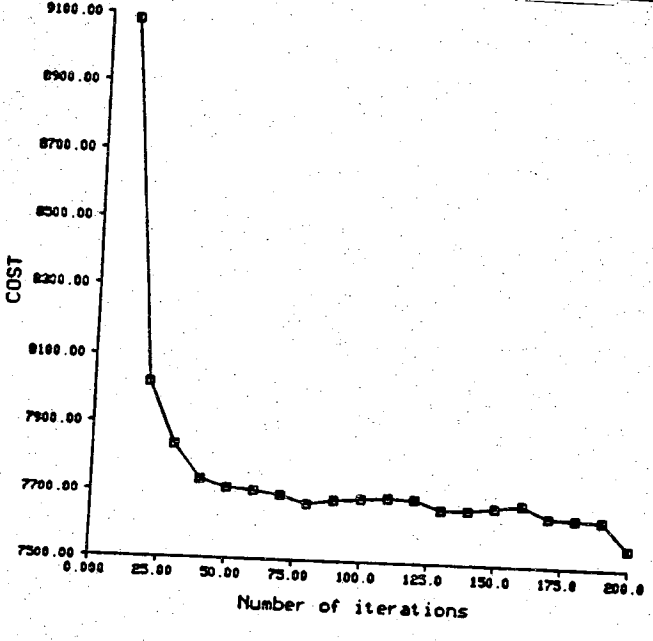


Figure 5.28 Detected edges and annealing process of airplanes image using 5 cost factors of the ACF. The weights were: w_c=0.2, w_d=2.0, w_e=1.0, w_f=2.0, and w_t=4.81. (Figure 5.21 shows the results using 6 cost factors).

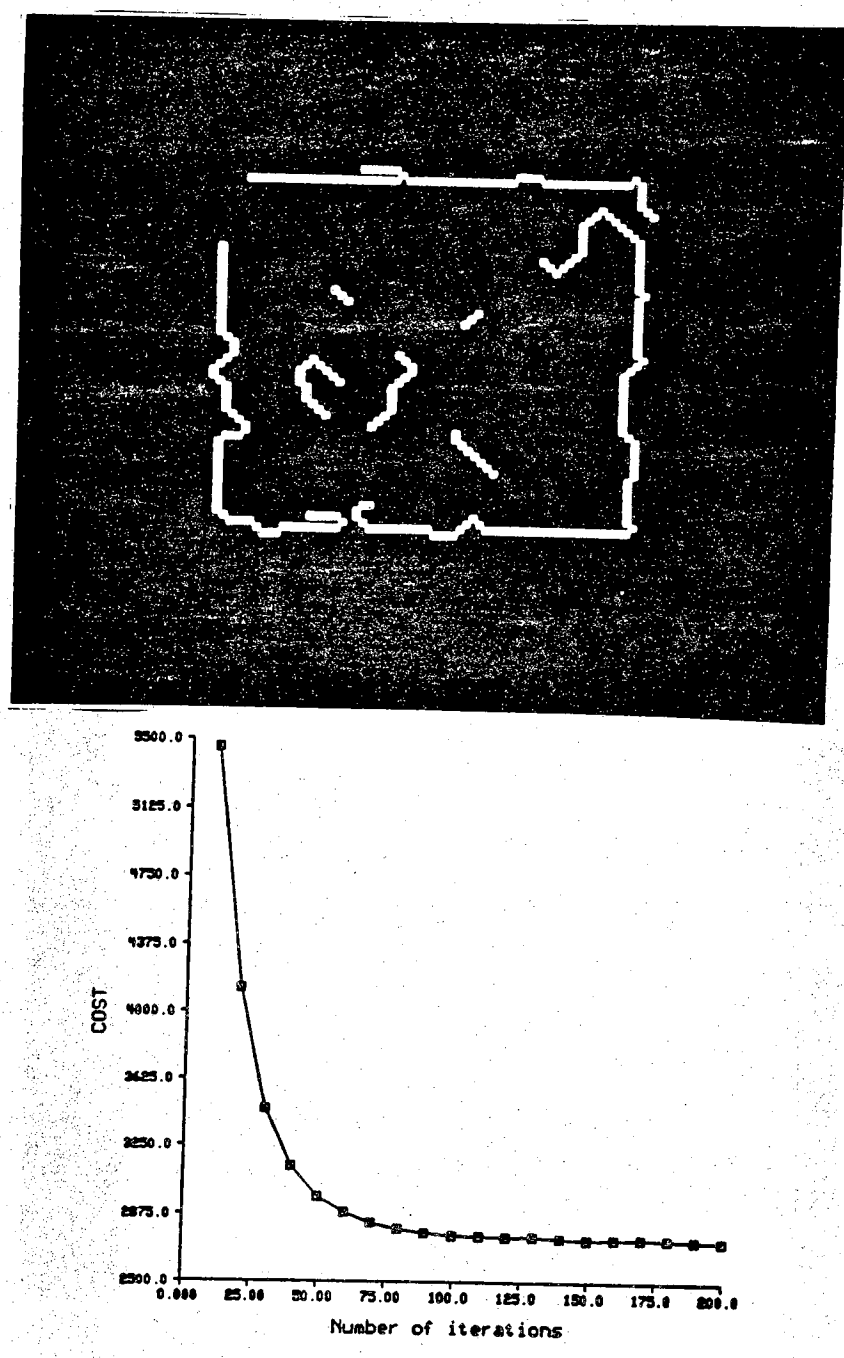


Figure 5.29 Detected edges and annealing process of texture box image using 5 cost factors of the ACF. The weights were: w_c=0.75, w_d=2.0, w_e=1.0, w_f=4.0, and w_t=8.25. (Figure 5.22 shows the results using 6 cost factors).

estimate of d^* based on Equation (4.32). The results indicate that the removal of the cost factor C_n alters the detected edges in a very minor way which is only slightly visible. Depending on the test image used, this alteration may improve or degrade detection performance.

5.7 Summary

We have shown examples of the detection of edges using both the CCF and ACF approaches. Comparison of the detection performance has been made with four other recent edge detection techniques: derivative of Gaussian, Laplacian of Gaussian, facet model, and Sequential Edge Linking. Both real and artificial images were used in evaluating the detection performance. Based on the Pratt figure of merit, it has been show that the detected edges of both the CCF and ACF techniques that are at least of comparable quality with other current techniques.

For the ACF approach, we have shown that all detected edges are thin, provided that the weight w_t is properly selected based on Propositions 3.15, 4.1 and 4.2. We have also demonstrated the usefulness of the cost factor for fragmentation in linking together fragmented edges, while at the same time suppressing short sporadic edges. This approach to edge detection is flexible in the sense that it allows for the detection of many different types of edges. In particular, we have shown examples of how the dissimilarity measure for the cost function can be defined to detect texture edges or other specific edge types.

CHAPTER 6
SUMMARY AND CONCLUSIONS

6.1 Summary of Results

The main emphasis of this work has been to cast edge detection as a problem in cost minimization. We have achieved this by the formulation of two cost functions that evaluate the quality of edge configurations. The first is a comparative cost function (CCF), which is a linear sum of weighted cost factors. It is heuristic in nature and can be applied only to pairs of similar edge configurations. It measures the relative quality between the configurations. The detection of edges is accomplished by a a heuristic iterative search algorithm which uses the CCF to evaluate edge quality.

The second cost function is the absolute cost function (ACF), which is also a linear sum of weighted cost factors. The cost factors capture desirable characteristics of edges such as accuracy in localization, thinness, and continuity. Edges are detected by finding the edge configurations that minimize the ACF. We have analyzed the function in terms of the characteristics of the edges in minimum cost configurations. These characteristics are directly dependent of the associated weight of each cost factor. Through the analysis of the ACF, we have provided guidelines on the choice of weights to achieve certain characteristics of the detected edges.

Minimizing the ACF is accomplished by the use of Simulated Annealing. Specifically, we have developed a set of strategies for generating next states for the annealing process. The method of generating next states allows the annealing process to be executed largely in parallel. We have also stated an estimate of the upper bound on the maximum cup depth of the cost function. This bound is useful in the design of an efficient temperature schedule for the annealing process.

Experimental results are shown which verify the usefulness of the CCF and ACF techniques for edge detection. In comparison, the ACF technique produces better edges than the CCF or other current detection techniques. A major difficulty with the annealing process is the large amount of computation

time required to minimize the ACF.

6.2 Suggestions for Further Work

Minimizing the absolute cost function is a novel approach to edge detection. Its usefulness has been both theoretically and experimentally justified. The following is a brief list of further research that could be undertaken in pursuit of this approach to edge detection.

(1) The approach is capable of detecting various kinds of edges, provided that a suitable measure of dissimilarity $f_a(R1,R2)$ can be defined. We could investigate the numerous possible ways of defining $f_a(R1,R2)$, and show how it can be applied to detect different edge types in real world situations.

(2) In this report, the basis set of edge structures for use in dissimilarity enhancement was constrained to be thin edge structures of 3 pixels. Investigation could be made into the use of other basis sets, possibly comprised of larger edge structures. This investigation should be performed in conjunction with (1) above.

(3) More experiments with Simulated Annealing could be undertaken. Five areas of possible investigation are listed below.

  (i) Choice of the probabilities p_i as given in Equation (4.21). These probabilities govern the frequency each strategy of generating next states is used.

  (ii) Alternate methods of generating next states.

  (iii) Alternate temperature schedules. The reference [63] could be consulted.

  (iv) Use of rapid cooling and different initial states.

  (v) Parallel implementation.

(4) It is not apparent that Simulated Annealing is the best algorithm for minimizing the absolute cost function. Other minimization techniques could be investigated.

(5) The investigation of how a priori information can be best incorporated into the cost function. This can be achieved either by direct incorporation into the dissimilarity measure $f_a(R1,R2)$, or by the inclusion of additional cost factors.

(6) The use of additional cost factors to capture desirable edge characteristics that have not already been mentioned.

REFERENCES

REFERENCES

[1] M. Brady, "Computational approaches to image understanding," *Computing Surveys*, vol. 14, No. 1, pp. 3-71, March 1982.

[2] P. J. Besl and R. C. Jain, "Three-dimensional object recognition," *Computing Surveys*, vol. 17, No. 1, pp. 75-145, March 1985.

[3] L. S. Davis and A. Mitiche, "Edge detection in textures," *Computer Graphics and Image Processing*, vol. 12, pp. 25-39, 1980.

[4] A. Rosenfeld and A. C. Kak, *Digital Picture Processing.* New York: Academic Press, 1982, 1 & 2 .

[5] T. Peli and D. Malah, "A study of edge detection algorithms," *Computer Graphics and Image Processing*, vol. 20, pp. 1-21, 1982.

[6] L. S. Davis, "A survey of edge detection techniques," *Computer Graphics and Image Processing*, vol. 4, pp. 248-270, 1975.

[7] V. Torre and T. A. Poggio, "On edge detection," *IEEE Trans. Pattern Anal. Machine Intelligence*, vol. PAMI-8, No. 2, pp. 147-163, March 1986.

[8] T. Poggio, "Early vision: From computational structure to algorithms and parallel hardware," *Computer Vision, Graphics, and Image Processing*, vol. 31, pp. 139-155, 1985.

[9] F. M. Dickey and K. S. Shanmugam, "Optimum edge detection filter," *Appl. Opt.*, vol. 16 no. 1, pp. 145-148, Jan. 1977.

[10] J. Canny, "A computational approach to edge detection," *IEEE Trans. Pattern Anal. Machine Intelligence*, vol. PAMI-8, No. 6, pp. 679-698, November 1986.

[11] D. Marr and E. Hildreth, "Theory of edge detection," *Proc. Royal Soc. London*, vol. B 207, pp. 187-217, 1980.

[12] D. Marr, *Vision.* New York: W. H. Freeman and Company, 1982.

[13] M. J. Brooks, "Rationalizing edge detectors," *Computer Graphics and Image Processing*, vol. 8, pp. 277-285, 1978.

[14] M. H. Hueckel, "An operator which locates edges in digitized pictures," *Journal of the Association for Computing Machinery*, vol. 18, No. 1, pp. 113-125, January 1971.

[15] R. M. Haralick, "Edge and region analysis for digital image data," *Computer Graphics and Image Processing*, vol. 12, pp. 60-73, 1980.

[16] R. M. Haralick and L. Watson, "A facet model for image data," *Computer Graphics and Image Processing*, vol. 15, pp. 113-129, 1981.

[17] R. M. Haralick, "Digital step edges from zero crossing of second directional derivatives," *IEEE Trans. Pattern Anal. Machine Intelligence*, vol. PAMI-6, No. 1, pp. 58-68, IEEE, January 1984.

[18] V. Nalwa and T. O. Binford, "On detecting edges," *IEEE Trans. Pattern Anal. Machine Intelligence*, vol. PAMI-8, No. 6, pp. 699-714, November 1986.

[19] B. J. Schachter and A. Rosenfeld, "Some new methods of detecting step edges in digital pictures," *Communications of the ACM*, vol. 21, No. 2, pp. 172-176, February 1978.

[20] R. Machuca and A. L. Gilbert, "Finding edges in noisy scenes," *IEEE Trans. Pattern Anal. Machine Intelligence*, vol. PAMI-3, No. 1, pp. 103-111, January 1981.

[21] A. Martellli, "An application of heuristic search methods to edge and contour detection," *Communications of the ACM*, vol. 19, No. 2, pp. 73-83, February 1976.

[22] G. P. Ashkar and J. W. Modestino, "The contour extraction problem with biomedical applications," *Computer Graphics and Image Processing*, vol. 7, pp. 331-355, 1978.

[23] P. H. Eichel and E. J. Delp, "Sequential edge detection in correlated random fields," *Proceedings of the IEEE Computer Vision and Pattern Recognition Conference*, pp. 14-21, San Francisco, June 1985.

[24] R. Nevatia and K. R. Babu, "Linear feature extraction and description," *Computer Vision and Image Processing*, vol. 13, pp. 257-269, 1980.

[25] A. Rosenfeld, *Image Modeling*. New York: Academic Press, 1981.

[26] A. Rosenfeld, "Iterative methods in image processing," *Pattern Recognition*, vol. 10, pp. 181-187, Pergamon Press Ltd., 1978.

[27] S. Ullman, "Relaxation and constrained optimization by local processes," *Computer Graphics and Image Processing*, vol. 10, pp. 115-125, 1979.

[28] S. Peleg, "A new probabilistic relaxation scheme," *IEEE Trans. Pattern Anal. Machine Intelligence*, vol. PAMI-2, No. 4, pp. 362-369, IEEE, July, 1980.

[29] J. A. Bondy and U. S. R. Murthy, *Graph Theory With Applications.* New York: North-Holland, 1976.

[30] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity.* Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1982.

[31] J. M. Hammersley and D. C. Handscomb, *Monte Carlo Methods.* London: London: Methuen, 1964.

[32] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equation of state calculations by fast computing machines," *The Journal of Chemical Physics*, vol. 21, No. 6, pp. 1087-1092, June 1953.

[33] C. J. Thompson, *Mathematical Statistical Mechanics.* New York: Macmillan Company, 1972.

[34] A. B. Brotz, M. H. Kalos, and J. L. Lebowitz, "A new algorithm for Monte Carlo simulation of Ising spin systems," *Journal of Computational Physics*, vol. 17, pp. 10-18, 1975.

[35] I. Z. Fisher, "Applications of the Monte Carlo method in statistical physics," *Soviet Physics Uspekhi*, vol. 2, No. 6, pp. 783-1012, June 1960.

[36] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, No. 4598, pp. 671-680, 13 May 1983.

[37] V. Cerny, "Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm," *Journal of Optimization Theory and Applications*, vol. 45, No. 1, pp. 41-51, January 1985.

[38] C. C. Skiscim and B. L. Golden, "Optimization by simulated annealing: A preliminary computational study for the TSP," in *Proceedings of the 1983 Winter Simulation Conference*, IEEE, 1983, pp. 523-535.

[39] M. P. Vecchi and S. Kirkpatrick, "Global wiring by simulated annealing," *IEEE Transactions on Computer-Aided Design*, vol. CAD-2, No. 4, pp. 215-222, October 1983.

[40] A. A. El Gamal, L. A. Hemachandra, I. Shperling, and V. K. Wei, "Using simulated annealing to design good codes," *IEEE Trans. Inform. Theory*, vol. IT-33, No. 1, pp. 116-123, January 1987.

[41] D. B. Paul, "Training of HMM recognizers by simulated annealing," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, vol. 1, 1985, pp. 13-16.

[42] P. Carnevali, L. Coletti, and S. Patarnello, "Image processing by simulated annealing," *IBM J. Res. Develop.*, vol. 29, No. 6, pp. 569-579, November 1985.

[43] W. E. Smith, H. H. Barrett, and R. G. Paxman, "Reconstruction of objects from coded images by simulated annealing," *Optics Letters*, vol. 8, No. 4, pp. 199-201, April 1983.

[44] H. Fleisher, J. Giraldi, D. B. Martin, R. L. Phoenix, and M. A. Tavel, "Simulated annealing as a tool for logic optimization in a cad environment," in *IEEE International Conference on Computer-Aided Design* , 1985, pp. 203-205.

[45] E. Cinlar, *Introduction to Stochastic Processes.* Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1975.

[46] D. L. Isaacson and Richard W. Madsen, *Markov Chains and Applications.* New York: John Wiley and Sons, 1975.

[47] M. Iosifescu, *Finite Markov Processes and Their Applications.* New York: John Wiley and Sons, 1979.

[48] E. Wong and B. Hajek, *Stochastic Processes in Engineering Systems.* New York: Springer-Verlag, 1985.

[49] F. Romeo and Alberto Sangiovanni-Vincentelli, "Probabilistic hill climbing algorithms: Properties and applications," in *1985 Chapel Hill Conference on VLSI*, 1985, pp. 393-417.

[50] F. Romeo, A. Sangiovanni-Vincentelli, and C. Sechen, "Research on simulated annealing at Berkeley," in *Proceedings of the IEEE International Conference on Computer Design: VLSI in Computers*, 1984, pp. 652-657.

[51] M. Lundy and A. Mees, "Convergence of an annealing algorithm," *Mathematical Programming*, vol. 34, pp. 111-124, 1986.

[52] S. Geman and D. Geman, "Stochastic relaxation, Gibbs distributions, and the bayesian restoration of images," *IEEE Trans. Pattern Anal. Machine Intelligence*, vol. PAMI-6, No. 6, pp. 721-741, November 1984.

[53] D. Mitra, F. Romeo, and A. Sangiovanni-Vincentelli, "Convergence and finite-time behavior of simulated annealing," in *Proceedings of 24th*

Conference on Decision and Control, 1985, pp. 761-767.

[54] D. Mitra, F. Romeo, and Alberto Sangiovanni-Vincentelli, "Convergence and finite-time behavior of simulated annealing," *Adv. Appl. Prob.*, vol. 18, pp. 747-771, 1986.

[55] S. B. Gelfand and S. K. Mitter, "Analysis of simulated annealing for optimization," in *Proceedings of 24th Conference on Decision and Control*, Ft. Lauderdale, FL., December 1985, pp. 779-786.

[56] B. Gidas, "Nonstationary markov chains and convergence of the annealing algorithm," *Journal of Statistical Physics*, vol. 39, Nos. 1/2, pp. 73-131, 1985.

[57] B. Hajek, "A tutorial survey of theory and applications of simulated annealing," in *Proceedings of 24th Conference on Decision and Control*, Ft. Lauderdale, FL., December 1985, pp. 755-760.

[58] B. Hajek, "Cooling schedules for optimal annealing," *Mathematics of Operations Research*, vol. 13 No.2, pp. 311-329, May 1988.

[59] J. Serra, *Image Analysis and Mathematical Morphology*. New York: Academic Press, 1982.

[60] W. Pratt, *Digital Image Processing*. New York: Wiley, 1978.

[61] I. Abdou and W. Pratt, "Quantitative design and evaluation of enhancement/thresholding edge detectors," *Proc. of IEEE*, vol. 67, pp. 753-763, May 1979.

[62] L. Kitchen and A. Rosenfeld, "Edge evaluation using local edge coherence," *IEEE Transactions on Systems, Man and Cybernetics*, vol. SMC-11, No. 9, pp. 597-605, IEEE, Sept. 1981.

[63] P. J. M. van Laarhoven and E. H. Aarts, *Simulated Annealing: Theory and Applications*. Dordrecht, Holland: D. Reidel Publishing Company, 1987.