# A Comparison of Image Segmentation Methods

Anna Blasiak

Advisor: Prof. Daniel Scharstein

Senior Thesis in Computer Scinece

Middlebury College

May 2007

# Abstract

The field of computer vision is concerned with extracting information from images. The task of image segmentation is a first step in many computer vision methods and serves to simplify the problem by grouping the pixels in the image in logical ways. Image segmentation is hard to clearly define because there are many levels of detail in an image and therefore many possible ways of meaningfully grouping pixels. Additionally, after choosing a definition for an optimal segmentation, there are many computational difficulties in finding such a segmentation. This thesis presents three different ways to approach image segmentation, explain an efficient implementation for each approach, and show sample segmentations results.

# Acknowledgements

I would like to thank Daniel Scharstein for advising me on my thesis despite that he was on leave for the semester. I truly appreciate the many hours he spent helping me understand various components of my thesis as well as the time he spent reading and making comments on various drafts. I would also particularly like to acknowledge Daniel's prompt replies to my many e-mails, especially when they were really late at night. I would also like to thank Dave Guertin and Daniel again for installing an image processing package in Matlab so that I could test the Normalized Cuts method. Additionally, I want to acknowledge Frank Swenton for his help in understanding and making sense of much of the incomprehensible math in the normalized cuts and mean shift papers. Finally, I am grateful to my friends and family who patiently listened to me grumble about my theses and supported me during all my four years at Middlebury.

# Contents

# Chapter 1

# Introduction

Image segmentation is a fundamental step in many areas of computer vision including stereo vision and object recognition. It provides additional information about the contents of an image by identifying edges and regions of similar color and texture, while simplifying the image from thousands of pixels to less than a few hundred segments. Additionally, image segmentation has applications separate from computer vision; it is frequently used to aid in isolating or removing specific portions of an image.

Although a seemingly simple first step in high level computer vision tasks, there are many challenges to an ideal image segmentation. There are many possible correct segmentations, and the one we are looking for depends on the application. For example, in object recognition, the goal is often to find segments that describe objects. However, the idea of "objects" is itself vague. Most of the time objects can be subdivided into smaller objects. For example, a tree can further divided into a trunk and its leaves. To clearly define an ideal segmentation we must decide on the level of detail that we want. Another application of segmentation is as a preliminary step to stereo vision. Here, we want to group together pixels that are the same distance

from the camera, and thus an ideal segmentation has relatively small segments where many put together define an object.

Even if we can formulate a definition for an ideal segmentation, there are many difficulties to overcome. The information we know directly about an image is the intensity/color and location of each pixel, and therefore these two variables are the central source of data for any segmentation. However, color and intensity can vary significantly over a single object. Also, shadows, reflections, and textures add sharp color contrasts to the same surface. Additionally, the image can be distorted or blurry. Edges that look clearly defined from far away are often blurry and hard to distinguish at the pixel level. Similarly, an area that looks like a solid color at a distance may be seen as mix of many colors at the pixel level and may be interpreted as a number of segments.

In the chapters that follow we will show three distinct methods of segmentation: graph cuts, normalized cuts, and mean shift. Currently, they are considered the three most popular methods for segmentation. Each has a different approach in defining what characterizes a good segmentation and uses a different technique to find the optimal segmentation. We will describe the three methods and compare them in terms of their strengths and weaknesses. However, it is almost impossible to make a categorical statement about one being better than the other because each has different goals, and may be useful in different applications. The main portion of this thesis explains the *Graph Cuts* segmentation method in Chapter 2. In Chapter 3 we present the *Normalized Cuts* segmentation method, and in Chapter 4 the *Mean Shift* segmentation method.

# Chapter 2

# Graph Cuts

The graph cuts algorithm for image segmentation has generated much interest since its introduction in 1998 [5]. It has spawned numerous papers that show its equivalence to related methods and reveal both its potential and its limitations. The algorithm is quite versatile and aside from image segmentation, it can be easily adapted to solve other problems in computer vision, and related problems in other areas of computer science. This chapter includes four sections: Section 2.1 presents the definition of an optimal segmentation used in the graph cuts algorithm. Section 2.2 provides a probabilistic explanation of the optimal segmentation definition. Section 2.3 explains the graph cut algorithm for binary segmentation and proves that the technique finds the optimal segmentation according to our definition. Section 2.4 shows how to find an approximate solution using graph cuts for a segmentation with more than two labels. And finally, Section 2.5 shows how to find the the minimum cut of a graph which is related because graph cuts reduces the work of image segmentation to finding the minimum cut of a graph.

## 2.1 Energy Model

In order to find the best segmentation, we first need to decide what it means to be "the best." One way to establish such a characterization is to create an *energy function* over the image, and define "the best" segmentation as the one that minimizes the energy. The energy model most commonly used in conjunction with graph cuts considers two factors: A *regional* term and a *boundary* term. The regional term, denoted $R_p(A_p)$, assigns a cost for pixel $p$ to belong to segment $A_p$, where $A$ is a segmentation, and $A_p$ is the segment that $p$ belongs to in $A$. Often this cost is determined by the intensity of $p$ relative to the distribution of intensities of the pixels known to be in segment $A_p$. The boundary term, denoted $B_{p,q}(A_p, A_q)$, assigns a cost for each pair of pixels $p$ and $q$ that are *neighbors* (denoted $\{p, q\} \in \mathcal{N}$). The *neighborhood* of a pixel $p$ is a set of pixels close in distance to $p$ and can be chosen as one likes, as long as we have the condition that if $p$ is a neighbor of $q$ then $q$ is a also a neighbor of $p$. Because we must have symmetry, we can say that if $q$ is in the neighborhood of $p$, then $p$ and $q$ are neighbors. A few examples of possible neighborhoods are shown in Figure 2.1. The most commonly used is a 4-neighborhood as depicted in Figure 2.1 (a). The boundary term accounts for the pairwise interactions of pixels and models how likely it is that two neighboring pixels will take on the same or different values. To model this, the boundary term assumes that in general the world is piecewise smooth. In other words, pixels near each other are often part of the same object, unless the two pixels have very different intensity, in which case it is likely that there is a division from one object to another. Therefore, the boundary cost is higher if the two pixels have different assignments and lower (generally 0) if they have the same assignment. Additionally, if the two pixels have different assignments, the cost is higher if the pixels are more similar. Notice that we

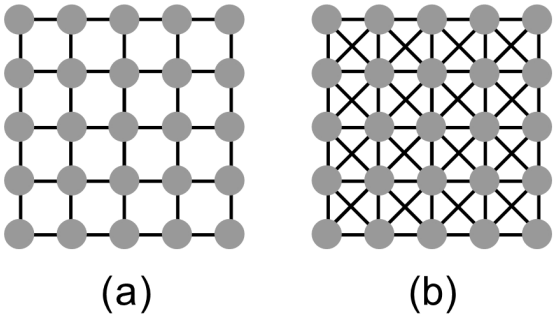



Figure 2.1: Example neighborhoods of a pixel: **(a)** A 4-neighborhood system; **(b)** An 8-neighborhood system. This Figure is based on image from [3].

only consider the relation between neighboring pixels and we do not care about the relative assignment of pixels who are not neighbors. Therefore, a good neighborhood system is one in which two pixels who are not neighbors are far enough away from each other that their relative assignment is insignificant. However, often we find that a small neighborhood gives sufficiently good results, and that a large neighborhood is too computationally intensive.

Finally, to bring together the regional and boundary terms, we can write the formal energy equation for the entire image as follows:

$$E(A) = \lambda R(A) + B(A), \tag{2.1}$$

where

$$R(A) = \sum_{p \in \mathcal{P}} R_p(A_p), \tag{2.2}$$

$$B(A) = \sum_{\{p,q\} \in \mathcal{N}} B_{p,q}(A_p, A_q), \tag{2.3}$$

and $\lambda$ is a constant that determines the relative importance of $R(A)$ and $B(A)$.

## 2.2 MAP Derivation of the Energy Model

Aside from being intuitive, a justification for the use of this energy model is that it can be used to optimize a *maximum a posteriori (MAP)* estimate of a *Markov Random Field (MRF)*. The MAP estimate determines the segmentation $A^*$ that has the highest probability of being correct given the data $D$ contained in the image, or formally,

$$A^* = \arg\max_A P(A|D). \tag{2.4}$$

In order to find the MAP estimate we need to make some assumptions about our data, otherwise we cannot simplify the estimate into anything useful.

The first assumption is that the label $L_p$ that indicates the segment to which each pixel $p$ belongs, is a random variable, and together the labels $L = (L_1, L_2, ..., L_{|\mathcal{P}|})$ make a Markov random field. An MRF is a group of random variables, each of which have a neighborhood, $\mathcal{N}$, that is a subset of the group of random variables. Just as in our previous definition of a neighborhood for the boundary term, this neighborhood must be symmetric. The essential property of an MRF is the *Markov assumption*: the probability of a random variable taking on a value given all other random variables in the field is equal to the probability given only its neighbors [13]. In image related problems, each pixel $p$ has an associated random variable $L_p$ that can take on some label $A_p$ from the set $\mathcal{L} = (l_1, ..., l_k)$, where every $l \in \mathcal{L}$ denotes a segment. The neighbors of each pixel $p$ are a set of pixels close in distance to $p$ in the image. We can write the Markov assumption as $P(L_p = A_p|A) = P(L_p = A_p|A_{\mathcal{N}_p})$ where $A = \{A_1, A_2, ...., A_{|\mathcal{P}|}\}$ is the labeling of the entire image, and $A_{\mathcal{N}_p}$ is the labeling of the pixels in $\mathcal{N}_p$.

To completely define an MRF we must also specify a clique size. If we view the entire neighborhood system as a graph where any two pixels that are neighbors are

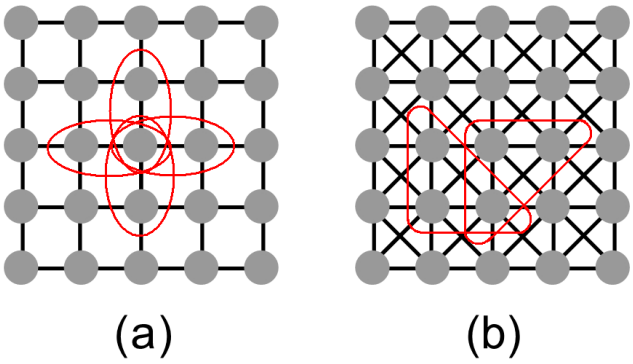



Figure 2.2: Sample neighborhood and clique configurations. **(a)** Cliques of size two in the 4-neighborhood of the pixel outlined in black. **(b)** Two of the cliques of size three in the 8-neighborhood of the pixel outlined in black.

connected, cliques are groups of pixels that are fully connected. The clique size of an MRF dictates the independence assumptions between the neighboring pixels of a pixel $p$ when computing the probability of a certain label at $p$. If our clique size is $c$ then we consider $P(L_p = A_p|A_{\mathcal{N}_p})$ to be a product of $P(L_p = A_p|C)$ for all cliques $C \in \mathcal{N}_p \cup p$ that have size $c$. The possible choices of clique size depends on the choice of neighborhood system. If we choose a 4-neighborhood system then there are no cliques of size bigger than two, and we must have $c = 2$. If we choose an 8-neighborhood system we can choose $c$ to be $2, 3$, or 4. Figure 2.2 illustrates a clique size of two in a 4-neighborhood system and a clique size of three in a 8-neighborhood system. For ease of computation, typically the clique size is chosen to be $c = 2$. If the clique size is two we have $P(L_p = A_p|A_{\mathcal{N}_p}) = \prod_{q \in \mathcal{N}_p} P(L_p = A_p|A_q)$. That is, given the label of one neighboring pixel, we assume the label of the pixel to be independent of the labels of all the other neighboring pixels.

The advantage of structuring our problem as an MRF is that we can use the theory associated with MRF's to assist us in the probabilistic derivation of the energy function. The main result in the area of MRF's is the Hammersly-Clifford Theorem. In the case that the clique size is two, it states that

$$P(L = A) \propto \prod_{\{p,q\} \in \mathcal{N}} V_{p,q}(A_p, A_q), \tag{2.5}$$

where $V_{p,q}(A_p, A_q)$ is a prior probability that describes how likely a given labeling is within the clique [5].

The second assumption that we make concerns the probability of the observed characteristics of a pixel $p$. Let $D = \{D_1, D_2, ..., D_{|\mathcal{P}|}\}$ be the data of the image where $D_p$ is, for example, the color, intensity, or texture at a pixel $p$. We make the assumption that the probability that pixel $p$ has data equal to $d_p$ is independent of all else given that we know the label at pixel $p$. Formally, this means

$$P(D|A) = \prod_p P(D_p = d_p|A_p). \tag{2.6}$$

This assumes that each segment is characterized by a distribution of data that is independently identically distributed. For example, this would be the case if all the data values in a segment vary according to Gaussian noise.

We are now ready to derive the energy function from a MAP estimate of our data model. Below we show the derivation of the MAP estimate $A^*$, where we take $\mathcal{H}$ to be all possible segmentations $A$:

$$
\begin{aligned}
A^* &= \arg\max_{A\in\mathcal{H}} \; P(A|D_{\mathcal{P}}) \\
&= \arg\max_{A\in\mathcal{H}} \; \frac{P(D_{\mathcal{P}}|A)P(A)}{P(D_{\mathcal{P}})} \quad \text{by Bayes' Law} \\
&= \arg\max_{A\in\mathcal{H}} \; P(D_{\mathcal{P}}|A)P(A) \quad \text{since } P(D_{\mathcal{P}}) \text{ is a constant independent of } A \\
&= \arg\max_{A\in\mathcal{H}} \; P(D_{\mathcal{P}}|A) \prod_{\{p,q\}\in\mathcal{N}} V_{p,q}(A_p, A_q) \quad \text{by the Hammersly-Clifford Theorem} \\
&= \arg\max_{A\in\mathcal{H}} \; \prod_{p\in\mathcal{P}} P(d_p|A_p) \prod_{\{p,q\}\in\mathcal{N}} V_{p,q}(A_p, A_q) \quad \text{by the cond. indep. of } d_p \text{ given } A_p \\
&= \arg\min_{A\in\mathcal{H}} \; \sum_{p\in\mathcal{P}} -\ln P(d_p|A_p) + \sum_{\{p,q\}\in\mathcal{N}} -\ln V_{p,q}(A_p, A_q).
\end{aligned}
\tag{2.7}
$$

We now have two terms: A term that is a sum over all the pixels that is a function of $P(d_p|A_p)$, which is our regional term. It gives us an estimate for how well the assignment at each pixel matches the data. The other term is a sum over all the pixels that is function of $V_{p,q}(A_p, A_q)$, which is our boundary term. This term gives probabilities of a specific labeling for every pair of pixels.

Thus, we arrived at the intuitive energy function through a probabilistic approach. We had to make many assumptions that may not always hold, however, doing this allows us to formalize exactly the assumptions that we make. This is helpful in analyzing method performance because we can more accurately assess what went right or wrong by determining whether the assumptions hold in our given image.

To make the concept of energy more concrete, one example of possible regional and boundary cost functions are the following:

$$
R_p = -\ln Pr(I_p|A_p) \tag{2.8}
$$

$$B_{p,q} = \begin{cases} e^{\left(\frac{1}{(X_p - X_q)^2}\right)\cdot\left(-\frac{(I_p - I_q)^2}{2\sigma^2}\right)} & \text{if } \{p,q\} \in \mathcal{N}, A_p \neq A_q \\ 0 & \text{otherwise} \end{cases} \tag{2.9}$$

where $I_p$ is the intensity of $p$ and $X_p$ is the loaction of pixel $p$ in the image.

The first equation is exactly as derived in the MAP MRF, where the data at pixel $p$ is taken to be the intensity $I_p$ of $p$. Typically this probability is computed from information that the user gives about the intensities of pixels in a segment. The second equation can be can be derived from our MAP MRF formula when we assume that the probability that two neighboring pixels have different labels is modeled by a Gaussian distribution over the intensity of the pixels.

## 2.3 Binary Image Segmentation

Now that we have established a formulation for what it means to be the best segmentation, i.e. the segmentation that minimizes Equation 2.1, we can present the algorithm that will compute the best segmentation. For clarity, we will first introduce graph cuts by describing how the algorithm is used in a simplified case where we consider only binary image segmentation, as presented in [1] and [2]. Later we will build upon this example and present graph cuts in full generality.

In binary image segmentation we are looking to separate a single object from the rest of the image. For example, consider the images in Figure 2.3. Suppose that the user is looking to separate the person from the leafy background in the image on the right. We begin by asking the user to provide *hard constraints* on the segmentation, i.e. to identify specific pixels that must belong to the object and must belong to the background. These pixels are called *seed pixels*. For example, a user might draw lines similar to those in Figure 2.3, where a black line indicates pixels that are part

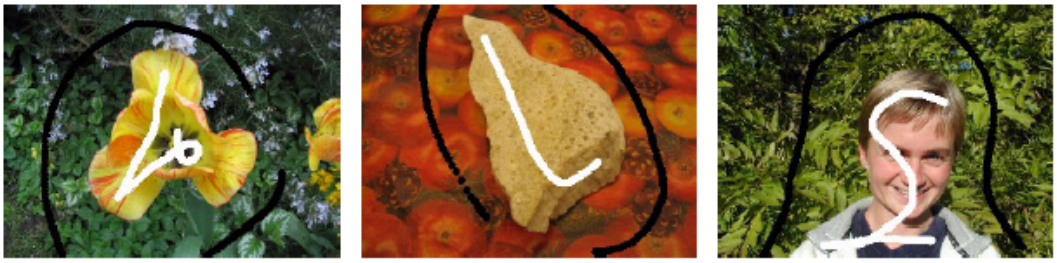


Figure 2.3: Example images for binary segmentation. We want to separate the flower, sponge, and person respectively from their backgrounds. The hard constraints are specified by the user with black lines for background and white lines for object. This image was taken from [20].

of the background and a white line, the object. We say that the pixels labeled as "object" are in a subset $\mathcal{O} \subset \mathcal{P}$ of all the pixels $\mathcal{P}$. Likewise, the pixels the user labels as "background" are in a subset $\mathcal{B} \subset \mathcal{P}$. The seed pixels are used both to reduce the number of possible segmentations as well as to create an intensity distribution that describes the segment and is used to compute the regional term in the energy equation.

Ideally, we would not need to ask the user for this kind of information, but even after the characterization in Sections 2.1 and 2.2 of an ideal segmentation, the question of segmenting an image is still too broad. Often, an image has many foreground objects and a binary segmentation problem is not clearly defined. Asking the user to enter hard constraints avoids this issue and allows us to formulate a well-defined problem. We can state this problem as follows: Given an image with a set of pixels, $\mathcal{P}$, find the vector $A = (A_1, A_2, ...., A_{|\mathcal{P}|})$ where

$$A_i = \begin{cases} \text{“object” or 1} \\ \\ \text{“background” or 0} \end{cases} \tag{2.10}$$

that minimizes the energy $E(A)$. As described in the previous section, we will take

$E(A)$ to be:

$$E(A) = \lambda \sum_{p \in \mathcal{P}} R_p(A_p) + \sum_{\{p,q\} \in \mathcal{N}} B_{p,q} \cdot \delta(A_p, A_q) \tag{2.11}$$

where

$$\delta(A_p, A_q) = \begin{cases} 1 & \text{if } A_p \neq A_q \\ 0 & \text{otherwise} \end{cases} \tag{2.12}$$

This energy function is slightly more specific then the one specified in Equations 2.1 to 2.3 because it adds the condition that the cost of two neighboring pixels having the same label is 0.

The fundamental idea of the graph cuts algorithm is to define a graph on the pixel grid with appropriate edge weights such that the *minimum cut* of the graph will define a segmentation that minimizes the energy function. The cut of a graph is defined as follows: Given a graph $G = (V, E)$ and a specified source ($S \in V$) and sink ($T \in V$), a *cut* is a set of edges $C \subset E$, such that in $G' = (V, E - C)$ $S$ and $T$ are in separate components. The cost of a cut $C$ is the sum of the weights of the edges in $C$, denoted by $|C|$:

$$|C| = \sum_{e \in C} w_e \tag{2.13}$$

where $w_e$ is the weight of edge $e$. The minimum cut of a graph is the cut with minimum cost. Alternatively, we can think of a minimum cut as a partition of the vertices into a *source set*, the component containing $S$, and a *sink set*, the component containing $T$, (there will be no other components by minimality of the cut) and thus is equivalent to a binary segmentation. This is a well studied problem in graph theory, and there are many known efficient solutions to it, one of which is presented in Section

2.4. One advantage of the way we formulate an optimal segmentation in this method is that it is possible to quickly find the exact global solution that minimizes the energy function. On the other hand, most other methods define an optimal segmentation in such a way that guaranteeing an exact solution is an NP-hard problem, and those methods must use approximation algorithms to find a segmentation.

Since most of the real work in finding the optimal segmentation is done by finding the minimum cut of a graph, the only thing we must do is define a graph where the minimum cut defines a segmentation with minimal energy. We can do this as follows: Let the vertices of the graph be all the pixels in the image plus two additional vertices $S$ and $T$, called the *sink* and *source* respectively. The sink and source are referred to collectively as *terminal nodes*. Vertex $T$ represents the background, while $S$ represents the object. Let the edge set in the graph include an edge from every pixel to both $S$ and $T$. We call these *t-links*, short for terminal-links. Additionally, let the edge set include an edge between vertices $p$ and $q$ for all pixels $p$ and $q$ that are neighbors. These are referred to as *n-links*, short for neighbor-links. An n-link between two vertices $p$ and $q$ is denoted by $\{p, q\}$. To visualize such a graph construction, Figure 2.4 (a) shows an example of an image with seed pixels and (b) shows the corresponding graph using a 4-neighborhood system.

Our goal is that a minimal cut will define a segmentation. Therefore, in the graph $G' = (V, E - C)$ it is natural to consider any vertex that is in the same component as $S$ part of the object and any vertex that is in the same component as $T$ part of the background. We can formalize this by defining a segmentation $A(C)$ that is determined by some cut $C$ as follows:

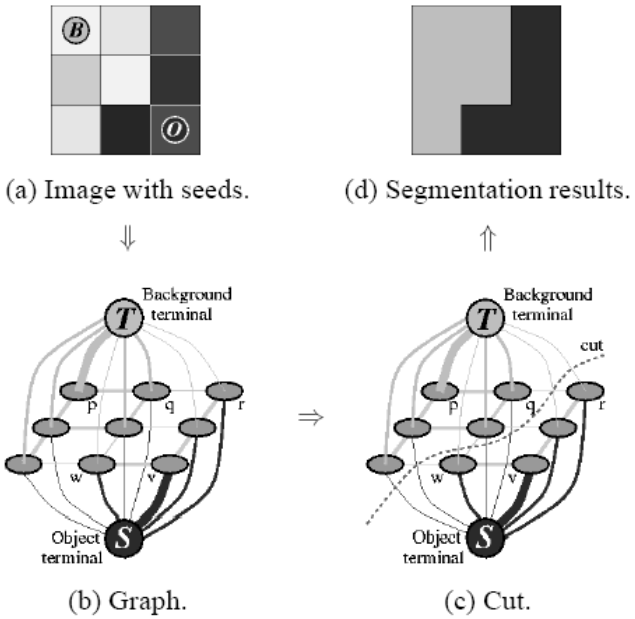



Figure 2.4: The graph and segmentation of a $3 \times 3$ image with $\mathcal{O} = \{v\}$ and $\mathcal{B} = \{p\}$. Figure taken from [2]

$$A_p(C) = \begin{cases} \text{“object” or 1, if } \{p, T\} \in C \\ \text{“background” or 0, if } \{p, S\} \in C. \end{cases} \tag{2.14}$$

An example of a cut and its corresponding segmentation is shown in Figure 2.4 (c) and (d).

The definition in Equation 2.14 that determines a segmentation given a cut will not work for any cut. If $\{p, T\}$ and $\{p, S\}$ are both in the cut then the segmentation is not properly defined because the pixel $p$ is placed in both segments. Additionally, we need that any seed pixel $p \in \mathcal{O}$ has $A_p(C) = 0$ so that it is placed in the object segment. Likewise, any seed pixel $p \in \mathcal{B}$ must have $A_p(C) = 1$. Thus, we must pick edge weights that not only minimize the energy function, but also ensure a minimal cut will satisfy the properties mentioned.

The intuitive way to define the edge weights in the graph is to make a cut $C$ have a cost equal to the energy of the segmentation $A(C)$. In other words, we assign weights that make $|C| = E(A(C))$. Then, when we minimize $|C|$, we will minimize $E(A(C))$. Therefore, it makes sense for the weight of n-links to be equivalent to the boundary costs between those pixels. If we can ensure that an n-link is cut if and only if the two neighboring pixels are in different segments, we can guarantee that we charge the appropriate boundary costs to the cut. Similarly, we can assign regional costs to the weights of the t-links. If $\{p, T\}$ is in the cut then $A_p(C) = 1$ and the weight of $\{p, T\}$ should be $R_p(1)$. Likewise, if $\{p, S\}$ is in the cut then $A_p(C) = 0$ and the weight of $\{p, S\}$ should be $R_p(0)$. Furthermore, for pixels in $\mathcal{O}$ and $\mathcal{B}$ we know which terminal link must be cut, so we should assign the weight of the t-link that needs to be cut to 0 and the assign weight of the the other t-link to be infinite so it is never in the minimum cut. To summarize, Table 2.3 displays the weights that are assigned to each edge in the graph construction.

| edge | weight | for |
|---|---|---|
| $\{p, q\}$ | $B_{p,q}$ | $\{p, q\} \in \mathcal{N}$ |
| $\{p, S\}$ | $\lambda R_p(0)$ | $p \in \mathcal{P}, p \notin \mathcal{O} \cup \mathcal{B}$ |
| | $\infty$ | $p \in \mathcal{O}$ |
| | 0 | $p \in \mathcal{B}$ |
| $\{p, T\}$ | $\lambda R_p(1)$ | $p \in \mathcal{P}, p \notin \mathcal{O} \cup \mathcal{B}$ |
| | 0 | $p \in \mathcal{O}$ |
| | $\infty$ | $p \in \mathcal{B}$ |

Table 2.1: The weights assigned to the edges in the graph construction.

In the following pages we will prove that this construction produces a well-defined segmentation $A$ where $E(A)$ is minimal for all segmentations $A$ that satisfy the hard constraints. The proof will closely follow the argument made in [1]. To prove the construction is correct we define a class $\mathcal{F}$ of cuts that characterize a proper segmentation. We then show that any minimum cut is in $\mathcal{F}$ and that a minimum cut defines

a segmentation $A$ that minimizes the energy $E(A)$. Together, this will prove that our construction is correct.

Let $\mathcal{F}$ be the class of graph cuts such that any $C \in \mathcal{F}$ satisfies the following properties:

1. For each pixel $p$ exactly one of $\{p, S\}$ and $\{p, T\}$ is in $C$.
2. For any two pixels $p, q$ with different t-links severed, $\{p, q\} \in C$.
3. For any two pixels $p, q$ with the same t-link severed, $\{p, q\} \notin C$.
4. For any pixel $p \in \mathcal{O}$, $\{p, T\} \in C$.
5. For any pixel $p \in \mathcal{B}$, $\{p, S\} \in C$.

Condition (1) guarantees that we will have a well-defined segmentation as specified in Equation 2.14. Conditions (2) and (3) ensure that we only charge the cost $B_{p,q}$ if $p$ and $q$ are in different segments. Finally, (4) and (5) guarantee that the seed pixels are placed in the correct segment. Proving that a minimum cut must belong to $\mathcal{F}$ will thus lay the ground work to show that a minimum cut defines a segmentation $A$ with $E(A)$ minimized.

**Theorem 2.3.1.** *The minimum cost cut belongs to $\mathcal{F}$.*

*Proof.* To prove that the minimum cut must be in $\mathcal{F}$ we will go through each property of $\mathcal{F}$ and show that if the property does not hold than the cut cannot be minimal.

1. Any minimal cut must include exactly one t-link for each pixel $p$. Clearly, a cut must include one, since otherwise $S$ and $T$ would be connected by edges $\{p, S\}$ and $\{p, T\}$, and we would not have a cut. But, suppose a cut $C'$ contains both t-links for pixel $p$. Cut $C'$ partitions the graph so $p$ belongs to either the

$S$ component, the $T$ component, or neither. If $p$ belongs to neither then we can simply add back either $\{p, S\}$ and $\{p, T\}$ and still have a cut. Otherwise, without loss of generality, suppose that $p$ belongs to the same component as $S$. We can return the edge $\{p, S\}$ to the graph and we still have a valid cut. The new cut $C = C'/\{p, S\}$ has cost $|C| = Cost(C') - w_{\{p,S\}}$ because $w_{\{p,S\}} > 0$. This implies that $C'$ is not minimal.

2. If $p$ and $q$ are connected to different t-links, then $\{p, q\}$ must be severed. Otherwise, we do not have a cut because $S$ and $T$ would be connected by a path through $p$ and $q$.

3. Whenever $p$ and $q$ are connected to the same terminal, then $\{p, q\}$ cannot be in a minimal cut. Suppose there is a minimal cut $C$ such that $\{p, q\} \in C$ and $p$ and $q$ are connected to the same terminal, then $p$ and $q$ both belong to the same component in $G' = (V, E \backslash C)$. This implies that if we return $\{p, q\}$ to the graph we still have a valid cut because $\{p, q\}$ lies within one component. But, just as in Item 1, this new cut has a smaller cost than the original, and thus $C$ can not be minimal.

4. Finally, in any minimal cut $C$, $p \in \mathcal{O}$ implies that $\{p, T\} \in C$. Suppose a minimal cut $C$ contains $\{p, S\}$ for $p \in \mathcal{O}$ and not $\{p, T\}$. The cost of $\{p, S\}$ is $\infty$, which is greater than all of the other edges out of $p$, since all the edges out of $p$ are less than $\infty$. Therefore, we can replace $\{p, S\}$ for $\{p, T\}$ and the appropriate n-links, and we have a smaller cost. This gives us a valid cut since we simply moved $p$ from the $S$ component to the $T$ component. We now have a cut with a smaller cost, implying that $C$ cannot be minimal.

5. A parallel explanation to Item 4 implies that any minimal cut includes $\{p, S\}$

if $p \in \mathcal{B}$.

□

**Theorem 2.3.2.** *Let $\mathcal{H}$ be the set of segmentations that satisfy the hard constraints. A minimal cut $C$ defines a segmentation $A = A(C)$ that minimizes the energy $E(A)$ among all $A \in \mathcal{H}$.*

*Proof.* First, notice that a cut $C \in \mathcal{F}$ will have a cost that is simply the sum of the weights of the t-links and n-links severed:

$$|C| = \sum_{A_p(C)=1} w_{\{p,T\}} + \sum_{A_p(C)=0} w_{\{p,S\}} + \sum_{\{p,q\}\in C} w_{\{p,q\}}. \tag{2.15}$$

Additionally, we know that for any $\mathcal{P} \in \mathcal{O} \cup \mathcal{B}$ the cost of its t-link is 0, since any cut in $\mathcal{F}$ connects the seed pixels to their correct terminal. Also, $\{p, q\} \in C$ if and only if $A_p \neq A_q$. So, given those two facts and by looking up costs in Table 2.3, we can rewrite the cost of a cut in $\mathcal{F}$ as follows:

$$|C| = \sum_{p\notin\mathcal{O}\cup\mathcal{B}} \lambda \cdot R_p(A_p(C)) + \sum_{\{p,q\};A_p(C)\neq A_q(C)} B_{p,q}. \tag{2.16}$$

Notice that this looks almost exactly like $E(A)$ as defined in Equations 2.11 and 2.12. In fact,

$$|C| = E(A(C)) - \sum_{p\in\mathcal{O}} \lambda \cdot R_p(1) - \sum_{p\in\mathcal{B}} \lambda \cdot R_p(0) \tag{2.17}$$

Now, we know the pixels that are in $\mathcal{O} \cup \mathcal{B}$ before we try to compute the segmentation. Therefore, we can precompute the regional term values for pixels in $\mathcal{O} \cup \mathcal{B}$, and so the term $\sum_{p\in\mathcal{O}} \lambda \cdot R_p(1) + \sum_{p\in\mathcal{B}} \lambda \cdot R_p(0)$ is just a constant. This gives us that:

$$|C| = E(A(C)) - b, \quad \text{where } b \text{ is a constant.} \tag{2.18}$$

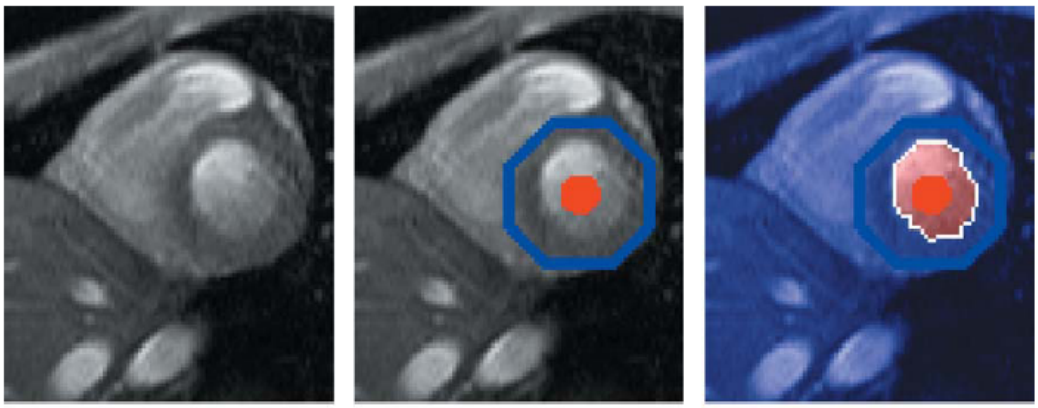
When we minimize $|C|$, $E(A(C)) = E(A)$ is minimal as well since a cut in $\mathcal{F}$ can represent any minimal segmentation $A \in \mathcal{H}$:

$$\min_{C \in \mathcal{F}}(|C| + b) = \min_{C \in \mathcal{F}} E(A(C)) = \min_{A \in \mathcal{H}} E(A). \tag{2.19}$$

□

The graph cuts algorithm presented thus far has many limitations but also many great properties. The most significant shortcomings is the limitation of a binary labeling. Other variations of the graph cuts algorithm address this problem and provide ways to segment an image into more labels. However, when we deal with more labels we cannot formulate the problem as the minimum cut between two vertices. If we try to find a partition into many sets by using multiple sources and sinks then the problem can be shown to be NP-hard [6]. Therefore, to segment into multiple labels we must use an approximation algorithm and we are not guaranteed a global energy minimization as we are in the binary case.

The need for seed pixels can be considered both a limitation and an advantage depending on the application. They can be advantageous in applications where we are looking to segment specific components of an image. The seed pixels allow the user to modify the segmentation to exactly what they want, and is much easier than segmenting an image completely by hand. However, when looking to fully automate a segmentation the need for seed pixels becomes a problem. As we mentioned before, the seed pixels are very important because they are not only used to limit the search space but also used to find prior probabilities to compute the $R_p(A_p)$ term. Therefore, in a



(a) Original image (b) Initialization (c) Segmentation

Figure 2.5: An example of segmentation for a medical image using graph cuts where the seed pixels are placed automatically. The Figure is taken from [1]

fully automated system, the computer will have to choose seed pixels, which is difficult task in and of itself. Although, it is possible to generate seed pixels automatically. Figure 2.5 shows a medical image where the circular ring of seed pixels is added automatically using template matching.

Finally, another advantage of the graph cuts algorithm is that there is no bias in the segmentation for each segment to take on any specific topology, unlike many algorithms that favor elliptical regions, or require that regions are connected. For example, a variation of the k-means algorithm used for segmentation in [22] requires that regions are connected, and favors elliptically shaped regions.

## 2.4 Multilabel Segmentation

The graph cuts algorithm can be expanded to segment an image into an arbitrary number of regions. In this case it is not possible to find the exact global minimum in a reasonable amount of time, but we can come very close with an approximation scheme. This section shows one approximation solution to the multilabeling problem

called $\alpha$-*expansion* [6] [12]. The main limitation of the algorithm is that we must know all the possible labels to which a pixels can be assigned. Specifically, in image segmentation, we must determine the number of segments to break the image into before we begin the segmentation. Although this is often a disadvantage, specifying the number of labels can be quite useful in medical and scientific imaging problems where there are specified types of objects we are looking to separate. Additionally, the algorithm has been quite useful in stereo vision.

The problem of assigning multiple labels to each pixel in an image can be formulated in an almost identical fashion to the binary case. We are looking for a vector $A = (A_1, A_2, ..., A_p)$ of length $|\mathcal{P}|$, where $\mathcal{P}$ is the number of pixels, such that $A_p \in \mathcal{L}$ where $\mathcal{L}$ is a set of $N$ labels. Specifically, we would like to find an $A$, such that $E(A) = \lambda \sum_{p \in P} R_p(A_p) + \sum_{\{p,q\} \in \mathcal{N}} B_{p,q} \cdot \delta(A_p, A_q)$ is minimal. Unlike in the binary case, we cannot solve for the minimum energy exactly, but instead we find an approximation. The original paper in which the approximation schemes are presented [6] describes two methods of approximation, both of which use the same general concept. They begin with some arbitrary labeling of the vertices and iteratively change the assignment of pixels, where at each step the algorithm solves a binary sub-problem. The first method, called $\alpha$-$\beta$ swap, considers only the subset of pixels in the image that have labels $\alpha$ and $\beta$. Let the current label of pixel $p$ be $f_p$. The method finds the optimal assignment of only $\alpha$ and $\beta$ among those pixels $p$ with $f_p = \alpha$ or $f_p = \beta$. As the name implies, at each step the algorithm is swapping the labels $\alpha$ and $\beta$. We repeat this process continually going through every pair of labels until convergence. We do not present this method in full detail because the other approximation scheme, $\alpha$-expansion, is both faster and comes closer to the optimal solution in most applications [20].

The idea of $\alpha$-expansion is very similar to $\alpha$-$\beta$ swap, except instead of picking

two labels, we pick one label and then take the other label to be the set of labels not equal to $\alpha$, which we denote $\overline{\alpha}$. In each iteration we find the optimal segmentation where each pixel can take on the value of $\alpha$ or its previous label, and thus the number of labels with $\alpha$ "expands". The graph construction is more complicated than it is for the binary case because when we consider two pixels beginning with two different labels, $f_p \neq \alpha$ and $f_q \neq \alpha$, the cost of $B_{p,q}(\alpha, \overline{\alpha}) = B_{p,q}(\alpha, f_q)$ and $B_{p,q}(\overline{\alpha}, \alpha) = B_{p,q}(f_p, \alpha)$ are usually not the same. In the binary case and in $\alpha$-$\beta$ swap, we always have that $B_{p,q}(\alpha, \beta) = B_{p,q}(\beta, \alpha)$. This difference makes the graph construction harder because the cost of cutting an n-link between pixels $p$ and $q$ depends on the respective assignments of $p$ and $q$. To deal with the more complicated energy function, the original paper [6] proposed a graph construction that adds nodes to the graph in addition to the pixels and two terminal nodes. However, a later paper [12] gives a simpler graph construction that does not add extra nodes, but instead compensates by using a directed graph. In fact, the paper [12] presents a general graph construction that minimizes any energy function that can be minimized with graph cuts. In what follows, we present the construction for the graph needed to perform $\alpha$-expansion from [12]. The construction is implied by [12], but never completely described in the paper.

Recall that in $\alpha$-expansion we iteratively optimize using each label $\alpha$ until convergence. For each $\alpha$ we dynamically change the edge weights of the graph to appropriately minimize the energy for the segmentation of labels $\alpha$ and $\overline{\alpha}$. To describe the graph construction that works first we need to introduce a little bit of notation. Given a label $\alpha$, we let $\mathcal{P}_\alpha$ denote the pixels that currently have the label $\alpha$ and we let $\mathcal{P}_{\overline{\alpha}}$ denote all other pixels ($\mathcal{P}_\alpha \cup \mathcal{P}_{\overline{\alpha}} = \mathcal{P}$). Also, as in the binary case we define a neighborhood for each pixel, which will be used to determine the edges in the graph construction. As mentioned above, we need a directed graph for this construc-

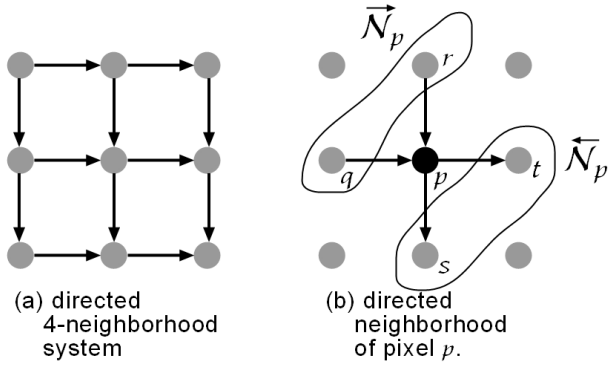



Figure 2.6: **(a)** An illustration of a directed 4-neighborhood system defined on the entire image. **(b)** The neighborhoods $\vec{\mathcal{N}}_p$ and $\overleftarrow{\mathcal{N}}_p$ of a single pixel $p$.

tion, and therefore we need to define a directed neighborhood. To define a directed neighborhood, we simply have to assign a direction to each edge in an undirected neighborhood. This can be done in an arbitrary fashion, but here, for simplicity and clarity, we define the following directed neighborhood: for a pixel $p$, all $q \in \mathcal{N}_p$ that are above and to the left of $p$ will be in the "in-neighborhood," denoted $\vec{\mathcal{N}}_p$. Likewise, all $q \in \mathcal{N}_p$ that are to the bottom and right of $p$ make up the "out-neighborhood," denoted $\overleftarrow{\mathcal{N}}_p$. It will be easy to remember the notation by thinking of the arrow over $\mathcal{N}$ pointing toward the subscript $p$ for the in-neighborhood, and away from the subscript $p$ for the out-neighborhood. Figure 2.6 shows an illustration of a directed 4-neighborhood system defined on the entire image and the neighborhoods $\vec{\mathcal{N}}_p$ and $\overleftarrow{\mathcal{N}}_p$ of a single pixel $p$.

Finally, we can begin to construct the graph. Let $G = (V, E)$ be a graph with $V = \mathcal{P} \cup \{S\} \cup \{T\}$, where $S$ and $T$ are terminal nodes. Since we are considering a directed graph we will denote an edge in $E$ by $(u, v)$, where $u, v \in V$ and $(u, v) \neq (v, u)$. Let $E$ contain directed t-links from $S$ to all $p \in \mathcal{P}$, denoted $(S, p)$, and from all $p \in \mathcal{P}$ to $T$, denoted $(p, T)$. Additionally, $E$ contains n-links $(p, q)$ between all pairs of pixels $p \in \mathcal{P}$ and $q \in \overleftarrow{\mathcal{N}}_p$. We want to assign appropriate edge weights to

the graph such that the minimal cut, $C$, will minimize the energy and determine a segmentation as follows:

$$A_p(C) = \begin{cases} \alpha \text{ if } (S,p) \in C \\ \overline{\alpha} = f_p \text{ if } (p,T) \in C. \end{cases} \tag{2.20}$$

From the binary case we know that any minimal cut will define such a segmentation because exactly one t-link is severed per vertex. From this assignment we determine that vertex $S$ represents $\overline{\alpha}$ and $T$ represents $\alpha$.

A natural way to make the cut minimize the energy is again to assign weights so that the cost of the cut $C$, $|C|$, is equal to $E(A(C))$. We will consider $R(A)$ and $B(A)$ separately, assign the appropriate edge weights to the graph for each term, and then "add" the graphs together by summing the weights of each edge. It is intuitive that we can do this since $E(A)$ is a sum. The full proof for this "summation theorem" is provided in [12].

Accounting for the $R(A)$ term in the graph construction is relatively simple. For any $p \in \mathcal{P}_{\overline{\alpha}}$ we want to charge a cost of $R_p(f_p)$ if $p$ does not change its label and $R_p(\alpha)$ if $p$ changes its label to $\alpha$. When $p$ keeps its label $f_p$ it is equivalent to cutting the t-link $(p,T)$, so we charge $R_p(f_p)$ if $(p,T)$ is in the cut, implying that $w_{(p,T)} = R_p(f_p)$. Likewise, $w_{(S,p)} = R_p(\alpha)$.

For $p \in \mathcal{P}_\alpha$, $p$ cannot be assigned to $\overline{\alpha}$, since $\overline{\alpha}$ is not a single label. Therefore for $p \in \mathcal{P}_\alpha$, the t-link $(p,S)$ should always be cut. To guarantee this we can make $w_{(p,T)} = \infty$ and $w_{(S,p)} = 0$. Table 2.4 summarizes the component of the edge weights assigned for the $R(A)$ term in the graph construction.

Now, we need to consider the $B(A)$ term. To simplify, we will consider the costs for each pair of pixels, and then "add" the graphs together. Again, the justification

| edge | weight | for |
|---|---|---|
| $(S,p)$ | $\lambda R_p(\alpha)$ | $p \in \mathcal{P}_{\overline{\alpha}}$ |
| | $0$ | $p \in \mathcal{P}_{\alpha}$ |
| $(p,T)$ | $\lambda R_p(f_p)$ | $p \in \mathcal{P}_{\overline{\alpha}}$ |
| | $\infty$ | $p \in \mathcal{P}_{\alpha}$ |

Table 2.2: The components of the edge weights in the graph construction for the regional term.

that we can do this is provided in [12], but it is intuitive since the boundary term is a sum of pairwise interactions.

For any pair of pixels $p$ and $q$, we have the following possible boundary costs based on the assignments $A_p$ and $A_q$:

$(A_p, A_q) = (\overline{\alpha}, \overline{\alpha}) : B_{p,q}(f_p, f_q) = V_{p,q}$ (for short),

$(A_p, A_q) = (\alpha, \overline{\alpha}) : B_{p,q}(\alpha, f_q) = V_{\alpha,q}$,

$(A_p, A_q) = (\overline{\alpha}, \alpha) : B_{p,q}(f_p, \alpha) = V_{p,\alpha}$,

$(A_p, A_q) = (\alpha, \alpha) : B_{p,q}(\alpha, \alpha) = V_{\alpha,\alpha}$.

Each of these possible assignments of $p$ and $q$ corresponds to a specific cut through the subgraph containing $S$, $T$ and pixels $p$ and $q$. Figure 2.7 shows the corresponding cut for each assignment of $p$ and $q$.

In order to charge the appropriate cost for all possible assignments, we need the sum of the weights of the edges in the cut to add up to the cost. For example, if $A_p = \overline{\alpha}$ and $A_q = \alpha$ we need $w_{(S,p)} + w_{(p,q)} + w_{(p,T)} = V_{p,\alpha}$. We can write down similar equations for each possible cut. Notice that because the edge $(p,q)$ is directed, it is not part of the cut when $A_p = \alpha$ and $A_q = \overline{\alpha}$. Remember that the cost of the cut is the sum of edges going from the source set to the sink set. For the case $(A_p, A_q) = (\alpha, \overline{\alpha})$ the source set is composed of $S$ and $p$, and the sink set is composed of $T$ and $q$. Therefore, the edge $(p,q)$ goes from the sink set to the source set and is not part of the cut.

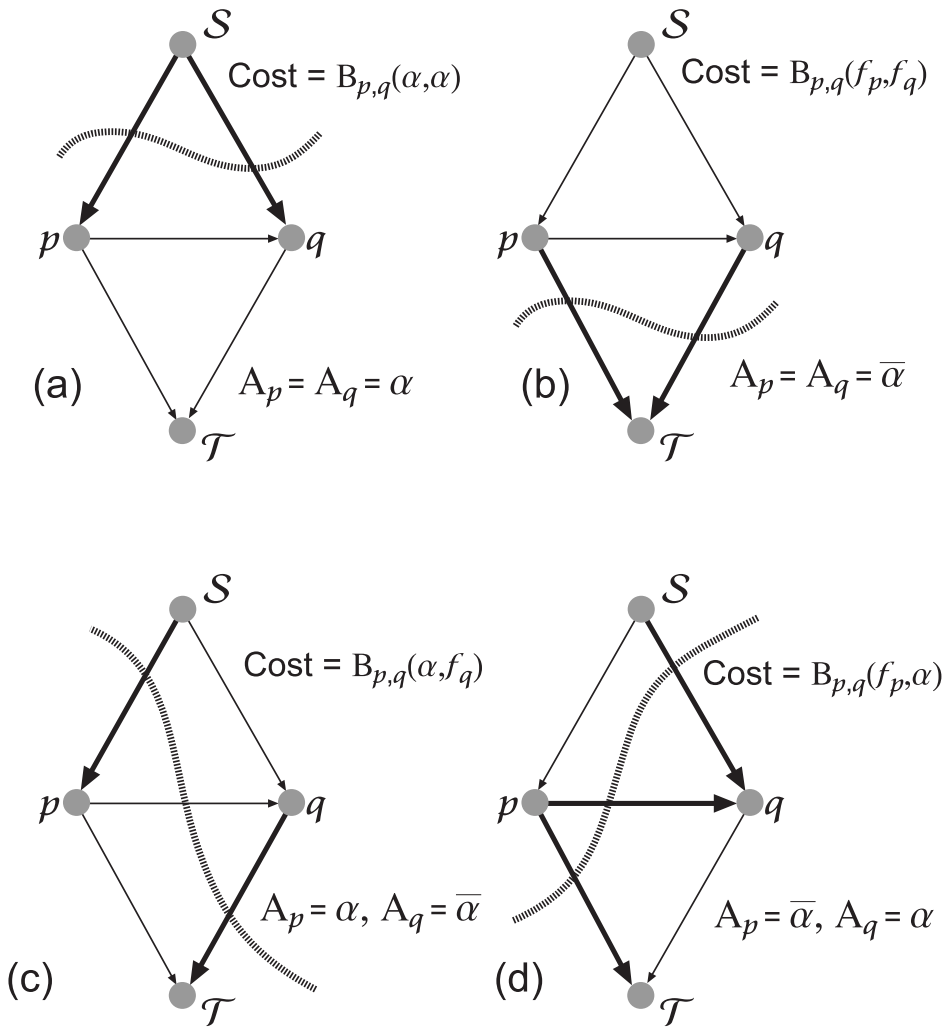



Figure 2.7: The possible cuts between two vertices. The cut line is drawn as a dotted line, and the edges included in each cut are in bold. Each cut corresponds to an assignment of $p$ and $q$. Next to each cut is the cost each cut needs to have so that the cost of the cut equals the boundary cost between the two pixels. Notice that edge $(p, q)$ is not included in the cut in image (c) even though the cut line severs this edge. This is because a cut is the edges from the source set to sink set, but the edge $(p, q)$ goes from the sink set to the source set.

The equations we write down for each possible assignment of $p$ and $q$, relating the desired cost of the cut to the edge weights, is simply a system of linear equations and can be solved. (Note that the set of equations cannot be solved if we do not have directed edges, and thus the reason we build a directed graph.) When we solve for the weights we get $w_{(S,p)} = V_{\alpha,q}$, $w_{(S,q)} = V_{\alpha,\alpha} - V_{\alpha,q}$, $w_{(p,T)} = V_{p,q}$, $w_{(q,T)} = 0$, and $w_{p,q} = V_{\alpha,q} + V_{p,\alpha} - V_{\alpha,\alpha} - V_{p,q}$. However, these weights cannot be used because some of them are negative. We need our graph to have positive weights to find the minimal cut. Typically, the value of $V(\alpha, \alpha)$ is zero since we only charge a boundary cost if the assignments are different, and the other terms are positive, implying that $w_{(S,q)} < 0$. Also, $w_{(p,q)}$ is only positive when $V_{\alpha,q} + V_{p,\alpha} \geq V_{p,q}$. We can solve the problem that $w_{(S,q)} < 0$ by adding a constant to each possible cut cost, since adding a constant will not change the minimal cut. If we add a constant to edges $(S, q)$ and $(q, T)$ we add the constant to each cost exactly once since exactly one of those edges in included in each cut. To make $w_{(S,q)} \geq 0$ we need to add at least $V_{\alpha,q}$, and this is sufficient. When we add $V_{\alpha,q}$ to $(S, q)$ and $(q, T)$ we have the following weights for the edges in the subgraph: $w_{(S,p)} = V_{\alpha,q}$, $w_{(S,q)} = 0$, $w_{(p,T)} = V_{p,q}$, $w_{(q,T)} = V_{\alpha,q}$, $w_{p,q} = V_{\alpha,q} + V_{p,\alpha} - V_{p,q}$. The weights are also shown graphically in Figure 2.8. The problem that $w_{(p,q)}$ can be negative can only be resolved by requiring that $V_{\alpha,q} + V_{p,\alpha} \geq V_{p,q}$ is true for the energy function used. More generally, Kolmogorov and Zabih [12] show that the only energy functions that can be minimized by graph cuts must satisfy the following inequality for all pairs $p, q$: $B_{p,q}(\alpha, \alpha) + B_{p,q}(f_p, f_q) \geq B_{p,q}(\alpha, f_q) + B_{p,q}(f_p, \alpha)$. Therefore, the edge weights shown in Figure 2.8 are the final pairwise boundary weights in the graph construction.

Now, we need to add the pairwise weights together to find weights for the entire graph. Figure 2.9 illustrates the merging of the pairwise graphs for a one dimensional image. To compute the weight of $(S, p)$ for a pixel $p$ we must add the weight of the

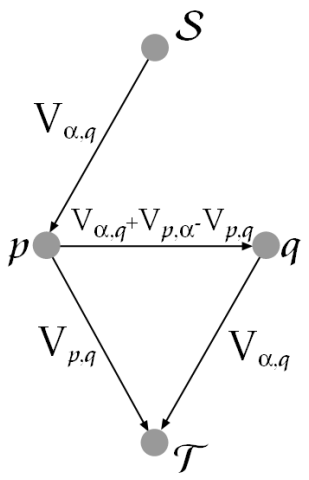



Figure 2.8: Edge weights for the pairwise interaction between pixels $p$ and $q$.

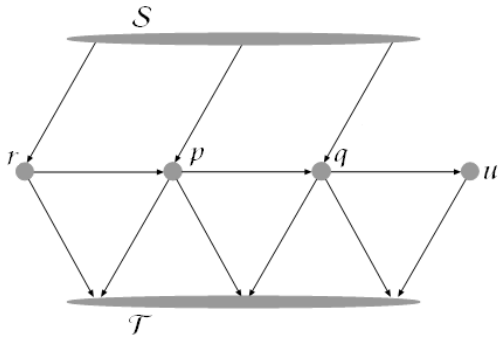



Figure 2.9: A view in a one dimensional image revealing how the pairwise edges must be added together when calculating the edge weights for the entire graph.

edge $(S, p)$ for all of the pairwise interactions that involve $p$. As Figure 2.9 shows, the edge $(S, p)$ exists only in the pairwise interactions to the right of the pixel, and thus part of the out-neighborhood. For each pixel $q \in \overleftarrow{\mathcal{N}}_p$, the pairwise graph with $p$ and $q$ gives weight $V_{\alpha,q}$ to $(S, p)$, therefore the weight of $(S, p)$ in the entire graph is the sum of $V_{\alpha,q}$ for every pixel $q \in \overleftarrow{\mathcal{N}}_p$. Likewise, to compute the weight of $(p, T)$ for a pixel $p$ we must add the weight of edge $(p, T)$ for all the pairwise interactions that involve $p$. Here, every pairwise interaction involving $p$ includes the edge $(p, T)$. For every pixel $q \in \overleftarrow{\mathcal{N}}_p$ the pairwise graph containing $p$ and $q$ assigns weight $V_{p,q}$ to edge $(p, T)$; for every pixel $r \in \overrightarrow{\mathcal{N}}_p$ the pairwise graph containing $p$ and $r$ assigns weight $V_{\alpha,r}$ to the edge $(p, T)$. Therefore, the weight of $(p, T)$ in the entire graph is the sum of $V_{p,q}$ for every pixel $q \in \overleftarrow{\mathcal{N}}_p$, plus $V_{\alpha,r}$ for every pixel $r \in \overrightarrow{\mathcal{N}}_p$. The final weights after we combine all the pairwise interactions and the $R(A)$ term are shown in Table 2.3.

| edge | weight | for |
|---|---|---|
| (p,q) | $B_{p,q}(\alpha, q) + B_{p,q}(p, \alpha) - B_{p,q}(p, q)$ | $p \in \mathcal{P}, q \in \overleftarrow{\mathcal{N}}_p$ |
| $(S, p)$ | $\lambda R_p(\alpha) + \sum_{q \in \overleftarrow{\mathcal{N}}_p} B_{p,q}(\alpha, q)$ | $p \in \mathcal{P}_\alpha$ |
| | $\sum_{q \in \overleftarrow{\mathcal{N}}_p} B_{p,q}(\alpha, q)$ | $p \in \mathcal{P}_\alpha$ |
| $(p, T)$ | $\lambda R_p(f_p) + \sum_{q \in \overleftarrow{\mathcal{N}}_p} B_{p,q}(p, q) + \sum_{q \in \overrightarrow{\mathcal{N}}_p} B_{p,q}(\alpha, q)$ | $p \in \mathcal{P}_{\overline{\alpha}}$ |
| | $\infty$ | $p \in \mathcal{P}_{\overline{\alpha}}$ |

Table 2.3: The weights of the graph used for $\alpha$-expansion for a given label $\alpha$ and directed neighborhood system $\mathcal{N}$.

The proof that the minimum cut in the graph construction for a label $\alpha$ finds the optimal assignment of $\alpha$ can be found in [12]. The proof has a similar flavor to the

proof in Section 2.3 that shows the construction for the binary case is correct. In order to find an approximate segmentation we must repeat the $\alpha$-expansion process for each label until nothing changes. A proof that this process closely approximates the optimal segmentation for multiple labels can be found in [6]. Specifically, the paper shows that if $\hat{f}$ is the segmentation found by this approximation and if $f^*$ is the optimal segmentation then $E(\hat{f}) \leq 2cE(f^*)$ where $c$ is a constant dependent on the values of $B_{p,q}$.

## 2.5 Minimum Cut Algorithms

One of the beauties of the graph cuts algorithm is that most of the work involved in computing the solution is accomplished by finding the minimum cut of the graph, which is a well studied problem with an efficient solution. In this section we will briefly outline one of the most famous minimum cut algorithms. The explanations are based on [4] and [8].

The key discovery that led to good algorithms to find the minimum cut is the theorem of Ford and Fulkerson (1956) that states that in a directed graph with a source and a sink, the minimum cut equals the maximum flow. The flow of a graph is a function $f$ from the edges to a set of real numbers such that $\forall e \in E, f_e \leq w_e$, and for all $v \in V \backslash \{S, T\}$, $\sum_{e \in (\cdot, v)} f_e = \sum_{e \in (v, \cdot)} f_e$. In words, the flow on a graph is value assigned to all the edges such that the value is less than the edge weight, and the sum of the flow into a vertex equals the sum of the flow out of a vertex. In these terms, the weight of the edge can be viewed as the maximum flow capacity of an edge. The intuition that min-cut equals max-flow is the following: If we can find a cut that has a sum of edge weights equal to $c$, then we know that no more that $c$ units can cross that cut, and thus that flow can be at most $c$. Likewise, if there is a flow $f$ from $S$ to

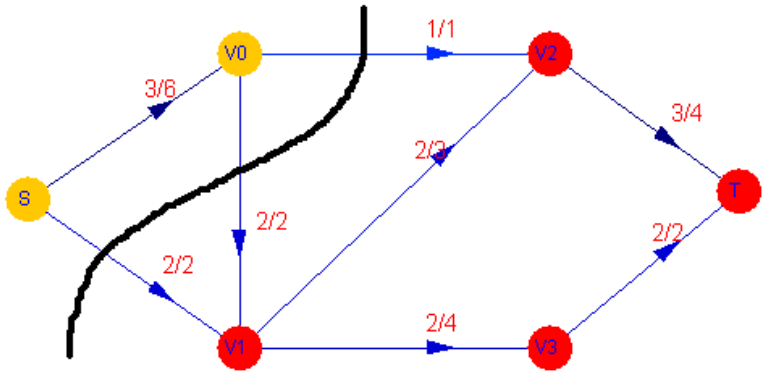



Figure 2.10: An example illustrating that maximum flow equals minimum cut. Edges are labeled with a fraction $x/y$ where $x$ is the flow on the edge and $y$ is the weight of the edge. The current flow on the graph is 5, which can be computed by adding the flow out of vertex $S$, or equivalently, into vertex $T$. A cut is drawn by the line through the graph. The cut includes all the edges going through that line from the "$S$ side" to the "$T$ side." By adding the weights of those edges one can see that the cost of the cut is also 5. Therefore, since the minimum cut equals the maximum flow the cut here must be minimal and the flow maximal. This figure was created using an applet illustrating the augmenting path algorithm [11].

$T$, the cost of any cut separating $S$ and $T$ must be at least $f$. Figure 2.10 shows an example illustrating that minimum cut equals the maximum flow. The reason that the min-cut max-flow theorem is useful is because if we find a cut and a flow with the same value we know that the cut must be minimal and the flow must be maximal. Figure 2.10 shows an example illustrating that minimum cut equals the maximum flow.

The fact that the minimum cut equals the maximum flow can be utilized in an algorithm to compute the minimum cut. The *augmenting path algorithm*, which was developed by Ford and Fulkerson, is a method that works by continually increasing the flow of the graph. When we can no longer increase the flow we also find a cut of the same value, and we find the max-flow and min-cut.

The algorithm works as follows: Beginning with zero flow, we search for a path

from $S$ to $T$, ignoring the direction of the edges, for which the following properties hold: $w_e - f_e > 0$ if on the path edge $e$ is directed toward the sink, and $f_e > 0$ if on the path edge $e$ is directed toward the source. We call such paths *augmenting paths*. Essentially, we are looking for any path to which we can add additional flow. We update the path by adding enough additional flow to the path so that at least one edge is *saturated*, i.e. $w_e = f_e$ for edges on the path directed toward the sink, or $f_e = 0$ in the opposite direction. We continue this until no more augmenting paths can be found, at which point we find the minimum cut and maximum flow. To see that we have found a minimum cut, consider the set of vertices $A$ such that there exists an augmenting path from $S$ to $v \in A$. This is some subset of the vertices since $T$ cannot be in $A$ or there would be an augmenting path. Now consider all the edges going from $A$ to $V - A$. This is a cut because all the vertices in $A$ are connected to $S$ and not $T$, and the vertices in $V - A$ are connected to $T$ but not $S$. This cut that has the value of the current flow because it is the set of edges restricting the flow from getting any larger. Figure 2.11 illustrates the steps of the augmenting path algorithm on a simple graph.

The Dinic variation of the original algorithm extends the algorithm by specifying that the search for augmenting paths be done with a breadth first search, finding the shortest augmenting path at every step. With this method the minimum cut can be found in time $O(n^2m)$ where $n$ is the number of nodes and $m$ is the number of edges in the graph. The original Ford and Fulkerson algorithm does not specify that we should find the shortest augmenting path, and without that condition we are not guaranteed to ever converge to a minimum cut.

In practice, graph cuts uses a minimum cut algorithm that runs faster for the types of graphs that are used in the image setting. This algorithm is specified in [4].

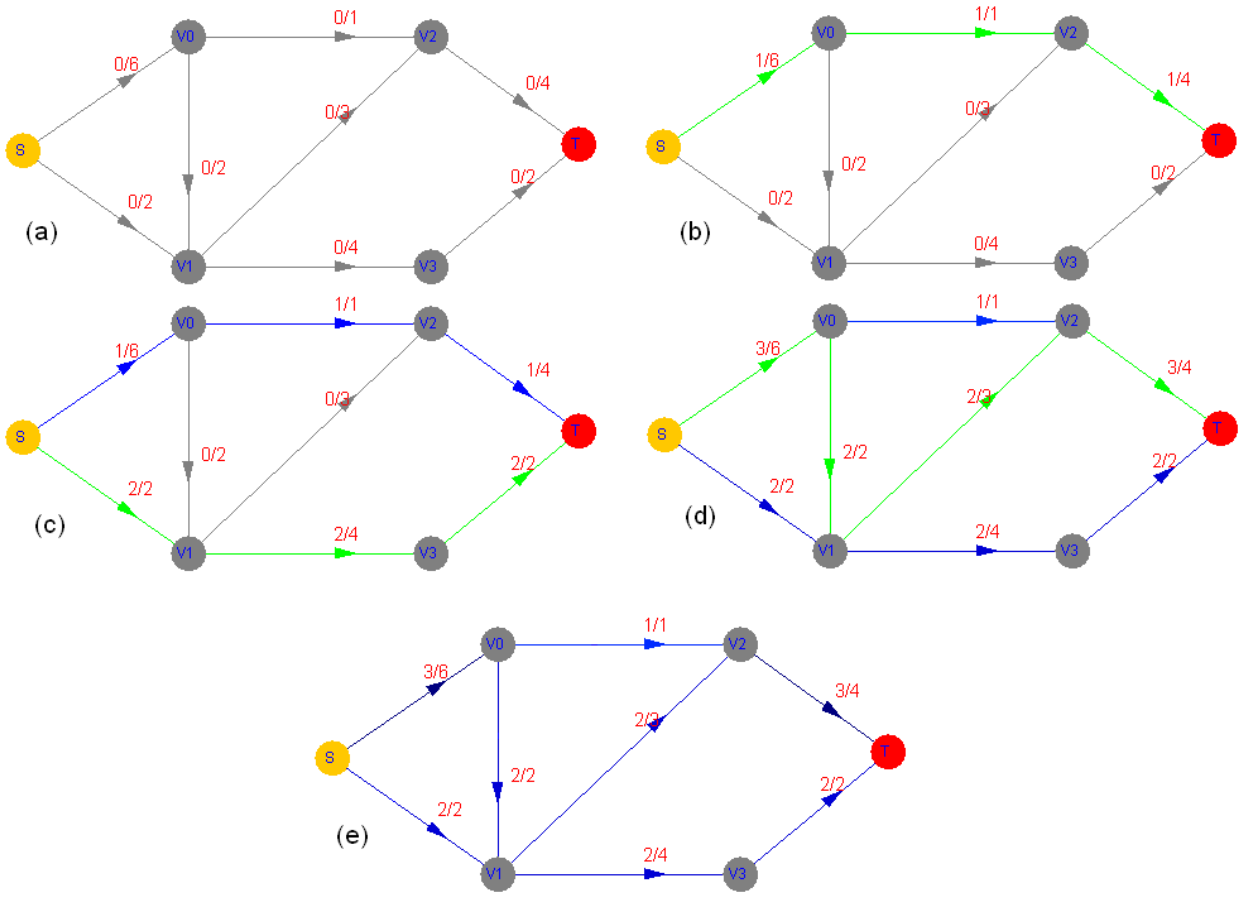



Figure 2.11: An example illustrating the steps of the augmenting path algorithm. The edges of the graphs are labeled with a fraction $x/y$ where $x$ is the flow on the edge and $y$ is the weight of the edge. **(a)** The initial graph; **(b)-(d)** Augmenting path updates; **(e)** The final flow. This figure was created using an applet illustrating the augmenting path algorithm [11]

.

# Chapter 3

# Normalized Cuts

The normalized cut method was introduced by Shi and Malik in their paper *Normalized Cuts and Image Segmentation* [18]. The paper is the one of the most-cited papers in computer vision according to Google Scholar, which indicates that the method is used frequently in practice. In Section 3.1 we will present the definition of an optimal segmentation used by the normalized cuts algorithm, and in Section 3.2 we will show how such a segmentation is computed.

## 3.1 Definition of Optimal Segmentation

Normalized cuts also uses a graph model, but it is quite different in nature than the graph cuts technique. The first difference lies in the way this technique defines an optimal segmentation. Whereas in graph cuts we take into account a regional term and a boundary term, in normalized cuts we just consider the boundaries. The advantage of this is that hard constraints are not needed in order to gain information for a data term. This makes normalized cuts more conducive for automatic segmentation, but also limits the potential segmentations that this method can produce because a user

only has limited input in determining the segmentation.

The normalized cuts algorithm defines the optimal segmentation in terms of a segmentation that divides the image into two parts. To achieve a full segmentation, it finds an optimal binary segmentation and then recursively finds an optimal segmentation within each segment.

The way the normalized cuts algorithm defines an optimal segmentation is quite intuitive. Given some measure of similarity between pairs of pixels, we define the optimal segmentation to be the one that maximizes the similarity between pixels that belong to the same segment, and minimizes the similarity between pixels that belong to different segments. Formally, we can define the optimization problem as follows: Let $G = (V, E)$ be a graph, where $V$ is the set of pixels $\mathcal{P}$, and $E$ is the set of all edges $\{p, q\}$ between every pair of pixels $p$ and $q$. We set the weight of an edge, denoted $w_{\{p,q\}}$, equal to a similarity measure between the pixels $p$ and $q$, where a higher weight corresponds to pixels that are more similar. The similarity measure used by Shi and Malik [18] is the following:

$$w_{i,j} = \begin{cases} e^{-\left(\frac{\|F(i)-F(j)\|^2}{\sigma_I^2} + \frac{\|X(i)-X(j)\|^2}{\sigma_X^2}\right)} & \text{if } \|X(i) - X(j)\|^2 < r^2 \\ 0 & \text{otherwise,} \end{cases} \tag{3.1}$$

where $X(i)$ is the spatial location of $i$ and $F(i)$ is the vector based on intensity, color, or texture information at $i$. This definition defines the similarity between two pixels to be proportional to the probability of pixel $i$ being generated by a Gaussian distribution centered at pixel $j$. Notice that with this definition we do not need a fully connected graph, but instead only place edges between two pixels that have a positive weight, i.e. when $\|X(i) - X(j)\|^2 < r^2$.

To measure the similarity between segments we use a *normalized cut*. Given a

graph as specified above, and sets $A$ and $B$, a normalized cut, denoted *Ncut*, is defined to be:

$$\text{Ncut}(A, B) = \frac{\text{cut}(A, B)}{\text{assoc}(A, V)} + \frac{\text{cut}(A, B)}{\text{assoc}(B, V)}, \tag{3.2}$$

where

$$\text{cut}(A, B) = \sum_{p \in A, q \in B} w_{\{p,q\}}, \tag{3.3}$$

and

$$\text{assoc}(A, V) = \sum_{p \in A, q \in V} w_{\{p,q\}}. \tag{3.4}$$

The associativity and cut both measure how similar two sets are to each other. The only difference is that a cut requires that the two sets are a partition of $V$, while associativity generalizes this to measure the similarity between any two sets. A normalized cut measures how similar $A$ is to $B$ compared to how similar $A$ is to the whole image and $B$ is to the whole image. Thus, when the value of the normalized cut is at a minimum, we are finding the segmentation where the difference between $A$ and $B$ is maximized. It may seem a bit odd that we do not consider the cut alone but rather the ratio of cut over associativity. However, this is necessary to avoid a bias resulting from a different amount of edges in the cut depending on the relative size of the segments. Unlike in graph cuts, where at least one terminal-link from every node must be in the cut, here if a segment is just one pixel, the value of the cut is just the sum of at most $|\mathcal{P}| - 1$ edges out of the pixel. This sums to significantly fewer edges than a cut that divides the image in half and cuts at most $\binom{\mathcal{P}/2}{2}$ edges. If we consider just the cut alone then we will falsely encourage one of the two parts to contain very few pixels.

To measure the similarity between pixels within the same segment we formalize something similar to the normalized cut called the *normalized associativity*. The nor-

malized associativity between sets $A$ and $B$, denoted $Nassoc(A, B)$, is the following:

$$\text{Nassoc}(A, B) = \frac{\text{assoc}(A, A)}{\text{assoc}(A, V)} + \frac{\text{assoc}(B, B)}{\text{assoc}(B, V)}. \tag{3.5}$$

Normalized associativity is the normalized sum of all the edges with each segment. The segmentation for which $\text{Nassoc}(A, B)$ is maximized will give us the segmentation with the most similarity within $A$ and $B$ because the weight of an edge is higher when two pixels are more similar,

Our definition of an optimal segmentation comes together when we observe that the normalized cut and normalized associativity are related, and finding the maximum of $\text{Nassoc}(A, B)$ is equivalent to finding the minimum of $\text{Ncut}(A, B)$. Notice that $\text{cut}(A, B) = \text{assoc}(A, V) - \text{assoc}(A, A)$: All the edges in the graph can be placed in one of three groups: (1) between $A$ and $B$, (2) within $A$, or (3) within $B$. By definition, $\text{cut}(A, B)$ sums exactly those in group (1), $\text{assoc}(A, A)$ sums exactly those in group (2), and $\text{assoc}(A, V)$ sums exactly those in (1) and (2). This allows us to rewrite Ncut as follows:

$$\begin{aligned}
\text{Ncut}(A, B) &= \frac{\text{cut}(A, B)}{\text{assoc}(A, V)} + \frac{\text{cut}(A, B)}{\text{assoc}(B, V)} \\
&= \frac{\text{assoc}(A, V) - \text{assoc}(A, A)}{\text{assoc}(A, V)} + \frac{\text{assoc}(B, V) - \text{assoc}(B, B)}{\text{assoc}(B, V)} \\
&= 2 - \left( \frac{\text{assoc}(A, A)}{\text{assoc}(A, V)} + \frac{\text{assoc}(B, B)}{\text{assoc}(B, V)} \right) \\
&= 2 - \text{Nassoc}(A, B).
\end{aligned} \tag{3.6}$$

Thus, we have shown that the when Ncut is minimized, $\text{Nassoc}(A, B)$ is maximized, justifying this notion of an optimal segmentation. Unfortunately, we do not have a fast way to compute a segmentation that optimizes these criteria exactly. In

fact, finding the exact solution is known to be NP-complete. This is one of the biggest disadvantages of normalized cuts because although the notion of the normalized cut method seems like an ideal way to define a segmentation, we can not use it directly in practice.

## 3.2 Approximating the Optimal Segmentation

Although we cannot find an exact solution to the optimization problem defined in the previous section, we can find an approximate solution. If we transfer the problem to an eigenvector problem, we can find an exact solution in the real-valued domain, and then use it to approximate the discrete-valued solution. To begin, we will slightly rephrase the problem. What we are looking for is a vector $\vec{x}$ of length $|V|$, where $x_i = 1$ if pixel $i$ belongs to segment $A$ of an optimal segmentation and $x_i = 0$ otherwise. We can phrase an eigenvalue problem to find a solution for $\vec{x}$ in the real-valued domain, that is, we let $x_j$ take on any value, not just 0 and 1. Precisely, the optimization problem is equivalent to finding the eigenvector with the second smallest eigenvalue of the following generalized eigenvalue equation:

$$(D - W)\vec{y} = \lambda D\vec{y}, \tag{3.7}$$

where $D$ is a $|V| \times |V|$ diagonal matrix with $d_{i,i} = \sum_j w_{\{i,j\}}$, $W$ is a $|V| \times |V|$ matrix with $w_{i,j} = w_{\{i,j\}}$, $\vec{y} = \vec{x} - b(1 - \vec{x})$, and $b = \frac{\sum_{i \in A} d_{i,i}}{\sum_{i \in B} d_{i,i}}$. The vector $\vec{y}$ is a linear combination of $\vec{x}$ and therefore $\vec{y}$ will define the same segmentation as $\vec{x}$. Below we will give an explanation, but not a complete proof, as to why this is the correct equation to solve. The explanation expands and clarifies the complete proof found in the paper by Shi and Malik [18]. The paper also provides evidence that the eigenvalue

problem is $O(n^{(3/2)})$ where $n$ is the number of pixels in the image, when using a sparse matrix where most of the edge weights are 0.

To give some intuition for why the eigenvector associated with the second smallest eigenvalue of $(D - W)\vec{y} = \lambda D\vec{y}$ minimizes $Ncut(A, B)$, consider a rewriting the equation as follows:

$$\frac{(D - W)\vec{y}}{D\vec{y}} = \lambda. \tag{3.8}$$

The eigenvector associated with the smallest eigenvalue will minimize $\frac{(D-W)\vec{y}}{D\vec{y}}$. We will soon show that $\frac{(D-W)\vec{y}}{D\vec{y}}$ is equivalent to Ncut$(A, B)$. But, if we assume it for now, we can observe a number of properties of $\lambda$ and the associated eigenvectors. First, Ncut cannot be negative because all the weights in the graph are positive, so $\frac{(D-W)\vec{y}}{D\vec{y}} = \lambda \geq 0$. Also, we always have the eigenvalue $\lambda = 0$ associated with the eigenvectors $\vec{y} = \vec{1}$ and $\vec{y} = \vec{0}$ because $D\vec{1} = W\vec{1}$ and $D\vec{0} = W\vec{0}$ . But, in both of these cases, the vector $\vec{y}$ only has one value and all the pixels are assigned to the same segment. The normalized cut of this segmentation is undefined because we need non-empty sets $A$ and $B$ so that we do not divide by zero in the normalized cut. Therefore, the smallest eigenvector is $\lambda = 0$, and we have that the second smallest eigenvalue is the one that minimizes the normalized cut.

Now, to see that $\frac{(D-W)\vec{y}}{D\vec{y}}$ is the same as the value of Ncut$(A, B)$ when $\vec{x}$ (and $\vec{y}$) are discrete we notice that:

$$\frac{(D - W)\vec{y}}{D\vec{y}} = 1 - \frac{W\vec{y}}{D\vec{y}} = 1 - \frac{\vec{y}^T W\vec{y}}{\vec{y}^T D\vec{y}} \tag{3.9}$$

Because $\vec{x}$ can only have the values 1 and 0, we have that $\vec{y} = \vec{x} - b(1 - \vec{x})$ can also only take on only two values. If $x_i = 1$ $(i \in A)$ then $y_i = 1$. If $x_i = 0$ $(i \in B)$ then $y_i = -b$. Also note that $\vec{y}^T W$ multiplies the $i^{th}$ column of $W$ by $y_i$. Because column $i$ contains the weights of all the edges going into a vertex $i$, we have that $\vec{y}^T W$ multiplies

weights of edges going into vertices in $A$ by 1, and weights of edges going into vertices in $B$ by $-b$. Likewise, $W\vec{y}$ multiplies weights of edges going out of vertices in $A$ by 1, and weights of edges going out of vertices in $B$ by $-b$. The expression $\vec{y}^T W\vec{y}$ is just a number that counts all of the weights in the graph exactly twice and weights each by a factor. We summarize the factors by which $\vec{y}^T W\vec{y}$ multiplies each edge weight in Table 3.1.

| FROM \ TO | $A$ | $B$ |
|---|---|---|
| $A$ | 1 | $-b$ |
| $B$ | $-b$ | $b^2$ |

Table 3.1: The factors $\vec{y}^T W\vec{y}$ multiplies the weight of an edge depending on its starting and ending set.

Notice that the sum of all the edge weights from $A$ to $A$ is exactly $\text{assoc}(A, A)$, and likewise sum of weights of edges from $B$ to $B$ is $\text{assoc}(B, B)$. Furthermore, the sum of the edge weights from $A$ to $B$ (and $B$ to $A$) is $\text{cut}(A, B)$. This implies that

$$\vec{y}^T W\vec{y} = \text{assoc}(A, A) + b^2\,\text{assoc}(B, B) - 2b\,\text{cut}(A, B). \tag{3.10}$$

Interpreting $\vec{y}^T D\vec{y}$ can be done in a similar fashion. $D$ is a diagonal matrix, so the expression $\vec{y}^T D\vec{y}$ simply sums the diagonal, weighting each term $d_{i,i}$ by 1 if $i \in A$ and by $-b$ is $i \in B$. Therefore, $\vec{y}^T D\vec{y} = \sum_{i\in A} d_{i,i} + b^2 \sum_{i\in B} d_{i,i}$. Now again, we can interpret this sum according to our earlier terminology. By definition of $d_{i,i}$, we have that $\sum_{i\in A} d_{i,i} = \sum_{i\in A}\sum_{j\in V} w_{\{i,j\}}$ which is the sum of the weights of edges out of $A$ to all of $V$ and exactly $\text{assoc}(A, V)$. The same is true for $B$, and so we have that

$$\vec{y}^T D\vec{y} = \text{assoc}(A, V) + b^2\,\text{assoc}(B, V). \tag{3.11}$$

Also, recall that $b = \frac{\sum_{i \in A} d_{i,i}}{\sum_{i \in B} d_{i,i}}$, and thus implies that

$$b = \frac{\text{assoc}(A, V)}{\text{assoc}(B, V)}. \tag{3.12}$$

We are now ready to prove that $\frac{(D-W)\vec{y}}{D\vec{y}} = \text{Ncut}(A, B)$. Remembering that $\text{assoc}(A, V) = \text{assoc}(A, A) + \text{cut}(A, B)$ (and likewise for B) the following manipulation will show that $\frac{(D-W)\vec{y}}{D\vec{y}} = \text{Ncut}(A, B)$. For ease of reading and writing we shorten the notation so that $(A, V)$ denotes $\text{assoc}(A, V)$, $(B, V) \equiv \text{assoc}(B, V)$, $(A, A) \equiv \text{assoc}(A, A)$, $(B, B) \equiv \text{assoc}(B, B)$, and $(A, B) \equiv \text{cut}(A, B)$.

$$\begin{aligned}
\frac{(D-W)\vec{y}}{D\vec{y}} &= 1 - \frac{\vec{y^T} W \vec{y}}{\vec{y^T} D \vec{y}} \\
&= 1 - \frac{(A, A) + b^2(B, B) - 2b(A, B)}{(A, V) + b^2(B, V)} \\
&= 1 - \frac{((A, V) - (A, B)) + (b^2(B, V) - b^2(A, B)) - 2b(A, B)}{(A, V) + b^2(B, V)} \\
&= 1 - \frac{(A, V) + b^2(B, V)}{(A, V) + b^2(B, V)} - \frac{-(A, B) - b^2(A, B) - 2b(A, B)}{(A, V) + b^2(B, V)} \\
&= \frac{(A, B)(b^2 + 2b + 1)}{(A, V) + b^2(B, V)} \\
&= \frac{(A, B)(1 + b)^2}{(A, V) + b\left(\frac{(A,V)}{(B,V)}\right)(B, V)} \\
&= \frac{(A, B)(1 + b)^2}{(A, V) + b(A, V)} \\
&= \frac{(A, B)(1 + b)^2}{(A, V)(1 + b)} \\
&= \frac{(A, B)}{(A, V)} + \frac{b(A, B)}{(A, V)} \\
&= \frac{(A, B)}{(A, V)} + \frac{(A, V)(A, B)}{(B, V)(A, V)} \\
&= \frac{(A, B)}{(A, V)} + \frac{(A, B)}{(B, V)} = \text{Ncut}(A, B).
\end{aligned} \tag{3.13}$$

If we could solve the eigenvalue equation for a discrete valued vector $\vec{y}$ that takes on only values 1 and $-b$ we would be able to solve this exactly. However, when we solve the eigenvalue equation we minimize over all possible vectors $\vec{y}$. Therefore, the main question that remains is how to turn a continuous valued vector into a binary segmentation. Let $\vec{y}^*$ denote the eigenvector associated with the second smallest eigenvalue of Equation 3.7. One way to find a discrete vector from the continuous $\vec{y}^*$ is to pick a splitting point $R$ and assign vertices $i$ with $y_i^* < R$ to $A$ and vertices $j$ with $y_j^* \geq R$ to $B$. There are many possible ways to pick $R$. One could simply take the median or mean as the splitting point. An approach that yields better results is to consider a number of splitting points, perhaps evenly spaced in the range of $\vec{y}^*$ and pick the one that has the minimum value for the normalized cut. In practice we get good results using either a small or large number of evenly spaced splitting points. Therefore, considering evenly spaced splitting points yields better results and does not take significantly longer than choosing $R$ to be the mean.

A problem that sometimes appears during the splitting process is that an eigenvector does not naturally lend itself to define a binary segmentation. Sometimes, the eigenvector will produce very similar values for the normalized cut for many different splitting points. In practice, such vectors rarely occur and we deal with the problem by throwing out such vectors and not dividing the segment further. It would be problematic if the first partition resulted in such a vector because then we would not be able to find any segmentation at all. The paper does not address this issue aside from briefly presenting a non-recursive solution for which they show no results.

In summary, the normalized cut algorithm can be written as follows:

- Decide on a similarity measure and compute matrices $D$ and $W$.
- Solve the generalized eigenvalue equation from Equation 3.7 for the eigenvector

$\vec{y}^*$ associated with the second smallest eigenvector.

- Find a splitting point of $\vec{y}^*$ such that the normalized cut is minimized and use it to partition the graph.

- Recursively partition each segment until some sufficient measure of similarity is reached inside the segment.

# Chapter 4

# Mean Shift

The mean shift technique builds upon a general machine learning approach called unsupervised clustering. Clustering is a technique used to classify a large amounts of data into different categories. It is unsupervised because we have no information indicating the right answer, for example, for most image segmentation problems there are no pixels marked as being part of a specific object. One of the simplest clustering techniques is called $k$-means. Mean shift and $k$-means both have the same idea of an optimal segmentation. Both assume that each pixel in the image is distributed according to some number of independent probability density functions (usually Guassians) The ideal segmentation is one where each pixel has the highest probability (according to the probability density functions associated with each segment) of belonging to the segment it is in, compared to any other possible segment we could make.

The difference between $k$-means and mean shift is the way in which we find the segmentation. To segment an image using $k$-means we do the following: First initialize the image to $k$ arbitrary segments and compute the mean of all the pixels in the segment, factoring in both spatial and intensity values. Then, for each pixel $p$, find the segment for which the mean is closest in Euclidean distance to $p$, and reassign $p$

to that segment. After all the pixels have been reassigned, restart the algorithm by recomputing the means and finding new assignments for each pixel. Continue until no pixels move segments.

Through this method we can effectively group similar regions. However, there are a few significant shortcomings of the technique. We must determine the number of segments and initialize them before we begin. This is not ideal because every image lends itself to be divided into a different number of segments. The mulitlabel graph cuts approximation problem has similar shortcomings. However, in the graph cut problem, no matter the arbitrary initialization we begin with, we are guaranteed to come within a constant of the the optimal solution; here, the beginning initialization has a significant influence on the final result. Figure 4.1 shows two possible segmentations of the same data points, each computed with $k$-means. The only difference between them is that they start with a different initial segmentations. Yet, the one on top is qualitatively a much better segmentation than the lower one. This is a major problem with the $k$-means algorithm.

The mean shift algorithm performs a $k$-means like clustering method except it overcomes the problem of initialization. Mean shift starts with a mean for each pixel. The means are shifted by taking a weighted average over all the pixels in the image. The weighted average is called a *kernel density estimator.* Essentially, it is a function that, given a point $x$ in the domain, weights all the points in the domain based on their distance to $x$. Typical kernel density estimators are Gaussians. By iteratively shifting the mean based on the kernel, all the pixels get drawn to a number of local points of convergence. We define segments by grouping together pixels whose means converge to similar intensities. The movement of each mean to its point of convergence can be compared to a particle moving through a force field until it reaches an equilibrium state, or can be seen as the points moving toward peaks and valleys in a landscape

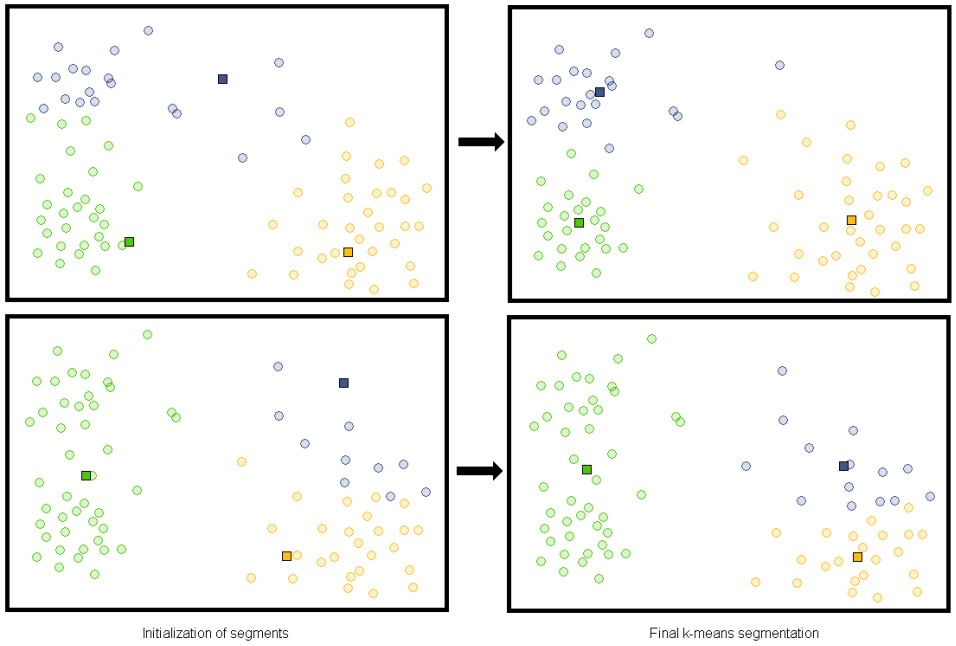



Figure 4.1: All four images show the same collection of data points indicated by the circles. The two images on left show different initial segmentations created by a random placement of the segment means, indicated by the squares. Each circle is then assigned to the square it is closest to and colored appropriately. The color of the circle indicates the segment it belongs to. The images on the right show the segmentations obtained by the $k$-means algorithm using the corresponding initialization of segments. Notice that the two segmentations are significantly different depending on the the initialization. These images are made from an applet illustrating $k$-means [21].

defined by the intensities in the image. This is well illustrated in Figure 4.4. Notice that in this process we have a clustering where the number of segments is determined by the image itself and there is no artificial initialization of segments.

In the pages that follow we will explain the mean shift algorithm in detail. This section is based on the 2002 paper by Comaniciu and Meer [7].

The first step in the mean shift procedure is to move each pixel to its mean. We call this step the *Filtering step*. Consider a $d$-dimensional image with $n$ pixels and $b$ pieces of information at each pixel, where $b$ is 3 for an image with rgb color and 1 for a gray scale image. We begin with a $d+b$ vector $\vec{x}_p$ for each pixel $p$. The vector $\vec{x}_p$ considers both spatial and color information. For each $\vec{x}_p$ we create a sequence of means, $\vec{y}_{j,p}$, beginning at $\vec{y}_{1,p} = \vec{x}_p$. Our final $\vec{y}$ is denoted $\vec{y}_{c,p}$ and is the vector to which our sequence of $\vec{y}_{j,p}$'s converges. At each step we find $\vec{y}_{j+1,p}$ using $\vec{y}_{j,p}$ as follows:

$$\vec{y}_{j+1,p} = \frac{\sum_{q=1}^{n} \vec{x}_q g\left(d_{qp_j}^T H d_{qp_j}\right)}{\sum_{q=1}^{n} g\left(d_{qp_j}^T H d_{qp_j}\right)}, \tag{4.1}$$

where

$$H = \begin{pmatrix} \frac{1}{h_1^2} & 0 & \dots \\ 0 & \ddots & 0 \\ \vdots & 0 & \frac{1}{h_{b+d}^2} \end{pmatrix}, \tag{4.2}$$

$\vec{h}$ is a vector for which element $\vec{h}_i$ gives the relative importance of element $\vec{x}_{p_i}$ in vector $\vec{x}^p$, and $d_{qp_j} = \vec{x}_q - \vec{y}_{j,p}$. The expression $d_{qp_j}^T H d_{qp_j}$ is simply finding $||\vec{x}_q - \vec{y}_{j,p}||^2$, but before we take the norm, we multiply each term in the vector by the respective diagonal element of $H$. The function $g(x)$ is the negative derivative of a kernel function. Here we take our kernel function to be $k(x) = e^{-\frac{1}{2}x}$, and thus $g(x) = -(1/2)e^{-\frac{1}{2}x}$. However, we can factor out the constant $-\frac{1}{2}$ from both the top and the

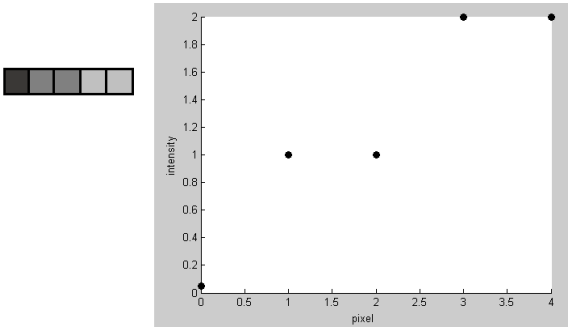



Figure 4.2: To the left is an enlarged $1 \times 5$ pixel grayscale image. To the right is a graph plotting possible intensity values of the $1 \times 5$ image.

bottom of Equation 4.1 and take $g(x)$ to be:

$$g(x) = e^{-\frac{1}{2}x}. \tag{4.3}$$

Equation 4.1 is essentially a weighted mean of the points of the image centered at $\vec{y}_{j,p}$. It is easier to see how Equation 4.1 is changing $\vec{y}_{j,p}$ with an example. Suppose we have a gray scale image that is $1 \times 5$ pixels. The pixels 0 to 4 take on the following intensity values: 0.05,1,1,2,2 as shown in Figure 4.2. Therefore, we have five vectors each of length two, with the following values: $\vec{x}_0 = (0, 0.05)$, $\vec{x}_1 = (1, 1)$, $\vec{x}_2 = (2, 1)$, $\vec{x}_3 = (3, 2)$, $\vec{x}_4 = (4, 2)$.

For simplicity, we will take $\vec{h}$ to be $(1, 1)$. Because the $\vec{h}$ is uniform, the spatial and intensity components are equally weighted. We can think of $\vec{h}$ as weighting the points on Figure 4.1 based on which concentric circle centered around $\vec{y}_{j,p}$ they fall into. The closer the circle, the larger the weight. If the components of $\vec{h}$ are different, then we use an ellipse rather than a circle. The choice of magnitude of the $\vec{h}$ vector effects the size of each concentric circle. In one dimension, $\vec{h}$ is the variance, determining how wide the Gaussian curve stretches. As $\vec{h}$ gets big, points father away have a bigger influence. Later figures will show the effect of changing the parameter $\vec{h}$. To continue

with our example, if we are trying to compute the sequence for $\vec{y}_{j,0}$ (the sequence for pixel 0) then we start with $\vec{y}_{1,0} = \vec{x}_0 = (0, 0.05)$, and we compute $\vec{y}_{2,0}$ as follows:

$$\begin{aligned}
\vec{y}_{2,0} &= \frac{\sum_{q=0}^{4} \vec{x}_q g\left((0 - x_{q_0}, 0.05 - x_{q_1})\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}\begin{pmatrix} 0-x_{q_0} \\ 0.05-x_{q_1} \end{pmatrix}\right)}{\sum_{q=0}^{4} g\left((0 - x_{q_0}, 0.05 - x_{q_1})\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}\begin{pmatrix} 0-x_{q_0} \\ 0.05-x_{q_1} \end{pmatrix}\right)} \\
&= \frac{(0, 0.05)e^0 + (1, 1)e^{-1.9025/2} + (2, 1)e^{-4.9025/2} + (3, 2)e^{-12.8025/2} + (4, 2)e^{-19.8025/2}}{e^0 + e^{-1.9025/2} + e^{-4.9025/2} + e^{-12.8025/2} + e^{-19.8025/2}} \\
&= \frac{(0, 0.05) + 0.3863(1, 1) + 0.0862(2, 1) + 0.0017(3, 2) + 0.0001(4, 2)}{1 + 0.3863 + 0.0862 + 0.0017 + 0.0001} \\
&= \frac{(0.5642, 0.5261)}{1.4743} \\
&= (0.3827, 0.3568)
\end{aligned} \tag{4.4}$$

The above equation shows that the points that are farther away both spatially and in color are weighted less. The process of filtering is exactly a Gaussian smoothing that preserves discontinuities. Instead of blurring pixels at a boundary, if there is a sharp color contrast, the pixels on one side of the boundary have very little influence on the mean of the pixels on the other side because even they are close spatially, their "color distance" is very large. To finish the $1 \times 5$ image example, we find the sequence of $\vec{y}$'s for all five pixels. The sequences are shown in Figure 4.3 for many different values of $\vec{h}$. Observe that if $\vec{h}$ is big all the pixels converge to one point, and if $\vec{h}$ is small then all the pixels converge to a mean very close to their starting location.

The final part of the filtering step is to create a new vector $\vec{z}_p$ for each pixel where the spatial component of $\vec{z_p}$ is the spatial component of the original vector $\vec{x}_p$ and the range (color or intensity) component is the range component of $\vec{y}_{c,p}$. One potential concern is that the $\vec{y}$'s will not converge and $\vec{y}_{c,p}$ will not exist. However, Comaniciu and Meer [7] prove that not only does the sequence of $\vec{y}$'s converge, but also that the sequence moves in a smooth trajectory.

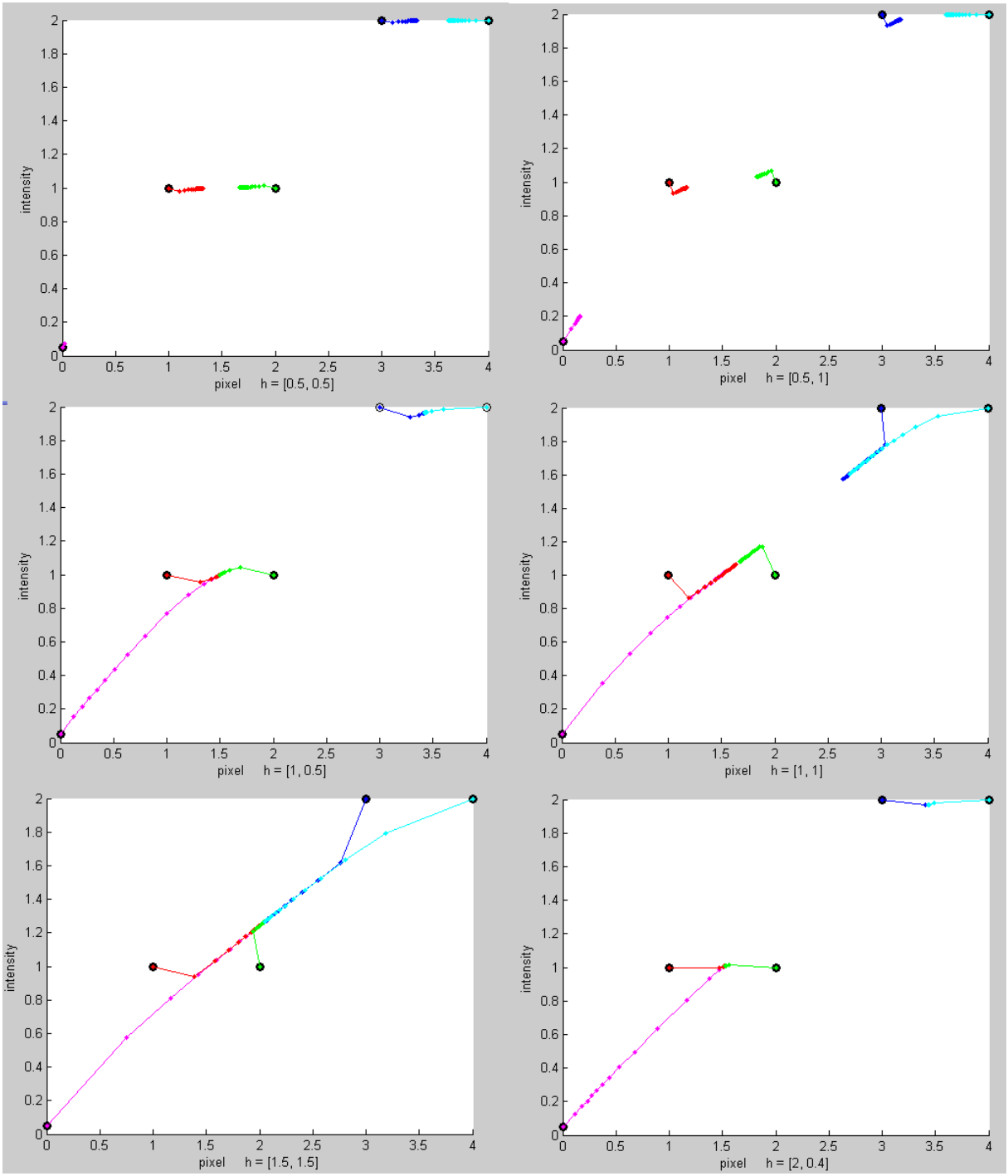



Figure 4.3: The sequence of vectors $\vec{y}_{j,p}$ for each pixel $p$ in the $1 \times 5$ image where each pixel is represented by one of the black dots plotted at its intensity. We show the sequence of $\vec{y}_{j,p}'s$ for six different values of $\vec{h}$.

The last step in the mean shift procedure, referred to as the *Segmentation step* uses the vectors $\vec{z}_p$ to delineate segments. To do this we compare the vectors $\vec{z}_p$ pairwise. For every pair of pixels $p$ and $q$ we place $p$ and $q$ in the same segment if the difference between $\vec{z}_p$ and $\vec{z}_q$ is less than $\vec{h}$, i.e. $\vec{z}_p - \vec{z}_q < \vec{h}$. Essentially, we need the spatial component of the two pixels to be less than the spatial component of $\vec{h}$, and we need the pixels to shift to means that have intensity differences less than the range component of $\vec{h}$. Also, to avoid segments with only a small number of pixels, after we are done grouping segments, pixels that are assigned to segments with fewer than $M$ pixels are places in a neighboring segment. The value of $M$ is a parameter that the user can specify; typically $M$ ranges from 20 to 100.

Figure 4.4 illustrates each step of the mean shift process for a two dimensional gray scale image. We can see the effect of the segmentation step in Figure 4.4 (e)-(h). In (e) and (f) we show the image after the filtering step, and in (g) and (h) we show the final segmentation. Notice that bumpy regions in the topology of image (f) are smoothed over to create clearly defined plateaus in (h), and correspondingly, image (e) has many variations in intensity, while the image (f) has just a few intensity values, each of which represent a clearly defined segment.

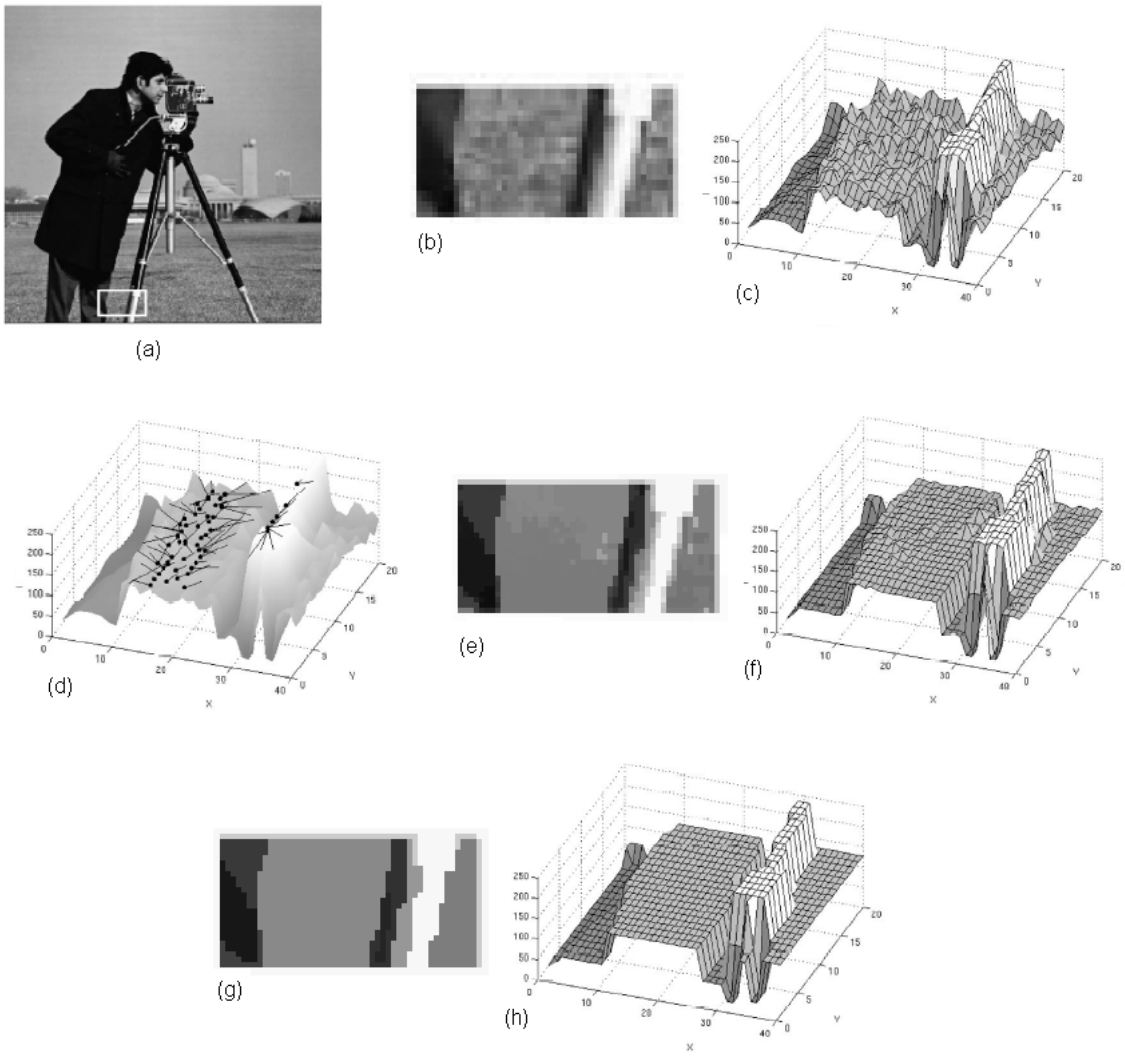



Figure 4.4: An illustration of each step in the mean shift process for a 2-D gray scale image. **(a)** An image of a camera man. **(b)** A zoomed-in view of the section in the white box of image (a). **(c)** A plot of image (b) with the the pixel grid on the x-y plane and the gray values on the z-axis. **(d)** A plot of image (b) with lines showing the path of the means for a selection of pixels ending at a black dot which is the point of convergence of the mean. **(e)** The image (b) after the filtering step. The intensity component of each pixel has been replaced with the intensity of the mean to which the pixel converged. **(f)** A plot of image (e), which we can also think of as a plot of the vectors $z_p$. **(g)** The final segmentation of the image (b), where pixels in the same segment all have the same intensity. **(h)** A plot of image (g). This Figure was created using [7] and the mean shift code [10].

# Chapter 5

# Results

In this section we show segmentation results for each of the three methods discussed in Chapters 2-4 on the same images. To generate the segmentations for normalized cuts and mean shift methods we use code produced by the authors of the methods that is posted on their websites [17], [10]. The mean shift code is implemented in C++, and the normalized cut code is implemented in Matlab. The normalized cut code can only handle images that have less than about 20,000 pixels. To deal with this, the normalized cut implementation downsamples the input images before it computes the segmentation. Also, the normalized cut code converts the images to grayscale before finding a segmentation. The mean shift code is implemented in C++. For the graph cuts method the authors have not made their code available, but in [20] we found results for the interactive binary segmentation version of graph cuts on three images that we call the "flower," "person," and "sponge" images. To compare the three methods we computed segmentations of these three images using the normalized cuts and mean shift code.

The results for the three images are shown on the following three pages in Figures 5.1, 5.2, and 5.3. Each figure shows: **(a)** the input image; **(b)** the input image

marked with seed pixels used for graph cuts (black for background, white for object); **(c)** the downsampled, grayscale image used for normalized cuts input; the images are downsampled to be 160x120 pixels; **(d)** the segmentation produced by graph cuts showing the foreground segment in its original colors and the background segment in black; **(e)** the segmentation produced by graph cuts showing the foreground segment in black and the background segment in its original colors; **(f)** the segmentation produced by normalized cuts with the edges of the segments drawn in red; **(g)** a segmentation produced by mean shift with $h = [30, 30]$ and $M = 20$ with the edges of the segments are drawn in red; **(h)** a segmentation produced by mean shift with $h = [7, 6.5]$ and $M = 20$.

Looking at the results it is easy to conclude that graph cuts is by far the best algorithm because it produces the best segmentation. However, graph cuts has a distinct advantage because it uses user input. Considering this, some of the mistakes it makes are surprising. Looking at the flower image it is odd that the corner of the petal is included in the background instead of the object. Likely this is an issue of balancing the regional and boundary terms.

The normalized cuts and mean shift methods produce only mediocre segmentations, but they are solving a much harder problem then the graph cuts method. For normalized cuts, often the segments are too equal in size. The method uses a cut that is normalized to avoid large differences in the size of the segments, but the result is that segments seem to be forced to be very similar sizes. This is especially apparent in the flower image, where instead of segmenting the white flowers, or the piece of the yellow flower, the segmentation divides the region into two relatively equal parts along a seemingly arbitrary line. Also, it seems that often the recursion is not stopped appropriately. For example in the sponge image it would have been better to stop after the first partition.

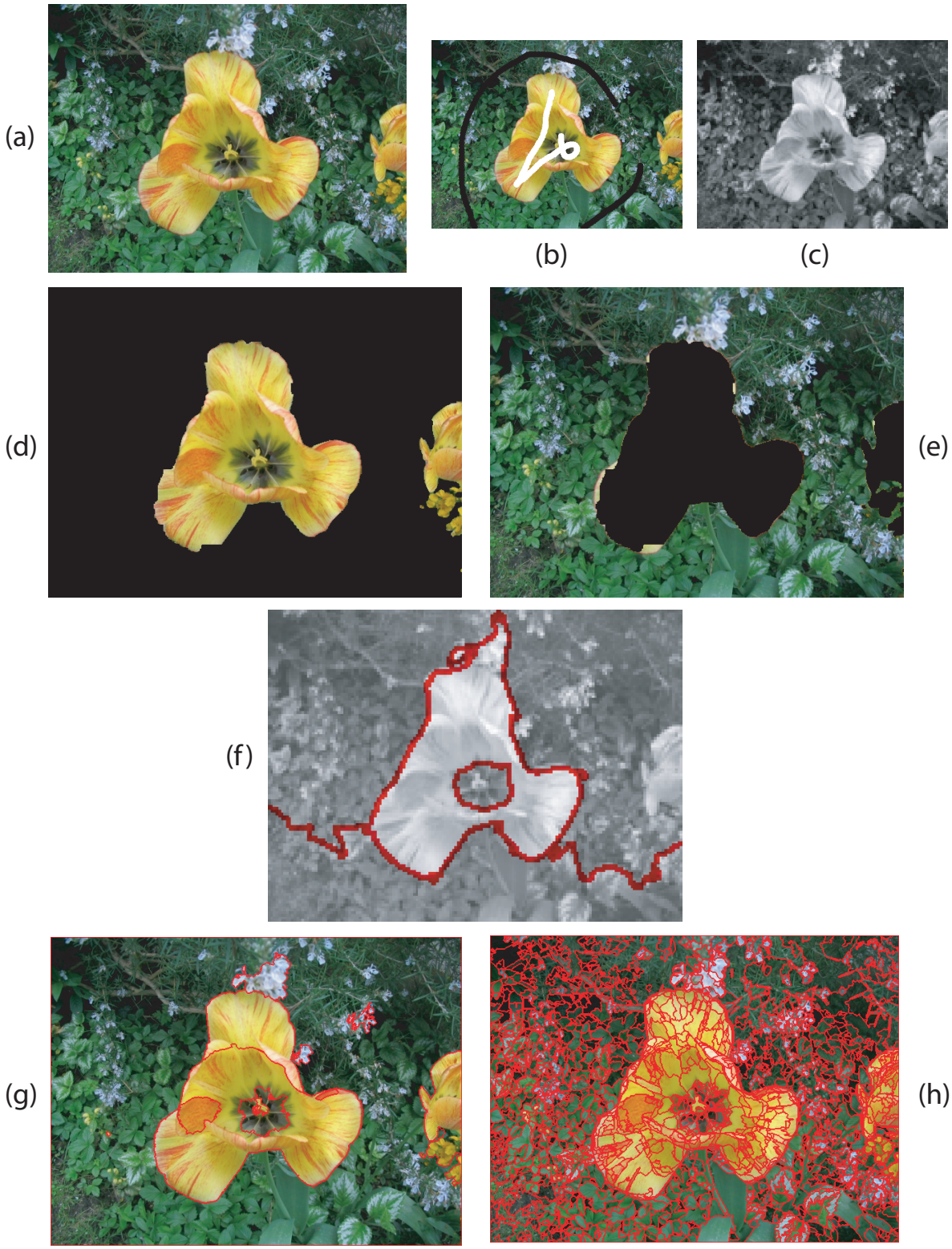



Figure 5.1: The flower image: 600x450 pixels. **(a)-(c)** Input images. **(d)-(e)** Graph cut segmentations. **(f)** Normalized cut segmentation. **(g)-(h)** Mean shift segmentations.

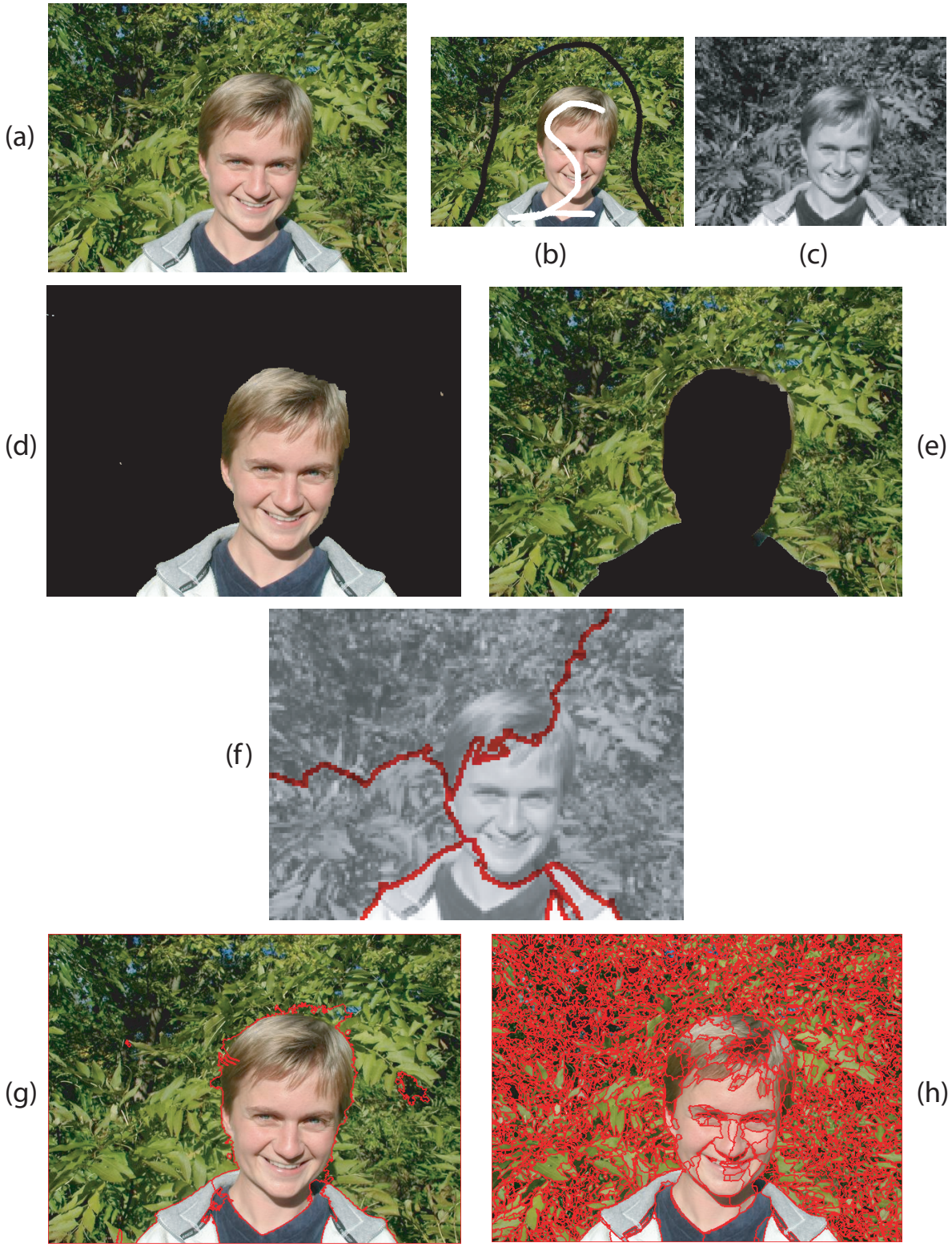



Figure 5.2: The person image: 600x450 pixels. **(a)-(c)** Input images. **(d)-(e)** Graph cut segmentations. **(f)** Normalized cut segmentation. **(g)-(h)** Mean shift segmentations.

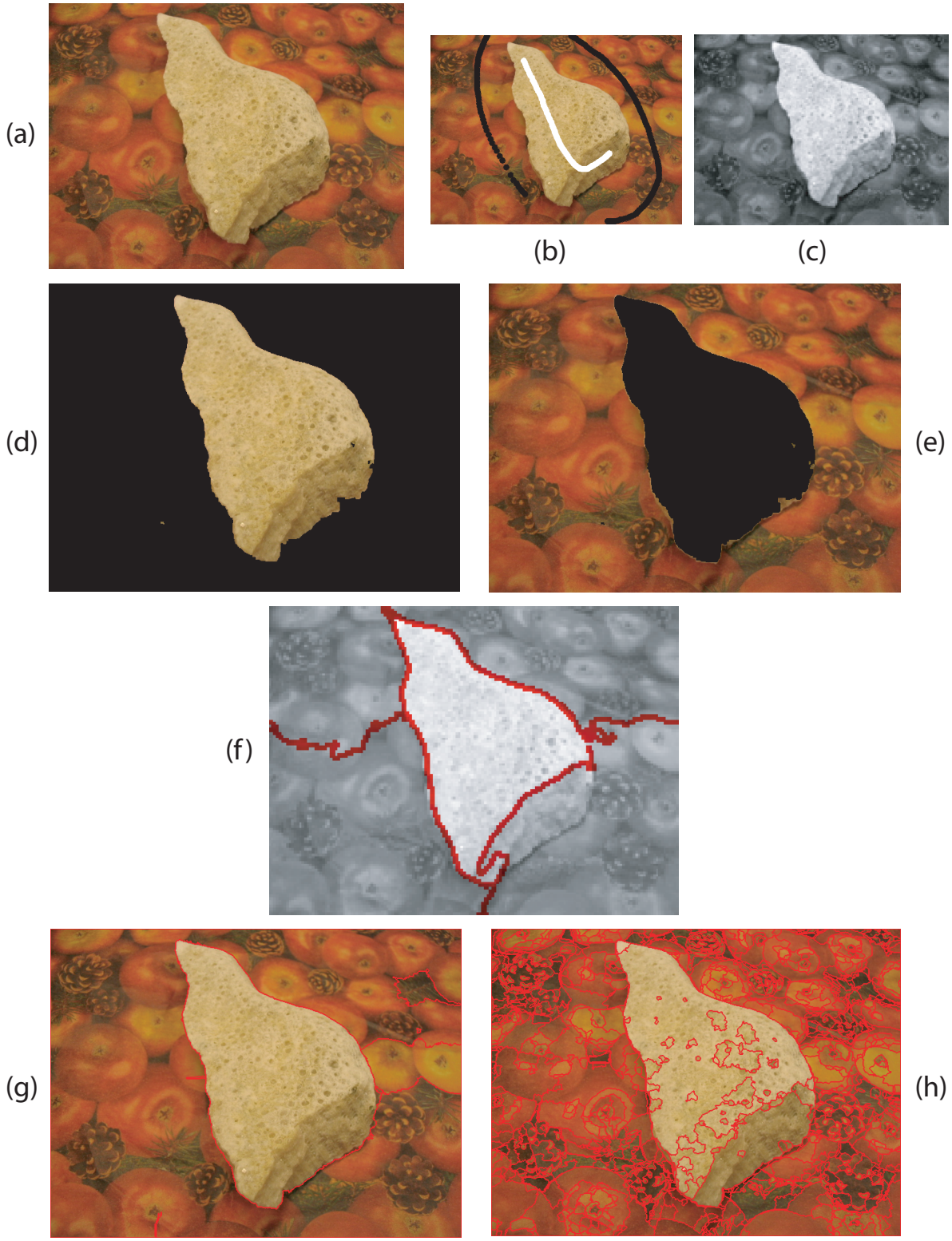



Figure 5.3: The sponge image: 640x480 pixels. **(a)-(c)** Input images. **(d)-(e)** Graph cut segmentations. **(f)** Normalized cut segmentation. **(g)-(h)** Mean shift segmentations.

| Method | Flower | Person | Sponge |
|---|---|---|---|
| Graph Cuts | 0.188 | 0.140 | 0.142 |
| Normalized Cuts | 11.290 | 10.633 | 9.987 |
| Mean Shift $h = [7, 6.5]; M = 20$ | 5.56 | 5.92 | 8.1 |
| Mean Shift $h = [30, 30]; M = 20$ | 77.14 | 77.52 | 86.75 |

Table 5.1: A comparison of the runtime for the three methods.

For mean shift, the segmentations that are generated with small $h$ values are difficult to analyze only by visual examination. In general the components seem to capture the boundaries well, which is the most important part. But often there are two segments where one would be better, or vice versa. Without applying the segmentation to be used for a specific application, for example stereo vision, or comparing it to another method that produces an "over-segmentation" it is hard to evaluate it. It is much easier to asses the results of the segmentation using large $h$ values. The flaw that stands out the most in the segmentations produced by mean shift is that often small, unimportant, and seemingly random segments are part of the final segmentation.

Table 5.1 shows the runtime for each of the three methods for each image. This table should be analyzed with hesitation because the timing methods are not always accurate and vary each time the segmentation is computed. Also, the comparison is not completely fair. As noted earlier, the normalized cut method is implemented in Matlab, and often Matlab is slightly slower than C++. On the other hand, the normalized cut method is using a much smaller image size which makes its runtime on each image significantly smaller than if the original image was used. Additionally, the graph cuts method should run significantly faster than the other two methods because the input to the graph cuts method includes seed pixels which considerably decreases the space of possible segmentations. Although, one of the main things to notice from the table is how fast the graph cuts method is in comparison to the others.

| Method | Lily Pad | Trumpet |
|---|---|---|
| Normalized Cuts | 10.1 | 10.9 |
| Mean Shift | 1.53 | 1.37 |

Table 5.2: A comparison of the runtime between the normalized cut and mean shift methods on small images.

In order to obtain a more fair runtime comparison of mean shift and normalized cuts, we tested the two methods on very small, grayscale images that the normalized cut code does not need to modify. Table 5.2 shows the runtimes for the segmentation results in Figures 5.4 and 5.5. In this experiment both methods found a segmentation on an image with the same number of pixels. We see that even when $h$ is large, mean shift is significantly faster than normalized cuts.

In terms of a comparison of the segmentations produced, again, it is hard to say which method is better. Mean shift picks out the water better in the lily pad image, but normalized cuts picks out the face of the man better in the trumpet image. One other thing to consider is that it is possible to adjust the parameters and get much better results for the mean shift. We chose to use $h = [20, 20]$ because it gave decent results for both the lily and trumpet images. It is also possible that a parameter adjustment would yield better results for normalized cuts, but the code for normalized cuts was not as conducive to making such changes.


Figure 5.4: The lily pad image: 147x135 pixels. **(a)** The input image. **(b)** The segmentation obtained by normalized cuts. **(c)** The segmentation obtained by mean shift using $h = [20, 20]$ and $M = 20$.

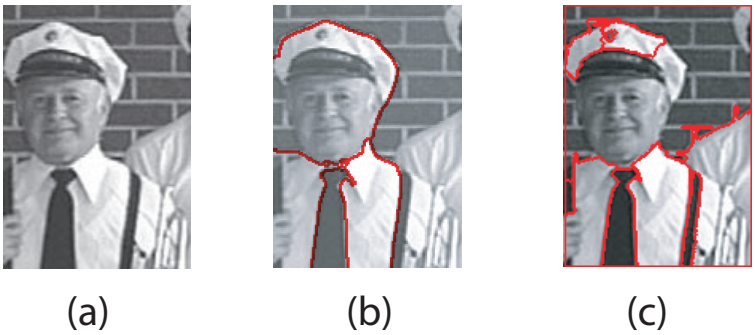



Figure 5.5: The trumpet image: 118x166 pixels. **(a)** The input image. **(b)** The segmentation obtained by normalized cuts. **(c)** The segmentation obtained by mean shift using $h = [20, 20]$ and $M = 20$.

# Chapter 6

# Conclusion

The three image segmentation methods presented in this thesis are the primary methods used in practice which indicates that in general they give good results. It is hard to make any sort of quantitative comparison between the methods because for any image there are an infinite number of segmentations that could be considered correct. Therefore we have no "correct" answer to compare to the results, and we cannot attempt to count the pixels that are mislabeled, as we can for stereovision. As we did in the Results chapter we can try to make a quantitative comparison and remark that one method performs better on a given image than another. However, even this is hard because all three methods have different applications and achieve very different segmentations.

The graph cuts method is good for interactive segmentation, for example in an image editing application like Photoshop. It is also good for medical and scientific imaging where we are looking to segment very specific items from many images. For example, graph cuts was implemented (but never used) for a project in the Middlebury geology department that looked at noisy soil samples. They wanted to segment the air, soil, and water in images that showed a slice of a sample of soil.

The normalized cut method and mean shift method are both better suited for automatic segmentation. Both can achieve an over-segmentation of the image into "super-pixels" or a segmentation that divides the image into only a few main regions, hopefully picking out the main objects from the scene. However, mean shift is the fastest when it finds an over-segmentation because the $\vec{h}$ values used are smaller and we only need to look at small area around a pixel in the segmentation step. Additionally, often to use mean shift to find a segmentation that delineates objects it is necessary to try a number of different values for $\vec{h}$. On the other hand, normalized cuts is the fastest when we divide the image into only a few components because normalized cuts uses recursion, so the more components, the more times we must perform the normalized cut procedure.

We can also compare the methods in terms of their similarities and differences from a theoretical standpoint. We already noted the differences by pointing out particular advantages and disadvantages of each method in their respective chapters. Therefore, here we will bring all three methods together and discuss some of the similarities between the methods. Many of the similarities highlight some of the biggest shortcomings of all the methods and what needs to be done to improve image segmentation algorithms.

The most striking similarity between the methods is that they all use the Gaussian distribution in some way to model segments. This is especially interesting considering that none of the methods explicitly state the use of a Gaussian in their models of an optimal segmentation. Graph cuts uses a Gaussian to determine the cost between neighboring pixels with a different labeling. Normalized cuts uses a Gaussian to compute the similarity measure between pixels, and mean shift uses a Gaussian kernel to "shift" the mean of each pixel. Because all these methods work relatively well, it seems that the Gaussian is a good way to model segments. It gives a probability

function that two pixels should belong to the same segment by incorporating the Euclidean distance in both the spatial and color domain as well as accounting for a variance, which can be thought of as image noise. The Gaussian model is a good choice from a probabilistic standpoint because the Central Limit Theorem tells us that the distribution of many identically distributed independent random variables is a Gaussian distribution, no matter the probability distribution of the random variables. In image segmentation, if we view every pixel to be an independent random variable, then a Gaussian is the appropriate model for each group of pixels the the same probability distribution. However, it is doubtful that pixels in an image are independent random variables. Often the assumption is justified with the idea that image noise usually does behave like a Gaussian. However, most objects do not have intensities that can be well modeled with a Gaussian. Additionally, when used in the spatial domain, the Gaussian model encourages elliptically shaped regions. Despite the problems with the Gaussian function, it seems that for now it is the best we can do. To make a model of similarity that is better than a Gaussian, it is likely that we would need a highly complex, computationally intensive model that is unlikely to have practical value.

Another similarity is that all three methods are highly sensitive to a few parameters. In graph cuts we can adjust the constant $\lambda$ and also the standard deviation for the boundary term. In normalized cuts we can adjust the standard deviation for the similarity measure, and in mean shift we can adjust the vector $\vec{h}$, which is essentially the standard deviation in the Gaussian kernel. The ability to adjust parameters to change results is sometimes desirable because we can obtain many segmentations from the same method. However, the problem is that the same parameters do not give us the same results across many images. It seems impossible to find a set of parameters that produce "good" segmentations for any image. Both the mean shift [7]

and normalized cut [18] papers specify vastly different parameters for all the results shown in their papers. (Neither of the graph cut segmentation papers [2] [1] give the parameters that they used to generate their results.) The sensitivity to parameters means that the methods are not robust and cannot always achieve good results. In many ways it makes the methods not fully automatic because a human is used to pick the best parameters.

Finally, all three methods use simple models. This needs to be the case in order to have the algorithms run in a reasonable amount of time, however, the models need to be more complex to be able to segment images as well as humans. It is clear that the human visual system uses much more complex models and makes many more assumptions about how it expects an image to look. Also, it is unlikely that humans ever specifically segment an image, rather it is more likely that humans simultaneously perform many vision tasks. One example that gives a little insight into how the human brain does this is the Kenisza triangle shown in Figure 6.1. Here we, as humans, see two clearly defined triangles even though there is no intensity difference along many edges of the triangle. The three methods described in this thesis are not designed to find the triangle in this image. Even though this image may seem artificial, real-world images often have areas that are partly occluded or objects that blend into the background, and it is necessary to have models that will produce correct segmentations in these cases.

Another more difficult example is shown in Figure 6.2. Here, when we look at this image we see two possible interpretations: a few black lines on a white background, or the letter 'E'. In order to infer the letter 'E' we not only need to have some model of occluded regions and be able to fill them in, but we also need to have a model for the letter 'E.' This image also reinforces the idea that image segmentation is an ambiguous problem. Depending on which part of the image we focus on we see

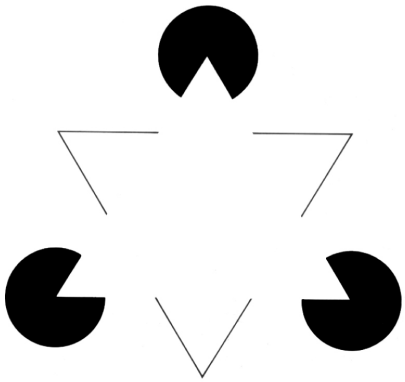


Figure 6.1: The Kenisza triangle: A human sees two triangles in this image even though neither triangle is defined completely by intensity differences.

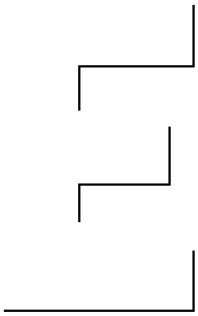


Figure 6.2: An optical illusion: A human interprets this as either black lines on a white background or an 'E.' None of the three segmentation methods are designed to find the 'E.'

different segmentations.

Although it would not be too difficult to design an algorithm that can find a segmentation that includes triangles the in Figure 6.1 and the letter 'E' in Figure 6.2, we are a long way from finding an image segmentation that applies similar notions of how to connect lines and group objects in real-world images. There are many theories of how humans perceive and group objects formulated by the Gestalt school of psychology [9]. If we can find a way to translate some of these ideas into algorithms, this would be one way to tackle the problem. Another possibility is to try to have

computers learn more complex models by training on thousands of images segmented by humans. But all the while, we must consider that it is hard to create a complex image model that is computationally efficient. Even as computing power increases, we still need efficient algorithms because the size of images will increase as well.

In all, no matter the similarities and differences, advantages and disadvantages of graph cuts, normalized cuts, and mean shift, all three segmentation methods are just small steps toward the end goal of making a computer understand images.

# Bibliography

[1] Boykov, Y. and Funka-Lea, G. Graph cuts and efficient N-D image segmentation. *International Journal of Computer Vision*, 70(2):109-131, 2006.

[2] Boykov, Y. and Jolly, M. Interactive graph cuts for optimal boundary and region segmentation of objects in N-D images. In *International Conference on Computer Vision*, vol. 1:105, July 2001.

[3] Boykov,Y. and Kolmogorov, V. Computing geodesics and minimal surfaces via graph cuts. In *International Conference on Computer Vision*, vol. I:26-33, 2003.

[4] Boykov,Y. and Kolmogorov, V. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(9):1124-1137, 2004

[5] Boykov, Y., Veksler, O. and Zabih, R. Markov random fields with efficient approximations. In *IEEE Conference on Computer Vision and Pattern Recognition*, 648-655, 1998.

[6] Boykov, Y., Veksler, O. and Zabih, R. Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence,* 23(11):1222-1239, 2001.

[7] Comaniciu, D. and Meer, P. Mean shift: A robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(5), 2002.

[8] Cook, W. J., Cunningham, W. H., Pulleybank, W. R., and Schrijver, A. *Combinatorial Optimization.* NY, John Wiley & Sons, Inc., 1998.

[9] Forsyth, D. and Ponce, J. *Computer Vision: A Modern Approach.* NJ, Pearson Education, 2003.

[10] Georgescu, B. and Christoudias, C. *Code Available to Download*, Online, Internet. Rutgers University: Robust Image Understanding Laboratory. 15 May 2007 ⟨http://www.caip.rutgers.edu/riul/research/code.html/⟩.

[11] Huang, H., Wang H, Wang, X. *Max-Flow Min-Cut*, 2006, Online. Internet. York Univeristy. 15 May 2007. ⟨http://www.cse.yorku.ca/~aaw/Wang/MaxFlowStart.htm/⟩.

[12] Kolmogorov, V. and Zabih, R. What energy functions can be minimized via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(2):147-159, 2004.

[13] Li, S. *Markov Random Field Modeling in Image Analysis.* Springer-Verlag New York, Inc., Secaucus, NJ, 2001.

[14] Mitchell, T. *Machine Learning.* Singapore: The McGraw-Hill Companies, Inc., 1997.

[15] Russell, S. and Norvig, P. *Artificial Intelligence: A Modern Approach.* NJ, Pearson Education, 2003.

[16] Shapiro, L. and Stockhman, G. *Computer Vision.* NJ, Prentice Hall, 2001.

[17] Shi, J. *MATLAB Normalized Cuts Segmentation Code*, Online. Internet. University of Pennsylvania. 15 May 2007 ⟨http://www.cis.upenn.edu/ jshi/software/⟩.

[18] Shi, J. and Malik, J. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888-905,2000.

[19] Sonka, M., Hlavac, V., Boyle, R. *Image processing, analysis, and machine vision.* California: Brooks/Cole Publishing Company, 1999.

[20] R. Szeliski, R. Zabih, D. Scharstein, O. Veksler, V. Kolmogorov, A. Agarwala, M. Tappen, and C. Rother. A comparative study of energy minimization methods for markov random fields. In *Ninth European Conference on Computer Vision* (ECCV 2006), vol. 2: 19-26, Graz, Austria, May 2006.

[21] *A Tutorial on Clustering Algorithms: K-means - Interactive demo*, Online, Internet. 15 May 2007. ⟨http://www.elet.polimi.it/upload/matteucc/Clustering/tutorial_html/AppletKM.html/⟩.

[22] Zitnick, L. and Kang, S. *Stereo for Image-Based Rendering using Image Over-Segmentation. International Journal of Computer Vision* 2007, to appear.