

AN EFFICIENT IMAGE SEGMENTATION ALGORITHM USING
BIDIRECTIONAL MAHALANOBIS DISTANCE

A Dissertation
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
Electrical Engineering

by
Rahul Suresh
December 2012

Accepted by:
Dr. Stanley Birchfield, Committee Chair
Dr. Brian Dean
Dr. Adam Hoover

Abstract

In this thesis, we present an efficient graph-based image-segmentation algorithm that improves upon the drawbacks of the minimum spanning tree based segmentation algorithm [9], namely leaks that occur due to the criterion used to merge regions, and the sensitivity of the output to the parameter k . To address these problems, we propose the use of bidirectional Mahalanobis distance, along with a Gaussian model for each region, and an intuitive normalized parameter τ that replaces k and works for all images without having to be changed. Furthermore, we propose an approximation to the algorithm that enables it to run efficiently in $O(N \log N)$ time (N represents the number of pixels in the image), without compromising on the performance. Experiments on a wide variety of images demonstrates the ability of the algorithm to achieve accurate results in an efficient manner.

Dedication

To my wonderful parents- Usha and Suresh, and to the lotus feet of Sriman Narayana.

Acknowledgments

I would firstly like to express my sincere gratitude towards my advisor Dr.Birchfield, without whom this thesis would not have been possible. His support and guidance was always inspiring.

I would also like to convey by gratitude towards Dr.Dean and Dr.Hoover for agreeing to serve in my committee and for their valuable feedback.

I would also like to acknowledge JungPhil Kwon for participating in our research meetings and brainstorming ideas. Additionally, I would like to thank all the members of the computer vision research group at Clemson University for creating an inspirational research environment.

Last but not the least, I would like to thank all my friends at Clemson who have supported me- Shivram, Sumod, Vibhor, Balu, Aditya, Darshana, Pallavi and Neeraja.

Table of Contents

Title Page	i
Abstract	ii
Dedication	iii
Acknowledgments	iv
List of Figures	vii
1 Introduction	1
1.1 Image segmentation	1
1.2 Applications of image segmentation	2
1.3 What defines a good segmentation?	3
1.4 Thesis outline	4
2 Related Work	6
2.1 Split and merge approaches	6
2.2 Clustering and mean shift segmentation	6
2.3 Spectral graph theory and normalized cuts	7
2.4 Minimum spanning tree based segmentation	8
3 Background theory	10
3.1 Image as a graph	10
3.2 Minimum Spanning Tree	11
3.3 Minimum spanning tree segmentation algorithm	12
3.4 Results and Drawbacks	16
4 Proposed Algorithm	18
4.1 Constructing the image grid	18
4.2 Region growing	20
4.3 Updating weights in the list	24
4.4 Approximation	24
4.5 Pseudocode	25
5 Results	27
5.1 Results on synthetic images	27
5.2 Analysis of optimal segmentation	28
5.3 Results on the BSDS dataset	29
6 Conclusions and Discussion	48
6.1 Future work	48

Bibliography 50

List of Figures

1.1	Example of image segmentation. a) Original image b) Segmented image [20]	2
1.2	(a) Deer image (b),(c),(d) Manual segmentation performed by various human subjects. Image courtesy- [20]	4
3.1	The leak of the MST algorithm demonstrated on a synthetic image. a) Noisy image with 2 partitions and a thin grayscale ramp between them. b) MST result, in which the ramp enables the two sides to be merged despite their very differences in appearance.	17
3.2	Effect of the choice of k on the granularity in the MST segmentation algorithm output. (a)-(d) MST results for various values of k , showing the difficulty of selecting the proper value.	17
4.1	Criterion used for merging 2 regions a) MST algorithm compares the edge (u, v) with the maximum edge weights of regions R_u and R_v b) Our algorithm models R_u and R_v as Gaussian models and compares the Mahalanobis distance between them before merging.	22
5.1	The leak of the MST algorithm demonstrated on a synthetic image. a) Noisy image with 2 partitions and a thin grayscale ramp between them. b) MST result c) Our algorithm's result	28
5.2	(a) Gradient image (b) MST result (c) Our algorithm's result	28
5.3	Effect of the choice of k on the granularity in the MST segmentation algorithm output. (a)-(d) MST results for various values of k , showing the difficulty of selecting the proper value. (e) Our algorithm for $\tau = 2.5$, which is the same value used for all images, despite scene content and image size.	28
5.4	Our algorithm- different granularities of segmentation for a) Monalisa b) Man	29
5.5	Graph showing the relationship between threshold and number of components for Monalisa (TOP) and Man (BOTTOM)	30
5.6	BSDS300 Segmentation Results (Mean Colors)- I[20]	32
5.7	BSDS300 Segmentation Results (Contours)- I [20]	33
5.8	BSDS300 Segmentation Results (Mean Colors)- II[20]	34
5.9	BSDS300 Segmentation Results (Contours)- II [20]	35
5.10	BSDS300 Segmentation Results (Mean Colors)- III[20]	36
5.11	BSDS300 Segmentation Results (Contours)- III [20]	37
5.12	BSDS300 Segmentation Results (Mean Colors)- IV[20]	38
5.13	BSDS300 Segmentation Results (Contours)- IV [20]	39
5.14	BSDS300 Segmentation Results (Mean Colors)- V[20]	40
5.15	BSDS300 Segmentation Results (Contours)- V [20]	41
5.16	BSDS300 Segmentation Results (Mean Colors)- VI[20]	42
5.17	BSDS300 Segmentation Results (Contours)- VI [20]	43
5.18	BSDS300 Segmentation Results (Mean Colors)- VII[20]	44
5.19	BSDS300 Segmentation Results (Contours)- VII [20]	45

5.20	BSDS300 Segmentation Results (Mean Colors)- VIII [20]	46
5.21	BSDS300 Segmentation Results (Contours)- VIII [20]	47

Chapter 1

Introduction

1.1 Image segmentation

Image segmentation refers to the process of dividing an image into components or regions such that the pixels within a region share certain visual characteristics with each other [10][27]. Mathematically, image segmentation involves partitioning an image I into K components- R_1, R_2, \dots, R_K such that:

$$R_1 \cup R_2 \cup R_3 \dots \cup R_K = I \quad (1.1)$$

$$R_i \cap R_j = \phi \quad \forall i, j : i \neq j \quad (1.2)$$

Each segment R_i is a subset of the image I . Its size can be as small as one pixel (complete segmentation) and as large as the entire image itself (no segmentation).

$$1 \leq |R_i| \leq |I| \quad \forall i \quad (1.3)$$

A good segmentation algorithm would divide the image into segments of intermediate sizes such that similar objects or backgrounds are grouped together. If $f(R_i)$ represents a function that measures whether region R_i is homogeneous or not, then a good segmentation algorithm would divide image I into segments such that:

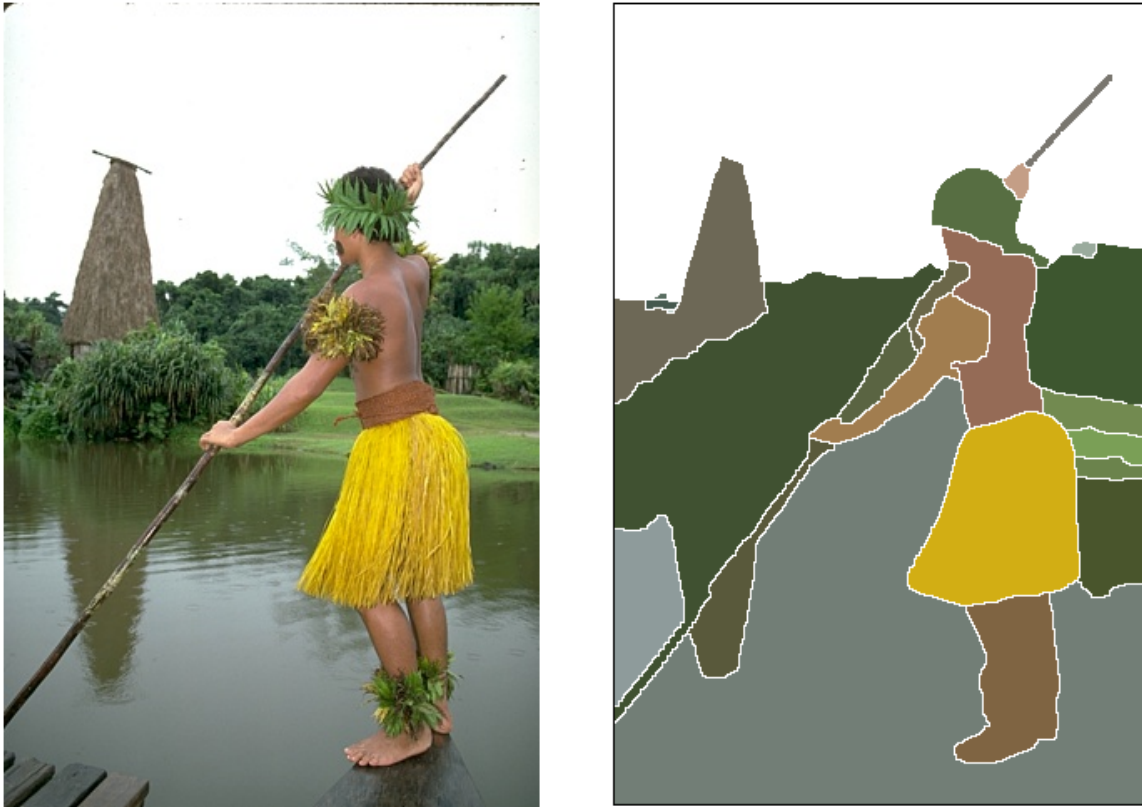


Figure 1.1: Example of image segmentation. a) Original image b) Segmented image [20]

$$f(R_i) = TRUE \quad \forall i \tag{1.4}$$

$$f(R_i \cup R_j) = FALSE \quad \forall i, j : i \neq j \tag{1.5}$$

In other words, pixels within a region share certain visual traits that are not found in the pixels of another region. Figure 1.1 is a good illustration of image segmentation. Notice that pixels adjacent with similar color values are grouped together.

1.2 Applications of image segmentation

Image segmentation is a widely used technique and a major area of research in computer vision. It is used in a wide range of applications from lower level tasks such as finding connected components to higher level applications such as large-scale image classification systems.

Image segmentation is a very important preprocessing step in many biomedical analysis problems [3] [24] [21]. The requirements of biomedical segmentation varies widely depending on the modality of imaging used and the part of the body involved. However, in almost all cases, there is a need to segment various anatomical organs to facilitate further diagnosis and treatment. Image segmentation is also increasingly used these days in computer-guided surgery [12]. Image segmentation is seen as a very important preprocessing step in object recognition systems. Lower level features such as texture, color and intensity are used to segment the image into several regions. Relevant segments (or subset of pixels) are alone used by the recognition systems. This considerably saves the computational cost involved, especially when running on a large database [23]. Another important application where image segmentation is used extensively today is content based image retrieval (CBIR) [7]. Some of the other areas where image segmentation is popular are face recognition [13], iris recognition [14], and astronomy [22].

1.3 What defines a good segmentation?

Consider the image containing the deer in Figure 1.2. Various human subjects were asked to manually segment the image. Notice that there is a lot of variation in the results. The number of segments is considerably higher in Figure 1.2-(d) than in Figure 1.2-(b). This variation in the segmentation results does not represent an error in the output, but shows the varied levels of granularity perceived by the different human subjects [30]. In other words, we have many segmentations for the same image and all of them are “correct”. This ambiguity in defining “correctness” of segmentation makes the evaluation of segmentation algorithms difficult. Other areas in computer vision such as object detection and classification are well defined and hence no difficulty exists in evaluating them. This problem is unique to image segmentation [1] .

One of the primary objectives of this thesis is to address the above problem by defining “optimal” granularity and providing a mathematical intuition for it . Yu and Hoover have done extensive work in determining “stable” segmentation [32] [15]. They define stable segmentation as a configuration where the number of regions are least sensitive to parameter changes. The segmentation algorithm is run repeatedly on the same image for a wide range of its parameters and the best configuration is chosen based on the invariance to parameter changes. We use a method

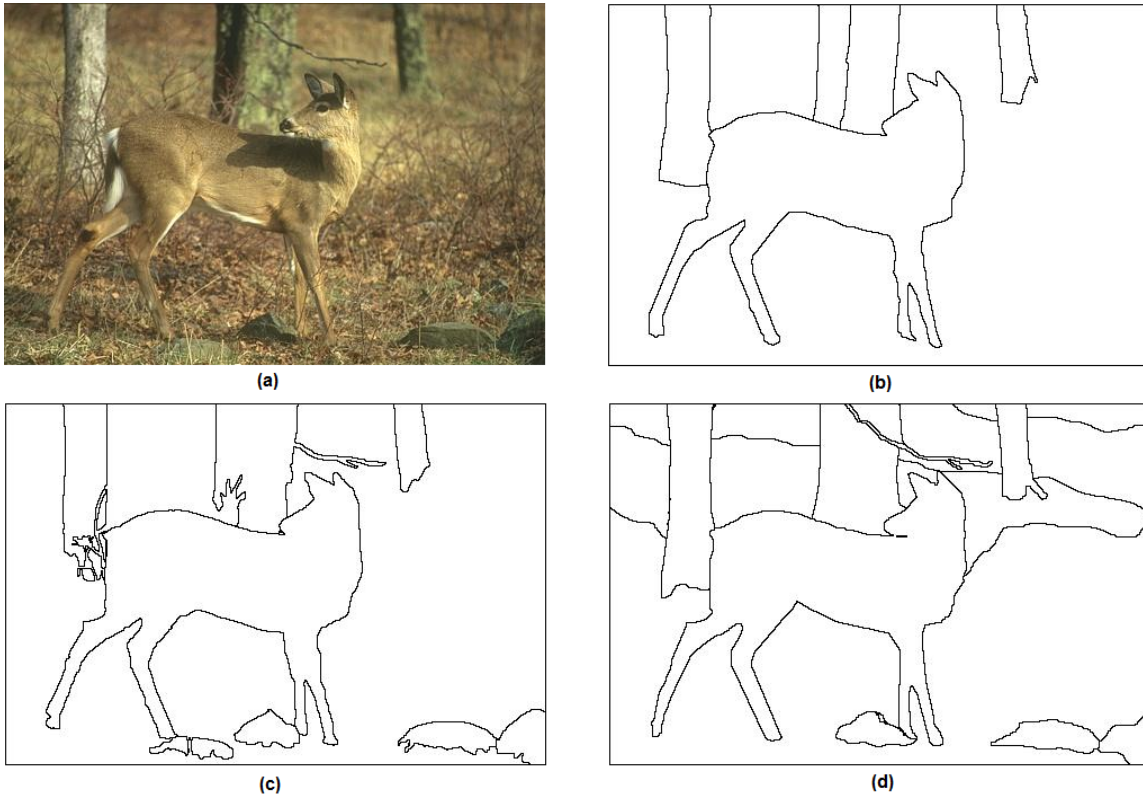


Figure 1.2: (a) Deer image (b),(c),(d) Manual segmentation performed by various human subjects. Image courtesy- [20]

along these lines to define “optimal” or “stable” granularity for our system.

1.4 Thesis outline

This thesis proposes an efficient image segmentation algorithm that runs real time and produces good segmentation results. The mathematical justification for what constitutes the “optimal” granularity is also provided. Chapter 2 provides an overview about the various segmentation algorithms that are popular today and their relative merits and demerits. Emphasis is laid on graph based segmentation algorithms as it closely corresponds to this work. Chapter 3 provides a detailed background on the minimum spanning tree (MST) based segmentation algorithm. The major drawbacks of this algorithm are also discussed in detail. In Chapter 4, we introduce a novel, efficient, graph-based segmentation algorithm that improves upon the drawbacks of the MST algorithm. In Chapter 5, we discuss the results of our proposed segmentation algorithm. We validate its perfor-

mance by testing on a large database of images and comparing our algorithm to other segmentation algorithms. Chapter 6 provides a summary of this thesis work and possible future improvements.

Chapter 2

Related Work

There has been a considerable amount of research in the area of image segmentation. This chapter will provide an overview of some of the popular image segmentation techniques that exist today, with special emphasis on graph based approaches.

2.1 Split and merge approaches

Split and merge algorithms begin by considering the entire image as one region. The image is then iteratively split into regions based on the evidence of a boundary. The algorithms that follow this approach are generally based on quadtrees [28] [8] [16]. In the quadtree based approach, the image is recursively split into 4 equal sized regions based on the existence of non-uniformity in features and again recursively combined based on the existence of similarities until a stable segmentation is obtained.

2.2 Clustering and mean shift segmentation

Another class of segmentation algorithm is based on clustering and mean shift. While clustering and mean-shift based methods are considered to be different algorithms, they still have a lot of underlying similarities. In the mean-shift segmentation algorithm, each pixel is represented as feature vector consisting of spatial and color components. The algorithm assumes that there is a probability distribution underlying the data. The modes of the distribution would represent the

centers of different regions. Each data point has a window defined around it and is shifted towards the mean of all the data points within that window. This is repeated iteratively until the data points converge to the modes of the distribution [4] [6].

The k -means segmentation algorithm, as the name suggests, uses k -mean clustering to segment data. Pixels are represented as vectors in an n -dimensional feature space. Feature space could be a combination of color and spatial information. k cluster centers are chosen randomly and each pixel is assigned to its closest cluster. Then the cluster means are recomputed. This process is repeated iteratively until the cluster centers converge. The main disadvantage of k -means is that the number of regions (clusters) must be known beforehand. Furthermore, the segmentation results are extremely sensitive to the initial choice of cluster centers. There is a lots of research happening of late that tries to address the above problems [33].

2.3 Spectral graph theory and normalized cuts

Another class of image segmentation algorithms seek to represent the image as a graph and find a “cut” that would partition the graph into different regions [19]. The objective of a graph cut problem is to find the set of edges to be removed such that similar vertices are grouped together. Consequently, the edges forming the cut should be the edges with the largest weights (lowest similarity).

Suppose we partition a graph G into disjoint sets S_1 and S_2 . The objective of a min-cut problem is to minimize the sum of the weights of edges that form the cut. We want to minimize the following function:

$$f(S_1, S_2) = \sum_{i \in S_1} \sum_{j \in S_2} W_{ij} \tag{2.1}$$

The above optimization problem has a bias to create single node partitions. In order to ensure that reasonable sized partitions are obtained through the cut, the objective function that needs to be minimized is modified as follows:

$$RatioCut(S_1, S_2) = \frac{f(S_1, S_2)}{|S_1|} + \frac{f(S_1, S_2)}{|S_2|} \quad (2.2)$$

Spectral graph theory based approaches including Normalized cuts use eigenvalues to find the graph partition [5] [31] [26] [29] [11]. The eigenvector of the Laplacian matrix L with second highest eigenvalue is chosen to form the cut .

$$L = D - W \quad (2.3)$$

where D is a diagonal matrix containing the row sums of the similarity matrix W .

$$D_{ii} = \sum_j W_{ij} \quad (2.4)$$

2.4 Minimum spanning tree based segmentation

Minimum spanning tree based segmentation algorithm is an efficient graph based segmentation algorithm [9]. It is a variant of Kruskal's MST algorithm. In Kruskal's algorithm, edges E of the graph G are sorted in the non-decreasing order of their weights. These sorted edges are added to the minimum spanning tree in order as long as the new edge added does not form a cycle (thereby destroying the tree).

Kruskal's algorithm adds all vertices of the graph G into one tree. However, the objective of the MST based segmentation algorithm is to obtain K trees from the graph G , where each tree would represent a region. In this algorithm, an undirected graph $G = (V, E)$ is constructed by building an image grid or a nearest neighbor graph [9]. Pixels constitute vertices while edges are formed by connecting neighboring pixels in $x - y$ space (image grid approach) or by connecting every pixel to its nearest neighbors in (x, y, R, G, B) space. The edges are sorted in non-decreasing order of weights. An approach similar to Kruskal's minimum spanning tree algorithm is used to add vertices into the spanning tree. However, MST based segmentation algorithm adds an edge into a tree only if there is no evidence of boundary between the vertices forming the edge. In other words, regions R_1 and R_2 are merged only if:

$$Diff(R_1, R_2) < \min \left(Int(R_1) + \frac{k}{|R_1|}, Int(R_2) + \frac{k}{|R_2|} \right) \quad (2.5)$$

$Diff(R_1, R_2)$ is the weight of the edge under consideration. $Int(R_1)$ and $Int(R_2)$ refer to the maximum internal difference within the region (maximum edge-weight) while $\frac{k}{|R|}$ ensures that the merge criteria is relaxed for relatively small regions.

A forest of trees are obtained from the graph, where each tree represents a region in the image. The algorithm runs in $O(N \log N)$ time, N representing the number of pixels in the image. The primary drawbacks of the algorithm include its sensitivity to parameter k and also the problem of “leak”. Since this algorithm is very closely related to this thesis, a very detailed description has been provided in Chapter 3.

Chapter 3

Background theory

3.1 Image as a graph

A graph $G = (V, E)$ is an abstract data type containing a set of vertices V and edges E . Vertices, often called nodes, are connected to each other through edges. A weight is associated with every edge in the graph and its value depends on the degree of similarity/ dissimilarity between the nodes connected by it. The list of edges between the various vertices are represented using Adjacency Matrix for a dense graph and Adjacency list for a sparse graph. Several useful operations can be performed on a graph. These include:

1. Checking if a path exists between vertices v_i and v_j .
2. Finding the connected components in a graph.
3. Finding the strongly connected components in a graph
4. Determining if a cycle exists in the graph
5. Computing the shortest path between vertices v_i and v_j .
6. Computing the minimum spanning tree
7. Partitioning a graph into various regions using graph-cut algorithms.

Graph techniques have become very popular in computer vision. In the graph representation of an image, a single pixel or a group of pixels normally form the vertices while the degree of dissimilarity

between the vertices determines the weight associated with the edges. There are multiple ways to construct an image graph. These include:

1. **Image grid:** In this scheme, every pixel is assigned to a vertex in the graph. Edges are connected between adjacent pixels in $x-y$ space, thereby creating a grid. The weight associated with the edges is normally the Euclidean distance between the pixels in color space. An advantage of this representation is that it is extremely easy to construct. Furthermore, the number of edges $m = O(N)$, thereby by reducing the time complexity of algorithms that normally run in $O(m)$ time. One of the main disadvantages of this representation is that an edge exists between two vertices only if they are adjacent in the $x - y$ space, thereby global properties are often not captured while running a segmentation algorithm.
2. **Complete graph:** In a complete graph, there exists an edge e between every pair of vertices $v_i, v_j \in V$. Thus, an image with n nodes would have $O(N^2)$ edges. The feature space for computing the distance between the nodes forming an edge can be RGB color space or $x - y$ space or a combination of both. Most of the graph operations in this scheme would be very expensive.
3. **Nearest neighbor graph:** This representation acts as a compromise between the image-grid representation (that fails to capture global properties in $x - y$ space) and a complete graph (which has $O(N^2)$ edges). For every vertex $v_i \in V$, its $k = O(1)$ neighbors are computed in some feature space using Approximate Nearest Neighbors (ANN) algorithm [17]. (x, y, R, G, B) is an example of a 5-D features space that captures both color and spatial properties. Each vertex is then connected to its k nearest neighbors, thereby creating a graph that has $m = O(n)$ edges.

3.2 Minimum Spanning Tree

A tree is a connected graph with no cycles. Given a connected unweighted graph $G = (V, E)$, a tree can be constructed using every vertex $v_i \in V$ such that there exists a path between every pair of vertices. Such a tree is called a spanning tree. A graph can have many spanning trees. A spanning tree that is obtained by choosing a subset of edges from the graph G such that the sum of weights of the edges forming the spanning tree is minimal is called a Minimum Spanning Tree.

Kruskal's [18] and Prim's [25] algorithm are popularly used to find the minimum spanning tree of graphs. Kruskal's algorithm has been discussed in the next section as it very closely relates to the image segmentation algorithm proposed in this thesis.

3.2.1 Kruskal's Algorithm

In this algorithm, edges E of the graph G are sorted in the non-decreasing order of their weights. The sorted edges are added to the minimum spanning tree in the order of their weights as long as the new edge added does not form a cycle (thereby destroying the tree). In other words, we start off by considering graph G to be a forest of N trees, each with exactly one node. After every iteration, two trees are merged as long as they do not form a cycle. This process is repeated until all the vertices of the graph are merged into one spanning tree. It can be observed that the algorithm makes greedy choices at every stage to locally optimize the solution.

The pseudocode for the Kruskal's algorithm is given below:

```

KRUSKAL( $G = (V, E)$ )
1   $\langle e_1, \dots, e_m \rangle \leftarrow \text{SORTASCENDINGBYWEIGHT}(E)$ 
2  for  $(u, v) \leftarrow e_1$  to  $e_m$  do
3       $u' \leftarrow \text{FINDSET}(u)$ 
4       $v' \leftarrow \text{FINDSET}(v)$ 
5      if  $u' \neq v'$  then
6          Merge( $u', v'$ )

```

There are three main operations in this algorithm- Sorting, FindSet and Merge. The FindSet and Merge operations can be accomplished in $O(m\alpha(N))$ time, where α is an extremely slow growing function called the inverse Ackermann function. The bottleneck in the algorithm is the initial sorting process, thus the overall time complexity is $O(m \log(m))$ time.

3.3 Minimum spanning tree segmentation algorithm

Felzenszwalb and Huttenlocher [9] proposed an efficient graph based algorithm that uses a variant of Kruskal's algorithm to segment images. The objective of Kruskal's algorithm is to compute the minimum spanning tree of a weighted connected graph. However, if Kruskal's algorithm were

directly applied to image segmentation, then we would end up with one image segment (consisting of the entire image). Thus, the objective of the MST based segmentation algorithm is modified to obtain K trees from the graph G . Each tree would represent a region in the image. Specific details about the algorithm are explained in the subsequent sub-sections.

3.3.1 Constructing the image graph

Given an image I of size (w, h) , we construct an image-grid G as follows:

1. Every pixel $I(x, y)$ is mapped with vertex $v_i \in V$ such that:

$$i = (y - 1) * w + x \tag{3.1}$$

The information about the vertices are stored using a disjoint-set data structure D .

2. Edge set E is constructed by linking every vertex v_i with its four immediate neighbors in $x - y$ space. The list of edges are maintained using an array.
3. The weight w_i associated with every edge $e_i \in E$ is the Euclidean distance in the RGB feature space between the vertices $u_i, v_i \in V$ that forms the edge.

Note that there are N nodes and $m = O(N)$ edges in the graph.

3.3.2 Disjoint-set data structure

Information about vertices (or regions) are maintained using a disjoint-set data structure D of size N . For every vertex v_i , we store information about its root, size of the region to which it belongs and the maximum edge weight in the region. During initialization of the image-grid G , every region consists of exactly one pixel. Information about vertex v_i is stored at the i th index of D . $D[i]$ is initialized as follow:

$$D[i].root = -1 \tag{3.2}$$

$$D[i].numPixels = 1 \tag{3.3}$$

$$D[i].maxEdgeWeight = 0 \tag{3.4}$$

While the length of the disjoint set is N (corresponding to the number of pixels in the image), their values are stored, accessed and updated at their root vertices only. The root of any region R_i is obtained using the following recursive algorithm:

```

GETEQUIV( $i$ )
1  if  $equiv[i] == -1$  then
2      return( $i$ )
3  else
4      return(GETEQUIV( $equiv[i]$ ))

```

3.3.3 Merge Criterion

As discussed in the previous section, every pixel is assigned to a separate region in the image. The objective of the segmentation algorithm is to divide the image into K regions such that $1 < K \ll N$. First, the edges are sorted in the non-decreasing order of weights. An approach similar to Kruskal's minimum spanning tree algorithm is used to add vertices into the spanning tree. However, MST based segmentation algorithm adds an edge into a tree only if there is no evidence of boundary between the vertices forming the edge. In other words, regions R_u and R_v are merged only if:

$$D(R_u, R_v) < \min \left(Int(R_u) + \frac{k}{|R_u|}, Int(R_v) + \frac{k}{|R_v|} \right) \quad (3.5)$$

$D(R_u, R_v)$ is the weight of the edge under consideration. $Int(R_u)$ and $Int(R_v)$ refer to the maximum internal difference within the region (maximum edge-weight) while $\frac{k}{|R|}$ ensures that the merge criteria is relaxed for relatively small regions. In other words, regions R_u and R_v are merged only if the edge weight connecting the regions is smaller than the the internal difference of both the components.

3.3.4 Merging regions

Merging can be easily accomplished using the disjoint set data structure. Suppose edge $e_i \in E$ connecting vertices $u_i, v_i \in V$ is being added to the MST. Then the regions containing u_i and v_i can be merged by updating the disjoint set data structure as follows:

$$u' = \min(FindSet(u), FindSet(v)) \quad (3.6)$$

$$v' = \max(\text{FindSet}(u), \text{FindSet}(v)) \quad (3.7)$$

$$D[v'].root = u' \quad (3.8)$$

$$D[u'].maxEdgeWeight = \max(D[u'].maxEdgeWeight, D[v'].maxEdgeWeight) \quad (3.9)$$

$$D[u'].numPixels = D[u'].numPixels + D[v'].numPixels \quad (3.10)$$

3.3.5 Pseudo code for the MST based segmentation algorithm

MST-SEGMENTATION(I, k)

Input: image I , parameter k

Output: segmentation S

Data structures:

D is a DSDS with an element for each pixel in image

$D[i]$ contains the base/root ID, max-edge-weight and number of pixels for vertex i

E is a vector of edges.

$E[i]$ contains information about edge e_i including the vertex ID's and weight

```

1   $D$ .INITIALIZE( $width * height$ )
2   $E \leftarrow$  CONSTRUCTEDGES( $I$ )
3   $\langle e_1, \dots, e_m \rangle \leftarrow$  SORTASCENDINGBYWEIGHT( $E$ )
4  for  $(u, v) \leftarrow e_1$  to  $e_m$  do
5       $u' \leftarrow$  min (FINDSET( $u$ ), FINDSET( $v$ ))
6       $v' \leftarrow$  max (FINDSET( $u$ ), FINDSET( $v$ ))
7      if  $u' \neq v'$  and ISSIMILAR( $u', v', w(u, v)$ ) = TRUE then
8          MERGE( $u', v', w(u, v)$ )

```

FINDSET(u)

```

1  if  $D[u].root = -1$  then
2      return  $u$ 
3  else
4      return FINDSET( $D[u].root$ )

```

ISSIMILAR($u', v', w; k$)
1 **return** $w < \min \left(D[u'].maxEdgeWeight + \frac{k}{D[u'].numPixels}, D[v'].maxEdgeWeight + \frac{k}{D[v'].numPixels} \right)$

DISJOINTSET:INITIALIZE(N)

1 **for** $i \leftarrow 0$ **to** $N - 1$ **do**
2 $D[i].root \leftarrow i$
3 $D[i].maxEdgeWeight \leftarrow 0$
4 $D[i].numPixels \leftarrow 1$

DISJOINTSET:MERGE(u, v, w)

1 $D[b].root \leftarrow a$
2 $D[a].maxEdgeWeight \leftarrow \max(w, D[a].maxEdgeWeight, D[b].maxEdgeWeight)$
3 $D[a].numPixels \leftarrow D[a].numPixels + D[b].numPixels$

3.4 Results and Drawbacks

MST algorithm runs in real time and uses greedy approach to achieve segmentation. The running time of this algorithm is $O(m \log(m))$ and since $m = O(N)$, the asymptotic runtime is $O(N \log N)$. Some of the drawbacks of MST based segmentation are:

1. The use of minimum edge weight to determine whether to merge regions or not. This can often lead to faulty segmentations when there is a leak. (See Figure 3.4).
2. Its sensitivity to parameter k that determines the granularity of segmentation. This is again apparent in Figure 3.2

In this thesis, we propose a graph based segmentation algorithm that improves upon the shortcomings of the MST segmentation. There are two primary contributions in this thesis. Our first main contribution is the use of bidirectional Mahalanobis distance to determine the existence of a boundary. Within the framework of the MST based approach, we represent every image region as a Gaussian distribution. Edges are added to a tree only if the Mahalanobis distance between the Gaussian distributions is less than a predetermined value. This ensures that region merging does not happen when there is a “leak” from one region to another.

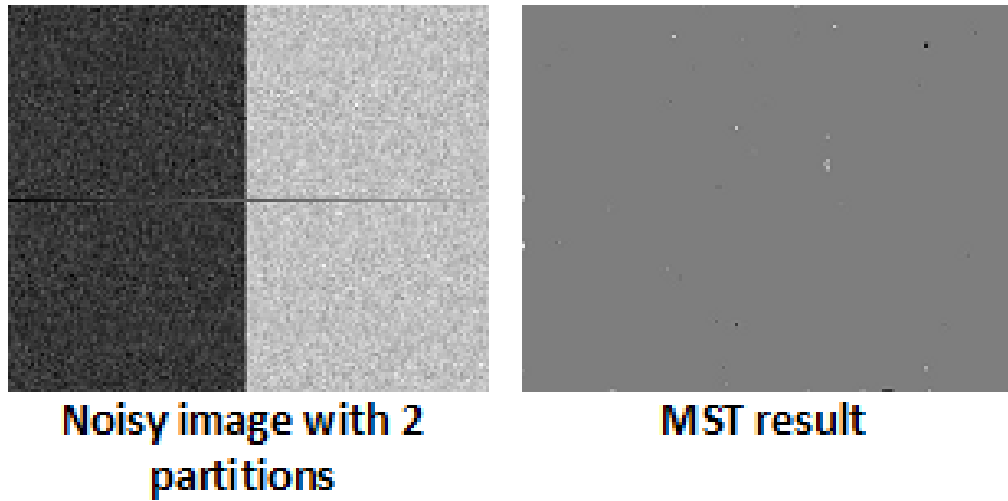


Figure 3.1: The leak of the MST algorithm demonstrated on a synthetic image. a) Noisy image with 2 partitions and a thin grayscale ramp between them. b) MST result, in which the ramp enables the two sides to be merged despite their very differences in appearance.

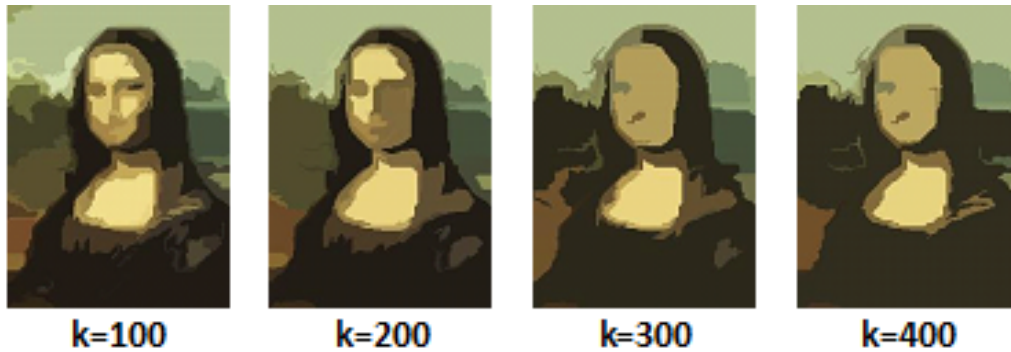


Figure 3.2: Effect of the choice of k on the granularity in the MST segmentation algorithm output. (a)-(d) MST results for various values of k , showing the difficulty of selecting the proper value.

The second contribution in this thesis is to provide an intuition regarding the granularity of segmentation. In the MST algorithm, the value of k is chosen arbitrarily. Any change in k can affect the granularity of segmentation significantly. Furthermore, k is dependent on the size of the image. We propose the use a threshold τ that would depend on the Mahalanobis distance between the Gaussian distributions. Mahalanobis distance is similar to Euclidean distance but is normalized by σ . We show that stable segmentations are obtained for $2 < \tau < 2.5$, which sounds mathematically justifiable. The details about this new algorithm is provided in the next chapter.

Chapter 4

Proposed Algorithm

In this chapter, we propose an efficient graph-based segmentation algorithm that would improve upon the problems associated with the MST based segmentation approach- namely its sensitivity to parameter k and the criterion used to merge regions. Later on, we propose an approximation that would enable the algorithm to run in real-time while still improving upon the problems associated with the MST based approach.

4.1 Constructing the image grid

4.1.1 Initialize vertices V

Given an image I with N pixels, we want to create a segmentation S consisting of regions $S = (R_1, R_2, \dots, R_K)$. We begin by constructing an image-grid graph $G = (V, E)$, where every pixel $I(x, y)$ is mapped to a vertex $v_i \in V$ such that:

$$i = (y - 1) * w + x \tag{4.1}$$

The information about vertices (or regions) are maintained using a disjoint-set data structure D of size N . Information about every vertex v_i such as its root node, zeroth-, first- and second- order moments are stored at the i th index of D . During initialization of the image-grid G , every region consists of exactly one pixel. Thus, $D[i]$ is initialized as follows:

$$D[i].root = -1 \tag{4.2}$$

$$D[i].zerothMoment = 1 \tag{4.3}$$

$$D[i].firstMoment = \{v[i].r, v[i].g, v[i].b\} \tag{4.4}$$

$$D[i].secondMoment = \{v[i].r^2, v[i].g^2, v[i].b^2\} \tag{4.5}$$

4.1.2 Initialize edges E

Edges are created by connecting every pixel to its 4 immediate neighbors. Thus, the number of edges $m = O(N)$. Every edge e_i connects vertices $u_i, v_i \in V$ and has a weight w_i associated with it. The edges are stored as a doubly linked list. Each item in this linked list is a *struct* with the following members:

```
struct Edge
{
    int u; //vertex u
    int v; //vertex v
    double w; //edge weight
    Edge *previous; //pointer to the previous edge in the list
    Edge *next; //pointer to the next edge in the list
}
```

Note that there are $m = O(N)$ number of edges in the list.

4.1.3 Initialize edge weights

The weight of an edge e_i represents the degree of dissimilarity between u_i and v_i in *RGB* color space. During initialization, since every region consists of exactly one pixel, a simple Euclidean distance in *RGB* space is used to compute w_i . However, as the regions grow, the weight w_i associated with $e_i \in E$ is the distance between the Gaussian distributions containing $u_i \in V$ and $v_i \in V$.

In our algorithm, we use bidirectional Mahalanobis distance measure described in [2] to compute the distance between two Gaussian distributions. If the region is too small to be modelled accurately as a Gaussian distribution, then we approximate its variance to unity. We merge regions if the distance between the two Gaussian distributions is less than a predetermined value τ . Note that τ would be in the range of 2-2.5 in Mahalanobis units and $2\sigma - 2.5\sigma$ units in Euclidean space. While initializing Gaussian parameters to the regions, we set its $\mu_i = I(x, y)$ and $\sigma_i^2 = 1$.

$$\mu_i = \begin{bmatrix} I(x, y).red \\ I(x, y).green \\ I(x, y).blue \end{bmatrix} \quad (4.6)$$

$$\Sigma_i^2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (4.7)$$

The distance between two Gaussian distributions $\mathcal{N}(\mu_u, \sigma_u^2)$ and $\mathcal{N}(\mu_v, \sigma_v^2)$ is given by:

$$w_{Gaussian} = \sqrt{(\mu_u - \mu_v)^T \Sigma^{-1} (\mu_u - \mu_v)} \quad (4.8)$$

where:

$$\Sigma = \frac{(\Sigma_u + \Sigma_v)}{2} \quad (4.9)$$

By substituting 4.7 in equation 4.8, we can notice that the Mahalanobis distance has reduced to a simple Euclidean distance for small regions (whose $\Sigma = I$).

$$w_{Gaussian} = \sqrt{(\mu_u - \mu_v)^T (\mu_u - \mu_v)} \quad (4.10)$$

4.2 Region growing

Having intialized every pixel to a seperate region in the image, we now follow the following steps in our segmentation algorithm:

1. Sort the edges in edge-list E in the non-decreasing order of their weights.

2. While the edge-list is not empty
 - (a) Pop edge $e_i \in E$ from the top of the list.
 - (b) If the regions containing $u_i, v_i \in V$ are **Similar**, then **Merge**.
 - (c) **Re-sort** the edge list.

Step 1 (sorting) would take $O(N \log N)$ time. Step 2.1 takes $O(1)$ time for every iteration (assuming that the list is sorted) while every operation in step 2.2 can be achieved in less than $O(\log N)$ time per iteration using the disjoint-set data structure [9], [18]. However, the primary bottle neck in the algorithm is to re-sort at every iteration. If we adapt a naive approach, then the worst case running time would be $O(N^2)$. However, in the later part of this chapter, we propose a data-structure for reducing the overall time complexity.

4.2.1 Checking for similarity between regions

Let the edge $e_i \in E$ popped from the top of the edge-list connect vertices $u_i, v_i \in V$. Let R_u and R_v represent the regions to which u_i, v_i belong respectively.

$$R_u = \text{FindSet}(u_i) \tag{4.11}$$

$$R_v = \text{FindSet}(v_i) \tag{4.12}$$

In the MST algorithm, regions $R_{u'}$ and $R_{v'}$ are merged if the following condition is satisfied:

$$D(R_u, R_v) < \min \left(\maxEdge(R_u) + \frac{k}{|R_u|}, \maxEdge(R_v) + \frac{k}{|R_v|} \right) \tag{4.13}$$

The above is not a good measure to compare the similarity between two regions. The maximum edge weights of R_u and R_v are compared with the weight of edge e_i . Even if there is a small leak, i.e. if there exists one edge e_i connecting regions R_u and R_v such that w_i is less than the maximum edge weights within R_u and R_v , we will end up merging the regions.

A more accurate measure would be to compare the Gaussian distributions to which the vertices belong and then decide if they are similar or not. Regions R_u and R_v are said to be similar if the following condition is true:

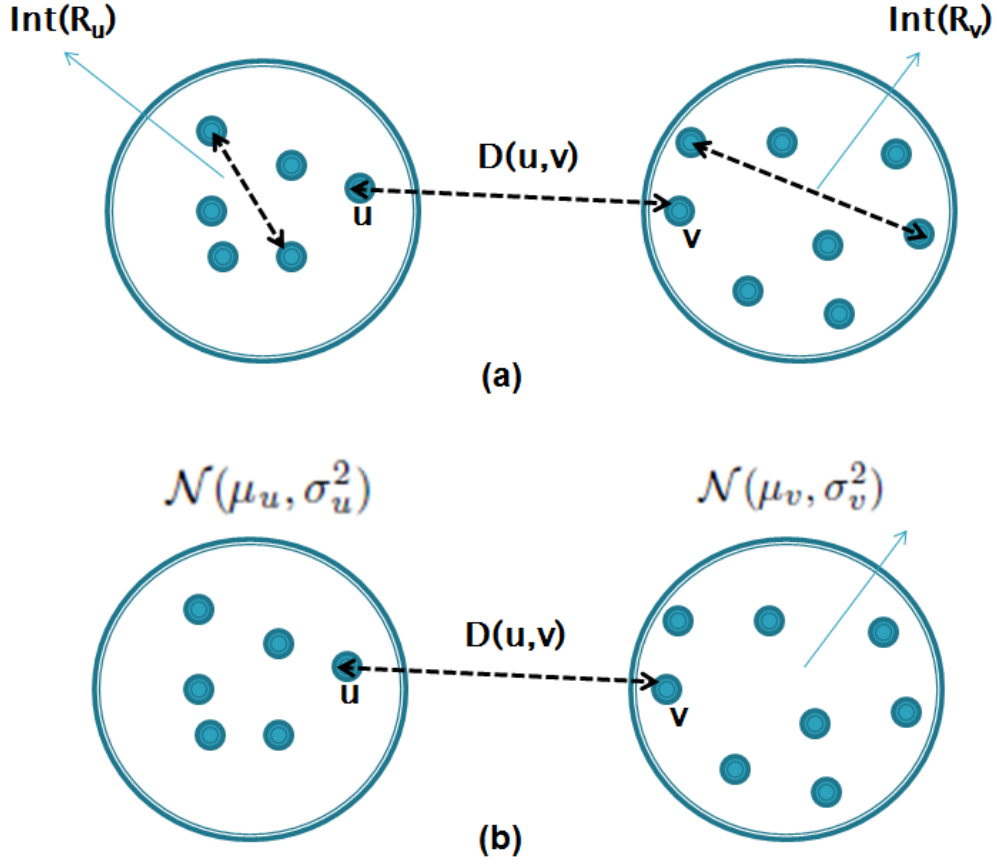


Figure 4.1: Criterion used for merging 2 regions a) MST algorithm compares the edge (u, v) with the maximum edge weights of regions R_u and R_v b) Our algorithm models R_u and R_v as Gaussian models and compares the Mahalanobis distance between them before merging.

$$D(\mathcal{N}_{R_u}, \mathcal{N}_{R_v}) - \frac{50}{\max(|R_u|, |R_v|)} \leq \tau \quad (4.14)$$

The distance between two Gaussian distributions $\mathcal{N}(\mu_u, \sigma_u^2)$ and $\mathcal{N}(\mu_v, \sigma_v^2)$ is given by:

$$D(\mathcal{N}_{R_u}, \mathcal{N}_{R_v}) = \sqrt{(\mu_{R_u} - \mu_{R_v})^T \Sigma^{-1} (\mu_{R_u} - \mu_{R_v})} \quad (4.15)$$

$$\Sigma = \frac{(\Sigma_{R_u} + \Sigma_{R_v})}{2} \quad (4.16)$$

The above Mahalanobis distance is nothing but Euclidean distance that is normalized by the variance of the distribution.

4.2.2 Merging regions

If regions R_u and R_v are found to be similar, then they have to be merged. Every vertex $v_i \in V$ has information about its root node $v'_i \in V$. The root node stores information about zeroth-, first- and second- order moments. Updating all the above information while merging is fairly straightforward using the disjoint-set data structure D . Without loss of generality, assuming that $u'_i < v'_i$, the 2 regions are merged as follows:

$$D[v'_i].root = u'_i \quad (4.17)$$

$$D[u'_i].zerothMoment = D[u'_i].zerothMoment + D[v'_i].zerothMoment \quad (4.18)$$

$$D[u'_i].firstMoment = D[u'_i].firstMoment + D[v'_i].firstMoment \quad (4.19)$$

$$D[u'_i].secondMoment = D[u'_i].secondMoment + D[v'_i].secondMoment \quad (4.20)$$

All the above operations take $O(1)$ time only. However, we want to update the weights of all edges that are connected to either R_u or R_v to reflect the weights w.r.t $R_u \cup R_v$. This can be accomplished efficiently by maintaining a list of neighbors for every region in D .

4.2.3 Merging neighbor list

For every entry in D , we maintain a list containing the pointers to all its edges. When regions R_u and R_v are merged, the corresponding lists $D[u'_i].list$ and $D[v'_i].list$ are also merged as follows:

1. Sort the neighbors of region R_u and R_v separately, according to the index of their root nodes.
2. Remove duplicate regions from both the lists separately.
3. Merge the two sorted neighbor lists.
4. Remove the duplicate regions from the combined list.

If the above update is performed after every merge operation and if the size of the neighbor list is a constant, then all the above operations would still run in $O(1)$ time. However, if a situation is encountered where one region grows very big and is surrounded by $O(N)$ small regions, then the

task of merging the neighbor lists could slow down considerably. However, our merge condition encourages multiple regions to grow simultaneously, thereby preventing the above problem.

4.3 Updating weights in the list

After merging regions R_u and R_v , the weights of all the edges outgoing from the merged region has to be updated. Note that the neighbor list for every region contains the pointers to the edges that are connected to it. Hence, every edge from the list can be accessed in $O(1)$ time and its weight can be updated. However, the bottleneck in the algorithm is to reorder the edges so that list remains sorted.

The above operation will be expensive to implement using a doubly-linked list. In the worst case scenerio, if every region has an average of p neighbors and it takes $O(N)$ time to readjust the updated edge, then the total time taken after every merge operation would be $O(Np)$. The overall running time of the algorithm would be $O(N^2p)$. This would become very expensive and make real-time implementation impossible. However, this step can be speeded up using skip-lists to maintain edges in the sorted order. Skip list is a data structure that permits quick search operations on a sorted list ($O(\log N)$ time). It stores an heirarchy of linked-lists that becomes more and more sparse. Using skip lists, the time complexity of every re-sorting operation can be reduced to $O(p \log N)$ and assuming that $p = O(1)$, the overall running time of the algorithm to $O(N \log N)$ time.

4.4 Approximation

As noted in the previous section, the primary bottleneck in the proposed algorithm is the updating of edge weights during every merge operation and to maintain the edges sorted according to their weights. However, in practice, we have found that even if we skip the updating of weights and re-sorting after every merge operation, the algorithm still fixes the primary drawbacks of the minimum spanning tree based segmentation algorithm, namely its sensitivity to parameter k and the merge criterion used. This approximated algorithm would run in $N \log N$ time, thereby enabling real-time implementaion.

4.5 Pseudocode

SEGMENT(I)

Input: image I

Output: segmentation S

Data structures:

D is a DSDS with an element for each pixel in image

$D[i]$ contains the base/root ID, 0th-, 1st-, and 2nd-order moments, and pointer to *neighbor-list*

neighbor-list stores the ID of the neighbor along with a pointer to the edge in the *edge-list*

edge-list is a skip list of edges; each each is a pair of region IDs and weight, along with valid bit

```

1  Compute and sort edge-list
2  while edge-list is not empty do
3       $(u, v, w) \leftarrow \text{edge-list} . \text{pop-front}()$  ; get min weight edge
4      if FINDSET( $u$ )  $\neq$  FINDSET( $v$ ) then
5          if ISSIMILAR( $\text{FindSet}(u), \text{FindSet}(v)$ ) then
6              MERGE(FINDSET( $u$ ), FINDSET( $v$ ))
7          return  $I'$ 

```

ISSIMILAR(u, v)

```

1  return  $\left( \text{MAHALANOBIS}(u, v) - \min \left( \frac{50}{D[u].\text{numPixels}}, \frac{50}{D[v].\text{numPixels}} \right) \right) < 2.5$ 

```

MERGE(u, v)

Input: region IDs a and b

Output: none

```

1   $a = \min(u, v)$ 
2   $b = \max(u, v)$ 
3   $D[b].\text{root} = a$ 
4   $D[a].\text{zeroMoment} = D[a].\text{zeroMoment} + D[b].\text{zeroMoment}$ 
5   $D[a].\text{firstMoment} = D[a].\text{firstMoment} + D[b].\text{firstMoment}$ 
6   $D[a].\text{secondMoment} = D[a].\text{secondMoment} + D[b].\text{secondMoment}$ 
7   $D[a].\text{neighbor} = \text{MERGENEIGHBORS}(a, b)$ 

```

MERGE_NEIGHBORS(a, b)

Input: region IDs a and b

Output: Merged neighbor-list

- 1 SORT($D[a].neighbor - list$) ; Sort neighbors of a according to their region ID, inplace sort
- 2 UNIQUE($D[a].neighbor - list$) ; Remove regions with duplicate ID's from the sorted list
- 3 SORT($D[b].neighbor - list$)
- 4 UNIQUE($D[b].neighbor - list$)
- 5 UNION($D[a].neighbor - list, D[b].neighbor - list$) ; Merged neighbor list is stored at $D[a]$
- 6 UPDATE_WEIGHTS($D[a].neighbor - list$) ; Update the weights of all edges connected to region ' a '
- 7 ; Since we have pointers to all the edges from region ' a ' in the neighbor list,
- 8 ; we can directly access the edges in the $edge - list$ and update it
- 9 MAINTAIN_SORTED_ORDER($edge - list$) ; Can easily accomplished using the skip lists

Chapter 5

Results

In this chapter, we analyze the performance of our algorithm and compare its output to the MST based segmentation algorithm. Initially, we analyze the segmentation results on some synthetic images. Later, we evaluate the algorithm by testing its performance exhaustively on the Berkeley Segmentation dataset.

5.1 Results on synthetic images

As noted in the previous chapters, “leak” is one of the main problems with the MST based segmentation. Even if there exists one edge connecting regions R_i and R_j such that its weight is less than the maximum edge weight in either of the two regions, we end up merging them. Figure 5.1 clearly illustrates this problem. Consider the noisy image with 2 partitions and a thin grayscale ramp between them. The MST based segmentation algorithm ends up merging the entire image because of the ramp, despite their very different appearances. This problem is solved in our approach as we compare the similarity of the regions (modelled as Gaussian distributions) and not individual pixels connected by an edge. Thus, even if there were a leak, it will not adversely affect the segmentation results. As shown in Figure 5.1 , our algorithm ends up with two stable regions, which is more accurate.

Similarly, consider the gradient image shown in figure 5.2. MST based segmentation algorithm, because of its merge criterion, ends up merging the entire image. However, our algorithm produces two regions, which seems more intuitive.

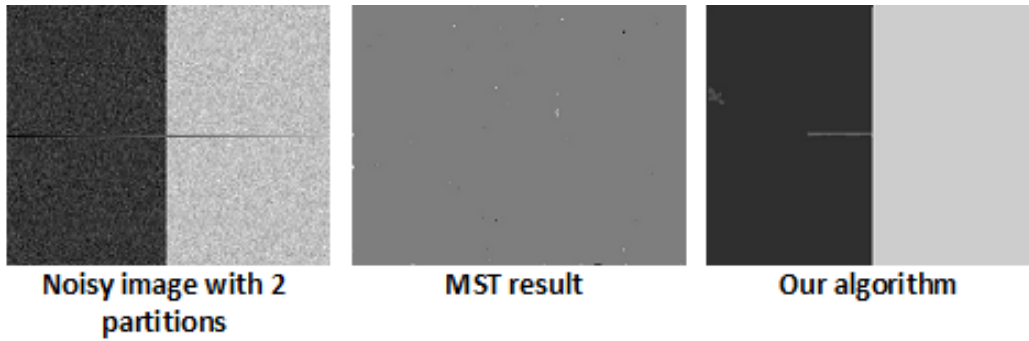


Figure 5.1: The leak of the MST algorithm demonstrated on a synthetic image. a) Noisy image with 2 partitions and a thin grayscale ramp between them. b) MST result c) Our algorithm’s result



Figure 5.2: (a) Gradient image (b) MST result (c) Our algorithm’s result

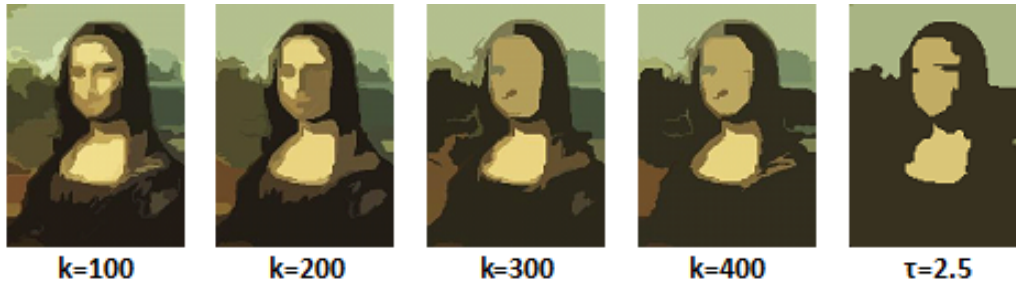


Figure 5.3: Effect of the choice of k on the granularity in the MST segmentation algorithm output. (a)-(d) MST results for various values of k , showing the difficulty of selecting the proper value. (e) Our algorithm for $\tau = 2.5$, which is the same value used for all images, despite scene content and image size.

5.2 Analysis of optimal segmentation

Another drawback of the minimum spanning tree algorithm is that the granularity of segmentation depends on the value of k . This parameter is often chosen arbitrarily and thus there is no control over the granularity of segmentation. This fact is clearly illustrated in Figure 5.3. In our algorithm, since Mahalanobis distance between Gaussian distributions is used to merge regions, we

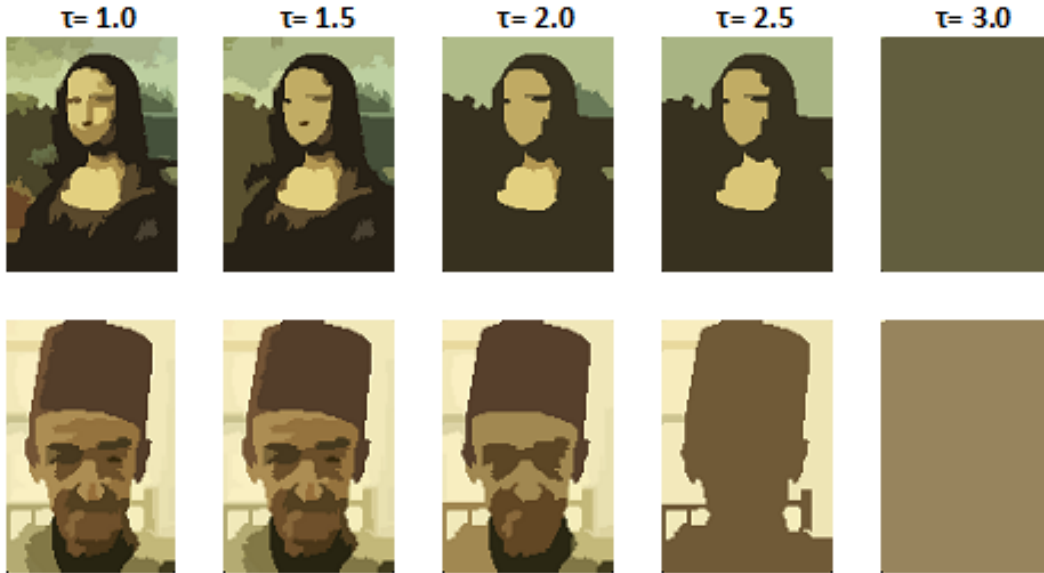


Figure 5.4: Our algorithm- different granularities of segmentation for a) Monalisa b) Man

hypothesize that a distance threshold of 2 – 2.5 would give us optimal results. This represents a good cutoff while comparing distributions because 2 – 2.5 Mahalanobis distance units corresponds to $2\sigma - 2.5\sigma$ in terms of Euclidean distance.

Figure 5.4 shows the segmentation obtained for different values of distance threshold τ . We can observe from Figure 5.5 that the number of segments decreases as τ increases. However, when the τ reaches 2 – 2.5, the curve flattens. In other words, the number of components become relatively stable to the changes in threshold. Such a segmentation represents “stable” regions because the regions are well formed and are not sensitive to parameter changes.

5.3 Results on the BSDS dataset

The algorithm was tested exhaustively on a subset of the Berkeley segmentation database [20]. This database contains 300 *RGB* images of size 481x321 pixels that are randomly chosen from the Corel database. These images are manually segmented by humans in a natural way. According to [20], the following instructions were given to the human subjects who segmented the image: “*Divide each image into pieces, where each piece represents a distinguished thing in the image. It is important that all of the pieces have approximately equal importance. The number of things in each*

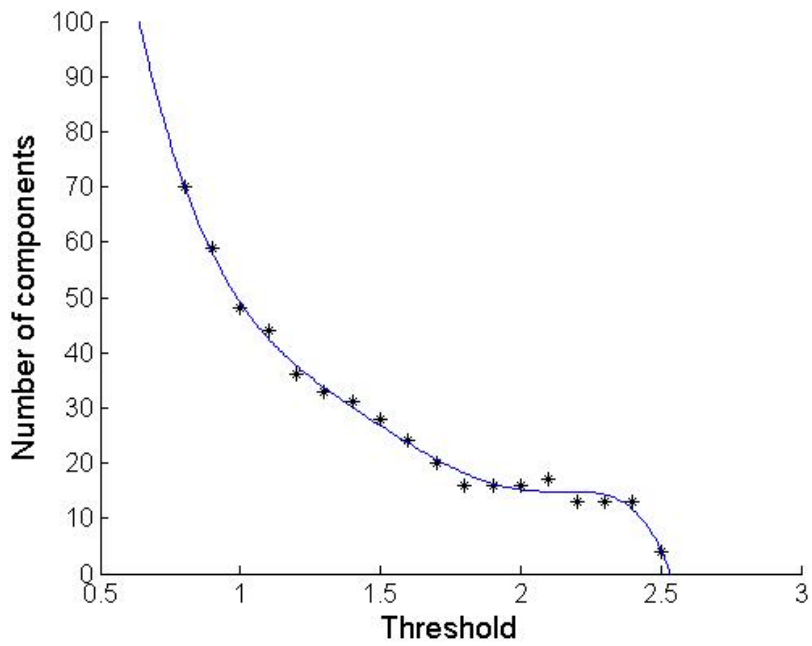
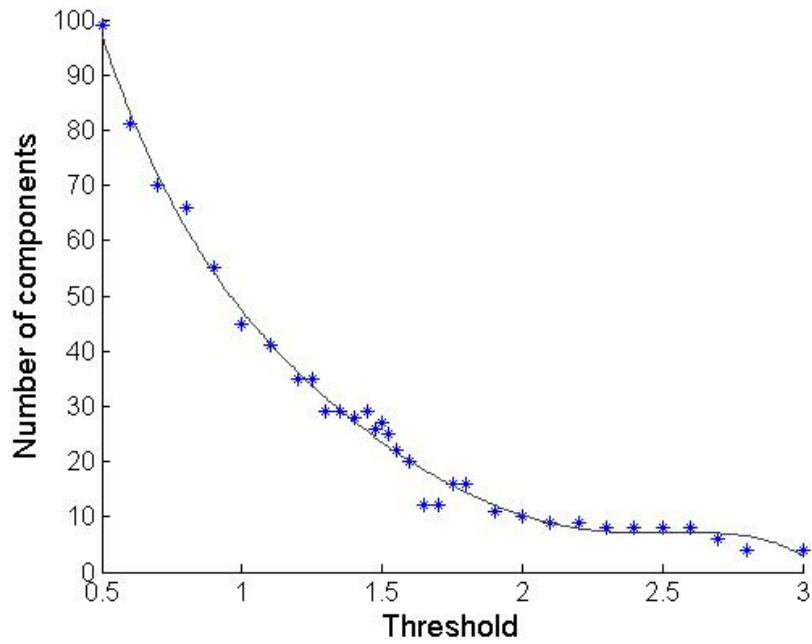


Figure 5.5: Graph showing the relationship between threshold and number of components for Monalisa (TOP) and Man (BOTTOM)

image is up to you. Something between 2 and 20 should be reasonable for any of our images.. ”

We ran the MST algorithm (authors' implementation) and our algorithm using the Berkeley

segmentation dataset [20]. The results are shown in Figures 5.6- 5.13. As can be seen, our results are noticeably improved in a wide variety of scenarios due to the modeling of each region with a Gaussian in RGB space, with no sacrifice in computational efficiency. It can be noticed that for most images, our algorithm produces results that are more closer to “correct” segmentation. Furthermore, our results are much “sharper” than the MST algorithm.

To justify the above claim, let us analyze Figure 5.6 in detail. MST algorithm fails to capture the person on the rock (second row), face of the man kneeling down (third row) and merges parts the bison’s body with the background (fourth row). All these problems are overcome by our algorithm. Furthermore, the airplane (first row) and cow (last row) results are much sharper in our algorithm.



Figure 5.6: BSDS300 Segmentation Results (Mean Colors)- I[20]



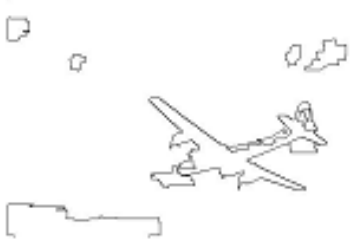

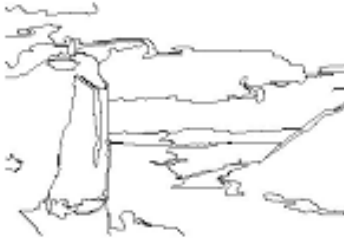








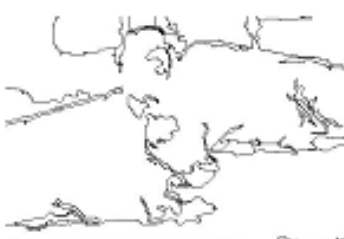

Original Image	MST	Our algorithm
		
		
		
		
		

Figure 5.7: BSDS300 Segmentation Results (Contours)- I [20]

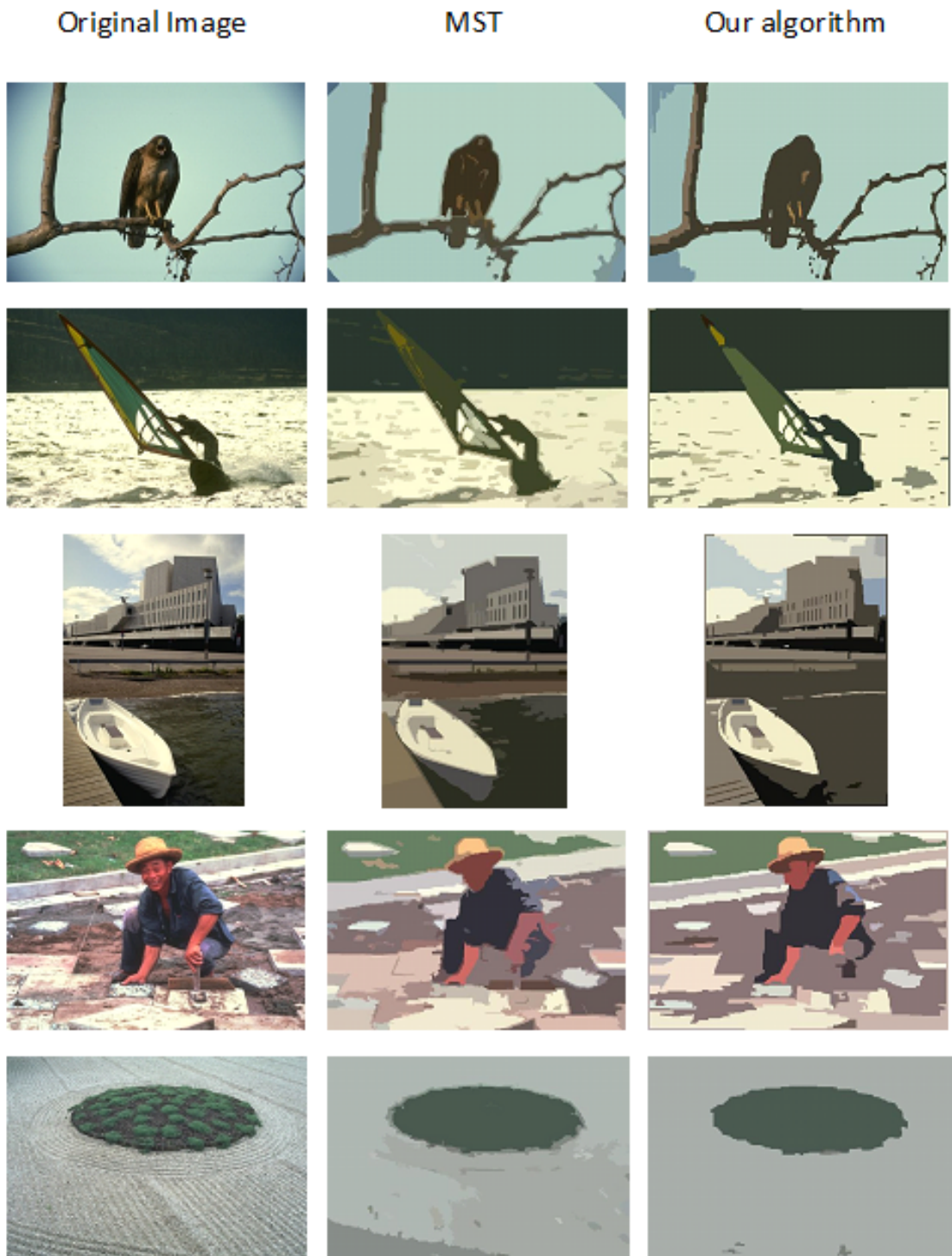


Figure 5.8: BSDS300 Segmentation Results (Mean Colors)- II[20]





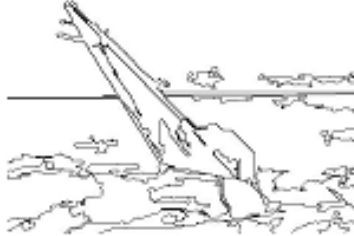


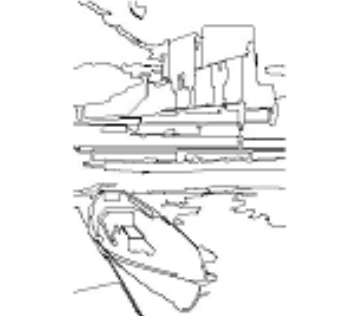
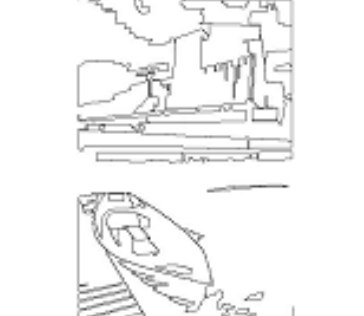






Original Image	MST	Our algorithm
		
		
		
		
		

Figure 5.9: BSDS300 Segmentation Results (Contours)- II [20]

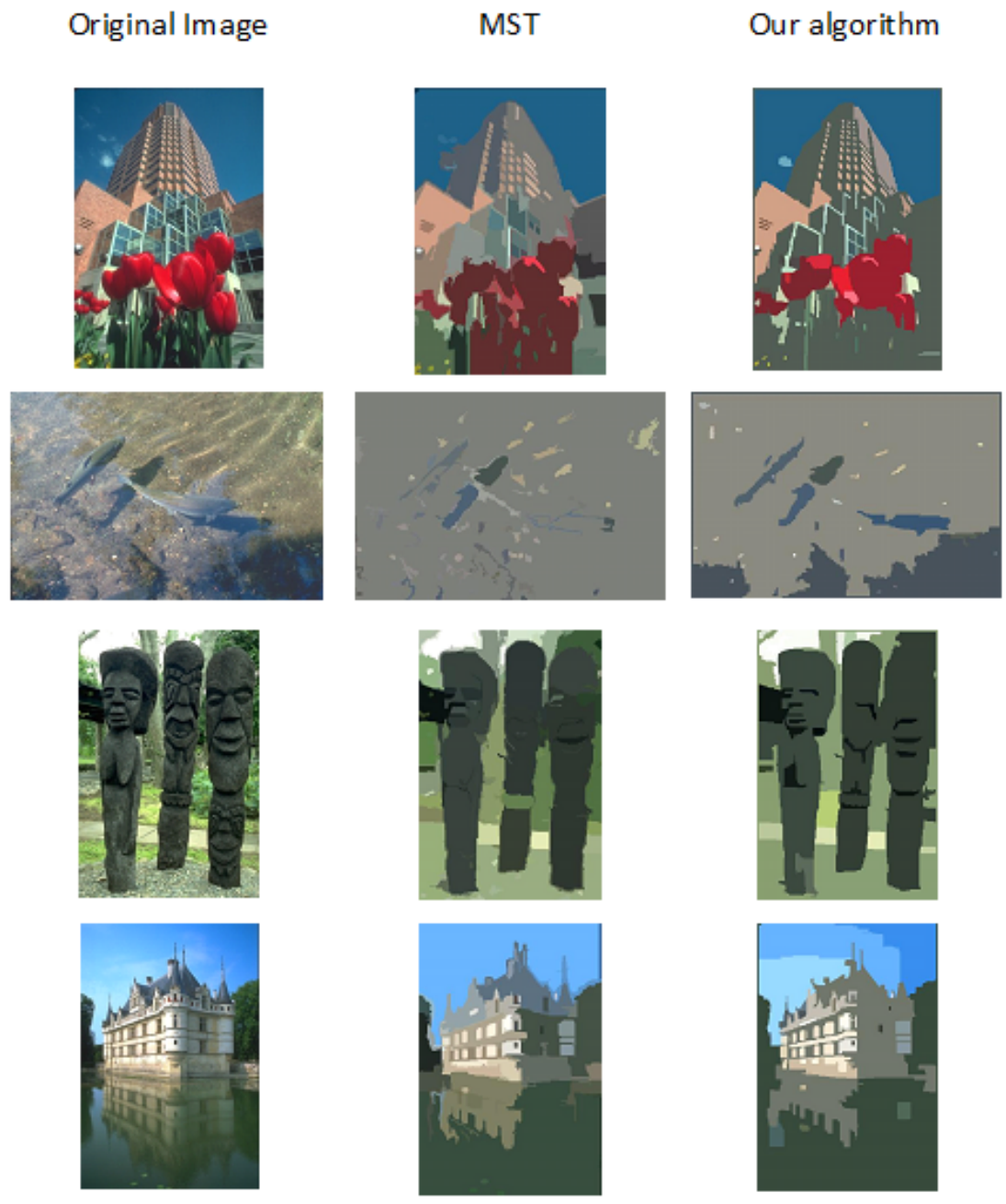


Figure 5.10: BSDS300 Segmentation Results (Mean Colors)- III[20]













Original Image	MST	Our algorithm
		
		
		
		

Figure 5.11: BSDS300 Segmentation Results (Contours)- III [20]

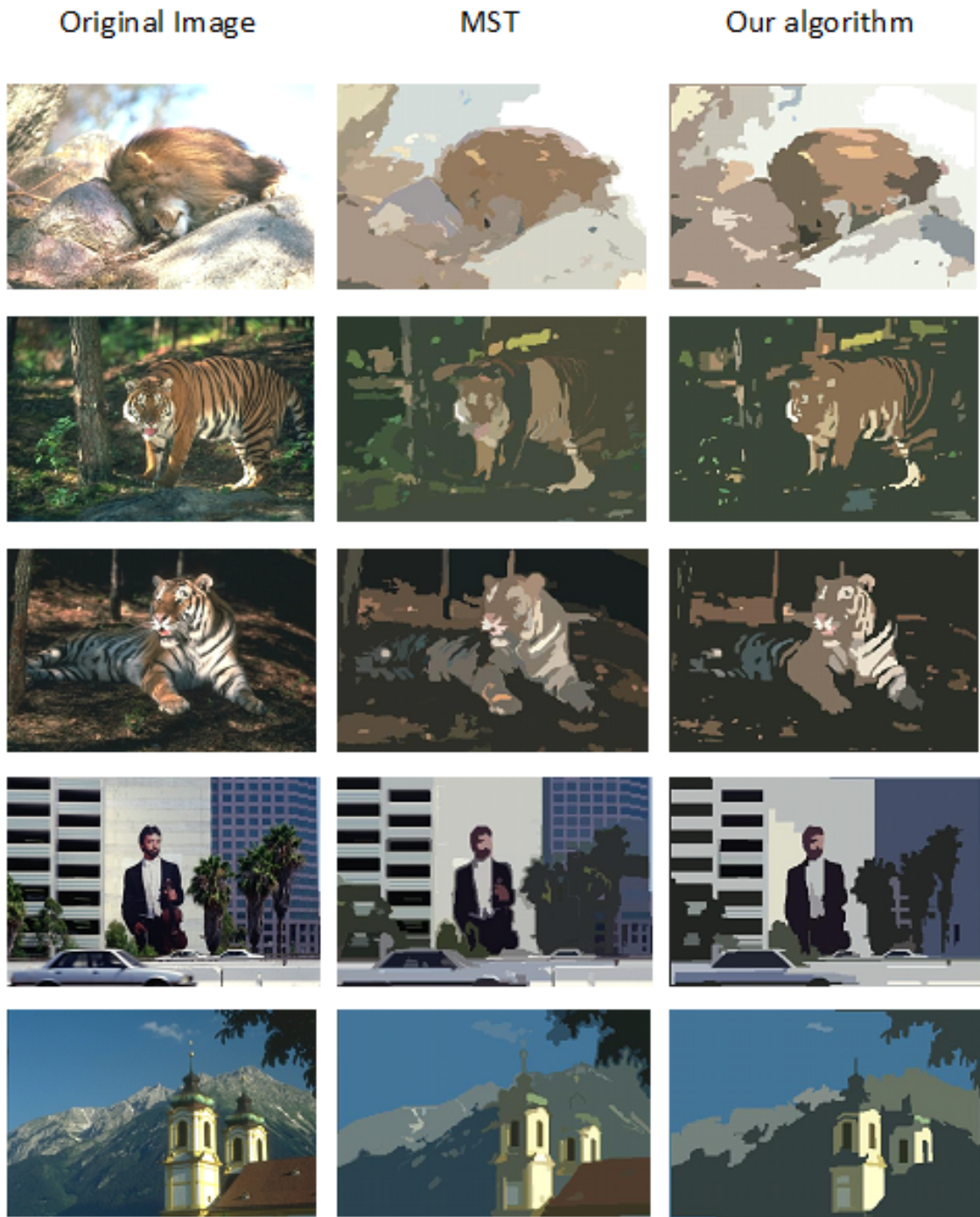


Figure 5.12: BSDS300 Segmentation Results (Mean Colors)- IV[20]







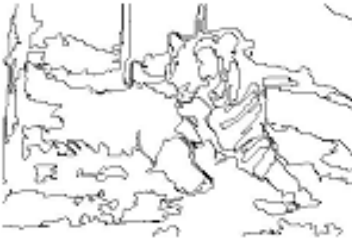






Original Image	MST	Our algorithm
		
		
		
		
		

Figure 5.13: BSDS300 Segmentation Results (Contours)- IV [20]

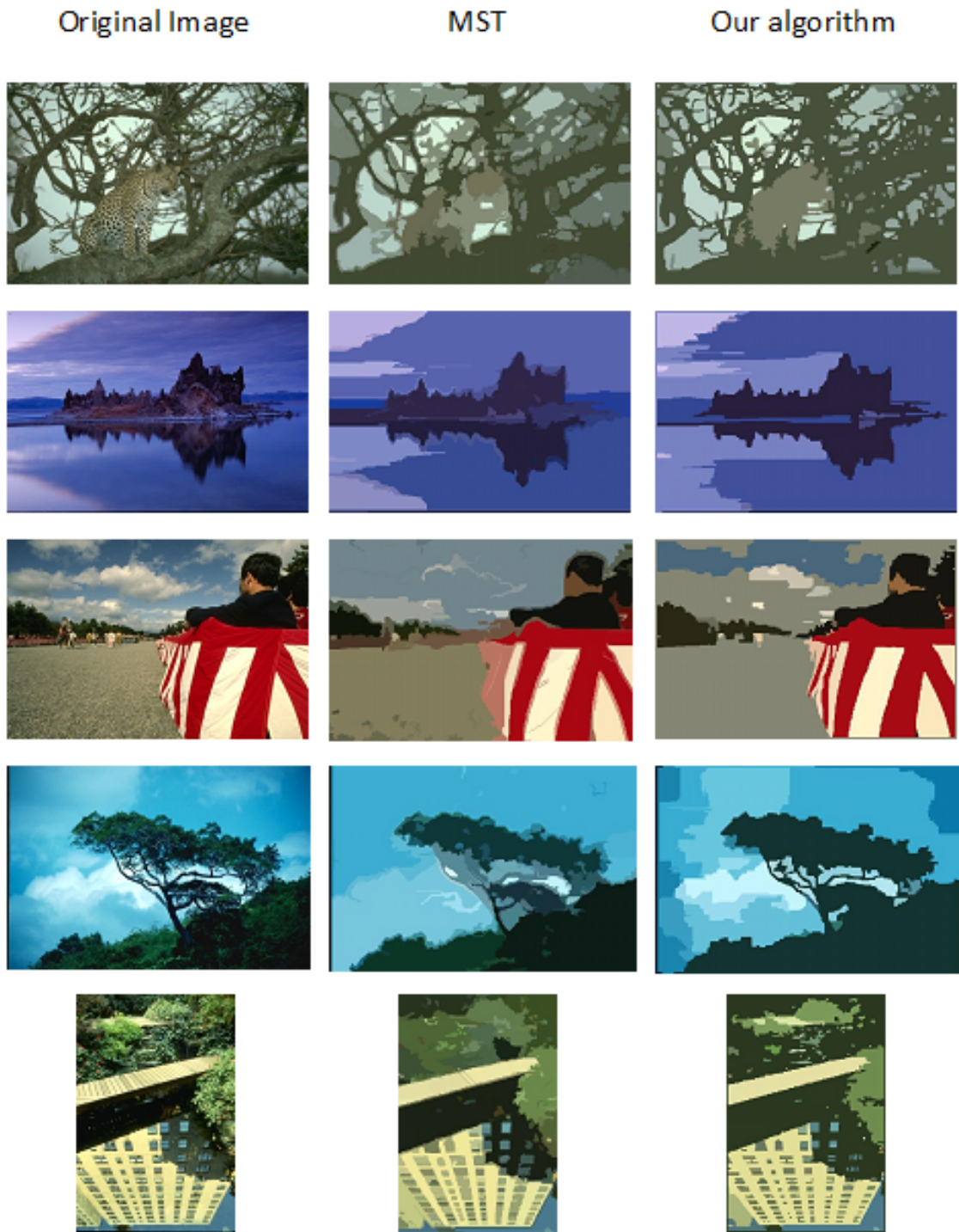


Figure 5.14: BSDS300 Segmentation Results (Mean Colors)- V[20]











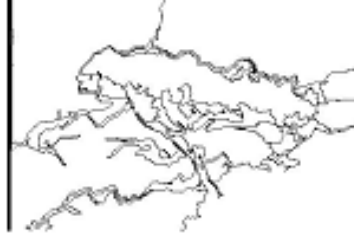




Original Image	MST	Our algorithm
		
		
		
		
		

Figure 5.15: BSDS300 Segmentation Results (Contours)- V [20]



Figure 5.16: BSDS300 Segmentation Results (Mean Colors)- VI[20]

Original Image	MST	Our algorithm
		
		
		
		
		

Figure 5.17: BSDS300 Segmentation Results (Contours)- VI [20]

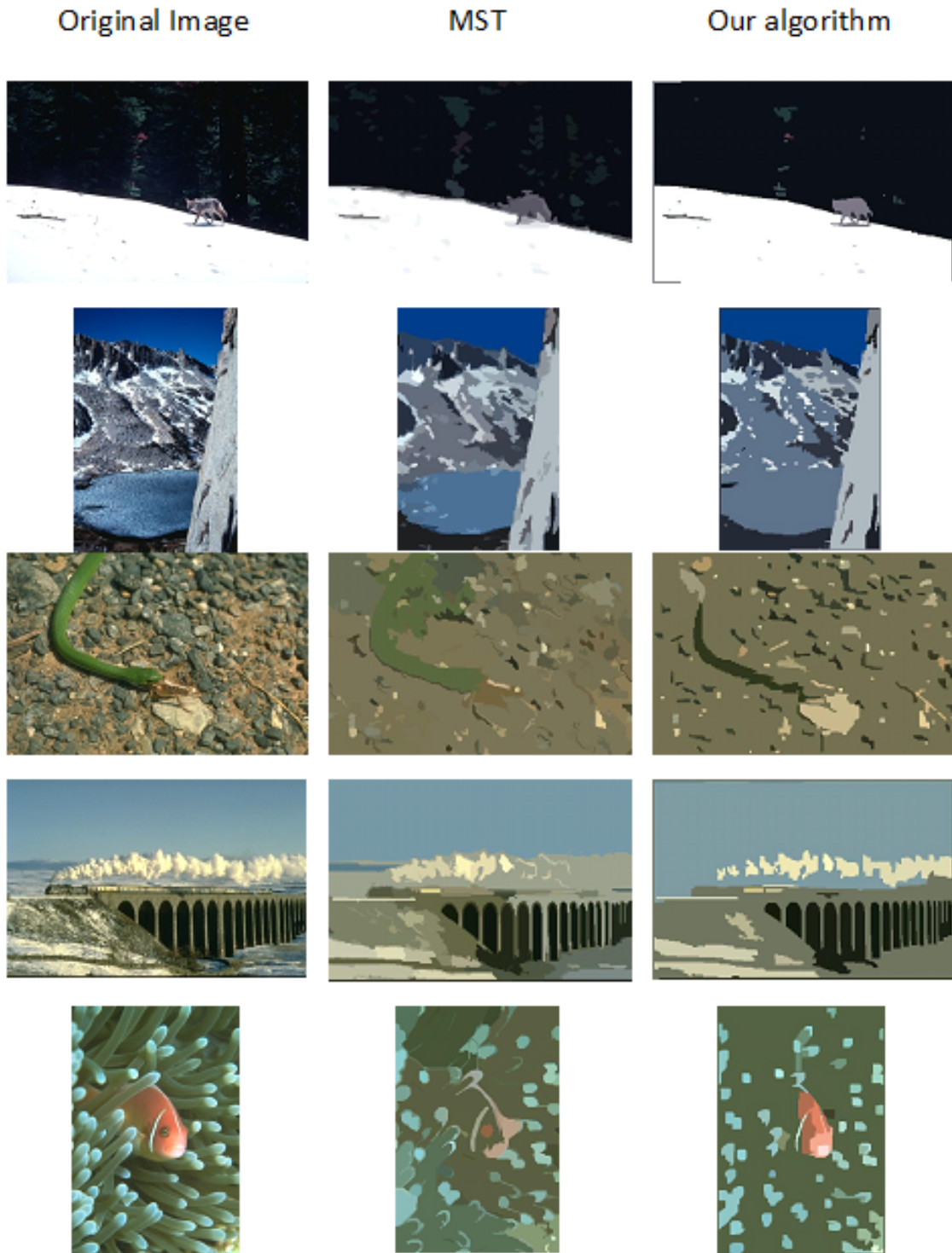


Figure 5.18: BSDS300 Segmentation Results (Mean Colors)- VII[20]











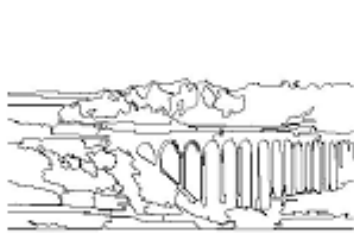




Original Image	MST	Our algorithm
		
		
		
		
		

Figure 5.19: BSDS300 Segmentation Results (Contours)- VII [20]



Figure 5.20: BSDS300 Segmentation Results (Mean Colors)- VIII [20]
















Original Image	MST	Our algorithm
		
		
		
		
		

Figure 5.21: BSDS300 Segmentation Results (Contours)- VIII [20]

Chapter 6

Conclusions and Discussion

We have proposed a novel segmentation algorithm that improves upon the two primary drawbacks of the MST based segmentation. Firstly, we replace the MST criterion for merging regions with bidirectional Mahalanobis distance, assuming that regions are modelled as Gaussian distributions in RGB space. Secondly, due to the use of Gaussians, we demonstrate that this choice of Gaussian distribution leads to a natural intuitive parameter for achieving good segmentation for a wide variety of images. All these results have been obtained without sacrificing the computational efficiency. We have validated our algorithm by testing on a wide variety of synthetic as well as real images. The performance of our algorithm is clearly superior to MST algorithm in most cases.

6.1 Future work

Following are some of the tasks that we plan to undertake in the near future to improve the results of this thesis:

1. Explore data structures that can possibly used to implement the segmentation algorithm such that updating the weights of edges and re-sorting can be performed efficiently.
2. Run the segmentation algorithm on the new Berkeley dataset (BSDS500) and evaluate its performance. Additionally, benchmark the different versions of the algorithm mathematically.
3. Extend the existing segmentation algorithm to use texture along with color features.

4. Preprocess the images of the dataset to make them less invariant to light. Homomorphic filtering would be one option to consider to improve our segmentation results.

Bibliography

- [1] Pablo Arbelaez, Michael Maire, Charless Fowlkes, and Jitendra Malik. Contour detection and hierarchical image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(5):898–916, May 2011.
- [2] Stanley T. Birchfield and Sriram Rangarajan. Spatiograms versus histograms for region-based tracking. In *Proceedings of Computer Vision and Pattern Recognition (CVPR'05) - Volume 2*, CVPR '05.
- [3] Rubén Cárdenes, Rodrigo de Luis-García, and Meritxell Bach-Cuadra. A multidimensional segmentation evaluation for medical image data. *Comput. Methods Prog. Biomed.*, 96(2):108–124, November 2009.
- [4] Yizong Cheng. Mean shift, mode seeking, and clustering. *IEEE Trans. Pattern Anal. Mach. Intell.*, 17(8):790–799, August 1995.
- [5] F. R. K. Chung. *Spectral Graph Theory*. American Mathematical Society, 1997.
- [6] Dorin Comaniciu and Peter Meer. Mean shift: A robust approach toward feature space analysis. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(5):603–619, May 2002.
- [7] Jérôme Da-Rugna, Gaël Chareyron, and Hubert Konik. About segmentation step in content-based image retrieval systems. In *World Congress on Engineering and Computer Science*, pages 550–554. IAENG, 2011.
- [8] Guillaume Damiand and Patrick Resch. Split-and-merge algorithms defined on topological maps for 3d image segmentation. *Graph. Models*, 65(1-3):149–167, January 2003.
- [9] Pedro F. Felzenszwalb and Daniel P. Huttenlocher. Efficient graph-based image segmentation. *Int. J. Comput. Vision*, 59(2):167–181, September 2004.
- [10] David A. Forsyth and Jean Ponce. *Computer Vision: A Modern Approach*. Prentice Hall Professional Technical Reference, 2002.
- [11] Charless Fowlkes, Serge Belongie, Fan Chung, and Jitendra Malik. Spectral grouping using the nystrm method. *IEEE Trans. Pattern Anal. Mach. Intell.*
- [12] C.L. Glisson, H.O. Altamar, S.D. Herrell, P. Clark, and R.L. Galloway. Comparison and assessment of semi-automatic image segmentation in computed tomography scans for image-guided kidney surgery. *Med Phys*, 38(11):6265–74, 2011.
- [13] Murad Al Haj, Andrew D. Bagdanov, Jordi Gonzàlez, and Xavier F. Roca. Robust and efficient multipose face detection using skin color segmentation. In *Proceedings of the 4th Iberian Conference on Pattern Recognition and Image Analysis*, pages 152–159, 2009.

- [14] Xiaofu He and Pengfei Shi. A new segmentation approach for iris recognition based on hand-held capture device. *Pattern Recogn.*, 40(4):1326–1333, April 2007.
- [15] Adam Hoover and Li Yu. Segmentation methods through the stability of region count in the scale space. In *Proceedings of International Conference on Image Processing, Computer Vision and Pattern Recognition*, pages 467–473, 2006.
- [16] Steven L. Horowitz and Theodosios Pavlidis. Picture segmentation by a tree traversal algorithm. *J. ACM*, 23(2):368–388, April 1976.
- [17] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, pages 604–613, New York, NY, USA, 1998.
- [18] J. B. Kruskal. On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem. In *Proceedings of the American Mathematical Society*, 7, 1956.
- [19] Jitendra Malik, Serge Belongie, Thomas Leung, and Jianbo Shi. Contour and texture analysis for image segmentation. *Int. J. Comput. Vision*, 43(1):7–27, June 2001.
- [20] 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. In *Proc. 8th Int'l Conf. Computer Vision*, volume 2, pages 416–423, July 2001.
- [21] Ines Njeh, Ismail Ben Ayed, and Ahmed Ben Hamida. A distribution-matching approach to mri brain tumor segmentation. In *ISBI*, pages 1707–1710, 2012.
- [22] Jorge Núñez and Jorge Llacer. Astronomical image segmentation by self-organizing neural networks and wavelets. *Neural Netw.*, 16(3-4):411–417, April 2003.
- [23] Caroline Pantofaru. *Studies in Using Image Segmentation to Improve Object Recognition*. PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, May 2008.
- [24] Dzung L. Pham, Chenyang Xu, and Jerry L. Prince. Current methods in medical image segmentation1. *Annual Review of Biomedical Engineering*, 2(1):315–337, 2000. PMID: 11701515.
- [25] R. C. Prim. Shortest connection networks and some generalizations. *The Bell Systems Technical Journal*, 36(6):1389–1401, 1957.
- [26] Jianbo Shi and J. Malik. Normalized cuts and image segmentation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(8):888–905, aug 2000.
- [27] M. Sonka, V. Hlavac, and R. Boyle. *Image Processing, Analysis, and Machine Vision*. Chapman & Hall, 2 edition, 1998.
- [28] Michael Spann and Roland Wilson. A quad-tree approach to image segmentation which combines statistical and spatial information. *Pattern Recognition*, 18(3-4):257–269, 1985.
- [29] David A. Tolliver and Gary L. Miller. Graph partitioning by spectral rounding: Applications in image segmentation and clustering. In *Computer Vision and Pattern Recognition, CVPR '06*.
- [30] R. Unnikrishnan, C. Pantofaru, and M. Hebert. A measure for objective evaluation of image segmentation algorithms. In *Computer Vision and Pattern Recognition (CVPR'05) - Workshops - Volume 03*, 2005.
- [31] Yair Weiss. Segmentation using eigenvectors: A unifying view. In *ICCV*, pages 975–982, 1999.

- [32] Li Yu. *On the stability of region count in parameter space of image analysis methods*. PhD thesis, Clemson University, Clemson, 2007.
- [33] Yu Zhao, Zhiqiang Gao, and Baoxiu Mi. Fast image segmentation based on k-means algorithm. In *Proceedings of the 4th International Conference on Internet Multimedia Computing and Service*, ICIMCS '12, pages 166–169, 2012.