ON EDGE LINKING

EDWARD CHOME

MS Dissertation

Graduate School of Sciences
Computer Engineering
Engineering Program
June, 2015

JÜRİ VE ENSTİTÜ ONAYI

Edward Chome'nin **On Edge Linking** başlıklı **Bilgisayar Mühendisliği** Anabilim Dalı **Bilgisayar** Bilim Dalındaki Yüksek Lisans tezi 08.07.2015 tarihinde aşağıdaki jüri tarafından Anadolu Üniversitesi Lisansüstü Eğitim - Öğretim ve Sınav Yönetmeliğinin ilgili maddeleri uyarınca değerlendirilerek kabul edilmiştir.

		Adı -Soyadı	**İmza**
Üye (Tez Danışmanı)	**:**	**Doç. Dr. Cüneyt Akınlar**	**.................**
Üye	**:**	**Yard. Doç. Dr. Muzaffer Doğan**	**.................**
Üye	**:**	**Yard. Doç. Dr. Mustafa Müjdat Atanak**	**.................**

Anadolu Üniversitesi Fen Bilimleri Enstitüsü Yönetim Kurulu'nun .... tarih ve .... sayılı kararıyla onaylanmıştır.

Enstitü Müdürü

ABSTRACT

MS Dissertation

ON EDGE LINKING

Edward Chome

Anadolu University
Graduate School of Sciences
Computer Engineering Program

Supervisor: Assoc. Prof. Dr. Cüneyt Akınlar
2015, 53 pages

Edge detection is a fundamental first step in many computer vision and image processing applications. Since traditional edge detection algorithms produce binary edge maps as output (which usually consist of multi-pixel wide, disconnected -especially in noisy images- edge fragments), an additional edge linking step is usually employed to clean up the resulting edge map and combine disjoint edge fragments. An edge linker takes a binary edge map as input and is expected to generate high-quality (one-pixel wide and contiguous) edge segments (chain of pixels), which are then used in such applications as line, arc and shape detection, image segmentation, tracking and registration, among many others.

In this thesis, two edge linking algorithms are proposed: The first algorithm makes use of the Smart Routing (SR) step of the recently proposed edge segment detection algorithm Edge Drawing (ED), to convert Canny's binary edge maps to edge segments; thus the name CannySR. The second algorithm takes in a binary edge map generated by any arbitrary traditional edge detection algorithm and converts it to a set of edge segments; filling in one pixel gaps in the edge map, cleaning up noisy edge pixel groups and thinning multi-pixel wide edge pixel formations in the process. The algorithm walks over the edge map based on the predictions generated from its past movements; thus the name Predictive Edge Linking (PEL).

We evaluate the performance of CannySR and PEL both qualitatively using visual experiments and quantitatively within the precision-recall framework of the Berkeley Segmentation Benchmark (BSDS 300), and compare its performance with ED, which is a natural edge segment detection algorithm. Both visual experiments and quantitative evaluation results show that both CannySR and PEL greatly improves the modal quality of binary edge maps produced by traditional edge detectors, and take a very small amount of time to execute making them suitable for real-time image processing and computer vision applications.

Keywords: Edge Detection, Edge Linking, Edge Segment Detection, Canny, Edge Drawing, Smart Routing.

ÖZET

Yüksek Lisans Tezi

KENAR BAĞLAMA ÜZERİNE BİR ÇALIŞMA

Edward Chome

Anadolu Üniversitesi
Fen Bilimleri Enstitüsü
Bilgisayar Mühendisliği Anabilim Dalı

Danışman: Doç. Dr. Cüneyt Akınlar
2015, 53 sayfa

Kenar tespiti birçok bilgisayarlı görü ve imge işleme uygulamalarında temel ilk adımdır. Geleneksel kenar tespit algoritmalarının ürettikleri ikili kenar haritaları çoğunlukla birden fazla piksel genişliğinde ve -özellikle gürültülü resimlerde- parçalı kenar fragmanlarından oluştuğu için, üretilen ikili kenar haritasındaki boşlukların doldurulması ve gürültülerin temizlenmesi için kenar bağlama işlemi kullanılmaktadır. Bir kenar bağlama algoritması ikili kenar haritasını işleyip yüksek kalitede (tek piksel genişliğinde ve bitişik) kenar bölütleri üretmelidir. Bu bölütler daha sonra çizgi, ark ve şekil tespiti, imge bölütleme, gibi birçok uygulamada kullanılabilirler.

Bu tezde iki adet kenar bağlama algoritması önerilmiştir. İlk önerilen algoritma Canny kenar tespit algoritması tarafından üretilen ikili kenar haritalarını, yakın zamanda önerilen Kenar Çizme algoritmasının Akıllı Rotalama adımını kullanarak çalışan, bu sebeple CannySR olarak adlandırılan bir algoritmadır. Ikinci önerilen algoritma ise girdi olarak herhangi bir kenar tespit algoritması tarafından üretilen bir kenar haritası alıp bunu kenar bölütlerine çevirir. Bu işlem esnasında kenar haritası içindeki bir piksel büyüklüğündeki boşlukları doldurur, gürültülü kenar piksel gruplarını temizler ve birkaç piksel genişliğindeki kenar piksel oluşumlarını inceltir. Bu algoritma kenar haritası üzerinde geçmiş hareketlerinden üretilen öngörüler ile hareket ettiği için Öngörülü Kenar Bağlama (PEL) olarak adlandırılır.

PEL ve CannySR'nin performansı öncelikle görsel deneyler vasıtasıyla nitel olarak değerlendirilmiştir. Nicel değerlendirme ise Berkeley Bölüt Kıyaslama (BSDS 300)'nın doğruluk-hatırlama çerçevesi içinde gerçekleştirilmiştir. Önerilen algoritmalar hem Canny ile hem de doğal bir kenar bölüt tespit algoritması olan Kenar Çizme algoritması ile karşılaştırılmıştır. Hem görsel, hem de nicel değerlendirmeler önerilen CannySR ve PEL kenar bağlama algoritmalarının geleneksel kenar tespit algoritmaları tarafından üretilen ikili kenar haritalarının şekilsel kalitelerini büyük ölçüde iyileştirdiğini göstermektedir. Ayrıca algoritmalar çok kısa zamanda çalışmaktadır, ve bu sebeple gerçek zamanlı uygulamalar için çok uygun olacakları düşünülmektedir.

Anahtar Kelimeler: Kenar Tespiti, Kenar Bağlama, Kenar Bölütü Tespiti, Canny, Kenar Çizme, AkıllıYönlendirme.

ACKNOWLEDGEMENTS

I would like to thank my supervisor Assoc. Prof. Dr. Cüneyt Akınlar for his supervision and guidance. I am extremely thankful and indebted to him for sharing his expertise, and for his sincere and valuable guidance and encouragement throughout this work.

I would also like to offer my most sincere thanks to all my teachers who contributed towards my education in my life.

I would also like to take this opportunity to express my gratitude to all the members of the Computer Engineering Department, Anadolu University, for their valuable help and support throughout my studies.

I also thank my parents for their unceasing encouragement, support and attention.

Finally, I am grateful to my friends Burak Benligiray, Muhammad Lemin and many others who supported me throughout this venture.

Edward Chome

29.06.2015

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

BEM	Binary Edge Map
CannySR	Canny using Smart Routing
ED	Edge Drawing
PEL	Predictive Edge Linking
SR	Smart Routing

1. INTRODUCTION

Edge detection is one of the most important and fundamental steps in many computer vision and image processing applications. Edges can be roughly described as image positions where the local intensity changes distinctly along a particular orientation. The stronger the intensity, the higher the evidence for an edge at that position [4].

Edges and contours play a dominant role in human vision and other biological vision systems. Edges are often vital clues toward the analysis and interpretation of image information, both in biological vision and in computer image analysis [5]. In fact, edge-like structures and contours seem to be so important for our human visual system that a few lines in a caricature or illustration are often sufficient to unambiguously describe an object or a scene [4].

The edges are not only visually striking but a complete object can be reconstructed by just a few key edges. Humans have a very good visual system that makes it possible to automatically detect boundaries within moments in an image; whereas, a considerable amount of effort is required for machines to replicate the same [6]. The human vision has a unique way of detecting boundaries in objects contained in images; however, in some cases the human visual system is affected by optical illusion.

Mathematically edge detection can be defined as a function that aims to identify points in a digital image at which the brightness or the intensity level changes sharply or has discontinuities. Edge detection is a useful, low-level form of image processing for obtaining a simplified image [7]. It serves to simplify the analysis of image by drastically reducing the amount of data to be processed and at the same time preserving useful structural information about the object boundaries [8].

Edge detection is a process which transforms a grayscale image to binary image, which indicates either the presents or absence of an edge [9]. More specifically, edge detection can be defined as the process of determining which pixels are the edge pixels [5]. The result of an edge detection is usually an edge map which is an image that describes each original pixel's edge classification and possible edge attributes such as magnitude and orientation [5]. A traditional edge detection algorithm takes a grayscale image as input and produces a binary edge map (BEM) as output, where an edge pixel (edgel) is marked (e.g., its value in the edge map is 255), and a non-edge pixel is unmarked (e.g., its value in the edge map is 0).

Due to its importance, edge detection remains to be an active area of research. It is not a trivial task, the research area has attracted much attention over the past decades. This is evidenced by a myriad of edge detection algorithms that have been proposed over the years. Unfortunately these algorithms depend on a wide range of external factors such as choice of appropriate filters and thresholds. Furthermore they are only limited to digital images. The other striking factor is that there is no a single edge detection algorithm which is applicable in all situations (different types of images and scenes contained in images). Despite the fact that there are a wide number of edge detection algorithms proposed in the literature, they are still far from matching the human eye.

There are a number of problems that can baffle the edge detection process in real images. These factors may include noise, crosstalk or interference between nearby edges and inaccuracies resulting from the use of a discrete grid [5], which result in missing edges, false detection and errors in edge location. Henceforth, there is a need for edge linking as there are usually gaps between the edgels, unattended edgels and noisy notch-like structures, ragged and multi-pixel wide edgels formations etc. This is the subject of this thesis. Our goal is to develop an edge linking algorithm that takes in a low-quality binary edge map produced by a traditional edge detection algorithm, and output a set of edge segments each of which is a chain of pixels. In the process, the proposed algorithm needs to fill up small gaps between edgel groups, clean up noisy edgel formations and thin down multi-pixel wide edgel groups to 1-pixel wide chains.

1.1 Applications of Edge Detection

Edge detection is of paramount importance in a wide range of applications such as in image processing and analysis, pattern recognition and computer vision applications. Boundary lines are considered one of the most important features of an object in an image. Many computer vision applications rely on boundary information for object and shape recognition tasks [10].

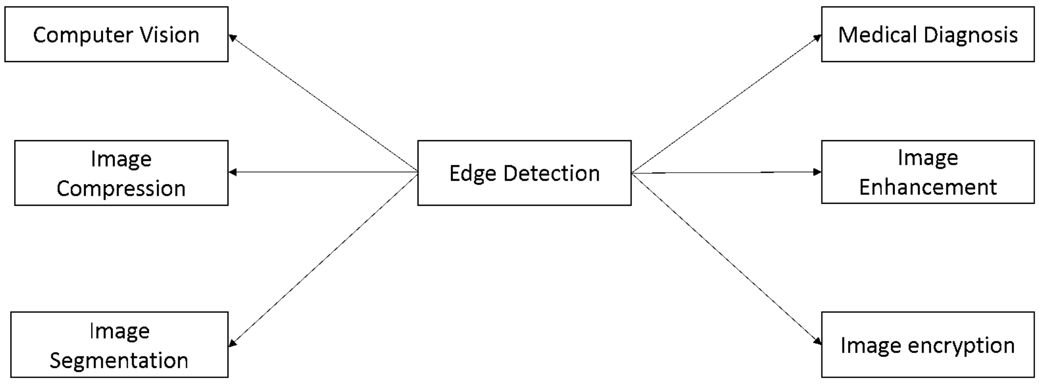


Figure 1.1: Some Applications of Edge Detection

Originally edge detection was used in object detection as the boundaries detected provide a set of features suitable for model matching. Nowadays it is being applied to a host of different ways [11]. It has been used as a pre-processing step in many applications such as image segmentation [9].

Edge detection algorithms may greatly reduce the amount of data to be processed or work to be carried out by filtering irrelevant data and at the same time preserving relevant data which will be processed. The purpose of edge detection algorithms is to extract useful information from an image in such a way that the image will be left with less but relevant information [12].

While edges can be used as objects of recognition or features for matching they can also be used for image editing [13]. A similar object can be reconstructed from the edge information obtained by edge detectors.

Edge detection is also particularly important in medical imaging, where it is used for body part recognition and also tumour identification such as in magnetic resonance angiography (MRA) [14]. It is of great importance in diagnostic detection and feature extraction. For example in medical imaging, edges may represent a tumour and other features of interest. Once the boundary features have been established high order reconstruction methods can be used to analyze internal tissues [15].

Steganography algorithms and digital water making algorithms use edges to hide secure information. Furthermore image resizing algorithms using the edge map of an image paired with a logical transform have yielded superior image resizing results [16]. Carlsson suggested a novel form of coding in which a compressed image is generated from the information contained in edges pixels [15, 17]. A recent JPEG-LS compression standard developed by the Joint Photograph Experts Group uses simple localized edge detection techniques in order to determine the predictive value of each pixel [15].

The correct detection of boundaries between overlapping objects allows for accurate object identification and precise motion analysis for several machine vision applications. This initial procedure often leads to further calculations such as area, perimeters and shape classification of scene elements once they have been isolated from the background this has shown to be particularly useful for military and surveillance applications [18].

Edge detection is a problem of fundamental importance to image analysis tasks. The aim is to find areas where there is high or large intensity changes. These changes often correspond to some boundaries of an object in an image. Edges characterize object boundaries which are useful for segmentation, registration and identification of objects in a scene [19].

Edge detection is also particularly important to image segmentation. It has been a staple of many segmentation algorithms for many years [20]. Examples of algorithms which make use of edge detection include geometric active contours, gradient vector floor and snakes use edge information to detect their curve evolution [18].

1.2 Edge Linking

Conventional edge detection algorithms always result in missing parts of the edges and spurious edges being added [9]. Efficient edge operators such as those based on partial derivatives fail to produce continuous edge maps. Traditional edge detection algorithms use a threshold and some filters to detect edges, consequently the edge maps produced thereafter consist of individual edge pixels with no real relationship and connectivity [21].

Some edge detection algorithms use a global threshold. The global threshold transforms the image into a binary image; however, this results in removal of thin edges in low contrast regions. Consequently the edges would be broken and important edges are lost [9].

Conventional edge detectors such as Sobel or Canny [22, 23] use local gradient operators and at times with additional smoothing for noise removal as a result some important edges might be regarded as noise and therefore discarded [9].

Common edge detectors which are supposed to extract object boundaries suffer from the problems posed by differentiation operations and noises contained in images except for images obtained in highly restricted environments [10]. To address this, supplementary edge linking step is required to complete the initial edge information [2]. Edge linking is of great importance and plays a pivotal role in ensuring that spurious edges are eliminated and the missing parts are added.

Edge detection involves four stages as explained by Gonzalez and Woods [24]. The first step is usually smoothing, which attempts to reduce noise and minimize the detection of false edges. Sharpening or enhancement then follows, which attempts to consolidate edges that might have been lost due to smoothing. Sharpening is usually optional some algorithms do not make use of it. Detection then follows that selects which non-zero pixels are to be considered as edges and which are not to be considered. A threshold is then applied to the image, which makes it a binary image as there will be two set of pixels those which are above the threshold and those which fall below. From the steps described above, at each stage some important edges will be lost.

Edge detection errors can occur in two forms, which can be grouped as false positive and false negative. False positives occur when wrong pixels are classified as edge pixels in other words they will be misclassified as edge pixels. False negative occurs when true edge pixels are not classified as edge pixels in other words they will be misclassified as not belonging to edge pixels [5]. Detection of both types increase proportionately with noise and as a result noise suppression is of great importance as it goes a very long in increasing the odds of accuracy detection. There is a need to compensate for those edges lost to ensure that edges are connected and to discard those false edges, thereby increasing the accuracy of edge detection. This will have a great impact on subsequent algorithms that depend on the outputs of an edge detection operation. This also depicts that edge linking is not a trivial task as it tries to ensure that edges are connected. Edge linking attempts to fill in the gaps and to connect the segments to a set of contiguous lines and also to eliminate spurious edges, enabling the precise description and accurate analysis of the boundary objects [10].

Edge linking and boundary detection operations are the fundamental important steps in image understanding. Edge linking process takes an unordered set of edge pixels produced by an edge detector as input to form an ordered list of edges [25]. If the edges have been detected using zero crossing of some function then linking them up is a very minute task since the edge elements share a common endpoint [13]. The edge elements are linked into chains by picking up an unlikely edge element and following its neighbors in both directions. Either a sorted list of edge elements or a 2D array is used to speed up the process of finding the neighbors. Since edge detection is a pre-processing step for many applications it is therefore necessary to ensure the successful detection of edges by following up with edge linking as other applications are dependent on the success of the detection.

Computer vision systems that rely on edge detection often have a hard time in doing their task if the edges contain gaps and are ambiguous. Henceforth edge linking is of great importance in eliminating such problems caused by non-contiguous and ambiguous edges. Detected edges are usually defragmented and in general they do not divide the image into sections henceforth edge linking is more than necessary [12]. On the other hand defragmented edges or edge elements produced by an edge detection operation can be useful in some applications such as line detection and sparse stereo matching; however, they are more useful when linked into contiguous contours [13].

Different techniques have been employed for linking edge points in order to recover closed contours. Edge linking methods can be classified into two main categories: (1) Local Edge Linking, (2) Global Edge Linking. Others group the edge linking into three categories, the third one being Regional Linking [24].

Local edge linking can be one of the simplest forms of edge linking as it involves analyzing the characteristics of pixels at every point in a small neighborhood (e.g., 3x3) that have been declared edge points. All the points that are similar or share some common properties according to some criteria are linked forming an edge of pixels. The two main properties used for establishing similarity of edge pixels are the strength (magnitude) of the edge and the direction of the gradient vector [24]. Local edge linking algorithms work over a single edge point by considering that particular point and its neighboring edge point's relationship with it [26, 27]. The basic process used by Local Edge Linking is that of tracking a sequence of edge points. Local Edge Linking has the advantage that it can be used to find arbitrary curves [27].

Local approaches have advantages; however, they are applicable in situations where knowledge or partial knowledge about pixels belonging to individual objects is at least known. Often times we have to work in situations where there is no knowledge of where the objects of interest might be. In these situations all pixels are considered as potential candidates to be included as edges for linking until when they fail to meet a certain predefined threshold or criteria put in place.

Regional processing is based on the idea that often times the locations of regions of interest in an image are known or they can be established. This implies that knowledge is available regarding the regional membership of pixels in the corresponding edge image. Techniques for linking pixels on a regional basis are used in such situations with the desired result being an approximation to the boundary of the region [24].

Global approaches consider the whole edge map at the same time and sets of edge points are sought according to some similarity constraint such as points which share the same edge equation [26, 27]. Global approaches apply mathematical modeling techniques to formulate the boundaries of the objects in the images [10]. Examples of global approaches include the Hough transformation [28] and the whole-boundary formulation [29].

Many edge linking algorithms have been proposed to compensate for the edges not fully connected. Broken edges are very difficult to fix [9]. The lack of information from edge images such as low number of endpoints limits the performance of edge linking algorithms. Although many edge linking algorithms have been proposed, they still have some shortfalls and are unable to link some edges. Edge linking remains to be an area of research as this is evidenced by the large number of researchers choosing the topic and large number of algorithms which have been proposed over the decades.

1.3 Objectives

The objective of this thesis is to develop edge linking algorithms that satisfy the following goals:

(1) Close small gaps (1-pixel gaps in our case) between edgel groups

(2) Clean-up noisy, unattended and notch-like structures from the edge map

(3) Thin down multi-pixel wide staircase edgel formations to 1-pixel wide chains

(4) Output a set of edge segments each of which is a chain of pixels. The edge segments can then be used in high level operations such as line, arc, circle, ellipse and corner detection.

1.4 Outline

The rest of this thesis is organized as follows:

- **Chapter 2** describes related work done by researchers in the literature. It gives a brief summary of the previous edge linking algorithms and how they operate. It then gives strengths and weaknesses of these previous linking algorithms and identifies more research opportunities.
- **Chapter 3** gives theory to the proposed solution. It clearly identifies the problem and explores the proposed algorithms in detail.
- **Chapter 4** gives the results and evaluation. The proposed algorithms are evaluated both qualitatively and quantitatively, and compared with some related work.
- **Chapter 5** presents the conclusions and suggestions for future work.

2. RELATED WORK

Edge detection is of great importance to image processing and computer vision applications. Edge detection algorithms always result in edge maps having broken edges or edges appearing where there are not supposed to appear; henceforth, many edge linking algorithms have been proposed to try to counter these shortcomings. Edge linking algorithms try to fix the broken edges, spurious edges and other problems encountered by edge detection algorithms in order for subsequent applications that depend on edges to perform at their best.

Edge linking algorithms can be divided into two main groups: Those based on global approach and those based on local approach. There also exist other edge linking algorithms that combine both approaches that may use additional information such as colour [26].

Although many edge linking algorithms have been proposed, there is not a single one that works well in all circumstances to date. This shows that edge linking is a challenging task and not a trivial one. A great many researchers who have devoted their time and effort to this research area only proves and provides evidence of how edge linking carries such a great weight.

In this chapter, we review some of the algorithms proposed over the decades. Only the most important and relevant edge linking algorithms are chosen here, and they are presented in chronological order.

2.1 Edge Linking By Using Causal Neighborhood Window

Xie [30] is one of the first researchers to present an edge linking algorithm that makes use of two concepts: the horizontal edge elements, and the concepts of casual neighborhood window. The algorithm performs two operations in one pass, which are contour chain creation and the contour chain linking. An edge map is transformed into a set of horizontal edge elements, and then grouped into contour chains [31].

A neighborhood window is referred to as being casual when all the neighboring edge elements have been processed and the chains where they belong to are known [31]. The main advantage of this approach is that it has a low computational cost; however, the drawback is that it performs poorly when dealing with textured images [2].

$$\begin{cases} x_{left0} - x_{gap} - 1 \le x \le x_{right0}, & \text{if } y = y_0 \\ x_{left0} - x_{gap} - 1 \le x \le x_{right0} + x_{gap} + 1, & \text{if } y_0 - y_{gap} - 1 \le y \le y_0 - 1 \end{cases} \tag{2.1}$$

The neighborhood window in region OXY is shown in equation 2.1, where x_{gap} denote the maximum gap value in X direction of boundaries and y_{gap} the maximum gap value in Y direction. Given a casual neighborhood window (x_{gap}, y_{gap}) positioned at a horizontal edge element $(x_{left0}, x_{right0}, y_0)$ [30].

2.2 Edge Linking By Sequential Search

Edge linking by sequential search considers linking as a graph search problem [26]. Each pixel is represented as a node. The set of pixels S is a lattice graph [1] as shown in equation 2.2.

$$S = \{(x, y) : 0 \le x \le M - 1, 0 \le y \le N - 1\} \tag{2.2}$$

Their approach uses the linear model as part of the linking algorithm and a path metric is used to guide the search. A* algorithm is then used for finding the best path along the edge points. The node $S(x, y)$ has 8 nearest neighbors, a tree then evolves having 8 branches. The depth into the tree indicates position along the path. The size of the search space for a path of Q nodes is 7Q; however, they [1] devised measures to reduce it to 3Q as noted by [32]. They limit the possible transitions to 3 ($\pi/4$ or 45^0), which leads to the path definition [32].

They state that succeeding node should differ by 45^0 or less from its predecessor as shown in Fig 2.1. This path definition reduces the search space significantly and ensures that the algorithm is fast. However, one of the problems with this path definition lies with images that have oscillating edges. The algorithm performs poorly as it only looks at 3 possible transitions, which is not the case with images that have oscillating edges. Important pixels may be left out. The main sources of errors occur at the corners since the edge path definition does not take into account abrupt or sharp changing edge transitions. As a result, the errors will be inherited in the linking algorithm resulting in broken edges or contours [33].

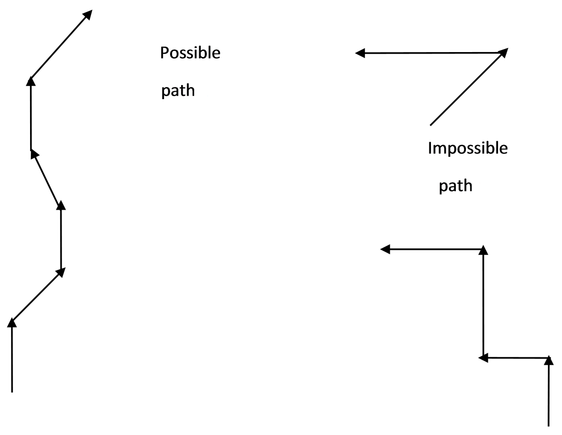


Figure 2.1: Edge path definition (Edge paths are such that connected segments can make ≤ 45^0 from each other).

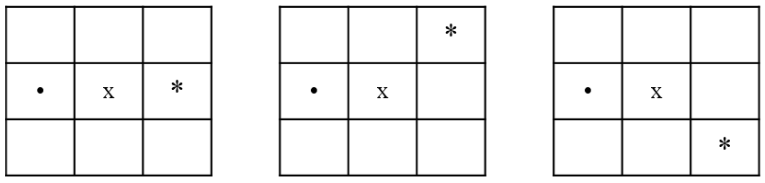


Figure 2.2: Possible node extension on 3x3 neighborhood according to the path definition. (The start node is denoted by x and the preceding node is denoted by • and the start direction is assumed to be horizontal) [1].

They define a path (edge path definition) as a connected set of nodes that has the following qualities: For any subset of three nodes on the path, the direction defined by the two preceding nodes and by the second node differs by $\frac{\pi}{4}$ or less.

They further put some criteria to define the path metric that follows the edge path definition defined above. The path metric is given by equation 2.3 for path $p^{(i)} \in S$ of length Q:

$$\gamma_Q(p^{(i)}) = \sum_{j=1}^{Q} \beta_j^i + h_i(p^{(i)}) \tag{2.3}$$

where β_j^i is a measure for the selection of the possible transitions along the j^{th} branch of the path $p^{(i)}$ that adheres to the path definition, and $h_j(p^{(i)})$ is the apriori measure.

The criteria to define the path metric is as follows:

(a) the path metric should not be biased by the path length

(b) the metric should have the necessary drift property (high on the correct path and low on the wrong path)

(c) the path metric should be easy to calculate

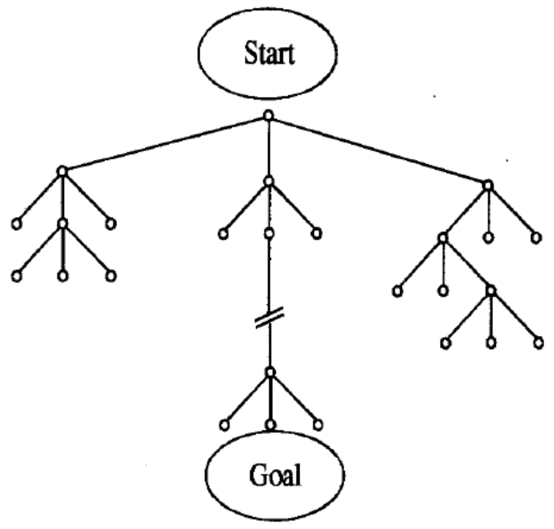


Figure 2.3: A simplified tree structure that satisfies the edge path definition.

Table 2.1: Edge Linking by Sequential Search

```
1. Smooth the image
2. Estimate the gradient of the smoothed image
3. Determine the swath (belt) of important information
4. Linking
    i. Choose the root node and find the initial direction using the
       magnitude and angle information obtained in step 2
    ii. A* algorithm is used within the belt of important information
        [a]. Calculate y (path metric) using the models
        [b]. Break the ties using the apriori measure as well as angle
             information
    iii. Stop the search when all goal nodes have been examined
```

They further reduce the search space to be the area inside the swath (belt) defined by a hypothesized boundary [1]. This further ensures that the algorithm performs fast. However, limiting the search to be just in the swath may affect the accuracy of the algorithm as it can lead to broken edges especially when some important edges are laying outside the swath of important information. Xiaomin Ji et al. [34] noted that the algorithm produced better results; however, it still had broken edges, which could be owing to the limitation of the search range provided by the swath of important information.

They also noted that the approach they use of applying the second gradient operator to the original image and considering the zero-crossing to be the hypothesized boundary has the advantage that the gradient operator provides closed boundaries and also it can be used as a by-product of the linking algorithm [1].

One of the weaknesses of the sequential edge linking algorithm is that it depends too much on the accuracy of the initial enhancement stages [1]. If the enhancement stage was done poorly, the errors will be inherited in the output image thereby producing less accurate results. However if done correctly, good results are almost guaranteed.

Another problem associated by the sequential search algorithm is the way in which the ties are broken in the event that the vertical model V, the horizontal model H or the diagonal models D1 and D2 are equal. The model that gives the smallest distance from the zero crossing boundaries is chosen. However, this does not necessarily mean that the correct decision was made or reached at, as noted in [1].

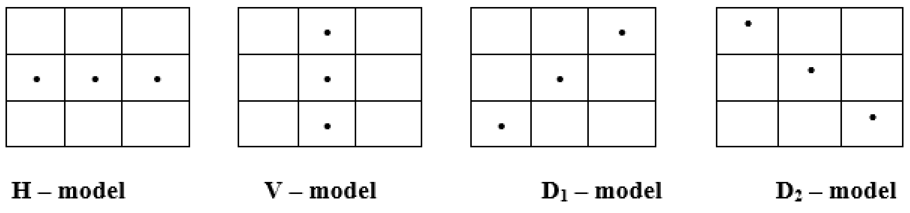


Figure 2.4: Four edge models on a 3x3 neighborhood.

Some researchers [2, 26, 35, 36] noted that the results presented by the sequential search algorithm are promising; however, the excessive CPU time and the large number of parameters that have to be adjusted before using the algorithm discourage its use.

2.3 Edge Linking By a Directional Potential Function (DPF)

Zhu et al. [10] proposed an algorithm by the potential function method that originated in physics. Their algorithm models an edge map as a potential field with energy dispositions at the detected edge positions. Pixels located at the broken edge points are charged with a potential force of energy proportionate to their relative distances and directions of neighboring pixels [10].

The algorithm looks at neighboring pixels, e.g., 3x3 or 5x5, in the whole image, and then links the segment to its most potentially connected segment [37]. This technique uses neighboring pixel information to help labeling an image pixel, it exploits the fact that noise pixels are normally not supported by their neighbors and as a result it uses this fact to suppress noise and reinforce detected structures. The drawback of this method is that it only deals with small gaps and no global shape model is involved in the process [38].

Tang et al. [39] noted that Zhu et al. [10] detected underlying boundaries by minimizing the directional potential function. They [39] also noted that these approaches are best suited at contour grouping in noisy images and fail to a great degree when it comes to dealing with clustered and textured regions.

Edge linking by DPF works as follows: Let x_i be an edge pixel at position (x_i, y_i) in an image $I(x, y)$. The assumption is that only positive energy is deposited in the image and as such non-edge pixels have null energy deposits. Energy charge at pixel x_i is given by equation 2.4:

$$q(x_i) = q(x_i)d(x_i) \tag{2.4}$$

where $q(x_i)$ is the energy charge and $d(x_i)$ is the directional component of the energy charge. The energy force is calculated as in equation 2.5:

$$g(x, x_i) = cq(x_i)\frac{cos(\alpha)}{||x - x_i||}n(x, x_i) \tag{2.5}$$

where $g(x, x_i)$ is the potential force generated by energy charge at point x_i, α is the angle between vector $d(x_i)$ and $x - x_i$.

The connection of edge points is inducted by the accumulation of $g(x, x_i)$ at the edge points x_i and x_j [10]. The accumulation of forces leads to the broken edges competing with each other to be included as the edge pixels. The process is repeated several times leading to some pixels (x and x_i) being discarded as edges and others being regarded as edge pixels.

2.4 A Very Large Scale Integration Architecture for Real-Time Edge Linking

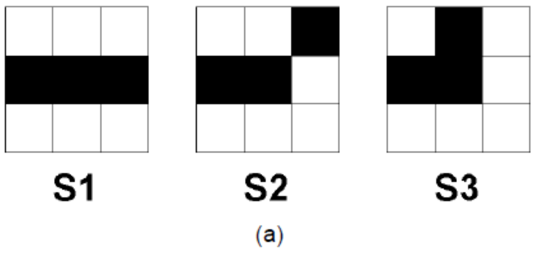
Hajjar and Chen [40] proposed a real-time algorithm and its VLSI implementation for edge linking. Their method is based on break point's direction and the weak level points. They define break points as a point at which an edge line is terminated. The gaps in the edges are filled according to the distance between the two compatible break points. The two compatible break points with the smallest distance are filled. Their approach has the advantage that it is simple and it increases significantly the level of connected pixels in the edge structure as observed by [2]. Their approach is based on a fixed scanning window and as such their implementation does not guarantee closed contours [2].

The break point is determined using the criteria given in equation 2.6 for a 3x3 window :

$$Break(x,y) = \begin{cases} 1, & \text{if } I_0 \sum_{i=1}^{8} I_i \sum_{j=(i+2)\%8}^{(6+i)\%8} I_j \neq 0 \\ 0, & \text{otherwise} \end{cases} \tag{2.6}$$

where I_i represents the existence of an edge point at position i.

Once the break points have been determined, their directions have to be determined as well. The direction of the break point takes one of the 8 directions depending on the preceding edge that it is connected to, as shown in Fig 2.5.



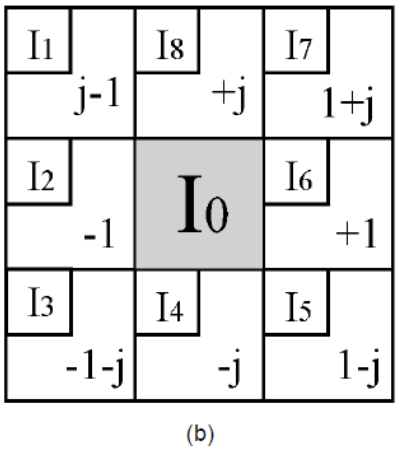


Figure 2.5: (a) The basic edge structures. (b) Break point direction representation.

They defined eight directions by using complex number notations as shown in Fig. 2.5(b). The real part of a break-point direction represents the horizontal shift that the linking edge point should take. The imaginary part represents the vertical shift. A break point at position (x, y) is said to have a direction a + jb, where a, b $\in -1, 0, +1$, if it is connected to a previous edge point located at the position (x - a, y - b) [40].

2.5 Computational Approach for Edge Linking

Ghita and Whelan [2] proposed an edge linking algorithm based on local information. After edge detection and selection of a threshold, an iterative stage of edge thinning follows.

First, small gaps will be closed and filled. The endpoints are recovered and labeled. The endpoints are linked together using local information, which takes into account the Euclidean distance based on the edge points to be linked (2D distance), and two reward coefficients: (1) if the points (to be linked) are both end points and (2) if the direction associated to the points to be linked is opposite from each other [26]. Sappa and Vintimilla observed that this technique proposed for edge linking does not guarantee closed contours [26]. Lu and Chen observed that a mask is adopted in the algorithm to acquire the direction of end points and in order to estimate the cost of the linking line [3].

Generally these methods are easy but their main drawback is that they return incomplete edge structures [3]. The diagram below summaries the algorithm.

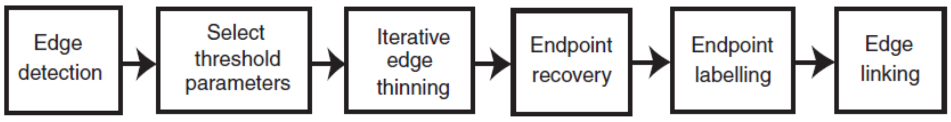


Figure 2.6: Outline of computational approach for edge linking courtesy of [2].

2.6 Adaptive Mathematical Morphology for Edge Linking

Shih et al. [33] applied mathematical morphology for edge linking to fill in the gaps between edge segments. Shih and Cheng [33] applied adaptive structuring to dilate the broken edges along their slope direction [9]. Thinning and pruning are applied. However, the common problem of thinning is that it distorts the image as noted by [26]. Shih and Cheng [33] solve this problem by iteratively linking the edges to ensure that broken edges are linked up gradually and smoothly so that shape of the object is not altered by the previous process of pruning and thinning. The algorithm steps are summarized in Table 2.2:

Table 2.2: Adaptive Mathematical Morphology Algorithm steps

1. Removing noisy edge segments
2. Detecting all the endpoints
3. Applying adaptive dilation operation at each endpoint
4. Thinning
5. Branch pruning
6. Decision The program terminates when no endpoints exist or when maximum number of iterations have been reached

2.7 Edge Point Linking by Means of Global and Local Schemes

Sappa and Vintimilla [26] suggested a technique for linking edge points to create closed contours. They use the original intensity image and the edge map as input to their algorithm. Their algorithm consist of two steps. The first step uses some global measure to compute the connecting edge point representation [26]. Sappa and Vintimilla proposed to use the Euclidean distance in 3D space taking into account the intensity (not only the point position in the edge map), unlike Ghita and Whelan [2] who used 2D Euclidean distance. The linking cost between two edge points (E(i,j), E(u,v)) is defined as follows:

$$LC(i,j)(u,v) = \|(i,j,I(i,j)) - (u,v,I(u,v))\| \tag{2.7}$$

where LC is the linking cost, which represents the 3D distance between the points to be linked, $I(r,c)$ is a 2D intensity array, i and u are rows, j and v are columns.

Their approach has an advantage that it is more accurate than only taking point positions in the edge map, which could lead to wrong results [26]. The last step is concerned with generating closed contours by linking broken edges using a local cost function. The first stage is based on graph theory and the second stage relies on local information. The spurious edges are removed by a morphological filter [26].

2.8 Edge Detection Improvement By Ant Colony Optimization

Edge detection improvement by ant colony optimization as the name suggests tries to improve edge detection by using ACO (Ant Colony Optimization). Initially ants are placed at the broken endpoints, the number of ants corresponding to the broken edges. Wong et al. observed that the algorithm uses the original intensity values to guide the ants [41], which are susceptible to noise as observed by [42].

Fig. 2.7 summaries the algorithm [3].

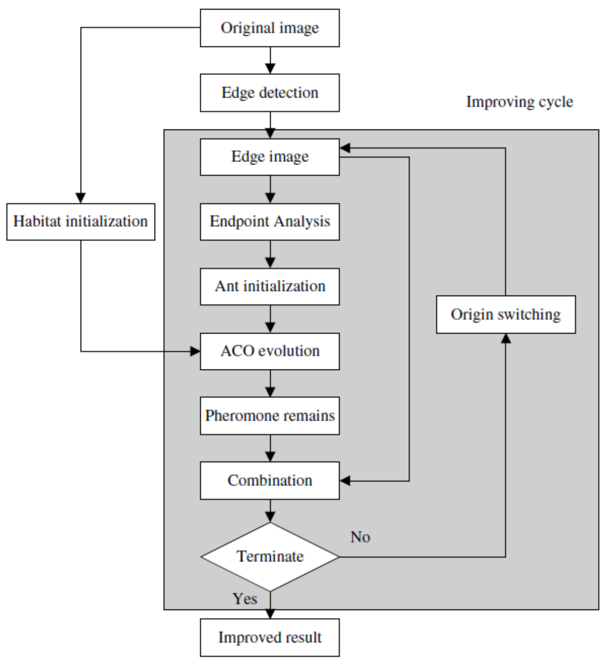


Figure 2.7: Flow chart for edge detection improvement by Ant Colony courtesy of [3].

2.9 Ant based Edge Linking Algorithm

Ant system is a swarm-based algorithm that exploits the self organizing nature of real ant colonies and their foraging behaviour to solve discrete optimization problems [43]. The ant based edge linking algorithm is based on mimicking the behaviour of biological ants. Biological ants leave a pheromone trail that attracts other ants when they are searching for food, in the same way the ant based algorithm uses artificial ants. The nodes (pixels) will be the food source. The artificial ants leave pheromone trail which in turn attracts other ants. Negative feedback is sent by pheromone evaporation which distracts other ants from following the same route. Initially, a number of ants corresponding to the number of endpoints are placed and each endpoint will be the starting pixel of each ant. The ant system uses the grayscale image and the Sobel edge image as its inputs, and the resultant image will be a sum of the Sobel edge image and the connecting edges [9]. A block diagram for the ant based linking algorithm is shown in Fig. 2.8 summarizing the processes involved in the ant based linking algorithm.

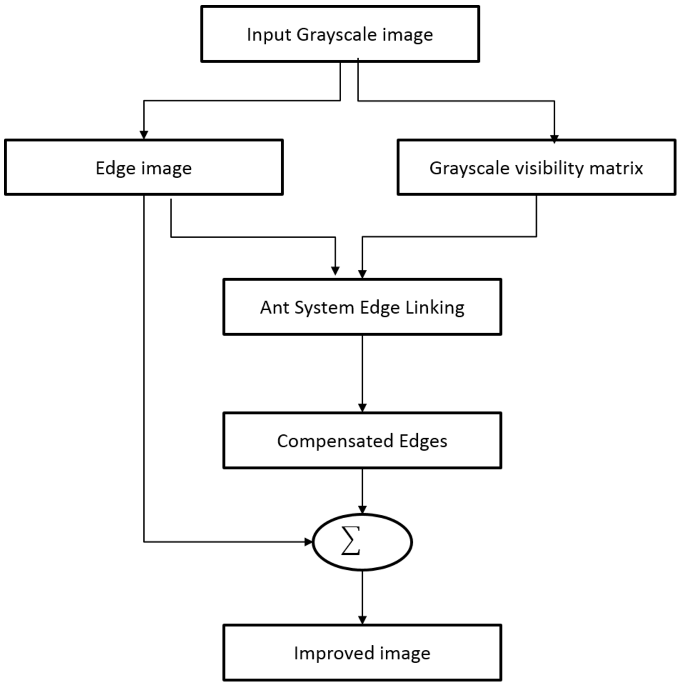


Figure 2.8: Block diagram of the Ant Based Edge Linking algorithm.

The ant based algorithm uses the original grayscale image and the Sobel edge image as inputs. The grayscale image is used to calculate the visibility matrix as shown in equation 2.8. It is used as the initial pheromone trail. Applying the visibility matrix as the initial pheromone trail has the advantage that it enhances the probability of a pixel belonging to the edge to be chosen and thereby reducing the computational overload [9]. On the other hand using the original grayscale image may present false edges being detected when the image has too much noise as shown in the formula (equation 2.8) for calculating the visibility matrix. However, this might be overcome by using the smoothed image. The grayscale value at $p(i, j)$ is calculated as follows:

$$\xi_{ij} = \frac{1}{I_{max}}.max\begin{bmatrix}|I(i-1,j-1) - I(i+1,j+1)|,\\ |I(i-1,j+1) - I(i+1,j-1)|,\\ |I(i,j-1) - I(i,j+1)|,\\ |I(i-1,j) - I(i+1,j)|\end{bmatrix} \tag{2.8}$$

where I_{max} is the maximum grayscale value in the image. Therefore, ξ_{ij} is normalized in $(0 \leq \xi_{ij} \leq 1)$.

The ant based algorithm does not use a global threshold; it uses a fitness value calculated as follows:

$$f_k = \frac{\overline{\xi}}{\sigma\xi.N_p} \tag{2.9}$$

where $\overline{\xi}$ and $\sigma\xi$ are the mean value and the standard deviation of the grayscale visibility of the pixels.

The fitness value is a measure of how fit a pixel is to the route it is supposed to belong to. This in turn gives other advantages in that weak edges may not be discarded as is the case when using a global threshold, so it avoids the shortcomings of a global threshold. The fitness value of a route is dependent on the mean value and the standard deviation of the grayscale visibility of the pixels in the route and the total number of pixels belonging to that route [9].

Node transition is based on probability. The probability of an ant following a route of some sort is a function of what the ant can see (visibility of the pixel from the endpoints) or the proximity to that particular endpoint and the pheromone trail laid as shown in the equation 2.10. Probability distributions change on each iteration. Probabilities are not constant and this can be a problem causing the algorithm to take more time to converge.

Pixel transition rule (probability) is defined as follows:

$$P_{ij}^{k} = \begin{cases} \frac{(\tau_{ij})^{\sigma}(\eta_{ij})^{\beta}}{\sum_{h \notin tabu_k}(\tau_{ih})^{\sigma}(\eta_{ih})^{\beta}}, & \text{if } j \notin tabu_k \\ 0, & \text{otherwise} \end{cases} \tag{2.10}$$

where τ_{ij} and η_{ij} are the intensity of the pheromone trail on edge (i, j) and the visibility of the node j from node i, respectively. $(\tau_{ij}, \eta_{ij} > 0; \tau_{ij}, \eta_{ij} \in \Re, \text{for } \forall i, j)$. α and β are the parameters that control the importance of the pheromone trail and the visibility, respectively $(\alpha, \beta > 0; \alpha, \beta \in \Re)$. $Tabu_k$ list contains the nodes that have already been visited by the k_{th} ant.

Tempering with α and β results in different outcomes. A large α/β ratio results in ants choosing the strongest edges. β parameter is of great importance as it inclines the ants towards closest endpoints [9]. At the end of each iteration the pheromone trail will be updated and a positive feedback will result in pheromone accumulation and negative feedback will result in pheromone evaporation. This has the advantage that it reduces poor quality solutions (wrong edges being detected).

Pheromone trail update rule:

$$\tau_{ij,(new)} = (1-p)\tau_{ij,(old)} + \sum_{k=1}^{m} \Delta\tau_{ij}^{k} \tag{2.11}$$

where p is the pheromone evaporation rate $(0 < p < 1 : p \in R)$, and $\Delta\tau_{ij}^{k}$ is the amount of pheromone laid on edge (i, j) by the k_{th} ant and is given by:

$$\Delta\tau_{ij}^{k} = \begin{cases} \frac{f_k}{Q}, & \text{if edge}(i,j)\text{is traversed by the } k^{th}\text{ant at the current cycle} \\ 0, & \text{otherwise} \end{cases} \tag{2.12}$$

where f_k is the fitness value of the solution found by k^{th} ant and Q is a constant.

One of the novelties of the ant based algorithm is that its convergence is guaranteed. However, one of the shortfalls is that there is no certainty on the time to converge. The number of iterations to be done is image dependent. Large resolution images require less iterations while low resolution images require quite a large number of iterations. As observed and stated by A. Jevtic et al. [9] the number of iterations that gave satisfactory results were 100 iterations, and lower resolution images such as 128x128 pixels required larger number of iterations. This also means the iterations have to be adjusted for each image and this can be cumbersome.

3. PROPOSED EDGE LINKING ALGORITHMS: CANNYSR, PEL

In this chapter we propose two edge linking algorithms: The first is an edge linking algorithm to convert Canny's binary edge maps to edge segments using the Smart Routing (SR) step of Edge Drawing (ED); thus the name CannySR [44]. The second is an edge linking algorithm that just takes in a binary edge map generated by any arbitrary traditional edge detection algorithm and converts it to a set of edge segments; filling in one pixel gaps in the edge map, cleaning up noisy edge pixel groups and thinning multi-pixel wide edge pixel formations in the process. The proposed edge linking algorithm walks over the edge map based on the predictions generated from its past movements; thus the name Predictive Edge Linking (PEL) [45].

Before we give a detailed explanation on how the proposed algorithms operate, we first give a brief overview of Canny and Edge Drawing [21, 46]. We then describe CannySR followed by PEL.

3.1 Canny, Edge Drawing and Smart Routing

In this section our goal is to give a brief overview of Canny, Edge Drawing and Smart Routing algorithms as they are the founding blocks of CannySR.

Table 3.1: Psuedocode for Canny

```
Symbols used in the algorithm:
I: Input grayscale image
sigma: of the Gaussian smoothing kernel
lowThresh: Low gradient threshold
highThresh: High gradient threshold
S: Smoothed image
G: Gradient magnitudes
Dir: Edge directions
BEM: Binary edge map

Canny(I, sigma, lowThresh, highThresh)
    1. S = SmoothImage(I, sigma);
    2. (G, Dir) = ComputeGradient(S, Sobel);
    3. BEM = NonMaximalSuppression(G, Dir);
    4. Hysteresis(BEM, lowThresh, highThresh);
    5. Return BEM;
End-Canny
```

The pseudocode for Canny is given in Table 3.1. As seen from the algorithm, Canny takes in 4 parameters, i.e., the input image I, sigma of the Gaussian smoothing kernel, low and high gradient thresholds, and performs edge detection in 4 steps. In the following, we briefly explain each step.

(1) Smoothing: Given an image I, the image is first smoothed by a Gaussian kernel with a given sigma. The main objective of this step is to suppress noise and remove some noisy artifacts from the image.

(2) Computation of the gradient: The next stage involves determining the gradient magnitude and directions. The gradient magnitude and directions are calculated over the smoothed image S. Well known operators such as the Prewitt, Sobel and Scharr operators can be used at this step.

(3) Non-maximal suppression: In this stage only the local maxima are marked as edges. The edges are computed by a method known as non-maximum suppression, where a pixel is considered to be an edgel if its gradient magnitude is greater than its neighbors in the direction of its gradient. For example, if the gradient direction of a particular pixel is 90 degrees, then pixels to its north and south are compared with it. If the gradient magnitude of the current pixel is higher then both neighbors, the current pixel is marked to be a possible edgel. Otherwise, the pixel is eliminated. At the end of this step a binary edge map is obtained where the white pixels are the pixels which survived the non-maximum suppression process.

(4) Hysteresis: This is the last step which is aimed at retaining the true edges and eliminating false ones in the binary edge map (BEM). It uses two thresholds, a lower one and a higher one. Edgels whose gradient magnitude is smaller than the lower threshold are eliminated as false detections while those edgels that have their gradient magnitude greater than the higher threshold are retained as strong edges. Edgels that fall in between the higher and lower threshold are only considered weak edgels, and they survice only if they are linked directly or indirectly to strong edges. Otherwise the weak edgels are eliminated as false detections.

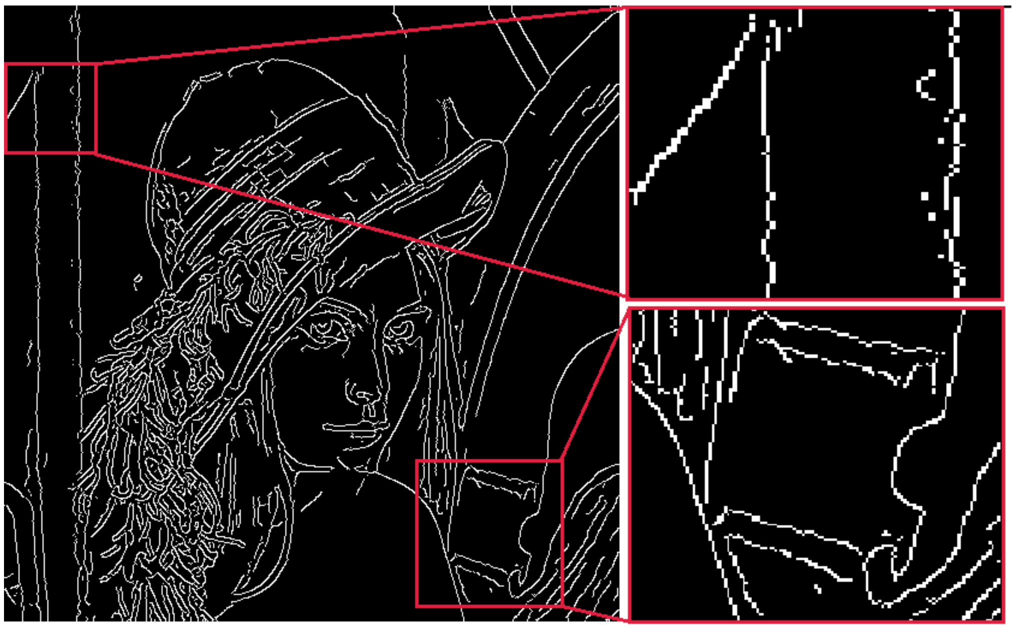


Figure 3.1: Edge map output by cvCanny for low and high threshold values of 20 and 40 respectively. The image was first smoothed by a Gaussian kernel with $\sigma = 1.5$ before cvCanny was called. A close-up view of a section of the Canny edge map, with missing, ragged and multi-pixel wide edgels.

Fig 3.1 shows the binary edge map for the famous Lena image. This edge map was obtained by the OpenCV implementation of the Canny edge detector (cvCanny), which is known to be the fastest Canny implementation. To obtain this edge map, the input image was first smoothed by a Gaussian kernel with $\sigma = 1.5$ (using cvSmooth from OpenCV), and cvCanny was called with low and high threshold values set to 20 and 40 respectively, and the Sobel kernel aperture size set to 3. Fig. 3.1 also shows the close-up views of two separate sections of the edge map to illustrate the low quality artifacts, which can be grouped in three categories as follows: (1) There are discontinuities and gaps between edgel groups as can clearly be seen in the close-up views of the two enlarged sections of the edge map. Some of these gaps need to be filled up. (2) There are noisy, unattended edgel formations and notch-like structures. This is more evident in the close-up view of the upper-left corner of the edge map. These noisy artifacts needs to be removed. (3) There are multi-pixel wide edgel formations in a staircase pattern especially around the diagonal edgel formations (both 45 degree and 135 degree diagonals). Such formations can be seen in many places in the edge map, and they need to be thinned down to 1-pixel wide chains.

Unlike traditional edge detectors, Edge Drawing (ED) is a unique algorithm in that it models the problem of edge detection similar to the childrens' dot completion games, where there will be marked anchor points, and the hidden picture is revealed after linking these boundary anchors. The algorithm does this by first computing the anchors, which are points of highest gradient in the edge map. The algorithm then links these anchors by a method called Smart Routing (SR), and outputs a set of segments each of which is a clean, contiguous, one pixel wide chain. The pseudo-code of ED is given in table 3.2.

Table 3.2: Psuedocode for Edge Drawing

```
Symbols used in the algorithm:
I: Input grayscale image
sigma: of the Gaussian smoothing kernel
thresh: Gradient threshold
S: Smoothed image
G: Gradient magnitudes
Dir: Edge directions
A: Anchors
ES: Edge segments

EdgeDrawing(I, sigma, thresh)
    1. S = SmoothImage(I, sigma);
    2. (G, Dir) = ComputeGradient(S, Sobel, thresh);
    3. A = ComputeAnchors(G, Dir);
    4. ES = SmartRouting(G, Dir, A);
    5. Return ES;
End-EdgeDrawing
```

(1-2) The first two steps of ED are similar to that of Canny algorithm.

The image is smoothed with the requested Gaussian sigma. The gradient magnitude and direction are calculated at each pixel. The algorithm differs from Canny in that the gradient directions are quantized into two directions, i.e., an edge can either be horizontal or vertical. Gradient magnitude of pixels which is less than the user supplied threshold are suppressed.

(3) ED computes the anchors, which are a set of points over the gradient map. These are the points where gradient is at peek and are assumed to be edgels.

(4) In the last step Smart Routing (SR) is employed. SR joins the anchors to form the edge segments.

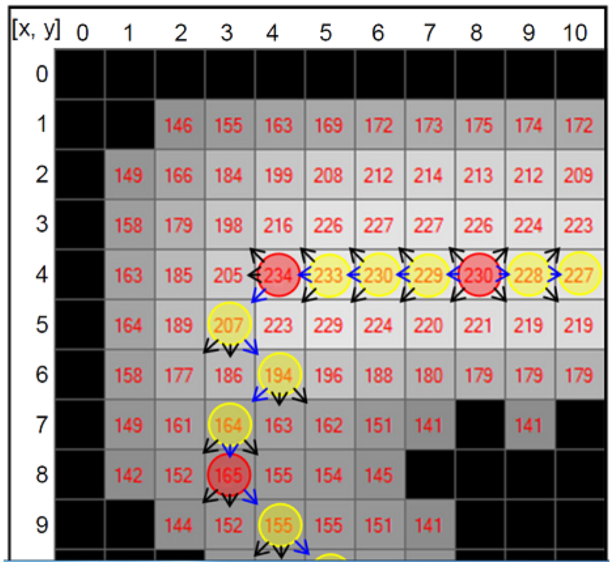


Figure 3.2: Smart Routing (SR) in action: Starting at an anchor (a red circle), SR follows a horizontal or a vertial path until it hits another anchor. SR stops when the end of the edge region is reached.

Fig. 3.2 illustrates SR in action. Starting at an anchor (a red circle), SR follows a horizontal or a vertical path depending on the edge direction until it hits another anchor. The walk continues until the end of the edge region is reached. In Fig. 2, assume that SR starts the walk at pixel (8, 4), whose gradient value is 230. Since the edge direction is horizontal, there is a walk to the left. At each step, only three neighboring pixels in the walk direction is considered, and SR moves to the pixel having the greatest gradient value. In the example, the pixel having the greatest value is (7, 4), so SR moves there. The walk continues horizontally to the left until pixel (3, 5) is reached. There the edge direction changes. Thereafter, SR starts walking vertically downwards until the end of the edge region is reached. Notice that as SR walk over the gradient map, the edgels are obtained as a chain of pixel linked one after the other. Therefore, the edge segment is a contiguous chain of pixels and it walks over the pixels having the largest gradient values. In a sense, this is like walking over the peeks of the gradient map mountain.

3.2 CannySR: An Edge Linking Algorithm to Convert Canny's Binary Edge Maps to Edge Segments

In this section we propose an edge linking algorithm to convert Canny's edge maps to edge segments using ED's Smart Routing (SR); thus the name Canny Smart Routing (CannySR). The motivation behind CannySR is to use a subset of the edgels in Canny's binary image as the anchors for SR (refer to Fig. 3.2), and output a set of edge segments each of which is a chain of pixels. The edge segment can then be used in other high level important tasks such as line [47], arc, circle, ellipse [48] and corner detection.

Table 3.3: Pseudocode for CannySR

```
Symbols used in the algorithm:
I: Input grayscale image
sigma: of the Gaussian smoothing kernel
lowThresh: Low gradient threshold
highThresh: High gradient threshold
MIN_SEG_LEN: Minimum segment length
BEM: Binary edge map
G: Gradient magnitudes
Dir: Edge directions
Anchors: Anchors
ES: Edge segments

CannySR(I, sigma, lowThresh, highThresh, MIN_SEG_LEN)
    1. BEM = Canny(I, sigma, lowThresh, highThresh);
    2. G = SmoothImage(BEM, 0.50);
    3. Dir = ComputeEdgeDirections(I, sigma);
    4.
    // Compute the anchors
    Set all Anchors to 0
    for y=2 to height-2 do
      for x=2 to width-2 do
          // Skip non-edgels
          if (BEM[y][x] == 0) continue;

          // Horizontal edgel group of 3
          if (BEM[y][x-1] && BEM[y][x+1]) Anchors[y][x] = 1;

          // Vertical edgel group of 3
          if (BEM[y-1][x] && BEM[y+1][x]) Anchors[y][x] = 1;

          // 45 degree edgel group of 3
          if (BEM[y-1][x+1] && BEM[y+1][x-1]) Anchors[y][x] = 1;

          // 135 degree edgel group of 3
          if (BEM[y-1][x-1] && BEM[y+1][x+1]) Anchors[y][x] = 1;
      end-for
    end-for
    5. ES = SmartRouting(G, Dir, Anchors, MIN_SEG_LEN);
    6. Return ES;
End-CannySR
```

The pseudocode for CannySR is given in Table 3.3. After the computation of the binary edge map (BEM) of an input image I using Canny with the user supplied parameters at step 1, the rest of the algorithm involves converting the obtained BEM to a set of edge segments using SR. We define the three things that SR needs to work, i.e., a gradient map, edge directions and anchors, as follows:

(1) Gradient map: We use a smoothed version of BEM as the gradient map. As seen from the pseudocode, a Gaussian kernel with $\sigma = 0.50$ is used for this purpose. The goal of this step is both to widen up the edgels and create narrow edge areas for SR to walk on, and also fill one pixel gaps between the edgels. The reason for using a small Gaussian kernel with a small sigma value is to prevent nearby but completely separate edgel regions to get connected, which would lead SR to jump to irrelevant edge regions during a walk and produce incorrect edge segments. Our experiments have shown that a Gaussian kernel with $\sigma = 0.50$ is a good choice for this purpose.

(2) Edge Directions: We compute the edge directions using the original image and the sigma of the Gaussian kernel that was used to smooth the image when BEM was obtained. That is, the original image is first smoothed by the Gaussian smoothing kernel having the supplied sigma, and then the edge direction for each pixel is calculated over this smoothed image by computing the horizontal and vertical gradients. The fact that edge directions have to be computed over the original image is a big drawback of this method. Not only do we need to have the original image for CannySR to work, but we also need to know how the image was smoothed before the BEM was obtained. That is, the edge linking method presented in this algorithm cannot be used if we only have a binary edge map and we do not know how it was obtained, or if we do not have the original image.

(3) Anchors: There are two alternatives here: (a) We can use all edgels in BEM as the anchors for SR, but our conclusion was that this produces some low quality crooked edge segments. (b) We can use a subset of the more stable edgels as anchors. This is what we propose as follows: Use an edgel as an anchor only if it is surrounded by two neighbor edgels, one in each edge direction. For example, within a horizontal edgel group of three, the middle pixel is taken to be an anchor if there is an edgel to the left and an edgel to the right. Similarly, within a vertical edgel group of three, the middle pixel is taken to be an anchor

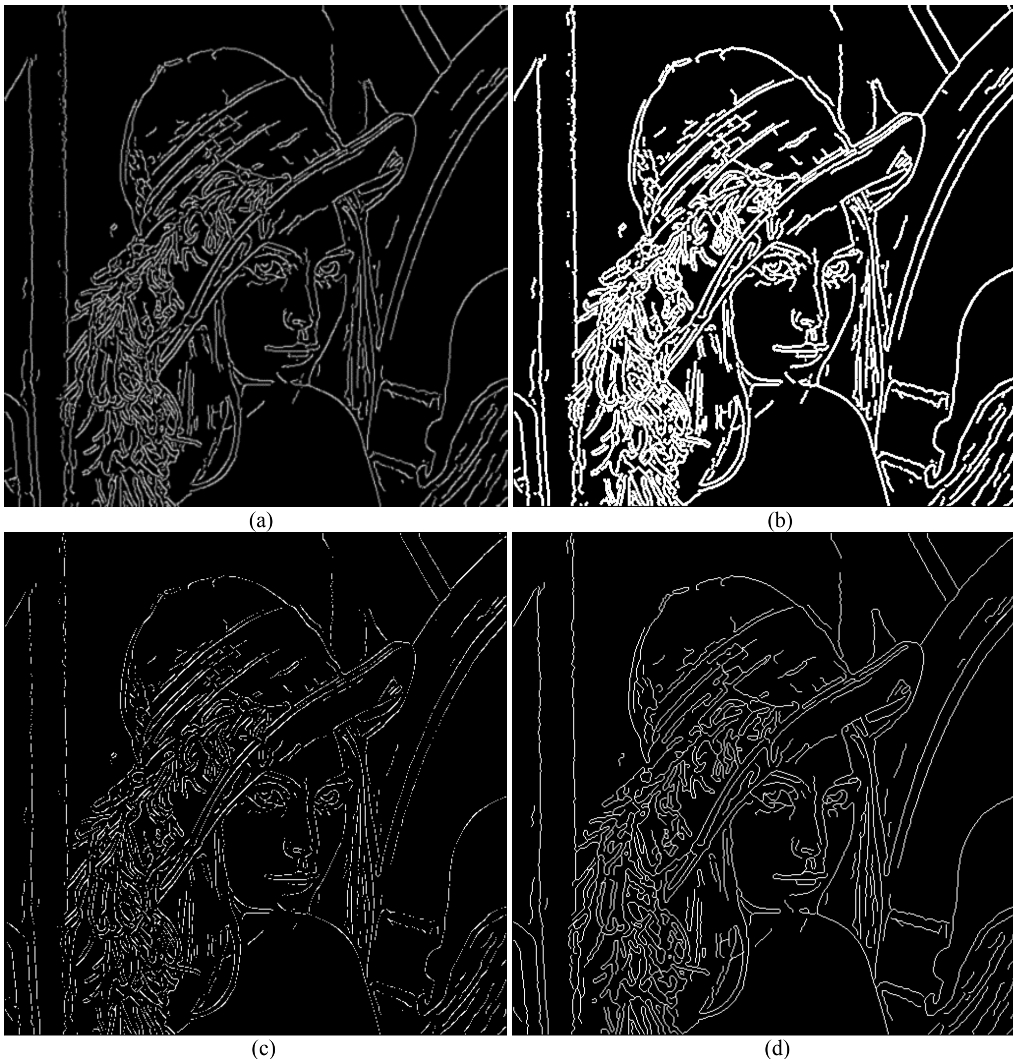


Figure 3.3: (a) Canny's BEM smoothed by a Gaussian kernel with $\sigma = 0.50$, (b) Thresholded smoothed edge map: the extended edge areas, (c) Anchors extracted from Canny's BEM, (d) Final edge segments after SR. Edge segments shorter than 8 pixels have been eliminated.

if there is an edgel upstairs and downstairs. The anchors for the two diagonal directions are computed similarly as shown in step 4 of the pseudocode.

Fig. 3.3 illustrates the steps of CannySR. Fig. 3.3(a) shows the smoothed BEM, which serves as the gradient map during SR. Fig. 3.3(b) shows the same smoothed edge map with non-zero values being set to 255. This, in a sense, is the extended edge regions during SR. That is, the final edgels will be located within these edge regions and will be located on top of the gradient map peeks. Fig. 3.3(c) shows the anchors computed at step IV of CannySR. Anchors are essentially a subset of the edgels in Canny's BEM and are assumed to be more reliable edgels due to our selection criteria. Finally, Fig. 3.3(d) shows the result of CannySR. Comparing Canny's BEM in Fig. 3.1 and the edge segments produced by CannySR in Fig. 3.3(d), we can clearly see the modal improvements and higher quality output of CannySR. We note that in Fig. 3.3(d), edge segments that are shorter than 8 pixels have been considered to be noisy artifacts and eliminated.

3.3 Predictive Edge Linking (PEL)

In this section, we propose a new edge linking algorithm named Predictive Edge Linking (PEL), which takes as input only the binary edge map (BEM) produced by any arbitrary traditional edge detector and returns a set of edge segments. Unlike CannySR, PEL neither requires the input image nor the sigma of the Gaussian smoothing kernel that was used to smooth the image before edge detection, which is an important advantage of PEL over CannySR.

Table 3.4: Pseudocode for PEL

```
Symbols used in the algorithm:
BEM: Binary edge map
MIN_SEG_LEN: Minimum segment length
ES: Edge segments
PEL(BEM, MIN_SEG_LEN)
    1. FillGaps(BEM);
    2. ES = CreateSegments(BEM);
    3. JoinSegments(ES);
    4. ThinSegments(ES, MIN_SEG_LEN);
    5. Return ES;
End-PEL
```

The pseudocode for PEL is given in Table 3.4. PEL only takes in a BEM as input and returns a set of edge segments (ES) as output. The algorithm consists of 4 steps, which are summarised below:

(1) In the first step one pixel gaps in BEM are filled.

(2) The second step involves the creation of edge segments

(3) In the third step edge segments whose endpoints are close to each other are then joined together to form longer edge segments.

(4) The fourth step involves thinning the multi-pixel wide edge segment to one-pixel wide edge segments. The minimum segment length supplied by the user specifies the length of the shortest segment to be returned by PEL. Any segments which are shorter than the minimum segment length are removed and not returned to the user. A detailed explanation of these steps is given in the following sections.

3.3.1 FillGaps: filling one pixel gaps in binary edge map

Recall from Fig. 3.1 that edge maps produced by traditional edge detectors contain gaps between edgel groups. Many proposed algorithms for edge linking found in the literature concentrate on this problem, and propose solutions to fill these gaps. Although our concentration in this paper is on obtaining edge segments, filling the gaps is also important. In this section, we propose a heuristic method to fill one pixel gaps between edgel groups.

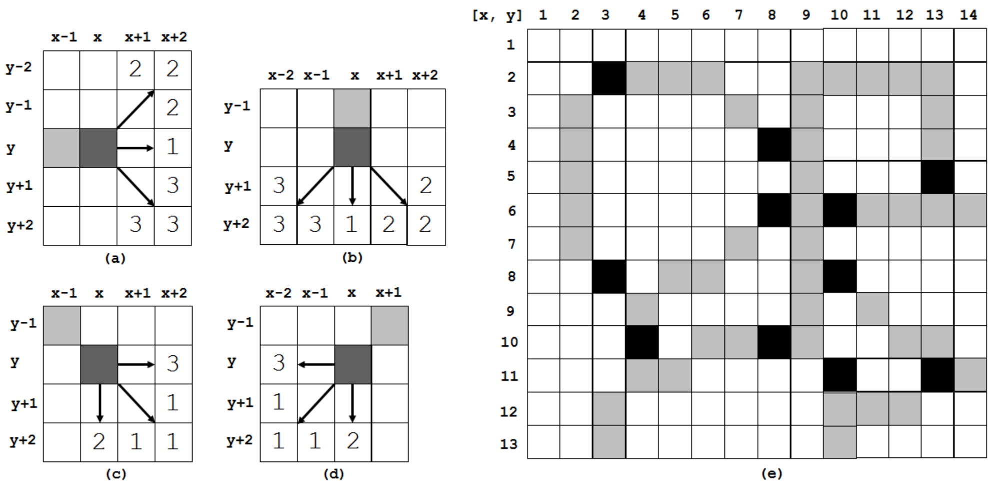


Figure 3.4: Filling one pixel gaps between the edgel groups. The dark gray pixels at (x,y) is the tip pixel of an edgel group, and the light gray pixel in each case is its neighbor. We perform a check along the direction where the tip is moving, and connect the tip to a neighboring edgel by filling the appropriate pixel.

Fig. 3.4 illustrates our heuristic for filling one pixel gaps between the edgel groups. Our idea is to first find the tips of the edgel groups, and then to connect each tip to a neighboring edgel that is one pixel away. The tip of an edge group is defined to be a pixel with only one neighbour. Of the eight possible scenarios, fig 3.1 depicts four cases, where the tip pixel marked in dark gray is located at (x,y), and its only neighbor is marked in light gray in each case. The other four cases (left, up, up-left, up-right) are simply symmetric versions of these four cases.

Our heuristic for connecting a tip pixel takes into account the direction the tip pixel is moving towards and connects it to a neighboring edgel group in that direction. For example, consider the case in Fig. 3.4(a), where the tip pixel is moving to the right because the neighbour pixel of the tip, marked in light gray, is located on the left at (x-1, y). Here we perform three checks to the right marked as 1, 2 or 3 in the figure. If the pixel at (x+2, y) is an edgel, then we connect the tip pixel to (x+2, y) by filling the pixel at (x+1, y). Otherwise, we check if one of the pixels at (x+1, y-2), (x+2, y-2) or (x+2, y-1) is an edgel, and if yes, then we connect the tip to one of these pixels by filling the pixel at (x+1, y-1). Finally, we check if one of the pixels at (x+1, y+2), (x+2, y+2) or (x+2, y+1) is an edgel, and if yes, then we connect the tip to one of these pixels by filling the pixel at (x+1, y+1). Although the other seven directions are not elaborated, we follow a similar procedure for each direction, and connect the tip pixel to a neighboring edgel in the direction that the tip is moving.

Fig. 3.4(e) illustrates how the proposed heuristic fills up one pixel gaps in the edge map. In the figure, light gray pixels are the original edgels in the input BEM, and the dark pixels are the filled pixels. Just to explain why the pixel (8, 10) is filled, consider the tip pixel at (7, 10), which is moving to the right because its only neighbour is located at (6, 10). According to Fig. 3.4(a), we need to first check the pixel at (9, 10), which is an edgel. Therefore, (7, 10) is connected to (9, 10) by filling (8, 10).

3.3.2 CreateSegments: linking contiguous edgels to create pixel chains

Now that some of the missing pixels in BEM have been filled up, we move on to the heart of the problem: that of linking the edgels in BEM and creating the edge segments, each a contiguous chain of pixels. The heuristic that we employ at this step is to start at an arbitrary edgel in BEM and create potentially two chains starting at that edgel: One in the forward direction and one in the reverse direction. We then combine these two chains together to create a single chain of pixels, which essentially makes up for one edge segment. We then start at another edgel and do the same thing until all edgels in BEM are converted to edge segments.

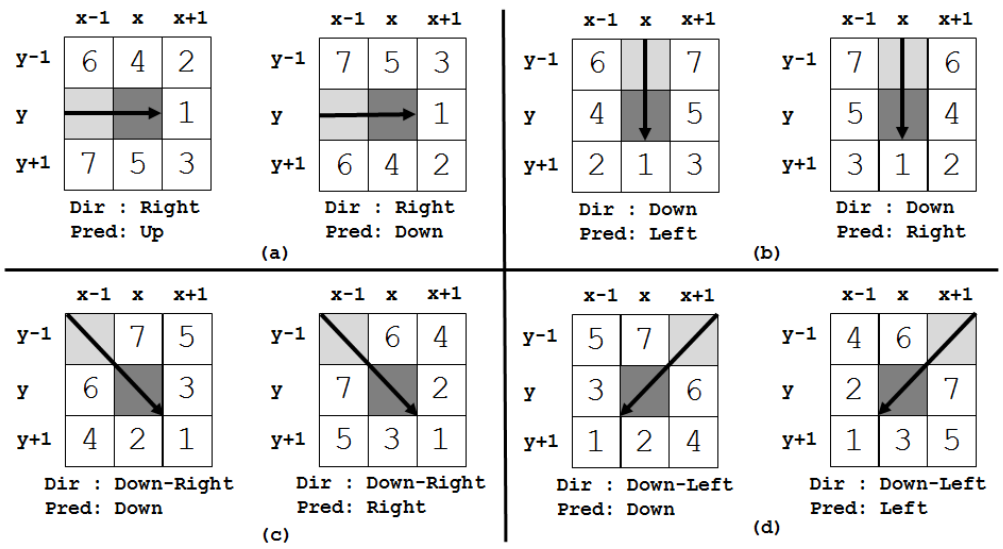


Figure 3.5: Walking in four directions with prediction. We are currently at pixel (x, y), marked with dark gray colour, and moving towards (a) Right, (b) Down, (c) Down-Right, (d) Down-Left. The other four directions, i.e., Left, Up, Up-Left, Up-Right, are simply symmetrical versions of these four directions respectively.

To create an edge segment, we perform eight-directional walk with prediction, therefore the name Predictive Edge Linking (PEL). The prediction is used when the current direction changes. By taking the last eight directions into account, the prediction engine tells us which direction to move on to after the current direction changes.

Fig. 3.5 illustrates the walk in four directions: (a) Right, (b) Down, (c) Down-Right and (d) Down-Left. The other four directions, i.e., Left, Up, Up-Left and Up-Right are simply symmetrical versions of these directions respectively and are not illustrated.

Starting with Fig. 3.5(a), we see a walk to the right. That is, we are currently at pixel (x, y), marked with dark gray colour, and we moved here from pixel (x-1, y) marked with light gray colour. Since the current direction is "right", we immediately check the pixel to the right, i.e., (x+1, y) regardless of our past moves. If there is an edgel at (x+1, y), then we add it to the current chain and move there. The current walk direction continues to be "right". If there is no edgel at (x+1, y), then we need to change the current direction and check the other six neighbours in some order. This is where the prediction comes into play. We can first check the "up-right" pixel at (x+1, y-1) or the "down-right" pixel at (x+1, y+1). To make this decision, we consult the prediction engine, which taking the last eight moves into account, tells us to either check the “up-right” or the “down-right” pixel first. If the prediction is “up”, then we first check the “up-right” pixel at (x+1, y-1) and move there if there is an edgel. The current direction is then changed from “right” to “up-right”. If the prediction is “down”, then we first check the “down-right” pixel at (x+1, y+1) and move there if there is an edgel. The current direction is then changed from “right” to “down-right”. Fig. 3.5(a) shows in detail the order in which the neighbours of the current pixel (x, y) are checked depending on the current prediction. While checking the neighbours in the depicted order, as soon as we encounter an edgel we move there and change the current direction accordingly. For example, if the current prediction is “up”, and there is no edgel at pixels marked 1, 2, 3, 4 but there is an edgel at pixel marked 5, then we move to (x, y+1) and change the current direction as “down”. Then in the next iteration of the loop, we will check the neighbours of the current pixel using the order shown in Fig. 3.5(b). The current pixel chain will come to an end when we check all 6 neighbours of the current pixel (x, y) and none has an edgel.

Although walking in all eight directions follow a logic similar to the one described in the previous paragraph, diagonal moves need a little more explanation. Consider the “down-right” walk depicted in Fig. 3.5(c). The first neighbour to be checked in the “down-right” pixel at (x+1, y+1) regardless of the prediction. If there is an edgel there, then we will move to that pixel and the current walk direction will continue to be “down-right”. In this case however, there is a little caveat that needs to be taken into account as follows. As we move to the “down-right” pixel marked as 1 in Fig. 3.5(c), most of the time we would also have an edgel at pixels marked 2 or 3 because the traditional edge detectors usually generate staircase edgel structures for diagonal edge groups. This can clearly be observed in Canny’s Lena edge map shown in Fig. 1. Since we need to collect all pixels in the input BEM during edge segment generation, as we make the “down-right” move in Fig. 3.5(c), we also check the pixels to the right and down, i.e., pixels (x+1, y) and (x, y+1) and if one of them contains an edgel, we also pick it up and add it to the current chain. If both neighbours have an edgel, only one if picked depending on the current prediction. It is important to note that this approach applies not only to the “down-right” move, but to all four diagonal moves. To be more specific, if we are making a “down-left” move and there is an edgel at the “down-left” pixel (x-1, y+1) marked as 1 in Fig. 3.5(d), then we check the neighbours (x, y+1) and (x-1, y). If any one of these contain an edgel, then it is picked up and added to the current chain before we move "down-left" to (x-1, y+1).

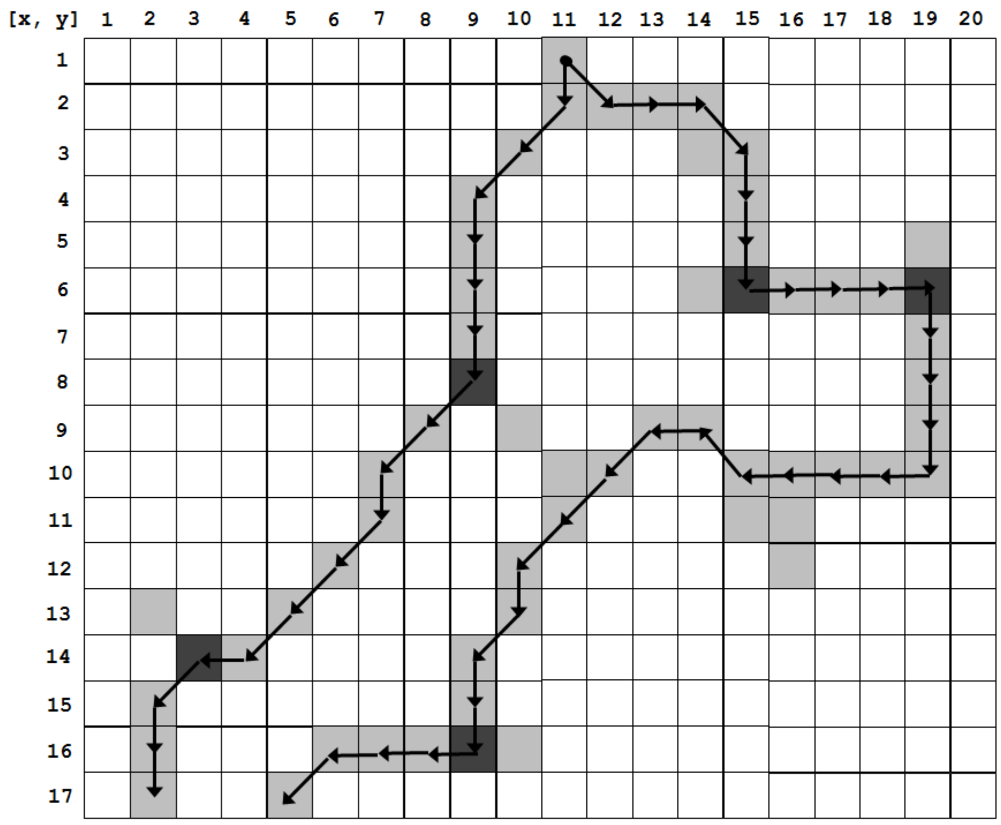


Figure 3.6: An example illustrating one edge segment creation starting at pixel (11, 1). Two chains are created, which are then combined together to create one edge segment. Coloured pixels have edgels. Dark gray pixels are places where the prediction guides us in the correct direction.

Fig. 3.6 illustrates the creation of an edge segment. Assume that we start the segment creation at pixel (11, 1). Looking at this pixel's neighbours, there are two possible walks: One going "down" through (11, 2), one going "down-right" through (12, 2). The arrows depict how PEL walks over the edgels (denoted with gray colour) and obtain two chains. At each pixel, the determination of the next pixel to move on to is made by the moves shown in Fig. 3.5. The dark gray pixels in Fig. 3.6 are where the prediction guides the walk one way rather than the other, and illustrates the importance of using the prediction to obtain longer chains.

Table 3.5: How PEL creates the chain going down from (11, 1)

Current Pixel	**Direction**	**Comment**
(11, 1)	Down	Check (11, 2): Full
(11, 2)	Down	Check (11, 3): Empty. Check (10, 3): Full
(10, 3)	Down-Left	Check (9, 4): Full
(9, 4)	Down-Left	Check (8, 5): Empty. Pred: Down. Check (9, 5): Full
(9, 5)	Down	Check (9, 6): Full
(9, 6)	Down	Check (9, 7): Full
(9, 7)	Down	Check (9, 8): Full
(9, 8)	Down	Check (9, 9): Empty. Pred: Left. Check (8, 9): Full
(8, 9)	Down-Left	Check (7, 10): Full
(7, 10)	Down-Left	Check (6, 11): Empty. Pred: Down. Check (7, 11): Full
(7, 11)	Down	Check (7, 12): Empty. Pred: Left. Check (6, 12): Full
(6, 12)	Down-Left	Check (5, 13): Full
(5, 13)	Down-Left	Check (4, 14): Full
(4, 14)	Down-Left	Check (3, 15): Empty. Pred: Down. Check (4, 15): Empty. Check (3, 14): Full
(3, 14)	Left	Check (2, 14): Empty. Pred: Down. Check (2,15): Full
(2, 15)	Down	Check (2, 16): Full
(2, 16)	Down	Check (2, 17): Full
(2, 17)	Down	End of chain

To better understand why PEL makes the moves depicted in Fig. 3.6, Table 3.5 gives the details of the decision engine as PEL creates the chain going down from (11, 1). During this chain creation, the prediction engine helps PEL to make the correct decision at two locations as follows: At pixel (9, 8), PEL is walking down and the pixel downstairs, i.e., (9, 9), is empty. At this point there are two alternatives: PEL can walk down-left to (8, 9) or down-right to (10, 9). PEL asks the prediction engine to guide it to the left or to the right. The prediction engine performs an analysis of the last 8 moves (in fact there are only 7 moves to this point, i.e., down, down-left, down-left, down, down, down, down) and concludes that PEL should check the down-left pixel before the down-right pixel and guides PEL to the down-left pixel at (9, 9). Similarly, at pixel (3, 14) PEL is moving left and the pixel to the left, i.e., (2, 14), is empty. At this point there are two alternatives:

PEL can walk down-left to (2, 15) or up-right to (2, 13). Again PEL consults the prediction engine, which recommends PEL to check the down-left pixel before the up-right pixel because over the last 8 moves, PEL made 2 "down" and 5 "down-left" moves but no "up" moves. Although not given in Table 3.5, a similar analysis can be performed for the chain on the right side of Fig. 3.6. At pixel (15, 6), the prediction engine guides PEL to the right rather than to the left; at (19, 6) PEL is guided down rather than up, and at (9, 16) PEL is guided to the left rather than to the right.

It is also important to note that after the two chains shown in Fig. 3.6 are obtained, they are joined together into one chain by taking the pixels from one chain in the forward direction and the pixels from the other chain in the backwards direction. This new chain makes up for the actual edge segment to be returned by PEL.

3.3.3 JoinSegments: extending nearby edge segments

Although PEL uses a prediction engine to try to make the correct decisions during a walk and obtain as long edge segments as possible, the structure of the input BEM may lead to two or more edge segments during segment creation that in fact should have been combined together into one edge segment.

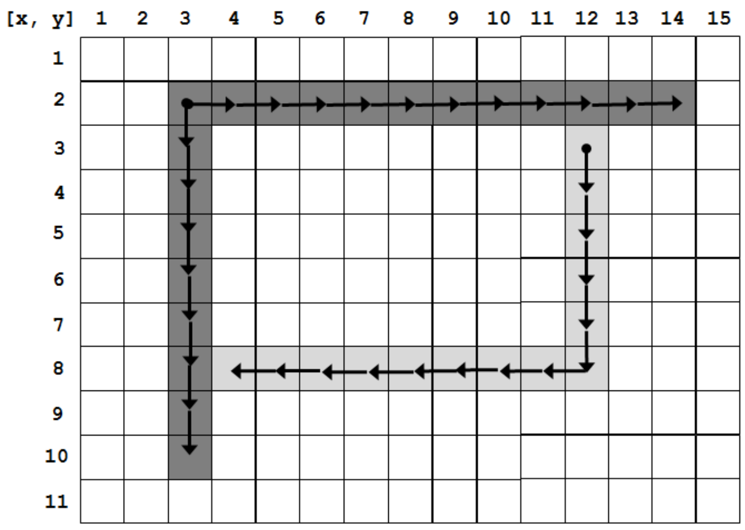


Figure 3.7: An example illustrating PEL creating two edge segments that should be joined together into one edge segment.

To understand the problem better, consider the illustration in Fig. 3.7.

When this BEM is fed into PEL, two edge segments are created: one marked with dark gray colour and the other marked with light gray colour. We can imagine however that this BEM belongs to the boundary a rectangular object and a single edge segment that traces the entire boundary of the rectangle would be better to return, rather than two edge segments as PEL's segment creation algorithm would do. So after segment creation, we have a new simple step named JoinSegments that groups neighbour edge segments and joins them together. We define that two edge segments are neighbours if the end point of one segment touches the other segment at some point and their end points are at most five pixels away from each other. Using this definition, the two edge segments in Fig. 3.7 are neighbours and can be combined together. To combine the two edge segments in Fig. 3.7, we first cut the superfluous pixels from the first segment; that is, pixels (13, 2), (14, 2) from one end, and pixels (3, 9), (3, 10) from the other. These are pixels beyond the point where the second segment touches the first. Joining the two neighbour segments after cutting of the superfluous pixels is simply done by attaching the two chains together. By joining neighbour edge segments together, PEL is able to create longer edge segments, which is important for high-level processing after edge linking.

3.3.4 ThinSegments: thinning down and cleaning up edge segments

The final step of PEL is to thin down the created edge segments and to remove the short ones from further consideration.

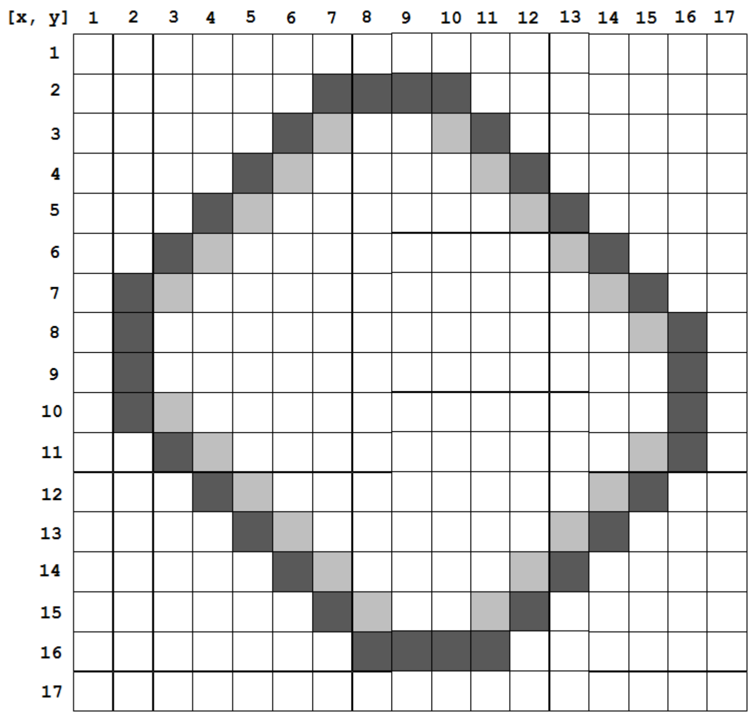


Figure 3.8: The need for thinning of the edge segments: Traditional edge detectors create multi-pixel wide edgels in a staircase pattern especially around diagonally placed objects. The goal of thinning is to remove superfluous edgels (light gray coloured ones) and return one-pixel wide edge segments.

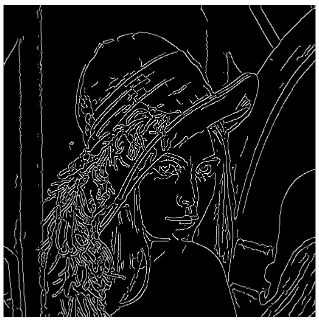
To see why thinning is necessary refer to Fig. 3.8, which shows a BEM output by a traditional edge detector for a rectangular object placed diagonally. As seen from the figure, there are multi-pixel wide edgels in a staircase pattern around the diagonals, which must be thinned down to one-pixel wide edgels. Specifically, the goal of thinning is to remove the superfluous edgels (light gray coloured pixels in Fig. 3.8) from the chain and to return a contiguous but one-pixel wide chain, i.e., the edge segment consisting only of the dark gray pixels in Fig. 3.8. This is an easy procedure to perform: We simply walk over the pixels of the edge segment and remove a superfluous pixel when we see the staircase pattern depicted in Fig. 3.8. After the edge segment goes through thinning, the last step is to check its length and remove the segment from consideration if it is shorter than the minimum segment length supplied by the user. This is important for cleaning up noisy edgel formations in a BEM. Our observation is that a minimum segment length of eight or ten pixels produces good, clean results without omitting any important details.

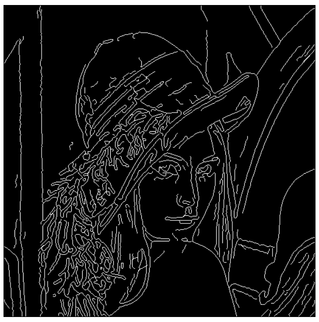
4. RESULTS AND EVALUATION

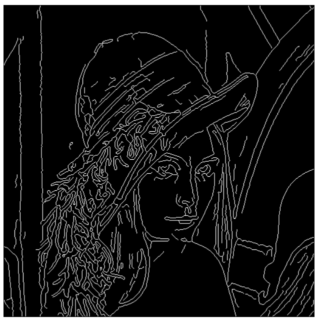
In this chapter we evaluate the performance of the proposed algorithms both qualitatively and quantitatively. We use the visual experiments for qualitative evaluation and the precision-recall within the framework of the Berkeley Segmentation Dataset (BSDS 300) [49, 50] for quantitative evaluation. Qualitative evaluation alone is not enough as there is more than meets the eye and further more qualitative evaluation does not express performance in numbers so we have to couple it with qualitative evaluation which measures performance numerically.

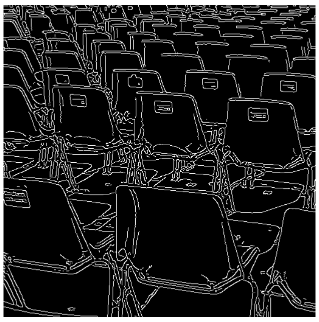
The Berkeley dataset is set-up as follows: The human segmented images provide the ground truth boundaries. They considered any boundary marked by a human subject to be valid. The ground truth constitute of a multiple segmentation of each image by different human subjects [50].

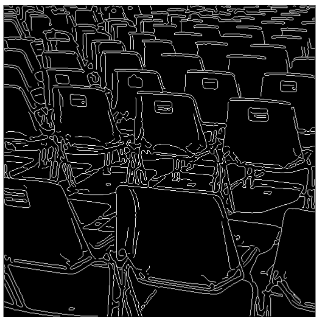
Precision is the probability that a machine-generated boundary pixel is a true boundary pixel. Precision measures how much noise is outputted in the final edge map. Recall is the probability that a true boundary pixel is detected. Recall measures how much of the ground truth is detected [50]. Precision-recall curves provide a good performance measure of the algorithm performance as they both quantify it but however we need to use a single number for measuring performance, this is achieved by using the F-measure which is the harmonic mean of precision and recall [50].

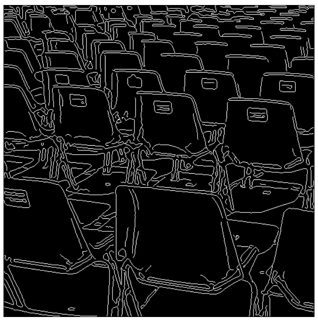

(a) OpenCV Canny (Low: 20, High 40)

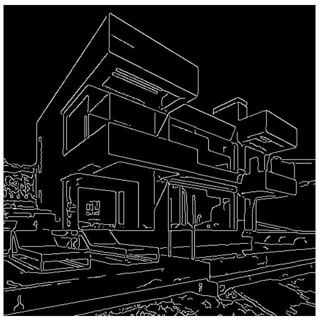

(b) CannySR: 303 edge segments

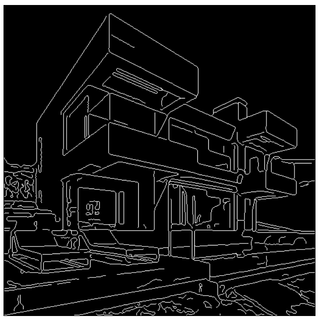

(c) PEL: 311 edge segments

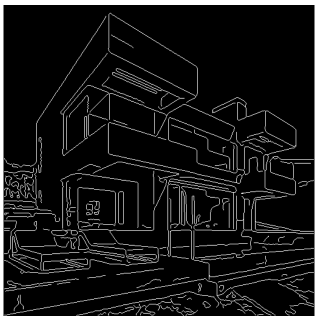

(d) OpenCV Canny (Low: 20, High 40)

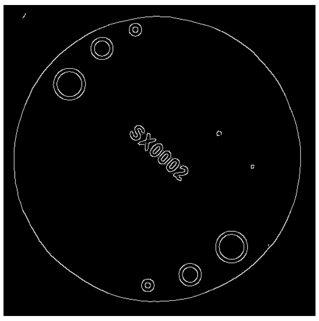

(e) CannySR: 405 edge segments


(f) PEL: 388 edge segments

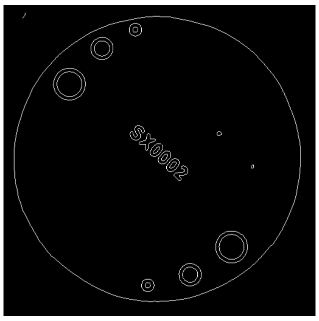

(g) OpenCV Canny (Low: 20, High 40)


(h) CannySR: 212 edge segments

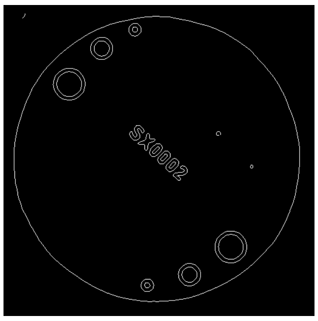

(i) PEL: 200 edge segments



(j) OpenCV Canny (Low: 20, High 40)



(k) CannySR: 25 edge segments



(l) PEL: 25 edge segments

Figure 4.1: Canny edge maps, CannySR and PEL edge segments for 4 images. The Canny edge maps were obtained by OpenCV Canny with low and high threshold parameters set to 20 and 40 respectively. The images were first smoothed by a Gaussian kernel with $\sigma = 1.5$. For CannySR and PEL, edge segments shorter than 8 pixels have been removed.

Fig 4.1 shows the Canny edge maps and the corresponding edge segments obtained by CannySR and PEL for 4 images. The Canny edge maps were obtained by OpenCV Canny implementation (cvCanny) with low and high threshold values set to 20 and 40 respectively, and the Sobel kernel aperture size set to 3. The images were first smoothed by a Gaussian kernel with $\sigma = 1{,}5$ (cvSmooth) before edge detection. For CannySR and PEL results look similar visually, it is very clear that both improve the modal quality of Canny's binary edge maps tremendously filling one pixel gaps and thus connecting disjoint edgel groups, removing noisy edgel formations, and thinning down multi-pixel wide edgel formations to 1-pixel wide edge segments. Last but not least, Canny's BEM has been converted to edge segments, which can now be used for higher level processing.

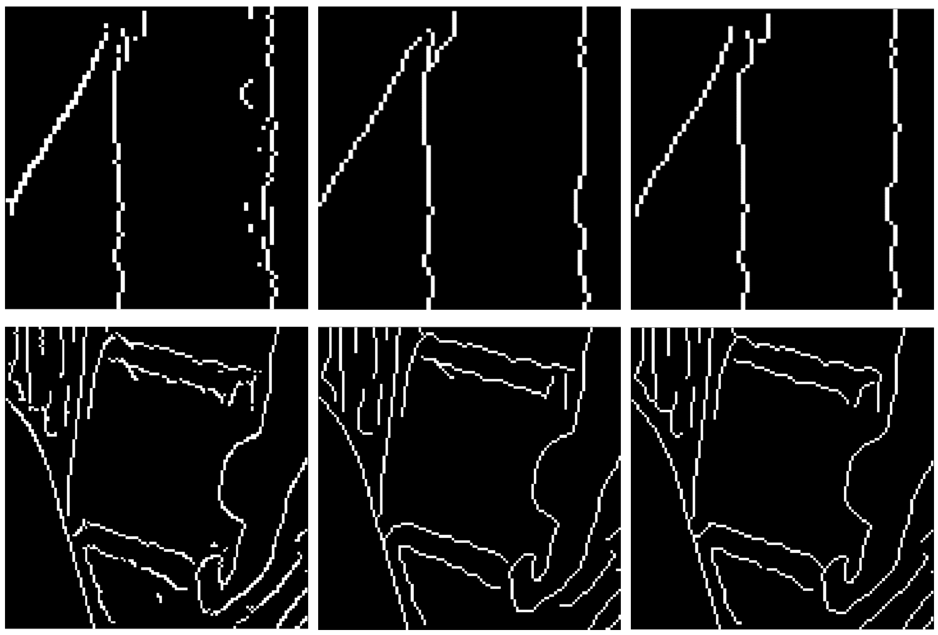


Figure 4.2: A close-up view of the two sections of Canny's edge map and the resulting edge segments by CannySR and PEL.

To better see the modal improvements made possible by CannySR and PEL to Canny's BEMs, consider Fig. 4.2 that shows a close-up view of the two sections of Canny's Lena BEM along with the resulting edge segments by CannySR and PEL. All three problems that we mentioned for binary edge maps have been solved: (1) One pixel gaps between edge groups have been filled up and long edge segments have been obtained. This is more evident in the vertical bar in the first row of Fig. 4.2. In Canny's BEM, this section contains a lot of 1 pixel wide gaps, all of which have been filled up and linked together by both CannySR and PEL as seen from their results. (2) Noisy edgel formations have been removed. This can clearly be seen in both Canny BEMs with many unattended, noisy edgel formations. All of these noisy edgel formations have been removed by both CannySR and PEL as seen in the second and third columns of Fig. 4.2. Recall that both CannySR and PEL have a minimum segment length parameter. Segments shorter than this threshold are removed after edge segment creation. In Fig. 4.2, edge segments shorter than 8 pixels have been removed as noise. (3) Multi-pixel wide edgel structures have been thinned down to one-pixel wide edge segments. This is more evident especially in diagonal edgel formations (both 45 degree and 135 degree diagonals). Looking at these diagonal edgel formations, we see the staircase pattern in Canny's BEMs, whereas both CannySR and PEL thin these edgel groups to 1-pixel wide edge segments as seen in the second and third columns of Fig. 4.2. All and all, it is very obvious from Fig. 4.1 and 4.2 that both CannySR and PEL greatly improve the modal quality of BEMs of traditional edge detectors. It is also important to stress once again that in addition to improving the modal quality of BEMs, both CannySR and PEL return the result as a set of edge segments, each of which is a contiguous chain of pixels. This makes it possible to post-process these segments for such higher level applications such as line, arc, circle, ellipse detection, image segmentation, edge segment validation etc.

Table 4.1: Running times of OpenCV Canny, CannySR, PEL and ED for the 4 test images in Fig. 9 on a Core i7-3770 CPU

Image (512x512)	**Canny (ms)**	**CannySR (ms)**	**PEL (ms)**	**ED (ms)**
Lena	5.20	5.64	2.62	4.32
Chairs	5.40	5.16	2.74	4.57
House	4.40	5.16	1.98	3.80
Circle	3.80	3.90	1.23	3.13

Table 4.1 shows the running time of OpenCV Canny, CannySR, PEL and ED for the 4 test images in Fig. 4.1. The running times were obtained on a PC with a Core i7-3770 CPU running at 3.40 GHz. For a fair comparison with PEL, the running times for CannySR include just the edge linking steps and not the edge detection step by Canny. We see from the table that CannySR takes as much time as Canny, while PEL is at least 2 times faster

than CannySR. Given that PEL's performance is as good as CannySR, if not better, and PEL requires but the binary edge map to be linked, we definitely recommend PEL over CannySR. Table 4.1 also gives the running time of Edge Drawing (ED) for comparison. We see that ED is faster than OpenCV Canny in all cases and is a natural edge segment detector. To obtain the edge segments for an image by first running Canny to get a binary edge map and then PEL to convert the binary edge map to edge segments would obviously be costlier. But in any case, PEL takes a very small amount of time and is very useful in converting binary edge maps to edge segments.

In the rest of this section our goal is to quantitatively evaluate the performance of the proposed edge linking algorithms to see the amount of improvements made possible by CannySR and PEL over Canny, which is the most widely used edge detection algorithm, and to compare and contrast their performance to that of Edge Drawing (ED), which is a natural edge segment detection algorithm. To this end, we make use of the Berkeley Segmentation Benchmark Dataset (BSDS 300) [49, 50] and its precision-recall evaluation framework.

BSDS has 300 images with 5 to 10 human annotated boundary ground truth information for each image. 200 of the images are test images and are used to tune up an algorithm's parameters. The other 100 images are used for testing an algorithm's performance. Let the boundaries returned by an algorithm for an image be A, and the ground truth boundary information be GT. Then precision P, recall R and $F-score$, which is essentially the harmonic mean of precision and recall, are defined as follows:

$$P = \frac{(A \cap GT)}{A} \tag{4.1}$$

$$R = \frac{(A \cap GT)}{GT} \tag{4.2}$$

$$F - score = \frac{2PR}{(P + R)} \tag{4.3}$$

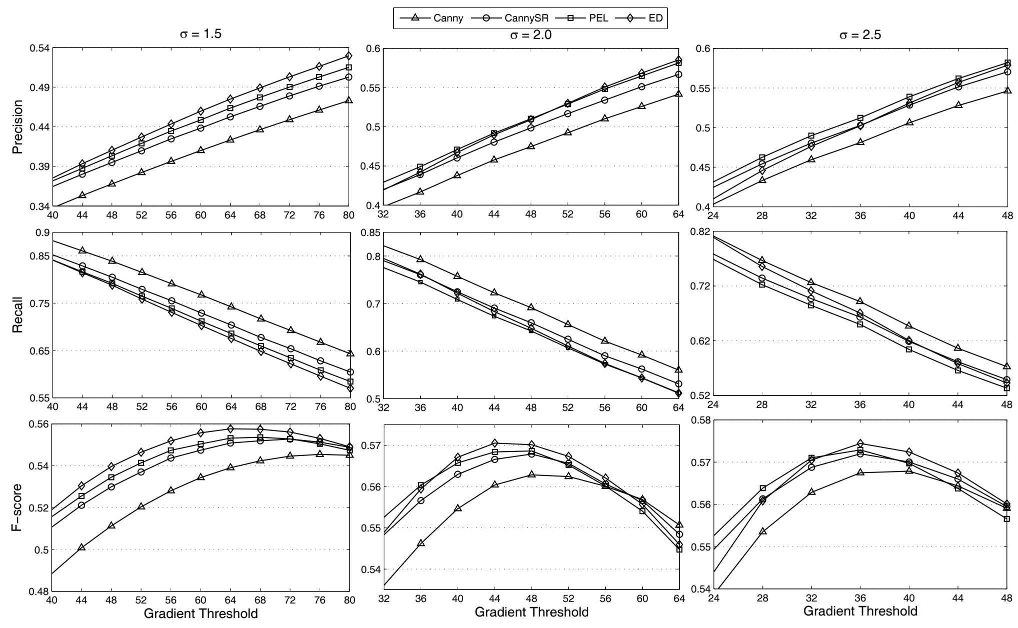


Figure 4.3: Precision, Recall and F-score curves for three Gaussian smoothing kernels with different sigma values as the gradient threshold changes for Canny, CannySR, PEL and ED. In all cases, ED produces the best F-score values with CannySR and PEL being close but much better than Canny.

Fig. 4.3 shows the precision, recall and F-score curves for Canny, CannySR, PEL and ED as the gradient threshold is increased. Each column in the figure represents the results for a Gaussian smoothing kernel with a different sigma value. Specifically, in the first column, an input image is smoothed with a kernel with $\sigma = 1.5$ before edge detection is performed. In the second and third columns, the smoothing sigma is increased to 2.0 and 2.5 respectively. The x-axis in the graphs, i.e., the gradient threshold, is the threshold used to suppress the pixels having a gradient value smaller than the threshold. To obtain the results, we fix the gradient threshold at a specific value and use the same threshold for all images in BSDS test set for Canny and ED. Then the threshold is increased and a new set of results are obtained until the maximum gradient value is reached.

As seen from Fig. 4.3, ED produces the best F-score values with CannySR and PEL being close but much better than Canny in all cases. The reason for CannySR and PEL's performance improvement can be seen in the precision curves. Since CannySR and PEL clean up Canny's edge maps to a great extent (as can also be visually observed in Fig. 4.1 and 4.2), the precision curves jump up for CannySR and PEL compared to Canny. Although the recall performance for CannySR and PEL drops a little compared to Canny, the big improvements in precision performance compensates the loss in recall resulting in a much better F-score. We can also observe that ED outperforms all other algorithms for most threshold values.

Table 4.2: Best F-scores for each algorithm for three Gaussian smoothing kernels with different sigma values

Gaussian Sigma	**Best F-score**			
	Canny	**CannySR**	**PEL**	**ED**
1.5	0.5454	0.5527	0.5536	0.5576
2.0	0.5628	0.5678	0.5686	0.5705
2.5	0.5678	0.5719	0.5728	0.5744

Table 4.2 lists the best F-score values for each algorithm for three Gaussian smoothing kernels. We can see from the table that both CannySR and PEL substantially improve the performance of Canny while ED performs the best.

5. CONLUSIONS

In this thesis we propose two edge linking algorithms. The first algorithm makes use of the Smart Routing (SR) step of the recently proposed edge segment detection algorithm, Edge Drawing (ED), to convert Canny's binary edge maps to edge segments; thus the name Canny SR. Both visual and quantitative experiments show that CannySR improves the modal quality of the binary edge maps produced by traditional edge detectors such as Canny. The problem with CannySR though is that in addition to the binary edge map on which the edge linking will be performed, it also requires the original source image and the Gaussian sigma that was used to smooth the image before the edge map was obtained. Although this may not be a problem in certain cases, it is a big problem if we only have the binary edge map or if we do not know how it was obtained. The second proposed edge linking algorithm, named Predictive Edge Linking (PEL) that requires only the binary edge map to work, thus overcoming the limitations of CannySR. PEL starts at an arbitrary edgel in the edge map and walks over the neighboring edgels until the end of an edgel chain is reached. During a walk, PEL consults a prediction engine that, based on the last several movements, makes a recommendation for the next move. The experimental results show that PEL performs as good as or even better than CannySR, substantially improves the modal quality of binary edge maps, takes a very small amount of time to execute and runs at least two times faster than CannySR. It is also important to stress that both CannySR and PEL return their results as a set of edge segments, each of which is a chain of pixels. The edge segments can then be used in many high-level processing applications. We believe that both CannySR and PEL will be very useful in many real-time image processing and computer vision applications.

REFERENCES

[1] A. Farage and E. Delp, "Edge linking by sequential search," *Pattern Recognition*, vol. 28, no. 5, pp. 611–633, 1995.

[2] G. Ovidiu and W. Pual, "Computational approach for edge linking," *The Journal of Electronic Imageging (JEI)*, vol. 11, no. 4, pp. 479–485, 2002.

[3] D.-S. Lu and C.-C. Chen, "Edge detection improvement by ant colony optimization," *Pattern Recognition Letters*, vol. 29, no. 4, pp. 416–425, 2008.

[4] W. Burger and M. J. Burge, *Principles of Digital Image Processing: Fundamental Techniques*. Springer-Verlag, 2009.

[5] P. A. Mlsna and J. J. Rodriguez, "Gradient and laplacian edge detection," *in The Essential Guide to Image Processing, 2nd ed., Al Bovik, ed. San Diego, CA: Elsevier*, pp. 495–524, 2009.

[6] P. Akhtar, T. J. Ali, M. I. Bhatti, and M. A. Muqeet, "A framework for edge detection and linking using wavelets and image fusion," *International Congress on Image and Signal Processing (CISP 2008), Sanya, Hainan, China*, pp. 273–277, 2008.

[7] E. Danahy, S. Agaian, and K. Panetta, "Directional edge detection using the logical transform for binary and grayscale images," *SPIE Defense and Security Symposium: Mobile Multimedia/Image Processing for Military and Security Applications*, vol. 6250, 2006.

[8] A. Elmabrouk and A. Aggoun, "A new edge detection algorithm," *Computational Intelligence and Security: International Conference, CIS 2005, Xi'an, China, December 15-19, 2005, Proceedings, Part 1*, 2005.

[9] A. Jevtic, I. Melgar, and D. Andina, "Ant based edge linking algorithm," *Proceedings of 35th Annual Conference of the IEEE Industrial Electronics Society (IECON 2009)*, pp. 3353–3358, 2009.

[10] Q. Zhu, M. Payne, and V. Riodan, "Edge linking by a directional potential function (dpf)," *Elsevier Image and Vision Computing*, vol. 14, pp. 59–70, 1996.

[11] P. L. Rosin and X. Sun, *Cellular Automata in Image Processing and Geometry*. Springer, 2014.

[12] M. Petrou and C. Petrou, *Image Processing: The Fundamentals*. John Wiley and Sons, 2010.

[13] R. Szeliski, *Computer Vision: Algorithms and Applications*. Springer, 2010.

[14] M. W. K. Law and A. C. S. Chung, "Weighted local variance-based edge detection and its application to vascular segmentation in magnetic resonance angiography," *IEEE Transactions on Medical Imaging*, vol. 26, no. 9, pp. 1224–1241, 2007.

[15] R. Saxena, "High order methods for edge detection and applications,"

PHD Thesis, Arizona State University, USA, 2008.

[16] E. Danahy, S. Agaian, and K. Panetta, "Algorithms for the resizing of binary and grayscale images using a logical transform," *SPIE Electronic Imaging*, 2007.

[17] S. Carlsson, "Sketch based coding of gray level images," *IEEE Transactions on Image Processing*, vol. 15, pp. 57–83, 1988.

[18] S. Nercessian, "A new class of edge detection algorithms with performance measure," *Masters of Science, TUFTS University, Electrical Engineering*, 2009.

[19] S. jayaraman, S. Esakkirajan, and T. Veerankumar, *Digital Image Processing.* Tata McGraw-Hill, 2009.

[20] R. C. Gonzalez, R. E. Woods, and S. L. Eddins, *Digital Image processing using Matlab.* Pearon Prentice Hall, 2004.

[21] C. Topal, C. Akınlar, and Y. Genç, "Edge drawing: A heuristic approach to robust real-time edge detection," *IEEE 20th International Conference on Pattern Recognition (ICPR)*, pp. 2424–2427, 2010.

[22] J. Canny, "A computational approach to edge detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, vol. 8, no. 6, pp. 679–698, 1986.

[23] I. Sobel and G. Feldman, "A 3x3 isotropic gradient operator for image processing," *presented at a talk at the Stanford Artificial Project, unpublished but often cited*, 1968.

[24] R. C. Gonzalez and R. E. Woods, *Image processing.* Pearon Prentice Hall, 3 ed., 2011.

[25] S. Nagabhushana, *Computer vision and Image processing.* New Age International Publishers, 2010.

[26] A. D. Sappa and B. X. Vintimilla, "Edge point linking by means of global and local schemes," *IEEE Int. Conf. On Signal-Image Technology and Internert-Based Systems*, 2006.

[27] D. Marshall, "Edge linking." `http://www.cs.cf.ac.uk/Dave/Vision_lecture/node30.html`, 1994-1997. Accessed: 2015-04-20.

[28] D. H. Ballard, "Generalising the hough transform to detect arbitrary shapes," *Pattern recognition*, vol. 13, no. 2, pp. 111–122, 1981.

[29] L. H. Staib and J. S. Duncan, "Boundary finding with parametrically deformable models," *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, vol. 14, pp. 1061–1075, 1992.

[30] M. Xie, "Edge linking by using causal neighborhood window," *Pattern Recognition Letters*, vol. 13, no. 9, pp. 647–656, 1992.

[31] B.-L. Yeo and S.-P. Liou, "From visualization to perceptual organization," *Visualization and Machine Vision*, pp. 62–73, 1994.

[32] P. Eichel and E. Delp, "Sequential edge linking," *Proc. Twenty-Second Allerton Conf. Commun Control and Computers*, pp. 782–791, 1984.

[33] F. Y. Shih and S. Cheng, "Adaptive mathematical morphology for edge linking," *Information Sciences Informatics and Computer Science: An international Journal*, vol. 167, no. 1-4, pp. 9–21, 2004.

[34] X. Ji, X. Zhang, and L. Zhang, "Sequantial edge linking method for

segmentation of remotely sensed imagery based on heuristic search," *21st International conference on Geoinformatics, 20-22 June*, 2013.

[35] A. Hajjar and T. Chen, "A new real time edge linking algorithm and its vlsi implementation," *Proceedings of Fourth IEEE International Workshop on Computer Architecture for Machine Perception (CAMP 1997)*, 1997.

[36] T. Guan, D. Zhou, K. Peng, and Y. Liu, "A novel contour closure method using ending point restrained gradient vector flow field," *Journal of Information Science and Engineering*, vol. 31, no. 1, pp. 43–58, 2015.

[37] Y. Weng and Q. Zhu, "Nonlinear shape restoration for document images," *Computer Vision and Pattern Recognition*, pp. 568–573, 1996.

[38] J. Wang and X. Li, "A content-guided searching algorithm for balloons," *Pattern Recognition*, vol. 36, pp. 205–215, 2003.

[39] Q. Tang, N. Sang, and T. Zhang, "Extraction of salient contours from cluttered scenes," *Pattern Recognition*, vol. 40, pp. 3100–3109, 2007.

[40] A. Hajjar and T. Chen, "A vlsi architecture for real-time edge linking," *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, vol. 21, no. 1, pp. 89–94, 1999.

[41] W. Ya-Ping, S. V. Chien-Ming, B. Kar-Weng, and B. Yoon-Teck, "Improved canny edges using ant colony optimization," *Computer Graphics, Imaging and Visualisation CGIV '08. Fifth International Conference*, 2008.

[42] J. Zhang, K. He, X. Zheng, and J. Zhou, "An ant colony optimization algorithm for image edge detection," *International Conference on Artificial Intelligence and Computational Intelligence*, vol. 2, pp. 215– 219, 2010.

[43] M. Dorigo, V. Maniezzo, and A. Colorni, "Ant system: Optimization by a colony of cooperating agents," *IEEE Transactions on Systems, Man and Cyberbatics Part B*, vol. 26, no. 1, p. 2941, 1996.

[44] C. Akinlar and E. Chome, "Cannysr: Using smart routing of edge drawing to convert canny binary edge maps to edge segments," *IEEE International Symposium on Innovations in Intelligent Systems and Applications (INISTA)*, 2015.

[45] "Pel web site." `http://ceng.anadolu.edu.tr/cv/pel`. Accessed: 2015-07-01.

[46] C. Topal and C. Akinlar, "Edge drawing: A combined real-time edge and segment dectector," *Journal of Visual Communication and Image Representation*, vol. 23, no. 6, pp. 862–872, 2012.

[47] C. Akinlar and C. Topal, "Edlines: A real-time line segment detector with a false detection control," *Pattern Recognition Letters*, vol. 32, no. 13, pp. 1633–1642, 2011.

[48] C. Akinlar and C. Topal, "Edcircles: A real-time circle detector with a false detection control," *Pattern Recognition*, vol. 46, no. 3, pp. 725–740, 2013.

[49] D. Martin, C. Fowlkes, D. Tal, and J. Malik, "A database of human

segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics," *IEEE International Conference on Computer Vision (ICCV)*, pp. 416–423, 2001.

[50] "The berkeley segmentation dataset and benchmark." `https://www.eecs.berkeley.edu/Research/Projects/CS/vision/bsds/`. Accessed: 2015-06-20.