Image Segmentation by Nested Cuts

Olga Veksler
NEC Research Institute, 4 Independence Way
Princeton, NJ 08540
olga@research.nj.nec.com

Abstract

We present a new image segmentation algorithm based on graph cuts. Our main tool is separation of each pixel p from a special point outside the image by a cut of a minimum cost. Such a cut creates a group of pixels C_p around each pixel. We show that these groups C_p are either disjoint or nested in each other, and so they give a natural segmentation of the image. In addition this property allows an efficient implementation of the algorithm because for most pixels p the computation of C_p is not performed on the whole graph. We inspect all C_p's and discard those which are not interesting, for example if they are too small. This procedure automatically groups small components together or merges them into nearby large clusters. Effectively our segmentation is performed by extracting significant non-intersecting closed contours. We present interesting segmentation results on real and artificial images.

1 Introduction

A popular framework for image segmentation (or data clustering) is graph partitioning. A weighted undirected graph $G = (V, E)$ is constructed, where V is the set of image pixels and edges E connect neighboring pixels according to some prescribed neighborhood system. The weight w on an edge $e = \{p, q\}$ measures the similarity between pixels p and q. Usually w in an increasing function of similarity. The goal of graph partitioning is to break V into a few disjoint sets $V_1, \ldots, V_k$ s.t. the similarity across pixels in $V_i \neq V_j$ is small.

A lot of prior work performs partitioning based on purely local properties ([5]). While methods based only on local properties are in general very efficient, they often fail to capture the important global properties of a scene (see [8] for a discussion). In recent years, graph cuts have emerged as a powerful optimization technique which allows extraction of global information ([10], [8], [3], [7], [4], [1]). We propose a new segmentation method based on graph cuts. We begin by reviewing graph cuts and segmentation methods based on graph cuts.

1.1 Graph Cuts

Suppose A and B are subsets of V s.t. $V = A \cup B$ and $A \cap B = \emptyset$. Then the (A, B)-cut is the subset of edges which connect A and B. If C is the (A, B)-cut then its cost is just the sum of its edge weights:

$$c(C) = \sum_{p \in A, q \in B} w(p, q).$$

A minimum cut is a cut of the minimum cost. It can be found efficiently, for example see [2]. In general, there may be several minimum cuts.

In a rooted variant of the minimum cut problem we are given two distinct nodes s and t which we call terminals. Here we want to find the minimum (A, B)-cut under the restriction that $s \in A$ and $t \in B$. We use notation (s, t)-cut to denote a cut under this restriction. If C is an (s, t)-cut then it splits vertices into two sets which we denote by C_s and C_t with $s \in C_s$ and $t \in C_t$.

1.2 Segmentation by cuts

Z. Wu and R. Leahy [10] proposed an algorithm which optimally partitions the graph into k subgraphs such that the maximum inter-subgraph cut is minimized. This solution minimizes the similarity across different subgraphs. The algorithm works recursively by splitting a segment in two parts by a minimum cut, until the whole graph is partitioned into k parts. To avoid cutting out a single pixel which is well connected to its neighbors the edge weights should decrease fast. That is severing just a few edges between pixels with similar intensities should be more expensive than severing many edges between pixels with different intensities. However even with such weight assignments, this approach prefers cutting out small isolated clusters of the graph and also the choice of the right k is difficult.

Figure 1.

(a) Original image

(b) Best contour normalized by length

Figure 2.

To avoid the problem of small clusters, J. Shi and J. Malik [8] propose to normalize the cost of an (A, B)-cut as follows:

$$Nc(C) = \frac{c(C)}{\sum_{p\in A, q\in V} w(p,q)} + \frac{c(C)}{\sum_{p\in B, q\in V} w(p,q)}. \quad (1)$$

In addition to minimizing the similarity between A and B, the normalized cut maximizes the normalized association, i.e. the similarity within each group. The problem of minimizing the normalized cut is NP-hard, and in [8] they find an approximate solution based on the generalized eigenvalue problem (for an interesting discussion of segmentation algorithms based on eigenvectors see [9]).

Normalized cut algorithm has been very successful and has been applied to many types of grouping problems. However there are cases when it has difficulties. Perona and Freeman [6] provide an example. They consider segmentation of a structured foreground from an unstructured background, see figure 4(a) for an illustration. The foreground pixels have large similarity to each other, but the background pixels are dissimilar. Bipartitioning into the foreground and the background fails either due to a bad approximation or the fact that normalized cut seeks to partition the image into two groups where pixels within each group are similar.

While removing the bias to small segments, normalized cut is biased towards splitting the image into two parts of equal weight. An extreme example is in figure 1. The background is 50 by 100 with intensity 120. The foreground consists of a 10 by 10 square with intensity 150. On the left side of the square there is a narrow ramp between foreground and the background, s.t. there is a gap 5 pixel wide in the edge between the square and the background. Assume that each pixel is connected to its 4 nearest neighbors. The normalized cost of the cut which separates the square from the background has cost at least $\frac{5w}{4\cdot 10\cdot 10\cdot w} = \frac{1}{80}$ where w is the cost of an edge between pixels of the same intensity (here we just considered the cost of the highest of two terms in equation (1)). However the normalized cost of a cut which splits the image vertically in two equal pieces is approximately $\frac{50w}{4\cdot 50\cdot 50\cdot w} + \frac{50w}{4\cdot 50\cdot 50\cdot w} = \frac{1}{100}$. Thus regardless of the weights, the cut which severs only high cost edges has cost smaller than the cost of a cut which severs just 5 high cost edges.

One can think about other ways to normalize the cut, for example by the total number of edges in the cut. Intuitively this corresponds to finding the highest contrast edge. First of all this approach still creates many small clusters. Second consider figure 2(a). This figure has 3 rectangles of different size stacked on top of each other The bottom and the top rectangles are shaded. Due to the shading, the inner and outer contours around the middle rectangle decrease in contrast from left to right. For many weight choices, the optimal cut normalized by number of edges is shown in figure 2(b). It consists of the brightest parts of the contours around the middle rectangle and cuts through the inside of the middle rectangle. The cost of including a few heavy edges is averaged out, and as a result the best contour found is a combination of the two distinct contours perceived by a human observer.

An interesting approach not based on bipartitioning is by Y. Gdalyahu, D. Weinshall and M. Werman [3]. They propose a stochastic segmentation algorithm which is based on k-way cuts, which is a generalization of the two way cut defined before.

1.3 Our approach

We propose a new algorithm based on rooted graph cuts. Out of the three methods discussed above, our algorithm is most similar to the one in [10]. However instead of partitioning the graph optimally into k subgraphs, we use graph cuts to directly search for a closed contour of a small cost around each pixel.

We introduce a new graph node t and connect the pixels on the boundary of the image to t with edges of appropriately chosen small weight. This new addition to the graph structure serves two purposes. First it assigns some low cost to the contour consisting of the image boundary (the cost of this contour is the sum of the weights of edges incident on t). Secondly node t intuitively represents the scene outside the image which is not similar to any of the pixel nodes. Thus to find a low cost contour around p we separate p from the external node t by a minimum (p, t)-cut. This

cut creates a group of pixels C_p containing p. We show that we can find minimum cuts s.t. C_p's are either nested in each other or disjoint. Thus C_p's give a natural partitioning of the graph. In addition this property allows us to implement the algorithm efficiently. We exclude from the segmentation all C_p's which are uninteresting, for example all C_p's which are smaller than some prescribed size. This procedure automatically groups small components together or merges them into larger clusters. Effectively our algorithm performs segmentation by finding significant closed contours which can touch but cannot intersect. We then recursively apply our algorithm to each segment until a certain stopping criterion is reached. This criterion implies that no more interesting contours can be found.

This paper is organized as follows. In section 2 we explain our algorithm, in section 3 we show how to implement it efficiently, and in section 4 we present the experimental results.

2 Nested Cuts

2.1 Graph Structure

We begin by describing the structure of our graph. Let P be the set of all image pixels and let $\mathcal{N} = \{\{p,q\} \mid p,q \in \mathcal{P}\}$ be a prescribed neighborhood system. A common choice for $\mathcal{N}$ is the set of all pixel pairs within some distance from each other. The set of vertices in our graph is $V = \mathcal{P} \cup \{t\}$. The set of edges is

$$E = \mathcal{N} \cup \{\{p,t\} \mid p \text{ is on the image border}\}.$$

The special node t always serves as one of the terminals. It is intuitively interpreted as a node outside the image dissimilar to every pixel node. The weight of edges $\{p,q\} \in \mathcal{N}$ is proportional to the similarity between pixels p and q. The weight of edges $\{t,p\}$ will be discussed later, but in general we will keep it low.

2.1.1 Main Theorems

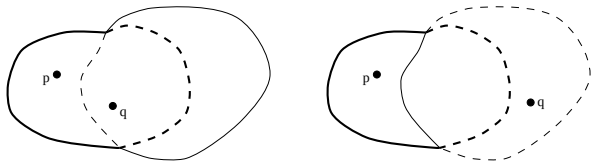
Given a pixel p, let C be a minimum (p,t)-cut. We have the following results:

Theorem 1 *If $q \in C_p$ then there is a minimum (q,t)-cut C' s.t. $C'_q \subset C_p$.*

Theorem 2 *If $q \notin C_p$ then there is a minimum (q,t)-cut C' s.t. either $C'_q \cap C_p = \emptyset$ or $C_p \subset C'_q$.*

We will give intuition about the theorems in a simple situation when cuts form paths.We give the general proofs in the appendix.

Suppose theorem 1 is false. Let C' be any minimum (q,t)-cut. This case is illustrated in figure 3(a). Here



(a) Theorem 1 (b) Theorem 2

Figure 3.

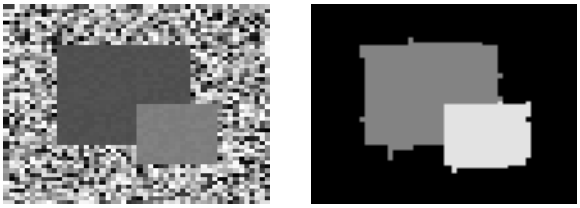
thick solid and dashed edges show cut C and thin solid and dashed edges show cut C'. Cut C' consists of the dashed thin path and the solid thin path. It prefers cutting along the thin solid path instead of the thick dashed path. That means that the thin solid path is cheaper than the thick dashed path. But then cut C should also prefer cutting along the thin solid path instead of the thick dashed path, and we get a contradiction.

Now suppose theorem 2 is false. Let C' be any minimum (q,t) cut. Consider figure 1(b). Again cut C' prefers cutting along the thin solid path instead of the thick dashed path. But that means that cut C should also prefer cutting the thin solid edge, which is again a contradiction.

2.2 Weights on edges $\{p,t\}$

We assign the same edge weight w_t for all edges in $\{\{p,t\} \mid p \text{ is on the image border}\}$. The choice of w_t plays an important role in the algorithm. In general we want to assign w_t a low weight, since each pixel p is not similar to t. Another way of looking at it is that the contour consisting of the border of the image should have a low cost. However, if w_t is too small, then for all p the optimal (p,t)-cut is s.t. $C_p = \mathcal{P}$. Therefore we choose a discrete range $W = \{w_{min}, w_{max}\}$ and we use binary search to find the smallest $w_t \in W$ s.t. $\exists C_p$ with $threshold \leq |C_p| \leq |\mathcal{P}| - threshold$. Here $|S|$ denotes the set size and $threshold$ is the smallest segment size we allow. If there is no such w_t the segmentation is stopped. Otherwise the image is segmented by C_p's and we apply the algorithm recursively to each resulting segment.

Thus w_t controls the maximum cost of cuts we are willing to include in the segmentation. We do not include in the segmentation any cut of cost more than w_t multiplied by the boundary length. So our choice of w_{min} and w_{max} is as follows: w_{min} is just the smallest possible edge weight; w_{max} is the largest possible edge weight s.t. the contour consisting of edges with weights w_{max} would be still considered a good contour to include in the segmentation. Notice now that in many cases the boundary length is longer than the length of contours inside the image(If we expect an image to contain contours longer than the boundary we



(a) Original image (b) Our results

Figure 4.

can create more edges to t from the boundary pixels). The segmentation is stopped when there is no cut cheaper than w_{max} times the boundary length which is a good decision because then most likely there are no good contours left. If $w_t \in W$ with the desired property is found and it is not equal to w_{max}, then after we are done segmenting there maybe still good contours which are not found (i.e. contours consisting of edges with weights in W but larger than current w_t). But then the algorithm proceeds recursively to find those contours.

2.3 Nested Cuts algorithm

0. Create a new graph
1. Use binary search to find smallest $w_t \in W$ s.t. there is a C_p with $threshold \leq |C_p| \leq |\mathcal{P}| - threshold$
2. If w_t at step 1 is not found, exit
3. For each p find C_p. Discard C_p if $|C_p| < threshold$ or $|C_p| > |\mathcal{P}| - threshold$
4. Recursively apply the algorithm to all segments.

At the deeper levels of recursion the costs of cuts get larger. Thus we get a hierarchy of segmentations where the deeper levels of the hierarchy in general contain weaker contours.

In the current implementation the decision on whether or not include C_p in the final segmentation is based on its size. However any other criterion can be used instead.

3 Efficient Implementation

As stated, the algorithm in the previous section would be very inefficient. In this section we discuss three steps we take to implement it efficiently.

3.1 Sampling

Recall that in step 1 of our algorithm for a given w_t we need to test if there is a C_p s.t. $threshold \leq |C_p| \leq |\mathcal{P}| - threshold$. To implement this test efficiently we sample r pixels at random. If after r trials no satisfactory C_p is found, we assume that no such C_p exists.

Suppose there are n pixels, and s of them satisfy the desired property. Then the probability that we make r random trials without replacement and do not find any of these s pixels is $\frac{C_r^{n-s}}{C_r^n}$. In practice we choose r so that if at least 10% of pixels have the desired property, the probability to miss all of them is less than 10%.

3.2 Graph Reduction

Theorems 1 and 2 allow us to reduce the graph size. Suppose we want to compute an (s,t)-cut and $s \in C_p$ for some p. Then all $p \in \bar{C}_p$ can be contracted into one node[1]. Furthermore if there is q s.t. $C_q \subset C_p$ then all nodes in C_q can also be contracted into one node.

The required storage is linear in the size of the graph. We keep a separate graph for each C_p which contains only the nodes of C_p. As soon as a new $C_q \subset C_p$ is found, a new graph for C_q is created and all nodes of C_q are contracted to one node in the graph for C_p.

3.3 Further speedups

We can further reduce the computations required in step 3 of our algorithm. Suppose for a pixel p we have already computed the minimum (p,t)-cut C and $q \in C_p$. It is easy to observe that if the cost of the minimum (q,p)-cut is larger than or equal to $c(C)$ then the minimum (q,t)-cut and (p,t)-cut are equal. We can exploit this fact.

The cost of the minimum (q,p)-cut is equal to the amount of flow we can push from q to p, see [2]. Thus if we can push flow of cost $c(C)$ from q to p then $C_q = C_p$. If neighboring pixels p and q have similar color, then we can push flow of cost $c(C)$ using just a few edges from the graph. Indeed we found that in many cases we can push enough flow just through a single edge from q to p.

The algorithm proceeds as follows. In the beginning all pixels are marked unprocessed. While there is an unprocessed pixel p we compute the minimum (p,t)-cut C and mark p to be processed. For all neighbors q of p which are close in color to p and are still unprocessed, we cut a small piece of the graph (usually of 40 nodes) around pixels p and q and check if we can push flow of size $c(C)$ from q to p. If yes we mark q as processed and continue this process now looking at the neighbors of q. If no we leave q unprocessed.

[1] To contract nodes in some set S we replace all nodes in S by a new node s, remove all edges with both end points in S, and replace multiple edges between s and $p \notin S$ by a single edge with weight equal to the sum of the multiple edge weights.

(a) Original image (b) Top of the hierarchy (c)Bottom of the hierarchy

Figure 5. Baseball image. Size 221 by 147. Running time 10 minutes. 5 levels in the hierarchy.

4 Experimental Results

In this section we present some experiments on segmenting intensity images. The most important parameters of the algorithm are the edge weights. Usually segmentation results are rather sensitive these weights. The important factor in choosing the edge weights is to make sure they decrease rapidly enough with the decrease in similarity. A common choice (used in [10],[8], [3]) is

$$w(p,q) = e^{-\frac{(I_p - I_q)^2}{\sigma_I}} \times e^{\frac{-d(p,q)^2}{\sigma_d}},$$

where I_p is the intensity of pixel p, $d(p,q)$ is the distance between pixels p and q and σ_I and σ_d are the control parameters. We chose different weights which have a similar functional form:

$$w(p,q) = max\left(\frac{2^{M-|I_p - I_q|}}{2^{d(p,q)}}, 1\right).$$

For all the experiments, we set $M = 25, threshold = 50$, $w_{max} = 2^{10}$, $w_{min} = 1$. For the image in section 4.1 we used 4 nearest neighbors, and for all other images we used 8 nearest neighbors. The results are displayed by assigning each segment a unique intensity.

4.1 Structured foreground and unstructured background

Figure 4 shows an example similar to the one in [6]. The intensities are distributed uniformly between 0 and 255 for the background, between 50 and 55 for the larger square and between 80 and 85 for the smaller square. Our algorithm achieves good segmentation, foreground squares are segmented out and the unstructured background is grouped together. A few spurious pixels which have similar intensities to the intensities of the squares get grouped together with the squares.

4.2 Baseball image

Figure 5(a) shows a baseball image from [8]. Figures 5(b) and (c) show our results at the top and the bottom level of the hierarchy. On the top level, the significant pieces of both players are segmented out. On the bottom level, the background and the players are split into more parts. Interestingly the shoe of the top player is segmented out, even though the pixels inside shoe have visibly large intensity variation. This is because the contour around the shoe is strong, and even though the contours inside the shoe are also strong they do not surround any segment of significant size.

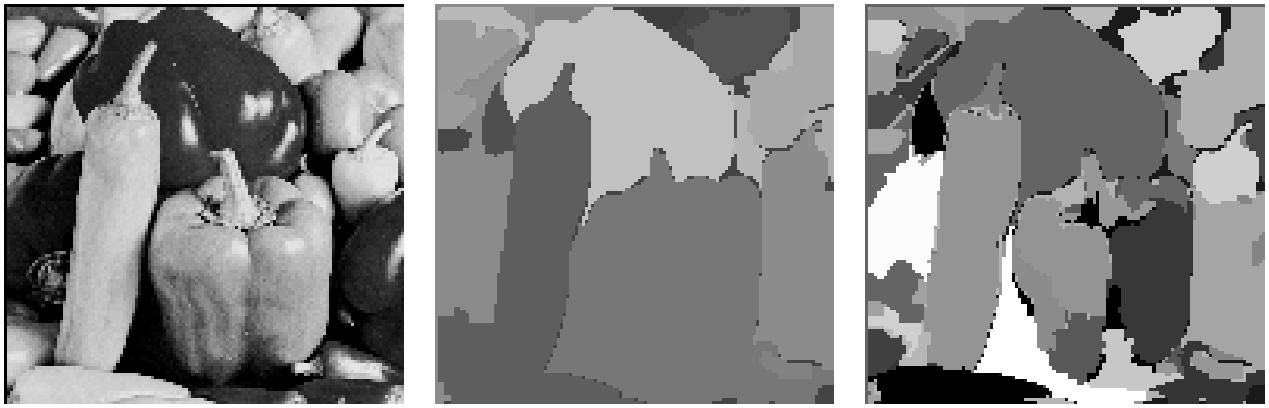
Notice that in spite the fact that we forbid C_p's of small size, there are segments of small size, especially on the higher levels of hierarchy around the intensity edges. Suppose $C_q \subset C_p$ and C_q passes our size requirements. Nevertheless C_q and C_p can overlap in such a way that C_q breaks C_p in small pieces, especially around the boundary. In principle this can be easily detected and corrected, but we have not implemented this yet.

4.3 Peppers image

Figure 6(a) shows an image with peppers taken from the machine vision textbook. Figures 6(b) and (c) show the results at the first and the last levels of the hierarchy. On the bottom of the hierarchy the thin long pepper in the foreground is segmented out from a similar pepper on the bottom of the image and from a small piece of similar pepper on the left. However the other big pepper on the foreground is split into several parts.

Acknowledgments

We would like to thank Yuri Boykov and David Jacobs for useful discussions.



(a) Original image (b) Top of the hierarchy (c) Bottom of the hierarchy

Figure 6. Peppers image. Size 128 by 128. Running time 2.5 minutes. 5 levels in the hierarchy.

Appendix

PROOF: (Theorem 1)

Suppose the theorem is false. Let C' be any minimum (q,t)-cut. Define edges set $S_{p\bar{q}}^{p\bar{q}}$, which contains all edges between pixels in $C_p \cap C'_q$ and $C_p \cap \bar{C}'_q$.

$$S_{p\bar{q}}^{p\bar{q}} = \{e \mid e \cap C_p \cap C'_q \neq \emptyset \text{ and } e \cap C_p \cap \bar{C}'_q \neq \emptyset\}$$

Similarly we define $S_{\bar{p}\bar{q}}^{\bar{p}\bar{q}}$, $S_{\bar{p}q}^{p\bar{q}}$,$S_{\bar{p}q}^{\bar{p}\bar{q}}$, $S_{pq}^{\bar{p}q}$, and $S_{p\bar{q}}^{\bar{p}\bar{q}}$ as follows:

$$S_{pq}^{\bar{p}\bar{q}} = \{e \mid e \cap C_p \cap C'_q \neq \emptyset \text{ and } e \cap \bar{C}_p \cap \bar{C}'_q \neq \emptyset\}$$
$$S_{\bar{p}q}^{p\bar{q}} = \{e \mid e \cap \bar{C}_p \cap C'_q \neq \emptyset \text{ and } e \cap C_p \cap \bar{C}'_q \neq \emptyset\}$$
$$S_{\bar{p}q}^{\bar{p}\bar{q}} = \{e \mid e \cap \bar{C}_p \cap C'_q \neq \emptyset \text{ and } e \cap \bar{C}_p \cap \bar{C}'_q \neq \emptyset\}$$
$$S_{pq}^{\bar{p}q} = \{e \mid e \cap C_p \cap C'_q \neq \emptyset \text{ and } e \cap \bar{C}_p \cap C'_q \neq \emptyset\}$$
$$S_{p\bar{q}}^{\bar{p}\bar{q}} = \{e \mid e \cap C_p \cap \bar{C}'_q \neq \emptyset \text{ and } e \cap \bar{C}_p \cap \bar{C}'_q \neq \emptyset\}$$

Using the sets defined above we can split C and C':

$$C = S_{pq}^{\bar{p}q} \cup S_{pq}^{\bar{p}\bar{q}} \cup S_{p\bar{q}}^{\bar{p}q} \cup S_{p\bar{q}}^{\bar{p}\bar{q}}$$
$$C' = S_{pq}^{p\bar{q}} \cup S_{\bar{p}q}^{\bar{p}\bar{q}} \cup S_{\bar{p}q}^{p\bar{q}} \cup S_{\bar{p}q}^{\bar{p}\bar{q}}.$$

We can define two new cuts, a (q,t)-cut C'' and a (p,t)-cut C''':

$$C'' = S_{pq}^{\bar{p}q} \cup S_{pq}^{p\bar{q}} \cup S_{pq}^{\bar{p}\bar{q}}$$
$$C''' = S_{p\bar{q}}^{\bar{p}\bar{q}} \cup S_{p\bar{q}}^{\bar{p}\bar{q}} \cup S_{\bar{p}q}^{\bar{p}\bar{q}}.$$

Observe that $C''_q = C_p \cap C_q$ and $C'''_p = C_p \cup C_q$. Now $c(C'') > c(C')$, or otherwise C'' is the minimum (q,t) cut required by the theorem. Since all S_{ab}^{cd}'s are disjoint, we get that

$$c(S_{pq}^{\bar{p}q}) > c(S_{\bar{p}q}^{p\bar{q}}) + c(S_{\bar{p}q}^{\bar{p}\bar{q}}).$$

Using this fact and writing out the costs of C and C''' in terms of the S's we derive that $c(C) > c(C''')$, which is a contradiction. ■

The proof of theorem 2 is very similar to the proof of theorem 1.

References

[1] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. In *International Conference on Computer Vision*, 1999.

[2] W. Cook, W. Cunningham, W. Pulleyblank, and A. Schrijver. *Combinatorial Optimization*. John Wiley and Sons, 1998.

[3] Y. Gdalyahu, D. Weinshall, and M. Werman. Stochastic image segmentation by typical cuts. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 596–601, 1999.

[4] H. Ishikawa and D. Geiger. Segmentation by grouping junctions. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 125–131, 1998.

[5] A. Jain and R. Dubes. *Algorithms for Clustering Data*. Prentice Hall, 1988.

[6] P. Perona and W. Freeman. A factorization approach to grouping. In *European Conference on Computer Vision*, pages 655–670, 1998.

[7] S. Roy and I. Cox. A maximum-flow formulation of the n-camera stereo correspondence problem. In *6th International Conference on Computer Vision*, 1998.

[8] J. Shi and J. Malik. Normalized cuts and image segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 731–737, 1997.

[9] Y. Weiss. Segmentation using eigenvectors: a unifying view. In *Seventh International Conference on Computer Vision*, pages 975–982, 1999.

[10] Z. Wu and R. Leahy. An optimal graph theoretic approach to data clustering: Theory and its application to image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11:1101–1113, 1993.