Motion Segmentation Using Spanning Trees and Graph Cuts

B. Tech Project Report
Submitted in partial fulfillment of the requirements
for the degree of
Bachelor of Technology

Abhishek Ranjan
Roll No: 99005033

under the guidance of
Prof. Sharat Chandran

Department of Computer Science and Engineering
Indian Institute of Technology
Bombay
May 26, 2003

Acknowledgement

I would like to thank Dr. Sharat Chandran, for his invaluable guidance and encouragement. Without his support the realization of this project would not have been possible.

Abhishek Ranjan

Abstract

Motion segmentation in videos involves identifying regions in the frames of the video that correspond to independently moving objects. This is one of the key techniques that help solve various problems encountered when dealing with image sequences, such as redundancy elimination in digital video and tracking of moving objects.

Graph theoretic approaches have been widely used for performing segmentation. In this report we have studied some such interesting graph algorithms. We propose an algorithm for segmentation which uses the minimum spanning tree algorithm and a special graph cut, called normalized cut.

Contents

Chapter 1

Introduction

Digital image and video analysis techniques form the basis for many new multimedia services. Recent applications of these techniques in the fields of medical imaging, automated surveillance, intelligent military defense systems, entertainment industries, etc. show their potential use. The key step in most of these applications is to take inputs as digital videos or images and find out "regions" in them. This is step is called image segmentation if the input is an image and video segmentation if the input is a video sequence. In general the problem of segmentation involves identifying various mutually disjoint sets of points in a space, with points in one set lying close to each other.

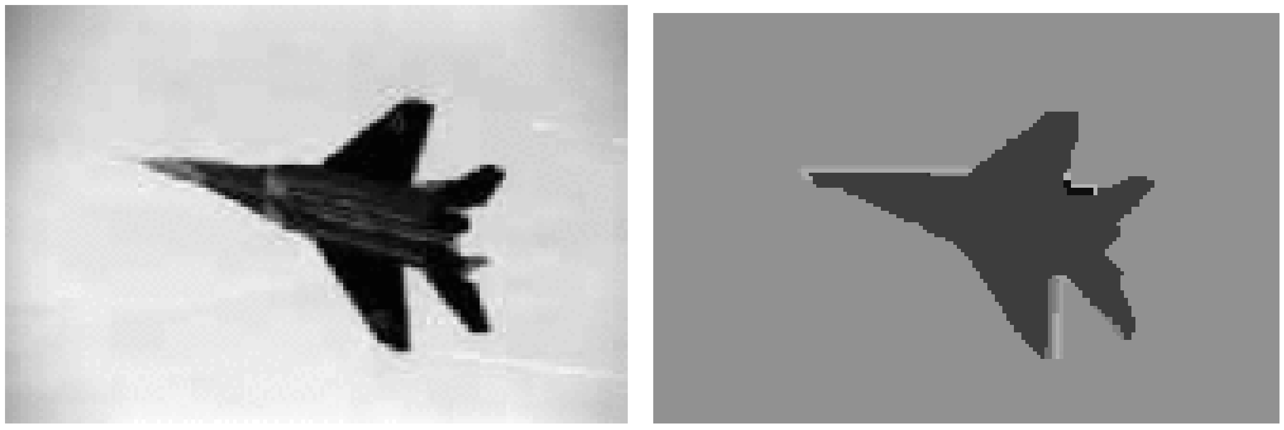

Figure 1.1: An application in intelligent military defense system: On the left an input image is given and on the right the fighter jet in the image has been identified as a region. This segmentation process forms an integral part of the target detection unit in automated anti-aircraft missile control.

Image segmentation has been among the most challenging problems in computer vision for the last three decades. From the Gestalt psychology of perception to the modern complicated mathematical formulations, there have been numerous philosophies for segmentation. These continued efforts made by various scientific communities suggest that the approach and the criteria of goodness of segmentation rely heavily on the end-application, e.g. a face detection application needs an accurate and fine segmentation of the face image, while a real time traffic control application needs a fast algorithm for detecting most prominently moving bodies on the road.

These requirements point to two basic characteristics of a segmentation algorithm

- **Feature Space.** Before starting the segmentation, features which decide the similarity or dissimilarity of pixels must be determined. These set of features form the feature space for the segmentation. Some common feature spaces are gray values of pixels for a gray scale image, RGB values for color images, motion profiles of pixels for sequence of images or videos.

- **Level.** This refers to the level of segmentation: fine or coarse. A fine segmentation forms a large number of small sized clusters, each cluster containing closely spaced pixels in some feature space. On the other hand, a coarse segmentation creates few large clusters. The ideal size of clusters depends upon the input. Segmentation algorithms often suffer from inappropriate level of segmentation, either under-segmentation or over-segmentation.

We address these issues in this report and analyze the solutions proposed.

1.1 Image versus video

As we have seen that the input to the problem of segmentation can either be a video or an image. But there is an inherent similarity between these two media. We can think of a video as a sequence of images so the basic unit on which the video segmentation algorithms operate is actually an image or a frame. The difference is that video segmentation must consider a larger feature space because they have moving objects. Informally we can say that video segmentation is essentially a segmentation problem, similar to the image segmentation problem, with pixel motion being an important dimension of the feature space. Figure 1.2 illustrates this similarity between videos and images. In this figure, region covered by athlete Carl Lewis has been segmented from the background in various frames of a video and these frames are shown separately. It can be seen in the figure that each frame is an image and the portion of athlete has been segmented in each of them.

In this report we explain some measures to characterize pixel motion as a feature space dimension, and then discuss general algorithms for segmenting an image in a given feature space.

1.2 Segmentation and graph theory

Graphs have been used for representing several systems in computer science. Images and videos are also represented as graphs. The pixels can be represented by nodes of the graph and similarity of two pixel values can be expressed as weighted edges. This representation of images as graphs allows the use of various well studied properties of graphs in the problem of segmentation. In this project we discuss the notion of normalized cuts and segmentation based on this cut. We further use spanning trees to speed up the normalized cut based segmentation algorithm.

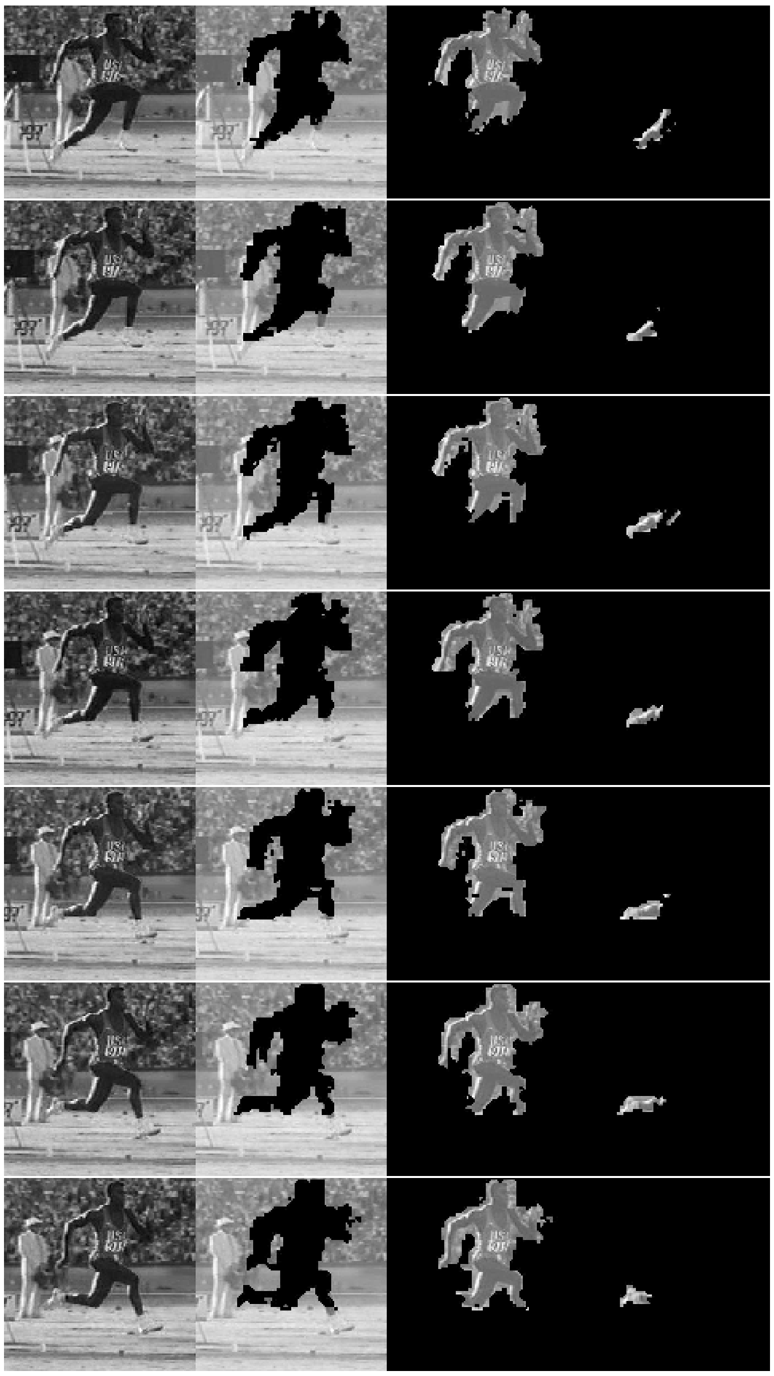

Figure 1.2: Segmentation of a sequence of frames of a video clip of Carl Lewis, using the normalized cut method [6]

1.3 Overview of the report

This report concentrates on the graph theoretic image segmentation, two philosophies of segmentation and our approach to combine the two for a faster algorithm:

- Chapter 2 discusses basic concepts relevant to video segmentation. We describe the feature space for the motion segmentation. Two broad approaches for motion segmentation have been briefly discussed.

- Chapter 3 deals with two different graph theoretic approaches for image segmentation, namely Normalized cuts technique and Local variation techniques.
- Chapter 4 describes our proposal for combining two approaches to get a faster hierarchical segmentation algorithm.
- Chapter 5 briefs the implementation of the algorithm proposed.
- Chapter 6 summarizes the work done in this project.

Chapter 2

Preliminary Ideas and Common Approaches

Video analysis involves several steps before we actually start operations on frames. In this chapter we will briefly discuss these steps and then move on to motion segmentation techniques.

2.1 Basics

Production process of video involves several complex movements of camera. These movements of camera cause the motion of the objects across various frames. A sequence of frames which are created by continuous motion of camera such as panning, zooming etc. is called a *shot*. These shots do not contain abrupt changes in scene. Common videos are sequences of such shots. For video analysis purposes we work at a lower level where video is separated into its shots. Overall process of video analysis involves two steps:

- *Shot detection:* In this step video is divided into its shots. Shots are separated based on the difference between pixel values of the frames. Several algorithms/systems have been developed for shot detection.
- *Analysis:* Frames of a shot are further analyzed based on the requirement.

In this project we assume that the shots have already been detected in the video. We address the video analysis problem of motion segmentation. This aims at breaking a scene into its most prominent moving groups.

2.2 Temporal analysis

In videos, objects and background keep changing over time. We refer to this aspect of videos as *temporal factor* in videos. Temporal factor is one of the most important factors which distinguishes motion segmentation in videos from image segmentation. The issue of temporal factor is addressed by characterizing the difference between frames in the frame sequence of the video. Common motion segmentation algorithms can be broadly divided into two categories on the basis of the way they deal with differences across frames:

1. *Approach based on Spatial Homogeneity:* In this approach, images (frames) are simplified using some filters, and then region boundary decision is made. Thus image now has distinct regions separated by the boundaries determined above. Next, for each region, *motion vector* is calculated and regions with similar motion vectors are merged together. Thus this approach stresses on tracking the boundary of objects, but is computationally costly because of boundary detection and motion vector calculation, which are themselves costly operations.

2. *Approach based on change detection:* In this approach change in the frames form the primary criteria for segmentation, rather than the spatial similarity. This approach is in some sense reverse of the previous one. Here moving objects are detected on the basis of differences between two consecutive frames. And boundary fine-tuning based on the spatial-temporal information is done later. Advantage with this approach is that it stresses on the motion of the object. It has been found to be better than the previous approach [7].

The concept of *motion vector*, used above, is based on the motion estimation techniques, i.e. we need to estimate the change in the coordinates of a particular pixel value across frames. There are several approaches for this estimation. We describe two of them below:

1. *Optical Flow:* This is the apparent motion of the brightness patterns across the image plane. This is based on the assumption that if an object changes its position across frames then its intensity pattern remains the same. Let the image sequence be parametrized by an intensity function $f(x, y, t)$, at coordinate (x, y) and time t. Then from above stated assumption, for a displacement (in spatial and temporal domain) of (dx, dy, dt), $f(x, y, t) = f(x + dx, y + dy, t + dt)$. Now, using Taylor expansion:

$$f(x+dx, y+dy, t+dt) = f(x,y,t) + \frac{\delta f}{\delta x}dx + \frac{\delta f}{\delta y}dy + \frac{\delta f}{\delta t}dt + \ldots \tag{2.1}$$

$$\begin{aligned}\Rightarrow\quad & \frac{\delta f}{\delta x}dx + \frac{\delta f}{\delta y}dy + \frac{\delta f}{\delta t}dt + \ldots = 0\\ \Rightarrow\quad & -\frac{\delta f}{\delta t} = \frac{\delta f}{\delta x}u + \frac{\delta f}{\delta y}v\\ \Rightarrow\quad & -\frac{\delta f}{\delta t} = \triangle.\underline{u}\end{aligned}$$

Here $\triangle = (\frac{\delta f}{\delta x}, \frac{\delta f}{\delta y}), u = \frac{dx}{dt}, v = \frac{dy}{dt}$ and $\underline{u} = (u, v)$. The aim is to assign the velocity vector $\underline{u}$ to each pixel. But here we have two unknowns (u, v) and one equation. So, we can not determine the variables. Though several approaches apply additional constraints to estimate the variables of $\underline{u}$, this is still a drawback in this approach and is known as the Aperture problem.

2. *Motion Profile:* This is a measure of the probability distribution of the image velocity at each pixel [6]. The advantage of motion profile is that it considers the direction of the motion along with the uncertainty associated with it. Let $I^t(X_i)$ denote the image pixel value at location $X_i \in R^2$ at time t. Now, at time $t+1$ this pixel value can be found at $X_i + dx, dx \in R^2$. So, we define $P_i(dx)$, for a particular dx, as the probability that $I^t(X_i)$ corresponds to $I^{t+1}(X_i + dx)$. $P_i(dx)$ is estimated by first computing the similarity between $I^t(X_i)$ and $I^{t+1}(X_i + dx)$ as $S_i(dx)$, and then normalizing it as :

$$P_i(dx) = \frac{S_i(dx)}{\sum_{dz} S_i(dz)} \tag{2.2}$$

   Thus with each pixel we have associated with it a measure of motion of that pixel.

2.3 Motion segmentation

Having dealt with the concepts of temporal analysis in the last section, in this section we will discuss motion segmentation and will look at an algorithm which is directly based on the *change detection*. A detailed survey of various strategies and frameworks for image and motion based video segmentation can be found in [8]. In this report we are chiefly concerned with *graph theoretic* approaches.

2.3.1 Segmentation using background registration technique

In [7], a video motion segmentation algorithm has been proposed based on change-detection. The algorithm is divided into the following five major steps:

1. *Frame Difference Mask Calculation:* For each pixel a *Frame Difference Mask(FD)* is calculated by considering the change in the pixel value for two consecutive frames. If the change in the pixel value is greater than a threshold then that pixel is masked as $changing$, and $stationary$ otherwise.

2. *Background Registration:* In this step FD calculated in the above step is used for detecting background region. If FD is $stationary$ for several consecutive frames then that pixel is marked as a part of the background. This is done by maintaining a stationary map M of pixels. Every time a pixel is found to be $stationary$, the count corresponding to that pixel in M is incremented. Thus the value of a pixel in M denotes the extent to which that pixel should belong to background.

3. *Background Difference:* This step generates a *Background Difference Mask(BD)*. This is calculated for each pixel by thresholding the difference between current frame value and the value stored in M. This value will be later used in object detection.

4. *Object Detection:* Using the values calculated in above steps, for each pixel an *Object Mask(OM)* is calculated under various region descriptions (e.g. Moving, Stationary, Uncovered Background etc.). So, OM assigns one of the regions to each pixel.

5. *Post Processing:* After step 4, an initial OM is generated. But it contains some noise in background and object both. In post processing step, these noise are removed by smoothing the boundary and applying other filters.

Figure 2.1 shows a result of running this background registration technique. Moving object and its shadow have been well segmented from the background.

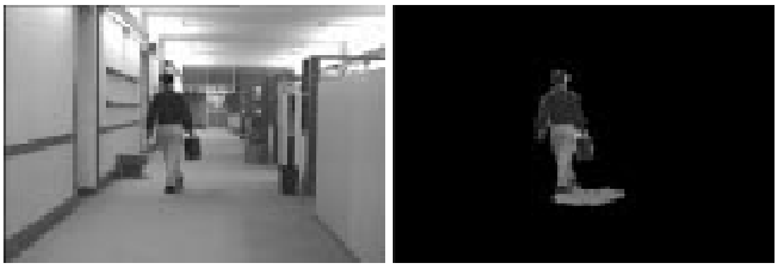

Figure 2.1: Moving object segmentation using background registration technique [7]

2.3.2 Segmentation based on graph theoretical approach

Graph theory is a well studied branch of computer science and several properties of graph are exploited for segmentation and clustering purposes. A general clustering algorithm based on flows has been proposed by Wu and Leahy [11]. [4], [1] use minimum spanning trees for segmentation. Similarly different forms of graph cuts has also been used for segmenting images, e.g. nested cuts [10], normalized cuts [5]. In graph theoretic motion segmentation algorithms temporal changes in the frames are characterized first. Thus, each pixel gets some spatial-temporal characteristic associated with it. Next, a graph is constructed from frames. So, the problem then is to identify sub-graphs in the graph with similar spatial-temporal properties (such as intensity variation in space or across frames). In the following chapters we will discuss graph theoretic image segmentation techniques.

Chapter 3

Graph Theoretic Image Segmentation Techniques

The problem of image segmentation is to partition the image into its *meaningful* components. Partition tries to cluster together the points which are closer to each other in some feature space. Based on the way the segmentation is done, these approaches can be put under two broad categories:

1. First approach can be called *hierarchical* or *top down*. In this approach image is partitioned in two parts based on some global criteria. And then it is segmented recursively in a hierarchical manner. Graph-cut based approaches, which minimizes the cut value based on some global criteria, fall under this category.

2. Second approach can be called *local approach*. In this approach segmentation process proceeds by creating clusters at several places in the image based on some local similarity criteria. The algorithm described in [4], where segmentation is done by growing a minimum spanning tree, is an example of this approach.

In this Chapter we will discuss two algorithms: "Image segmentation using Normalized Cuts" and "Image segmentation using Local Intensity Variation".

3.1 Segmentation using normalized cut (Ncut)

Segmentation using Ncut partitions the image in top down manner. For a partition V_1, V_2 of graph $G(V, E)$, the quantity Ncut, intuitively, denotes how strongly connected are the nodes of V_i among themselves as compared to the $cut(V_1, V_2)$. Thus, as Ncut value decreases, ratio of sum of edge-weights inside each partition to sum of edge-weights across partition increases. So, in case of graph representation of images, if weights of edges are the measure of similarity, then Ncut gives a measure of goodness of image segmentation, such that lower is the Ncut value, better is the segmentation. In this section we will discuss the algorithm which partitions the image by minimizing the ncut. We will first represent the Image as a graph and formally define normalized cut for the graph, then we will discuss the minimization of Ncut. For this we will first write Ncut in terms of some matrices and then will operate on those matrices.

3.1.1 Graph representation of image

Let I be an $n \times n$ pixel image. For representing it as a graph $G(V, E)$, V is same as the set of pixels. Connect each pixel to its 4 neighboring pixels. So the image is represented as a grid-graph. Weight $w(u, v)$ of the edge between vertices u, v is a measure of similarity between u, v : higher the similarity, higher is the weight. One such weight function could be $w(u, v) = e^{-k|I_u - I_v|}$. Once we have found a graph representation for the image, we will apply *normalized cuts* recursively to segment the image.

3.1.2 Normalized cuts

Let $G(V, E)$ be a graph with $|V| = n$, and $w(i, j)$ denote the weight of the edge joining v_i, v_j. *normalized cut* $V = \{A, B\}$ is defined as,

$$Ncut(A, B) = \frac{cut(A, B)}{assoc(A, V)} + \frac{cut(A, B)}{assoc(B, V)} \tag{3.1}$$

where,

$$cut(A, B) = \sum_{v_i \in A, v_j \in B} w(i, j) \tag{3.2}$$

$$assoc(X, V) = \sum_{v_i \in X, v_j \in V} w(i, j) \tag{3.3}$$

Next, we will write this $Ncut$ in the form of following matrices and variables:

$$x = \begin{bmatrix} x_1 & x_2 & \dots & x_n \end{bmatrix}, x_i = \left\{ \begin{array}{ll} 1, & x_i \in A \\ 0, & x_i \in B \end{array} \right\} \tag{3.4}$$

$$d = \begin{bmatrix} d_1 & d_2 & \dots & d_n \end{bmatrix}, \textit{where } d_i = \textstyle\sum_m w(i, m) \tag{3.5}$$

$$D = \begin{bmatrix} d_1 & 0 & \dots & 0 \\ 0 & d_2 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & d_n \end{bmatrix} \tag{3.6}$$

$$W = \begin{bmatrix} w(1,1) & w(1,2) & \dots & w(1,n) \\ w(2,1) & w(2,2) & \dots & w(2,n) \\ \vdots & \vdots & \vdots & \vdots \\ w(n,1) & w(n,2) & \dots & w(n,n) \end{bmatrix} \tag{3.7}$$

$$k = \frac{\sum_{i, x_i > 0} d_i}{\sum_i d_i} \tag{3.8}$$

$$b = \frac{k}{1-k} \tag{3.9}$$

$$b \;=\; \frac{\sum_{x_i>0} d_i}{\sum_{x_i<0} d_i} \tag{3.10}$$

Therefore, we can write $Ncut(A, B)$

$$\begin{aligned}
&= \frac{\sum_{i=1,x_i>0}^{n} \sum_{j=i+1,x_j<0}^{n} -w(i,j)x_ix_j}{\sum_{i,x_i>0} d_i} + \frac{\sum_{i=1,x_i<0}^{n} \sum_{j=i+1,x_j>0}^{n} -w(i,j)x_ix_j}{\sum_{i,x_i<0} d_i} \\
&= \frac{(1+x)^T(D-W)(1+x)}{4k1^TD1} + \frac{(1+x)^T(D-W)(1-x)}{4(1-k)1^TD1} \\
&= \frac{[(1+x)-b(1-x)]^T(D-W)[(1+x)-b(1-x)]}{4b1^TD1}
\end{aligned} \tag{3.11}$$

Setting,

$$y = \frac{(1+x)}{2} - b\frac{(1-x)}{2} \tag{3.12}$$

we can see that $y_i = \left\{ \begin{array}{ll} 1, & x_i \in A \\ -b, & x_i \in B \end{array} \right\}$
also,

$$y^TD1 = \sum_{x_i>0} d_i - b\sum_{x_i<0} d_i = 0 \tag{3.13}$$

$$\begin{aligned}
y^TDy &= \sum_{x_i>0} d_i + b^2\sum_{x_i<0} d_i \\
&= b(\sum_{x_i>0})d_i + b\sum_{x_i<0} d_i \\
&= b1^TD1
\end{aligned} \tag{3.14}$$

Therefore, using (3.12) and (3.14) in (3.11) we get :

$$Ncut(A, B) = \frac{y^T(D-W)y}{y^TDy} \tag{3.15}$$

and from (3.13) y is constrained by following conditions:

$$y_i \in \{1, -b\} \tag{3.16}$$
$$y^TD1 = 0 \tag{3.17}$$

Thus solving normalized cuts(Ncut) problem has been reduced to solving following minimization problem for $y_{n\times 1}$:

$$\min_{y^TD1=0, y_i\in\{1,-b\}} \frac{y^T(D-W)y}{y^TDy} \tag{3.18}$$

Where, for partition {A, B} of Vertex set $\mathcal{V}$

$$b = \frac{\sum_{x_i>0} d_i}{\sum_{x_i<0} d_i} = \frac{\sum_{v_i\in A}\sum_{v_j\in\mathcal{V}} w(i,j)}{\sum_{v_i\in B}\sum_{v_j\in\mathcal{V}} w(i,j)} \tag{3.19}$$

and $y = [y_1 y_2 \dots y_n]$. $\frac{y^T(D-W)y}{y^T D y}$ is called the *Rayleigh Quotient* $R(y)$. Our aim is to minimize the $R(y)$ in (1). For this, we will start with (3.18), and using *Theorem 1* we will find an approximate solution.

Theorem 1: Let $A \in R^{n\times n}$ be a Symmetric Matrix, with eigenvalues $\lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_n$ (*Eigenvalues of a real symmetric matrix are always real*), and the corresponding eigenvectors are $p_1, p_2, \dots p_n$, such that $p_i^T p_j = \delta_{ij}$ (δ_{ij} is *Kroneker Delta*). Let V be a vector space of finite dimension n, and for $k = 1, 2 \dots n$, let V_k denote the subspace of V spanned by vectors $p_1 p_2 \dots p_k$. Let $R_A(v) = \frac{v^T A v}{v^T v}$ $v \in V, v \neq 0$, then

$$\lambda_k = \min_{v \perp V_{k-1}} \frac{v^T A v}{v^T v} \tag{3.20}$$

(*Theorem and proof in* [2])

3.1.3 Approximate minimization

Let

$$D^{\frac{1}{2}} = \begin{bmatrix} \sqrt{d_1} & 0 & \dots & 0 \\ 0 & \sqrt{d_2} & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & \sqrt{d_n} \end{bmatrix} \tag{3.21}$$

$$D^{-\frac{1}{2}} = (D^{\frac{1}{2}})^{-1} = \begin{bmatrix} \frac{1}{\sqrt{d_1}} & 0 & \dots & 0 \\ 0 & \frac{1}{\sqrt{d_2}} & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & \frac{1}{\sqrt{d_n}} \end{bmatrix} \tag{3.22}$$

$$D^{\frac{T}{2}} = (D^{\frac{1}{2}})^T \tag{3.23}$$

$$D^{-\frac{T}{2}} = (D^{\frac{T}{2}})^{-1} \tag{3.24}$$

$$\Rightarrow D^{\frac{T}{2}} = D^{-\frac{T}{2}} = D^{-\frac{1}{2}} \tag{3.25}$$

Now, substituting

$$y = D^{-\frac{1}{2}} u \tag{3.26}$$

in (3.18) we get :

$$\min_{u^T D^{\frac{1}{2}} 1 = 0, u_i \in \{\sqrt{d_i}, -b\sqrt{d_i}\}} \frac{u^T D^{-\frac{T}{2}} (D - W) D^{-\frac{1}{2}} u}{u^T u} \tag{3.27}$$

Feasibility of this substitution is discussed in the next section. For the time being we assume that we can always do this substitution. Let $M = D^{-\frac{T}{2}}(D - W)D^{-\frac{1}{2}}$, so, from above equation we get

$$\min_{u^T D^{\frac{1}{2}} 1 = 0, u_i \in \{\sqrt{d_i}, -b\sqrt{d_i}\}} \frac{u^T M u}{u^T u} \tag{3.28}$$

So, solving (3.18) for y is equivalent to solving (3.28) for u. Next we will find approximate solution to (3.28). Eigenvalue equation for M is

$$Mz = \lambda z \tag{3.29}$$

Now we note that M has following properties:

1. M is symmetric. Here $(D-W)$ is a symmetric matrix. $D^{-\frac{1}{2}}, D^{-\frac{T}{2}}$ are Diagonal matrices, and $D^{-\frac{1}{2}} = D^{-\frac{T}{2}}$. So, M is also a symmetric matrix, because we have a result that *If $X_{n\times n}$ is a diagonal matrix and $A_{n\times n}$ is a symmetric matrix then XAX is also a symmetric matrix* (**Proof 1**, Appendix).
2. M has n Real eigenvalues, because we have the result that *Real Symmetric matrix has real eigenvalues* (**Proof 2**, Appendix)
3. M has n orthogonal eigenvectors, because we have a result that *Real Symmetric $n \times n$ matrix has orthogonal eigenvectors* (**Proof 3**, Appendix).
4. M is a Symmetric Positive Definite matrix (**Proof 4**, Appendix).
5. M has non-negative eigenvalues, because we have the result that *Symmetric Positive Definite matrix has non-negative eigenvalues* (**Proof 5**, Appendix).

Let M has eigenvalues $\lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n$ and corresponding eigenvectors are $z_1, z_2 \ldots z_n$. Since $M \in R^{n\times n}$ is Symmetric (by property 1), has n real eigenvalues (by property 2), has orthogonal set of eigenvectors (by property 3), therefore, from *Theorem 1*

$$\lambda_k = \min_{u\perp V_{k-1}, u\in V, u\neq 0} \frac{u^T M u}{u^T u} \tag{3.30}$$

For $k = 2$, V_2 is the subspace spanned by eigenvector z_1 which corresponds to the smallest eigenvalue λ_1. So, condition $u \perp V_1 \Rightarrow u^T z_1 = 0$. Thus we can write (5) as:

$$\lambda_2 = \min_{u^T z_1 = 0, u\in V, u\neq 0} \frac{u^T M u}{u^T u} \tag{3.31}$$

Next we will find z_1 and λ_1. For this, we put $z = D^{\frac{1}{2}}1$ in (3.29) to get

$MD^{\frac{1}{2}}1 = \lambda D^{\frac{1}{2}}1$
$\Rightarrow D^{-\frac{T}{2}}(D-W)D^{-\frac{1}{2}}D^{\frac{1}{2}}1 = \lambda D^{\frac{1}{2}}1$
$\Rightarrow D^{-\frac{T}{2}}(D-W)1 = \lambda D^{\frac{1}{2}}1$
$\Rightarrow 0 = \lambda D^{\frac{1}{2}}1$, since $(D-W)1 = 0$
$\Rightarrow \lambda = 0$

so eigenvector $z = D^{\frac{1}{2}}1$ corresponds to eigenvalue $\lambda = 0$. From property 4, M has only non-negative eigenvalues, so 0 is in fact the smallest eigenvalue. So,

$$z_1 = D^{\frac{1}{2}}1 \tag{3.32}$$

Therefore,

$$\lambda_2 = \min_{u^T D^{\frac{1}{2}} 1 = 0, u \in V, u \neq 0} \frac{u^T M u}{u^T u} \tag{3.33}$$

Now, comparing (3.28) and (3.33), if we *relax* the condition $u_i \in \{\sqrt{d_i}, -b\sqrt{d_i}\}$ in (3.28) to $u \in V$, λ_2 is the solution to (3.28). And, since solving (3.28) is equivalent to solving (3.18), λ_2 is the solution to (3.18), provided we *relax* the condition as stated above.

3.1.4 Overall algorithm

We can now use the result obtained in the previous subsection and write the overall image segmentation algorithm. Given an Image I, get the Graph representation $G(V, E)$ of I. Now, `Ncut(G)` is as follows:

Ncut(G)

1. Construct the eigenvalue problem $D^{-\frac{1}{2}}(D - W)D^{-\frac{1}{2}}y = \lambda y$
2. Solve the eigenvalue problem for second smallest eigenvalue. Use corresponding eigenvector to bipartition the graph G into G_1, G_2.
3. If (based on some prior criteria) current partition need to be further re-partitioned then recursively call Ncut(G_1), Ncut(G_2).

Figure 3.1 and 3.2 shows recursive segmentation of the left most image. In Figure 3.1 of *Arc de triumph*, the left most image has been segmented into sky and *Arc de triumph*. Further recursive segmentation of *Arc de triumph* causes the sky visible below the *Arc* to be separated. Again ground on the both sides of the *Arc* have been segmented.

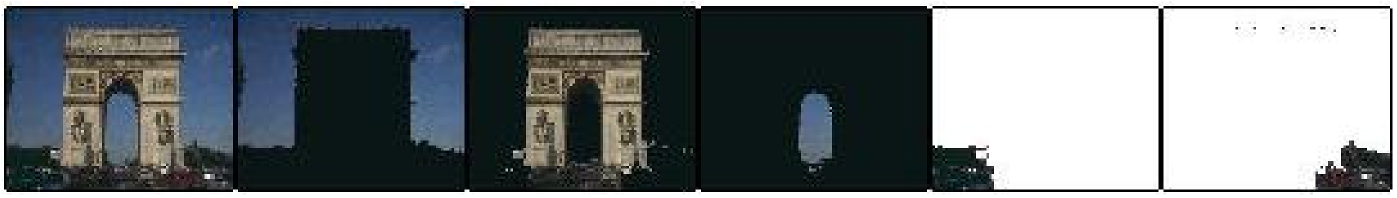

Figure 3.1: Segmentation of *Arc de triumph* using Normalized Cuts recursively [5]

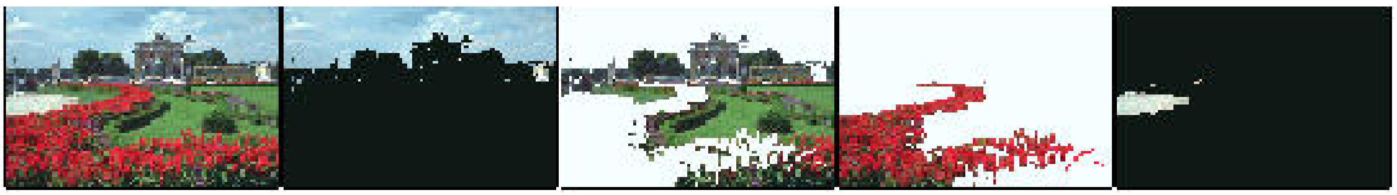

Figure 3.2: Segmentation of a color scene using Normalized Cuts recursively [5]

3.1.5 Discussion

Substitution described by (3.26) means :

$$\begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{bmatrix} = \begin{bmatrix} \sqrt{d_1} & 0 & \dots & 0 \\ 0 & \sqrt{d_2} & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & \sqrt{d_n} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} y_1\sqrt{d_1} \\ y_2\sqrt{d_2} \\ \vdots \\ y_n\sqrt{d_n} \end{bmatrix}$$

$$\Rightarrow y_i = \frac{u_i}{\sqrt{d_i}} \tag{3.34}$$

From (3.34), for y_i to be finite, $d_i > 0$, or, in terms of graph edges, vertex v_i should have at least one non-zero weight incident edge.

From the last section we find that λ_2 would have been the required value of (3.18), if y_i's had been relaxed to take any real value, instead of just 1 or $-b$. Now we check what happens if, fortunately, after solving for y it actually comes out to be $y_i \in \{1, -b\}$:

$$y_i = \frac{(1+x_i)}{2} - b\frac{(1-x_i)}{2} \tag{3.35}$$

1. $x_i = 1 \Rightarrow y_i = 1$
2. $x_i = -1 \Rightarrow y_i = -b$

So, we get a partition from vector x where for $x_i = 1$ we put the corresponding node in A, and for $x_i = -1$ we put it in B. So, this suggests one of the possible ways to interpret vector x. If we get $x_i \notin \{1, -1\}$, then in that case we can just look at its sign and $x_i > 0 \Rightarrow x_i \in A$ and $x_i > 0 \Rightarrow x_i \in A$.

3.1.6 Complexity analysis

Time complexity of this image segmentation algorithm is same as the time complexity of solving standard eigenvalue problem, which is cubic. But matrix M has following properties:

1. Since W is sparse, for locally connected graph, M is also sparse. This can be shown by following argument. We will show that if $W_{ij} = 0$ then $M_{ij} = 0$. We have

$$((D-W)D^{-\frac{1}{2}})_{ij} = \sum_k (D-W)_{ik}(D^{-\frac{1}{2}})_{kj} = (D-W)_{ij}(D^{-\frac{1}{2}})_{jj} \tag{3.36}$$

and

$$W_{ij} = 0 \Rightarrow (D-W)_{ij} = 0 \tag{3.37}$$

so,

$$\begin{aligned} &(D^{-\frac{1}{2}}(D-W)D^{-\frac{1}{2}})_{ij} \\ &\quad = \sum_k (D^{-\frac{1}{2}})_{ik}((D-W)D^{-\frac{1}{2}})_{kj} \end{aligned}$$

$$
\begin{aligned}
&= (D^{-\frac{1}{2}})_{ii}((D-W)D^{-\frac{1}{2}})_{ij} \\
&= (D^{-\frac{1}{2}})_{ii}(D-W)_{ij}(D^{-\frac{1}{2}})_{jj} \\
&= \left\{ \begin{array}{cc} 0, & W_{ij}=0 \\ (D^{-\frac{1}{2}})_{ii}(D-W)_{ij}(D^{-\frac{1}{2}})_{jj}, & W_{ij}\neq 0 \end{array} \right\}
\end{aligned}
$$

2. Only the top eigenvector is needed for partition.

Above properties are exploited in *Lanczos Method* for solving eigenvalue problems. The time complexity of *Lanczos Method* is $O(mn) + O(mM(n))$, where n is the dimension of matrix, m is the maximum number of matrix-vector multiplication required and $M(n)$ is the cost of a matrix-vector product computation of Mx. $M(n)$ is $O(n)$ because ith row of M has only d non-zero terms, where d is the degree of the v_i vertex in G. And in most practical cases d is constant (for a grid graph, in our case, $d = 4$). So each row-vector multiplication is $O(1)$ time, so Mx calculation is $O(n)$. So, this results in a complexity of $O(mn)$ for the overall algorithm. Further, empirically it has been found that $m < O(\sqrt{n})$ [5]. So, complexity of Image Segmentation algorithm is $O(n^{1.5})$.

3.2 Segmentation using local intensity variation

In this approach the notion of "Good" segmentation has been formalized [4]. We call this algorithm "K-algorithm". Based on the goodness criteria terms "Too-fine", i.e. over-segmentation and "Too-coarse", i.e. under-segmentation have been formally defined. Further, a graph theoretic approach is proposed which is based on the concept of variation inside a partition and variation across partitions. The algorithm finds the partition $\{G_1, G_2, \ldots, G_k\}$ for G such that intensity variation inside any two neighboring partitions G_i, G_j is less than the variation across them. The partition produced by this algorithm is neither over-segmentation nor under-segmentation.

3.2.1 Graph representation of image

The algorithm first constructs a graph $G(V, E)$ from a given image where V is a set of vertices v_i corresponding to each pixel in the image. An edge is defined between pixels p_i and p_j such that $\|p_i - p_j\| < d$ for some given distance d and a weight function w is defined as

$$w((v_i, v_j)) = \|I(v_i) - I(v_j)\|, (v_i, v_j) \in E \tag{3.38}$$

where $I(v_i)$ is defined as intensity of pixel corresponding to v_i. Here, we can note that $w((v_i, v_j))$ is direct measure of dissimilarity: higher the weight, higher is the dissimilarity, which is exactly opposite to the measure used in Ncut.

3.2.2 Basic definitions and criteria

Intuitively, this algorithm clusters the nodes which are joined by light weight edges (and hence combines "similar pixels" in the original image). For creating these clusters some criteria have been defined, and clusters are merged if they satisfy those criteria. Next, we define some terms and the criteria which create the basis for the algorithm.

If S is a segmentation of $G(V, E)$, C is some component in S, $M(C, E)$ is the Minimum spanning tree of C with edges in E, then the algorithm uses following notions of "variation" and "merging criteria":

- Internal variation of a component C

$$Int(C) = \max_{e \in MST(C,E)} w(e) \tag{3.39}$$

- Variation or difference across components

$$Dif(C_1, C_2) = \min_{v_i \in C_1, v_j \in C_2} w((v_i, v_j)) \tag{3.40}$$

- Criteria for merging two components C_1, C_2

$$Dif(C_1, C_2) \leq MInt(C_1, C_2) \tag{3.41}$$

  where,

$$MInt(C_1, C_2) = min(Int(C_1) + \frac{K}{\|C_1\|}, Int(C_2) + \frac{K}{\|C_2\|}) \tag{3.42}$$

  $\|C\|$ denotes the size of component C and K is some constant.

3.2.3 Overall algorithm and time complexity

Given an image with a corresponding graph $G = (V, E)$, the algorithm produces a segmentation S as follows :

K-Algorithm

1. Sort E into $\pi = (o_1, o_2, \ldots, o_k)$ by non-decreasing edge weight.
2. Start with $S^0 = \{v_1, v_2 \ldots v_n\}, n = \|V\|$.
3. Repeat Step 4 for $q = 1, \ldots, k, k = \|E\|$.
4. Construct S^q given S^{q-1} as follows. Let $o_q = (v_i, v_j)$, i.e., edge o_q connects vertices v_i and v_j. If v_i and v_j are in disjoint components of S^{q-1} and $w(o_q)$ is small compared to the internal difference of components containing v_i and v_j, then merge the two components, otherwise do nothing. That is if, $C_i^{q-1} \neq C_j^{q-1}$ and $w(o_q) \leq MInt(C_i^{q-1}, C_j^{q-1})$, then $S^q = S^{q-1} \cup \{o_q\}$ else $S^q = S^{q-1}$.

The running time of the algorithm is at max $O(n \log n)$.

Figure 3.3 presents an example image segmentation produced by K-Algorithm. The advantage of K-Algorithm is illustrated here in the sense that it is able to segment out regions with noise. Some improvements in the K-Algorithm have been made in [1]. The algorithm proposed in [1] is called P-Algorithm. This algorithm uses efficient *Prim Algorithm* for finding $Int(C)$. With these improvements P-Algorithm may run in time linear in the size of input image. P-Algorithm uses a notion of seed point and grows the region from that seed point. Seed points are picked from a priority queue. This procedure has the advantage that growth of the regions starts from different parts of the image. Several cases and results of better performance have been presented in [1].

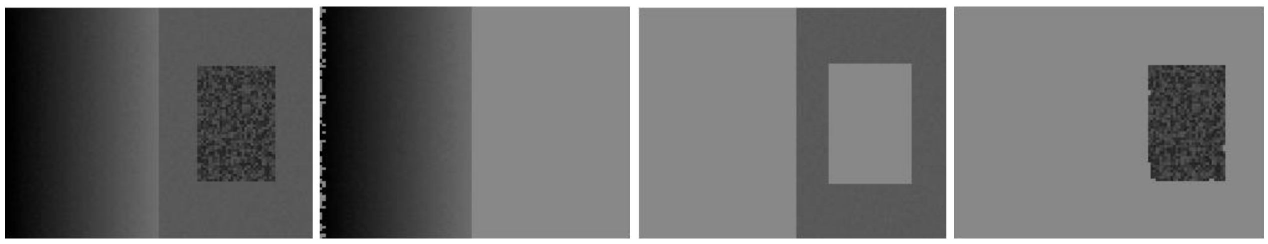
Figure 3.3: 3 regions produced by K-Algorithm [4]

3.3 Summary

In this chapter we have discussed Graph Theoretic image segmentation techniques. We discussed two algorithms which follow different approaches, and in some sense represent two classes of approaches. *Ncut* approach has an intention of partitioning image into two major regions and further sub-partitioning each region. But, *Local Variation* approach has an intention of merging small similar groups to eventually grow major regions of the image.

Chapter 4

Our Approach

In this project, we proposed a hierarchical image segmentation algorithm. This chapter discusses the motivation behind our approach and the algorithm.

4.1 Motivation

In the earlier chapters we have discussed two main philosophies behind image segmentation, viz. *hierarchical* and *local*. Each of them has its own advantages and disadvantages. The preference of one over the other depends on the final application. Hierarchical approach has a benefit that it gives an overall idea of the foreground and the background. Moreover, hierarchical approach gives the control on the level of segmentation which is quite desired in several interactive applications. But recursively using this approach for finding fine segmentation will be costly. Local approach starts from small clusters, and grows to larger ones. Its level of segmentation depends on some threshold decided in the beginning, so we can not stop the segmentation at some intermediate stage.

We think that combining these two approaches can lead to a faster and better segmentation. The idea is to use local approach for reducing the size of the problem. Ncut is a good measure of segmentation [5], so we use this as hierarchical approach. The idea of reducing the input size has been used in [3] through algebraic multi-grid solvers. We propose an algorithm which uses P-Algo for reducing the size of the input to Ncut. Let the input image be of size n. The modified P-Algo takes $O(n \log n)$ time and produces $\sqrt{n}$ clusters. These clusters can be assumed to be a coarse form of the original input image. We apply Ncut segmentation algorithm on these $\sqrt{n}$ clusters of pixels. Since running time of Ncut algorithm is typically $O(n^{1.5})$, for input size $\sqrt{n}$ time will be $n^{0.75}$.

4.2 Details of approach

We first describe the overall structure of the algorithm, then we will present it more concretely in pseudo code style. The algorithm has 3 major parts:

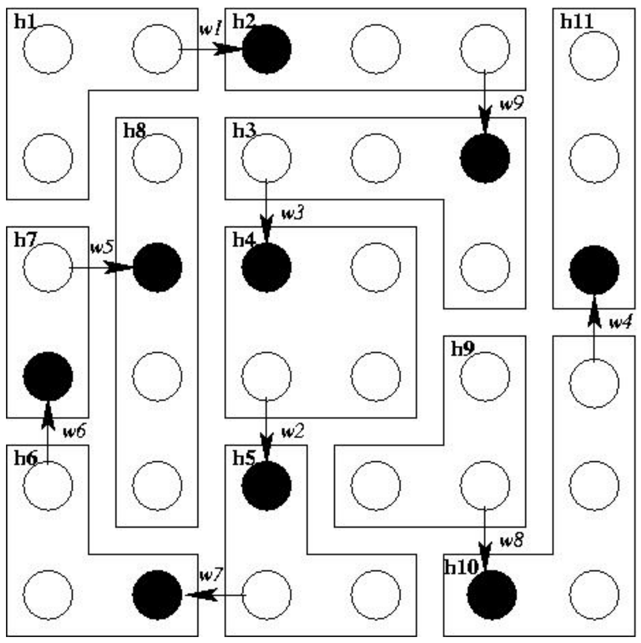

Figure 4.1: Illustration of Step 1. Clusters are shown as polygons and linking nodes are colored in black. Annotated arrows show the weighted links between two handles.

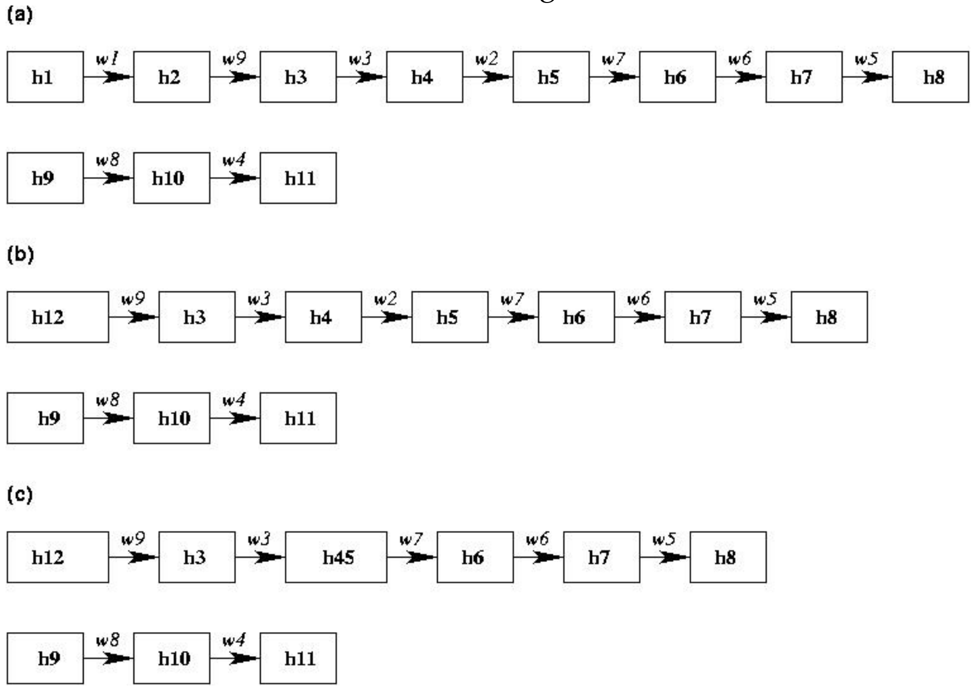
(a)
h1 w1 h2 w9 h3 w3 h4 w2 h5 w7 h6 w6 h7 w5 h8
h9 w8 h10 w4 h11
(b)
h12 w9 h3 w3 h4 w2 h5 w7 h6 w6 h7 w5 h8
h9 w8 h10 w4 h11
(c)
h12 w9 h3 w3 h45 w7 h6 w6 h7 w5 h8
h9 w8 h10 w4 h11

Figure 4.2: Illustration of Step 2 (continued from Figure 4.1). Handles are arranged as linked lists and edges are annotated according to their weights $w1 > w2 > w3 \ldots$. (a) Initial list after step 1, (b) List after merging $h1$ and $h2$ which are joined by maximum weight edge $w1$, (c) List after merging $h4$ and $h5$ which are joined by $w2$.

1. A weighted undirected graph $G(V, E)$ is constructed from the input image I of size n. V is the set of all pixels. 4-neighbors are connected and E is the set of all the edges.

Weight of the edge between vertices $u,\ v$ is $w(u,v) = |I_u - I_v|$. A modified version of P-Algo is applied on G. At the end of this step a weighted directed graph $G'(V', E')$ is obtained. V' is the set of data structure *handle*(h_i), which contains vertices of V (see **Appendix B**). These handles represent a cluster of pixels and the directed edges join two clusters that can be merged further. Weight of the edge from h_i to h_j is measured as $\frac{\text{Int}(h_i)}{\text{edge wt}}$ and it is an increasing function of the similarity between the two clusters it's joining. This step takes $O(n \log n)$ time. An example has been shown in Figure 4.1.

2. The edges of E' are sorted in decreasing order of weight. Starting from the maximum weight edge two adjacent clusters are merged sequentially. This is repeated as long as connected components are there in the graph and the number of clusters is greater than n. At the end of this step we get a set of clusters. This step takes $O(m \log m)$ time, where m is the number of edges and $m < n$. Some steps of this part are shown in Figure 4.2.

3. We apply Ncut algorithm on this cluster set of size $\sqrt{n}$.

4.3 Algorithm

In this section we present the pseudocode of the modified P-Algo. The data-structures involved are described in **Appendix B**

```
procedure overall()
initQ_2()
initQ_1()
i ← 0
h ← new(handle)
while Q_2 ≠ ϕ do
    s ← findMin(Q_2)
    Q_1.dec(s, 0)
    grow(h, i)
    i ← i+1
end while
```

```
procedure initQ_2
for all v ∈ V do
    x ← minAdjacent(v)
    Q_2.insert(v,x)
end for
```

```
procedure grow(h, i)
done ← false
while not done do
    u ← findMin(Q_1)
    if causeMerge(h, u) then
        if isLinkingNode(u) then
            h′ = handle pointing to u
            Link handles h, h′
            linkWeight ← $\frac{\text{Int(h′)}}{\text{edgeWeight}}$
        end if
        h.add(u)
        updateAdjacent(u)
        Q_1.remove(u)
        Q_2.remove(u)
    else
        u.setLinkingNode (true)
        u.setHandlePointingIt (h)
        done ← true
    end if
end while
```

```
procedure initQ$_1$
for all v $\in$ V do
    v.key $\leftarrow \infty$
    Q$_1$.insert(v, v.key)
end for
```

```
procedure causeMerge(h, u)
if u.key < (h.internalVariation + $\tau$)
then
    return true
else
    return false
end if
```

```
procedure updateAdjacent(u)
for all v $\in$ adjacent(u) do
    if (v $\in$ Q$_1$) $\wedge$ (v.key<w(u,v))
    then
        v.key $\leftarrow$ w(u,v)
        Q$_1$.decreaseKey(v,v.key)
    end if
end for
```

Pseudocode: Step 1

The above pseudocode is for *step 1.* Ncut part of the algorithm is the same as one described in previous sections.

```
procedure mergeClusters()
sort(edgeList) {sorted list of edges $e_i$'s looks like $(C_x e_0 C_{x'})(C_y e_1 C_{y'})\ldots$,
where $e_i \geq e_{i+1}$}
i $\leftarrow$ 1
count $\leftarrow$ number of clusters
while (count $\geq$ n) $\wedge$ (i $\leq$ no of edges) do
    merge($C_x,\ C_{x'}$) {$e_i$ joins $C_x,\ C_{x'}$}
    i $\leftarrow$ i+1
    count $\leftarrow$ count-1
end while
```

Pseudocode: Step 2

4.4 Discussion

The algorithm assigns weight of the link to be $\frac{\text{Int(h')}}{\text{edgeWt}}$ because this is an increasing measure of similarity between the handles joined by it; higher the ratio, higher the similarity. Int(h′) is a measure of internal variation of the handle h′ and is calculated during the growth of h′, so no extra calculation is needed for determining the link weight.

Overall complexity of the algorithm proposed can be found from the complexity of the 3 steps involved. Step-1 takes $O(n \log n)$, step-2 takes $O(n \log n)$ in the worst case, and the final step of normalized cut takes $O(m^{1.5})$, where m is the size of the input to the step-3. By the conditions put in the algorithm m will be approximately $O(n^{0.5})$, so this causes step-3 to be $O(n^{0.75})$. So, overall complexity becomes $O(n \log n) + O(n^{0.75})$, i.e. $O(n \log n)$. Thus we have achieved a better time complexity in most of the cases, and even if this can not do any better in some cases, it will not be worse.

Chapter 5

Implementation Details

The current implementation is being done in following steps.

- Modified P-Algorithm implementation (step 1)
- Cluster merging (step 2)
- Normalized cuts based segmentation (step 3)

The details of these implementations are discussed in following sections.

5.1 Modified P-Algorithm implementation

This is implemented as an independent class which forms a linked list of clusters. The current version of implementation uses "iMagick++" package for reading and writing images. This enables the interface to accept input images in varied formats. For time considerations, a faster but much restrained version is being implemented which takes some particular input image format.

5.2 Cluster merging

This step uses recursive quick sort for sorting edges.

5.3 Normalized cuts

The normalized cuts method uses Lanczos algorithm for finding solution to eigenvalue problem. This is being done by using LASO package [9] for matrix calculations.

Chapter 6

Summary

In this report, we have done a survey of image segmentation and motion segmentation in videos. Two approaches for the same have been studied in details. We have discussed the segmentation algorithms based on local variation and normalized cuts. The segmentation based on normalized cuts is a hierarchical algorithm but has high time complexity. We proposed an algorithm which can perform the segmentation in time less than that taken by normalized cut. The algorithm uses the segmentation based on local variation for creating small clusters and coarsens the input image, hence reduces the input size for normalized cuts algorithm.

This approach for coarsening the image can be extended to other feature spaces. Though we are implementing this for images with pixel's RGB value as the feature, for motion segmentation motion profile could be the feature space.

Appendix A

Key Proofs

Results used in Chapter 3 are proved here.

Proof 1: *If $D_{n\times n}$ is a diagonal matrix, $A_{n\times n}$ is a symmetric matrix, then DAD is symmetric*:

$(DAD)_{ij}$
$= (D(AD))_{ij}$
$= \sum_k D_{ik}(\sum_r A_{kr}D_{rj})$
$= \sum_k \sum_r (D_{ik}A_{kr}D_{rj})$
$= \sum_{k,k\neq i} \sum_{r,r\neq j}(D_{ik}A_{kr}A_{rj}) + D_{ii}A_{ij}D_{jj}$
$= D_{ii}A_{ij}D_{jj}$
$= D_{jj}A_{ji}D_{ii}$
$= (DAD)_{ji}$.

Proof 2: *If $A \in R^{n\times n}$ and symmetric matrix then A has real eigenvalues.*
$Ax = \lambda x \Rightarrow x^*Ax = \lambda x^*x$ Here $x^*x \in \mathcal{R}$, so if we prove that $x^*Ax \in \mathcal{R}$ then we are done. For this we will prove that $\overline{x^*Ax} = x^*Ax$.

x^*Ax
$= \sum_i \overline{x_i}(Ax)_{i1}$
$= \sum_i \overline{x_i}(\sum_k a_{ik}x_k)$
$= \sum_i \sum_k (a_{ik}x_k\overline{x_i})$, since A is symmetric, so $a_{ij} = a_{ji}$, and using this we get
$= (\sum_i a_{ii}x_i\overline{x_i}) + \sum_{i=1}^n \sum_{j=i+1}^n a_{ij}(x_i\overline{x_j} + \overline{x_i}x_j)$
$\overline{x^*Ax}$
$= \overline{(\sum_i a_{ii}x_i\overline{x_i}) + \sum_{i=1}^n \sum_{j=i+1}^n a_{ij}(x_i\overline{x_j} + \overline{x_i}x_j)}$
$= (\sum_i a_{ii}\overline{x_i}x_i) + \sum_{i=1}^n \sum_{j=i+1}^n a_{ij}(\overline{x_i}x_j + x_i\overline{x_j})$
$\Rightarrow \overline{x^*Ax} = x^*Ax$

Proof 3: *If $A_{n\times n}$ is a real symmetric matrix, then it has orthogonal set of eigenvectors.*
Consider any two eigenvalues λ_i, λ_j and corresponding eigenvectors x_i, x_j.

$x_j^T Ax_i = x_j^T\lambda_i x_i = \lambda_i(x_j^T x_i) = \lambda_i < x_i, x_j >$
$(x_j^T Ax_i)^T = x_i^T Ax_j = x_i^T\lambda_j x_j = \lambda_j(x_i^T x_j) = \lambda_j < x_i, x_j >$
but, $(x_j^T Ax_i)^T = (\lambda_i < x_i, x_j >)^T = \lambda_i < x_i, x_j >= x_j^T Ax_i$
$\Rightarrow \lambda_i < x_i, x_j >= \lambda_j < x_i, x_j >$
$\Rightarrow< x_i, x_j > (\lambda_i - \lambda_j) = 0$
$\Rightarrow< x_i, x_j >= 0$ for $i \neq j$

Proof 4: *M is Symmetric Positive Definite(SPD), i.e. $x^TMx \geq 0 \forall x$.*

$$x^TD^{-\frac{1}{2}}(D-W)D^{-\frac{1}{2}}x^T =$$

$$\begin{bmatrix} \frac{x_1}{\sqrt{d_1}} & \frac{x_2}{\sqrt{d_2}} & \dots & \frac{x_n}{\sqrt{d_n}} \end{bmatrix} \begin{bmatrix} d_1 - w_{11} & -w_{12} & \dots & w_{1n} \\ -w_{21} & d_2 - w_{22} & \dots & w_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ -w_{n1} & -w_{n2} & \dots & d_n - w_{nn} \end{bmatrix} \begin{bmatrix} \frac{x_1}{\sqrt{d_1}} \\ \frac{x_2}{\sqrt{d_2}} \\ \vdots \\ \frac{x_n}{\sqrt{d_n}} \end{bmatrix}$$

$= \{\frac{x_1}{\sqrt{d_1}}(d_1 - w_{11}) - \frac{x_2w_{21}}{\sqrt{d_2}} - \dots - \frac{x_nw_{n1}}{\sqrt{d_n}}\}\frac{x_1}{\sqrt{d_1}}$
$+\{-\frac{x_1w_{12}}{\sqrt{d_1}} + \frac{x_2}{\sqrt{d_2}}(d_2 - w_{22}) - \dots - \frac{x_nw_{n2}}{\sqrt{d_n}}\}\frac{x_2}{\sqrt{d_2}} + \dots$
$= \frac{x_1^2}{d_1}(d_1 - w_{11}) + \frac{x_2^2}{d_2}(d_2 - w_{22}) + \dots - \frac{2x_1x_2}{\sqrt{d_1d_2}} - \frac{2x_1x_3}{\sqrt{d_1d_3}} - \dots$
$= \frac{x_1^2}{d_1}(w_{12} + w_{13} + \dots + w_{1n}) + \frac{x_2^2}{d_2}(w_{21} + w_{23} + \dots + w_{2n}) + \dots$
$= w_{12}\{(\frac{x_1}{\sqrt{d_1}})^2 + (\frac{x_2}{\sqrt{d_2}})^2 - \frac{2x_1x_2}{\sqrt{d_1d_2}}\} + \dots$
$= \sum_{i=1}^n \sum_{j=i+1}^n w_{ij}(\frac{x_i}{\sqrt{d_i}} - \frac{x_j}{\sqrt{d_j}})^2$
$\geq 0 \forall x$
$\Rightarrow M$ is SPD.

Proof 5: *SPD matrix $A_{n\times n}$ has non-negative eigenvalues.* Consider

$Ax = \lambda x$
$\Rightarrow x^TAx = x^T\lambda x = \lambda x^Tx$
$\Rightarrow \lambda\|x\|^2 \geq 0$, because $x^TAx \geq 0$
$\Rightarrow \lambda \geq 0$

Appendix B

Key Data Structures and Implementation Details

Some important classes and their main members are explained in this chapter.

```
struc vertexNwt{
    BasicVertex *vertex;
    float edgeWt;
};
```

Structure used to represent the adjacent vertices of a node. It also stores the weight of the edge joining the two vertices.

```
class Handle{
    Handle *nextHandle;
    float linkWeight;
    short noOfNodesBelow;
    float internalVariation;
    LinkedList<BasicVertex> *verticesList;
    float getTau();
    bool causeMerge(BasicVertex *);
};
```

Class used to represent the handles obtained after running P-Algo on the initial input image. “nextHandle” is the pointer to the next handle determined by the algorithm’s step 1. Several “BasicVertex” objects cluster to make one “Handle” object, and all the constituent vertices are stored in “verticesList”. The function “causeMerge(BasicVertex *v)” returns if “BasicVertex *v” can be merged in this handle.

```
class BasicVertex{
    MyColor *pixelValue;
    bool linkingNode;
    int row, column;
    Handle *handlePointingIt, *handlePointedByIt;
    LinkedList<vertexNwt> *adjacentVertices;
    int getQ1Index();
    int getQ2Index();
};
```

Class used to represent the vertices of initial graph obtained directly from the image. "linkingNode" tells if this node is a link between two handles or not. "handlePointedByIt" points to the "Handle" object under which it comes, and "handlePointingIt" points to the "Handle' for which this is a linking node. "getQ1Index()" and "getQ2Index()" returns the index of the this object in priority queues Q_1 and Q_2 respectively.

```
class PriorityQ{
    bool indexSetting;
};
```

A modified class for priority queue. If "indexSetting" is set true in the constructor, index of the element is stored with the object. This facilitates searching of the object in the queue (given the pointer to the object) in $O(1)$ time. Hence an element can be removed from the queue in $O(\log n)$ time.

```
class ImageHandler{
    char *inFile;
    BasicVertex **getImageToGraph();
    bool writeGraphToImage(LinkedList<Handle> *graph, int ht, int wt, char* outFile);
};
```

Class used to create a graph from image file named "inFile" and writing the output to "outFile". "writeGraphToImage" takes a linked list of "Handle" objects and produces an image with all pixels under one "Handle" object colored in one color.

```
class PAlgo{
    PriorityQ<BasicVertex> *Q1, *Q2;
    bool initQ1();
    bool initQ2();
    void updateAdjacent(BasicVertex *u);
    void grow(Handle *hndl);
    short overall();
    void segment();
};
```

Class used to segment image according to P-Algorithm. "Q1" and "Q2" are priority queues as described in algorithm. The member functions work as described in the algorithm. Function overall() actually performs the segmentation on the graph and returns the end status. "segment()" is the actual public interface for using this class, and it hides all the internal details.

Bibliography

[1] S. Chandran and K. K. Madheshia. A fast segmentation algorithm revisited. *Proceedings of ICVGIP*, December 2002.

[2] F. R. K. Chung. Lectures on spectral graph theory.

[3] A. Brandt E. Sharon and R. Basri. Fast multiscale image segmentation. *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2000.

[4] P. Feleznszwalb and D. Huttenlocher. Image segmentation using local variation. *Proceedings of IEEE conf. Computer Vision and Pattern Recognition*, pages 98–104, 1998.

[5] J. Malik and J. Shi. Normalized cuts and image segmentation. *Proceedings of IEEE Conf. Computer Vision and Pattern Recognition*, 1997.

[6] J. Malik and J. Shi. Motion segmentation and tracking using normalized cuts. *International Conf. on Computer Vision*, 1998.

[7] S. Y. Ma S. Y. Chien and L. G. Chen. Efficient moving object recognition algorithm using background registration technique. *IEEE Transactions on Circuits and Systems for Video Technology*, 12, July 2000.

[8] P. Salembier and F. Marques. Region-based representations of image and video: Segmentation tools for multimedia services. *IEEE Transactions on Circuits and Systems for Video Technology*, 9(8), December 1999.

[9] D. S. Scott and B. N. Parlett. Laso. http://www.netlib.org/laso/.

[10] O. Veksler. Image segmentation by nested cuts. *IEEE Computer Vision and Pattern Recognition*, 2000.

[11] Z. Wu and R. Leahy. An optimal graph theoretic algorithm to data clustering: Theory and its application to image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(11), November 1993.