Normalized Cuts and Image Segmentation

Jianbo Shi and Jitendra Malik

Abstract

We propose a novel approach for solving the perceptual grouping problem in vision. Rather than focusing on local features and their consistencies in the image data, our approach aims at extracting the global impression of an image. We treat image segmentation as a graph partitioning problem and propose a novel global criterion, the *normalized cut*, for segmenting the graph. The *normalized cut* criterion measures both the total dissimilarity between the different groups as well as the total similarity within the groups. We show that an efficient computational technique based on a generalized eigenvalue problem can be used to optimize this criterion. We have applied this approach to segmenting static images as well as motion sequences and found results very encouraging.

Keywords: grouping, image segmentation, graph partition

1 Introduction

Nearly 75 years ago, Wertheimer[24] launched the Gestalt approach which laid out the importance of perceptual grouping and organization in visual perception. For our purposes, the problem of grouping can be well motivated by considering the set of points shown in the figure (1).

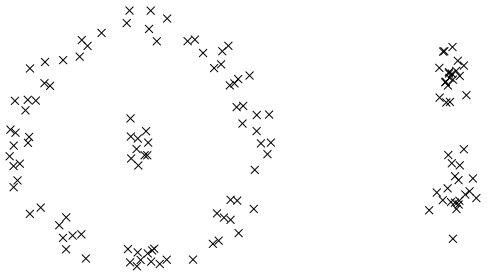

Figure 1: How many groups?

Typically a human observer will perceive four objects in the image–a circular ring with a cloud of points inside it, and two loosely connected clumps of points on its right. However

this is not the unique partitioning of the scene. One can argue that there are three objects– the two clumps on the right constitute one dumbbell shaped object. Or there are only two objects, a dumb bell shaped object on the right, and a circular galaxy like structure on the left. If one were perverse, one could argue that in fact every point was a distinct object.

This may seem to be an artificial example, but every attempt at image segmentation ultimately has to confront a similar question–there are many possible partitions of the domain D of an image into subsets D_i (including the extreme one of every pixel being a separate entity). How do we pick the "right" one? We believe the Bayesian view is appropriate– one wants to find the most probable interpretation in the context of prior world knowledge. The difficulty, of course, is in specifying the prior world knowledge–some of it is low level such as coherence of brightness, color, texture, or motion, but equally important is mid- or high-level knowledge about symmetries of objects or object models.

This suggests to us that image segmentation based on low level cues can not and should not aim to produce a complete final "correct" segmentation. The objective should instead be to *use the low-level coherence of brightness, color, texture or motion attributes to sequentially come up with hierarchical partitions.* Mid and high level knowledge can be used to either confirm these groups or select some for further attention. This attention could result in further repartitioning or grouping. The key point is that image partitioning is to be done from the big picture downwards, rather like a painter first marking out the major areas and then filling in the details.

Prior literature on the related problems of clustering, grouping and image segmentation is huge. The clustering community[12] has offered us agglomerative and divisive algorithms; in image segmentation we have region-based merge and split algorithms. The hierarchical divisive approach that we advocate produces a tree, the *dendrogram.* While most of these ideas go back to the 70s (and earlier), the 1980s brought in the use of Markov Random Fields[10] and variational formulations[17, 2, 14]. The MRF and variational formulations also exposed two basic questions (1) What is the criterion that one wants to optimize? and (2) Is there an efficient algorithm for carrying out the optimization? Many an attractive criterion has been doomed by the inability to find an effective algorithm to find its minimum–greedy or gradient descent type approaches fail to find global optima for these high dimensional,

nonlinear problems.

Our approach is most related to the graph theoretic formulation of grouping. The set of points in an arbitrary feature space are represented as a weighted undirected graph $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$, where the nodes of the graph are the points in the feature space, and an edge is formed between every pair of nodes. The weight on each edge, $w(\boldsymbol{i}, \boldsymbol{j})$, is a function of the similarity between nodes $\boldsymbol{i}$ and $\boldsymbol{j}$.

In grouping, we seek to partition the set of vertices into disjoint sets $\mathbf{V}_1, \mathbf{V}_2, ..., \mathbf{V}_m$, where by some measure the similarity among the vertices in a set $\mathbf{V}_i$ is high and across different sets $\mathbf{V}_i$,$\mathbf{V}_j$ is low.

To partition a graph, we need to also ask the following questions:

1. What is the precise criterion for a good partition?

2. How can such a partition be computed efficiently?

In the image segmentation and data clustering community, there has been much previous work using variations of the minimal spanning tree or limited neighborhood set approaches. Although those use efficient computational methods, the segmentation criteria used in most of them are based on local properties of the graph. Because perceptual grouping is about extracting the global impressions of a scene, as we saw earlier, this partitioning criterion often falls short of this main goal.

In this paper we propose a new graph-theoretic criterion for measuring the goodness of an image partition– the *normalized cut*. We introduce and justify this criterion in section 2. The minimization of this criterion can be formulated as a generalized eigenvalue problem; the eigenvectors of this problem can be used to construct good partitions of the image and the process can be continued recursively as desired(section 2.1) Section 3 gives a detailed explanation of the steps of our grouping algorithm. In section 4 we show experimental results. The formulation and minimization of the normalized cut criterion draws on a body of results from the field of spectral graph theory(section 5). Relationship to work in computer vision is discussed in section 6, and comparison with related eigenvector based segmentation methods is represented in section 6.1. We conclude in section 7.

2 Grouping as graph partitioning

A graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ can be partitioned into two disjoint sets, A, B, $A \cup B = \mathbf{V}$, $A \cap B = \emptyset$, by simply removing edges connecting the two parts. The degree of dissimilarity between these two pieces can be computed as total weight of the edges that have been removed. In graph theoretic language, it is called the *cut*:

$$cut(A, B) = \sum_{u \in A, v \in B} w(u, v). \tag{1}$$

The optimal bi-partitioning of a graph is the one that minimizes this *cut* value. Although there are exponential number of such partitions, finding the *minimum cut* of a graph is a well studied problem, and there exist efficient algorithms for solving it.

Wu and Leahy[25] proposed a clustering method based on this minimum cut criterion. In particular, they seek to partition a graph into k-subgraphs, such that the maximum cut across the subgroups is minimized. This problem can be efficiently solved by recursively finding the minimum cuts that bisect the existing segments. As shown in Wu & Leahy's work, this globally optimal criterion can be used to produce good segmentation on some of the images.

However, as Wu and Leahy also noticed in their work, the minimum cut criteria favors cutting small sets of isolated nodes in the graph. This is not surprising since the cut defined in (1) increases with the number of edges going across the two partitioned parts. Figure (2) illustrates one such case. Assuming the edge weights are inversely proportional to the

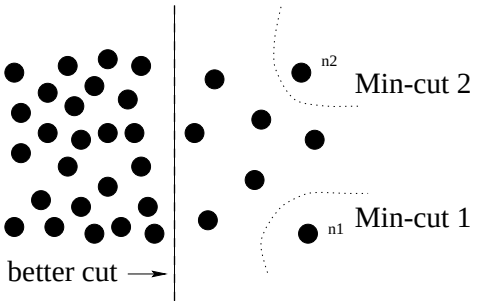

Figure 2: A case where minimum cut gives a bad partition.

distance between the two nodes, we see the cut that partitions out node n_1 or n_2 will have a

very small value. In fact, any cut that partitions out individual nodes on the right half will have smaller cut value than the cut that partitions the nodes into the left and right halves.

To avoid this unnatural bias for partitioning out small sets of points, we propose a new measure of disassociation between two groups. Instead of looking at the value of total edge weight connecting the two partitions, our measure computes the cut cost as a fraction of the total edge connections to all the nodes in the graph. We call this disassociation measure the *normalized cut* (*Ncut*):

$$Ncut(A,B) = \frac{cut(A,B)}{assoc(A,V)} + \frac{cut(A,B)}{assoc(B,V)} \tag{2}$$

where $assoc(A,V) = \sum_{u \in A, t \in V} w(u,t)$ is the total connection from nodes in A to all nodes in the graph, and $assoc(B,V)$ is similarly defined. With this definition of the disassociation between the groups, the cut that partitions out small isolated points will no longer have small *Ncut* value, since the *cut* value will almost certainly be a large percentage of the total connection from that small set to all other nodes. In the case illustrated in figure 2, we see that the cut_1 value across node n_1 will be 100% of the total connection from that node.

In the same spirit, we can define a measure for total normalized association within groups for a given partition:

$$Nassoc(A,B) = \frac{assoc(A,A)}{assoc(A,V)} + \frac{assoc(B,B)}{assoc(B,V)} \tag{3}$$

where $assoc(A,A)$ and $assoc(B,B)$ are total weights of edges connecting nodes within A and B respectively. We see again this is an unbiased measure, which reflects how tightly on average nodes within the group are connected to each other.

Another important property of this definition of association and disassociation of a partition is that they are naturally related:

$$\begin{aligned} Ncut(A,B) &= \frac{cut(A,B)}{assoc(A,V)} + \frac{cut(A,B)}{assoc(B,V)} \\ &= \frac{assoc(A,V) - assoc(A,A)}{assoc(A,V)} \\ &\quad + \frac{assoc(B,V) - assoc(B,B)}{assoc(B,V)} \\ &= 2 - (\frac{assoc(A,A)}{assoc(A,V)} + \frac{assoc(B,B)}{assoc(B,V)}) \end{aligned}$$

$$= \quad 2 - Nassoc(A, B)$$

Hence the two partition criteria that we seek in our grouping algorithm, minimizing the disassociation between the groups and maximizing the association within the group, are in fact identical, and can be satisfied simultaneously. In our algorithm, we will use this normalized cut as the partition criterion.

Unfortunately minimizing normalized cut exactly is NP-complete, even for the special case of graphs on grids. The proof, due to C. Papadimitriou, can be found in appendix A. However, we will show that when we embed the normalized cut problem in the real value domain, an approximate solution can be found efficiently.

2.1 Computing the optimal partition

Given a partition of nodes of a graph, V, into two sets A and B, let $\boldsymbol{x}$ be an $N = |\boldsymbol{V}|$ dimensional indicator vector, $x_i = 1$ if node i is in A, and -1 otherwise. Let $\boldsymbol{d}$(i) $= \sum_j w(i, j)$, be the total connection from node $\boldsymbol{i}$ to all other nodes. With the definitions $\boldsymbol{x}$ and $\boldsymbol{d}$ we can rewrite $Ncut(A, B)$ as:

$$\begin{aligned} Ncut(A, B) &= \frac{cut(A, B)}{assoc(A, V)} + \frac{cut(B, A)}{assoc(B, V)} \\ &= \frac{\sum_{(\boldsymbol{x}_i>0, \boldsymbol{x}_j<0)} -w_{ij}\boldsymbol{x}_i\boldsymbol{x}_j}{\sum_{\boldsymbol{x}_i>0} \boldsymbol{d}_i} \\ &\quad + \frac{\sum_{(\boldsymbol{x}_i<0, \boldsymbol{x}_j>0)} -w_{ij}\boldsymbol{x}_i\boldsymbol{x}_j}{\sum_{\boldsymbol{x}_i<0} \boldsymbol{d}_i} \end{aligned}$$

Let $\mathbf{D}$ be an $N \times N$ diagonal matrix with $\boldsymbol{d}$ on its diagonal, $\mathbf{W}$ be an $N \times N$ symmetrical matrix with W(i,j) $= w_{ij}$, $k = \frac{\sum_{x_i>0} \boldsymbol{d}_i}{\sum_i \boldsymbol{d}_i}$, and $\mathbf{1}$ be an $N \times 1$ vector of all ones. Using the fact $\frac{\mathbf{1}+\boldsymbol{x}}{2}$ and $\frac{\mathbf{1}-\boldsymbol{x}}{2}$ are indicator vectors for $x_i > 0$ and $x_i < 0$ respectively, we can rewrite $4[Ncut(\boldsymbol{x})]$ as:

$$\begin{aligned} &= \frac{(\mathbf{1}+\boldsymbol{x})^T(\mathbf{D}-\mathbf{W})(\mathbf{1}+\boldsymbol{x})}{k\mathbf{1}^T\mathbf{D}\mathbf{1}} + \frac{(\mathbf{1}-\boldsymbol{x})^T(\mathbf{D}-\mathbf{W})(\mathbf{1}-\boldsymbol{x})}{(1-k)\mathbf{1}^T\mathbf{D}\mathbf{1}} \\ &= \frac{(\boldsymbol{x}^T(\mathbf{D}-\mathbf{W})\boldsymbol{x}+\mathbf{1}^T(\mathbf{D}-\mathbf{W})\mathbf{1})}{k(1-k)\mathbf{1}^T\mathbf{D}\mathbf{1}} + \frac{2(1-2k)\mathbf{1}^T(\mathbf{D}-\mathbf{W})\boldsymbol{x}}{k(1-k)\mathbf{1}^T\mathbf{D}\mathbf{1}} \end{aligned}$$

Let $\alpha(\boldsymbol{x}) = \boldsymbol{x}^T(\mathbf{D}-\mathbf{W})\boldsymbol{x}$, $\beta(\boldsymbol{x}) = \mathbf{1}^T(\mathbf{D}-\mathbf{W})\boldsymbol{x}$, $\gamma = \mathbf{1}^T(\mathbf{D}-\mathbf{W})\mathbf{1}$, and $M = \mathbf{1}^T\mathbf{D}\mathbf{1}$, we can then further expand the above equation as:

$$\begin{aligned} &= \frac{(\alpha(\boldsymbol{x})+\gamma)+2(1-2k)\beta(\boldsymbol{x})}{k(1-k)M} \\ &= \frac{(\alpha(\boldsymbol{x})+\gamma)+2(1-2k)\beta(\boldsymbol{x})}{k(1-k)M} - \frac{2(\alpha(\boldsymbol{x})+\gamma)}{M} + \frac{2\alpha(\boldsymbol{x})}{M} + \frac{2\gamma}{M} \end{aligned}$$

Dropping the last constant term, which in this case equals 0, we get

$$\begin{aligned} &= \frac{(1-2k+2k^2)(\alpha(\boldsymbol{x})+\gamma)+2(1-2k)\beta(\boldsymbol{x})}{k(1-k)M} + \frac{2\alpha(\boldsymbol{x})}{M} \\ &= \frac{\frac{(1-2k+2k^2)}{(1-k)^2}(\alpha(\boldsymbol{x})+\gamma)+\frac{2(1-2k)}{(1-k)^2}\beta(\boldsymbol{x})}{\frac{k}{1-k}M} + \frac{2\alpha(\boldsymbol{x})}{M} \end{aligned}$$

Letting $b = \frac{k}{1-k}$, and since $\gamma = 0$, it becomes,

$$\begin{aligned} &= \frac{(1+b^2)(\alpha(\boldsymbol{x})+\gamma)+2(1-b^2)\beta(\boldsymbol{x})}{bM} + \frac{2b\alpha(\boldsymbol{x})}{bM} \\ &= \frac{(1+b^2)(\alpha(\boldsymbol{x})+\gamma)}{bM} + \frac{2(1-b^2)\beta(\boldsymbol{x})}{bM} + \frac{2b\alpha(\boldsymbol{x})}{bM} - \frac{2b\gamma}{bM} \\ &= \frac{(1+b^2)(\boldsymbol{x}^T(\mathbf{D}-\mathbf{W})\boldsymbol{x}+\mathbf{1}^T(\mathbf{D}-\mathbf{W})\mathbf{1})}{b\mathbf{1}^T\mathbf{D}\mathbf{1}} \\ &\quad + \frac{2(1-b^2)\mathbf{1}^T(\mathbf{D}-\mathbf{W})\boldsymbol{x}}{b\mathbf{1}^T\mathbf{D}\mathbf{1}} \\ &\quad + \frac{2b\boldsymbol{x}^T(\mathbf{D}-\mathbf{W})\boldsymbol{x}}{b\mathbf{1}^T\mathbf{D}\mathbf{1}} - \frac{2b\mathbf{1}^T(\mathbf{D}-\mathbf{W})\mathbf{1}}{b\mathbf{1}^T\mathbf{D}\mathbf{1}} \\ &= \frac{(\mathbf{1}+\boldsymbol{x})^T(\mathbf{D}-\mathbf{W})(\mathbf{1}+\boldsymbol{x})}{b\mathbf{1}^T\mathbf{D}\mathbf{1}} \\ &\quad + \frac{b^2(\mathbf{1}-\boldsymbol{x})^T(\mathbf{D}-\mathbf{W})(\mathbf{1}-\boldsymbol{x})}{b\mathbf{1}^T\mathbf{D}\mathbf{1}} \\ &\quad - \frac{2b(\mathbf{1}-\boldsymbol{x})^T(\mathbf{D}-\mathbf{W})(\mathbf{1}+\boldsymbol{x})}{b\mathbf{1}^T\mathbf{D}\mathbf{1}} \\ &= \frac{[(\mathbf{1}+\boldsymbol{x})-b(\mathbf{1}-\boldsymbol{x})]^T(\mathbf{D}-\mathbf{W})[(\mathbf{1}+\boldsymbol{x})-b(\mathbf{1}-\boldsymbol{x})]}{b\mathbf{1}^T\mathbf{D}\mathbf{1}} \end{aligned}$$

Setting $\boldsymbol{y} = (\mathbf{1}+\boldsymbol{x})-b(\mathbf{1}-\boldsymbol{x})$, it is easy to see that

$$\boldsymbol{y}^T\mathbf{D}\mathbf{1} = \sum_{x_i>0}\boldsymbol{d}_i - b\sum_{x_i<0}\boldsymbol{d}_i = 0 \tag{4}$$

since $b = \frac{k}{1-k} = \frac{\sum_{x_i>0} \boldsymbol{d}_i}{\sum_{x_i<0} \boldsymbol{d}_i}$, and

$$\begin{aligned} \boldsymbol{y}^T\mathbf{D}\boldsymbol{y} = & \quad \textstyle\sum_{x_i>0} \boldsymbol{d}_i + b^2 \sum_{x_i<0} \boldsymbol{d}_i \\ = & \quad b \textstyle\sum_{x_i<0} \boldsymbol{d}_i + b^2 \sum_{x_i<0} \boldsymbol{d}_i \\ = & \quad b(\textstyle\sum_{x_i<0} \boldsymbol{d}_i + b \sum_{x_i<0} \boldsymbol{d}_i) \\ = & \quad b\mathbf{1}^T\mathbf{D}\mathbf{1}. \end{aligned}$$

Putting everything together we have,

$$min_{\boldsymbol{x}} Ncut(\boldsymbol{x}) = min_{\boldsymbol{y}} \frac{\boldsymbol{y}^T(\mathbf{D}-\mathbf{W})\boldsymbol{y}}{\boldsymbol{y}^T\mathbf{D}\boldsymbol{y}}, \tag{5}$$

with the condition $\boldsymbol{y}(i) \in \{1, -b\}$ and $\boldsymbol{y}^T\mathbf{D}\mathbf{1} = 0$.

Note that the above expression is the Rayleigh quotient[11]. If $\boldsymbol{y}$ is relaxed to take on real values, we can minimize equation (5) by solving the generalized eigenvalue system,

$$(\mathbf{D}-\mathbf{W})\boldsymbol{y} = \lambda\mathbf{D}\boldsymbol{y}. \tag{6}$$

However, we have two constraints on $\boldsymbol{y}$, which come from the condition on the corresponding indicator vector $\boldsymbol{x}$. First consider the constraint $\boldsymbol{y}^T\mathbf{D}\mathbf{1} = 0$. We can show this constraint on $\boldsymbol{y}$ is automatically satisfied by the solution of the generalized eigensystem. We will do so by first transforming equation (6) into a standard eigensystem, and show the corresponding condition is satisfied there. Rewrite equation (6) as

$$\mathbf{D}^{-\frac{1}{2}}(\mathbf{D}-\mathbf{W})\mathbf{D}^{-\frac{1}{2}}\boldsymbol{z} = \lambda\boldsymbol{z}, \tag{7}$$

where $\boldsymbol{z} = \mathbf{D}^{\frac{1}{2}}\boldsymbol{y}$. One can easily verify that $\boldsymbol{z}_0 = \mathbf{D}^{\frac{1}{2}}\mathbf{1}$ is an eigenvector of equation (7) with eigenvalue of 0. Furthermore, $\mathbf{D}^{-\frac{1}{2}}(\mathbf{D}-\mathbf{W})\mathbf{D}^{-\frac{1}{2}}$ is symmetric positive semidefinite, since $(\mathbf{D}-\mathbf{W})$, also called the *Laplacian* matrix, is known to be positive semidefinite[18]. Hence $\boldsymbol{z}_0$ is in fact the smallest eigenvector of equation (7), and all eigenvectors of equation (7) are perpendicular to each other. In particular, $\boldsymbol{z}_1$ the second smallest eigenvector is perpendicular to $\boldsymbol{z}_0$. Translating this statement back into the general eigensystem (6), we have 1) $\boldsymbol{y}_0 = \mathbf{1}$ is the smallest eigenvector with eigenvalue of 0, and 2) $0 = {\boldsymbol{z}_1}^T\ \boldsymbol{z}_0 = {\boldsymbol{y}_1}^T\ \mathbf{D}\ \mathbf{1}$, where $\boldsymbol{y}_1$ is the second smallest eigenvector of (6).

Now recall a simple fact about the *Rayleigh quotient*[11]:

Let $\mathbf{A}$ *be a real symmetric matrix. Under the constraint that* $\boldsymbol{x}$ *is orthogonal to the j-1 smallest eigenvectors* $\boldsymbol{x}_1,...,\boldsymbol{x}_{j-1}$*, the quotient* $\frac{\boldsymbol{x}^T\mathbf{A}\boldsymbol{x}}{\boldsymbol{x}^T\boldsymbol{x}}$ *is minimized by the next smallest eigenvector* $\boldsymbol{x}_j$*, and its minimum value is the corresponding eigenvalue* $\boldsymbol{\lambda}_j$*.*

As a result, we obtain:

$$\boldsymbol{z}_1 = arg.min_{\boldsymbol{z}^T\boldsymbol{z}_0=0}\frac{\boldsymbol{z}^T\mathbf{D}^{-\frac{1}{2}}(\mathbf{D}-\mathbf{W})\mathbf{D}^{-\frac{1}{2}}\boldsymbol{z}}{\boldsymbol{z}^T\boldsymbol{z}}, \tag{8}$$

and consequently,

$$\boldsymbol{y}_1 = arg.min_{\boldsymbol{y}^T\mathbf{D}\mathbf{1}=0}\frac{\boldsymbol{y}^T(\mathbf{D}-\mathbf{W})\boldsymbol{y}}{\boldsymbol{y}^T\mathbf{D}\boldsymbol{y}}, \tag{9}$$

Thus the second smallest eigenvector of the generalized eigensystem (6) is the real valued solution to our normalized cut problem. The only reason that it is not necessarily the solution to our original problem is that the second constraint on $\boldsymbol{y}$ that $\boldsymbol{y}(i)$ takes on two discrete values is not automatically satisfied. In fact relaxing this constraint is what makes this optimization problem tractable in the first place. We will show in section (3) how this real valued solution can be transformed into a discrete form.

A similar argument can also be made to show that the eigenvector with the third smallest eigenvalue is the real valued solution that optimally sub-partitions the first two parts. In fact this line of argument can be extended to show that one can sub-divide the existing graphs, each time using the eigenvector with the next smallest eigenvalue. However, in practice because the approximation error from the real valued solution to the discrete valued solution accumulates with every eigenvector taken, and all eigenvectors have to satisfy a global mutual orthogonality constraint, solutions based on higher eigenvectors become unreliable. It is best to restart solving the partitioning problem on each subgraph individually.

It is interesting to note that, while the second smallest eigenvector $\boldsymbol{y}$ of (6) only approximates the optimal normalized cut solution, it exactly minimizes the following problem:

$$\inf_{\boldsymbol{y}^T\mathbf{D}\mathbf{1}=0}\frac{\sum_i\sum_j(\boldsymbol{y}(i)-\boldsymbol{y}(j))^2w_{ij}}{\sum_i\boldsymbol{y}(i)^2\mathbf{d}(i)}, \tag{10}$$

in real-valued domain, where $\mathbf{d}(i) = \mathbf{D}(i,i)$. Roughly speaking, this forces the indicator vector $\boldsymbol{y}$ to take similar values for nodes i and j that are tightly coupled(large w_{ij}).

In summary, we propose using the normalized cut criteria for graph partitioning, and we have shown how this criteria can be computed efficiently by solving a generalized eigenvalue problem.

3 The grouping algorithm

Our grouping algorithm consists of the following steps:

1. Given an image or image sequence, set up a weighted graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$, and set the weight on the edge connecting two nodes being a measure of the similarity between the two nodes.

2. Solve $(\mathbf{D} - \mathbf{W})\boldsymbol{x} = \lambda \mathbf{D}\boldsymbol{x}$ for eigenvectors with the smallest eigenvalues.

3. Use the eigenvector with second smallest eigenvalue to bipartition the graph.

4. Decide if the current partition should be sub-divided, and recursively repartition the segmented parts if necessary.

The grouping algorithm as well as its computational complexity can be best illustrated by using the following two examples.

3.1 Example 1: Point Set Case

Take the example illustrated in figure 3. Suppose we would like to group points on the 2D plane based purely on their spatial proximity. This can be done through the following steps:

1. Define a weighted graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$, by taking each point as a node in the graph, and connecting each pair of nodes by a graph edge. The weight on the graph edge connecting node i and j is set to be $w(i,j) = e^{-(d(i,j)/\sigma_x)}$, where $d(i,j)$ is the Euclidean distance between the two nodes, and σ_x controls the scale of the spatial proximity measure. σ_x is set to be 2.0 which is 10% of the height of the point set layout. Figure 4 shows the weight matrix for the weighted graph constructed.

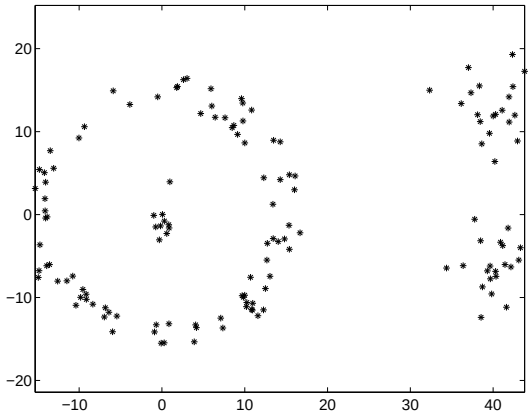

Figure 3: A point set in the plane.

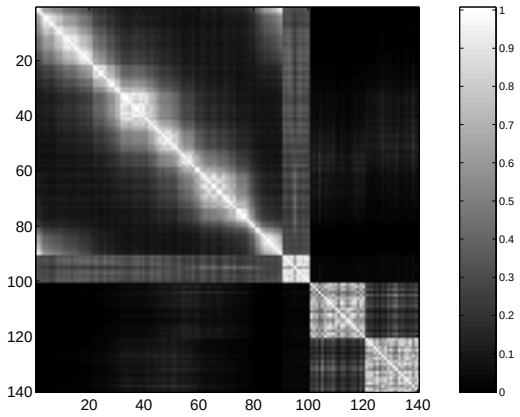

Figure 4: The weight matrix constructed for the point set in (3), using spatial proximity as the similarity measure. The points in figure (3), are numbered as: 1-90, points in the circular ring in counter-clockwise order, 90-100, points in the cluster inside the ring, 100-120, and 120-140, the upper and lower clusters on the right respectively. Notice that the non-zero weights are mostly concentrated in a few blocks around the diagonal. The entries in those square blocks are the connections within each of the clusters.

2. Solve for the eigenvectors with the smallest eigenvalues of the system

$$(\mathbf{D} - \mathbf{W})\boldsymbol{y} = \lambda \mathbf{D}\boldsymbol{y}, \tag{11}$$

or equivalently the eigenvectors with the largest eigenvalues of the system

$$\mathbf{W}\boldsymbol{y} = (1 - \lambda)\mathbf{D}\boldsymbol{y}. \tag{12}$$

This can be done by using a generalized eigenvector solver, or by first transforming the generalized eigenvalue system of (11) or (12) into standard eigenvector problems of

$$\mathbf{D}^{-\frac{1}{2}}(\mathbf{D} - \mathbf{W})\mathbf{D}^{-\frac{1}{2}}\boldsymbol{x} = \lambda \boldsymbol{x}, \tag{13}$$

or

$$\mathbf{D}^{-\frac{1}{2}}\mathbf{W}\mathbf{D}^{-\frac{1}{2}}\boldsymbol{x} = (1 - \lambda)\boldsymbol{x}, \tag{14}$$

with $\boldsymbol{x} = \mathbf{D}^{\frac{1}{2}}\boldsymbol{y}$, and solve it with a standard eigenvector solver.

Either way, we will obtain the solution of the eigenvector problem as shown in figure 5.

3. Partition the point set using the eigenvectors computed. As we have shown, the eigenvector with the second smallest eigenvalue is the continuous approximation to the discrete bi-partitioning indicator vector that we seek. For the case that we have constructed, the eigenvector with the second smallest eigenvalue(figure 5.3) is indeed very close to a discrete one. One can just partition the nodes in the graph based on the sign of their values in the eigenvector. Using this rule, we can partition the point set into two sets as shown in figure 6.

To recursively subdivide each of the two groups, we can either 1) rerun the above procedure on each of the individual groups, or 2) using the the eigenvectors with the next smallest eigenvalues as approximation to the sub-partitioning indicator vectors for each of the groups. We can see that in this case, the eigenvector with the third and the fourth smallest eigenvalue are also good partitioning indicator vectors. Using zero as the splitting point, one can partition the nodes based on their values in the eigenvector into two halves. The sub-partition of the existing groups based on those two subsequent eigenvectors are shown in figure 7.

In this case, we see that the eigenvectors computed from system (11) are very close to the discrete solution that we seek. The second smallest eigenvalue is very close to the optimal

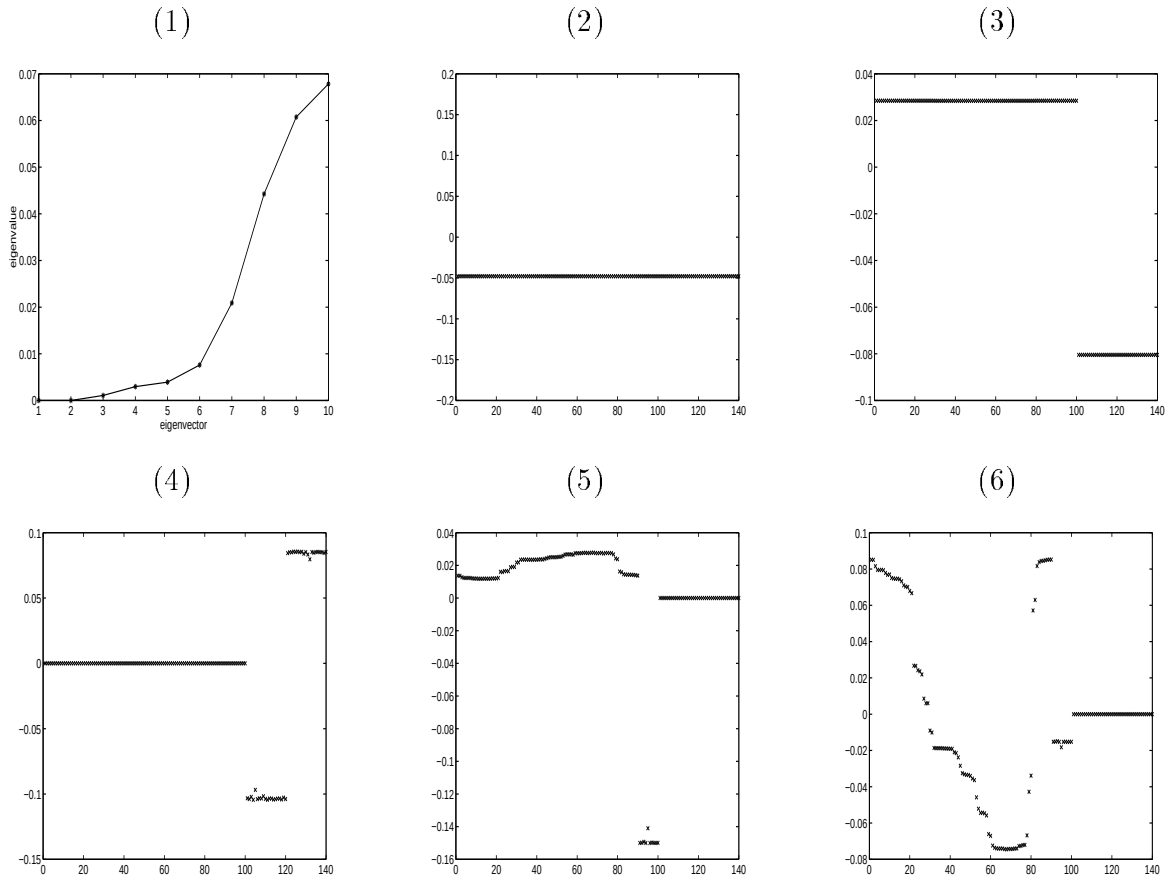

Figure 5: Subplot (1) plots the smallest 10 eigenvalues of the generalized eigenvalue system (11). Subplot (2) - (6) shows the eigenvectors corresponding the the 5 smallest eigenvalues of the system. Note the eigenvector with the smallest eigenvalue (2) is a constant as shown in section 2, and eigenvector with the second smallest eigenvalue (3) is an indicator vector: it takes on only positive values for points in the two clusters to the right. Therefore, using this eigenvector, we can find the first partition of the point set into two clusters: the points on the left with the ring and the cluster inside, and points on the right with the two dumb bell shaped clusters. Furthermore, note that the eigenvectors with the third smallest eigenvalue, subplot (3) and the fourth smallest eigenvalue, subplot(4), are indicator vectors which can be used to partition apart the ring set with the cluster inside of it, and the two clusters on the right.

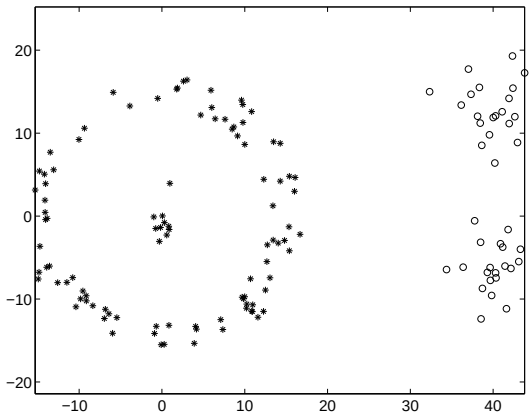

Figure 6: Partition of the point set using the eigenvector with the second smallest eigenvalue.

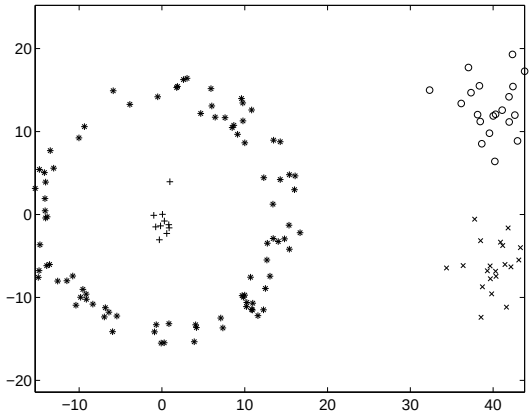

Figure 7: Sub-partitioning of the point sets using the eigenvectors with the third and fourth smallest eigenvalues.

normalized cut value. For the general case, although we are not necessarily this lucky, there is a bound on how far the second smallest eigenvalue can deviate from the optimal normalized cut value, as we shall see in section 5. However, there is little theory on how close the eigenvector is to the discrete form that the normalized cut algorithm seeks. In our experience, the eigenvector computed is quite close to the desired discrete solution.

3.2 Example 2: Brightness Images

Having studied an example of point set grouping, we turn our attention to the case of static image segmentation based on brightness and spatial features. Figure 8 shows an image that we would like to segment.

Figure 8: A gray level image of a baseball game.

Just as in the point set grouping case, we have the following steps for image segmentation:

1. Construct a weighted graph, $\mathbf{G} = (\mathbf{V}, \mathbf{E})$, by taking each pixel as a node, and connecting each pair of pixels by an edge. The weight on that edge should reflect the likelihood of the two pixels belong to one object. Using just the brightness value of the pixels and their spatial location, we can define the graph edge weight connecting two nodes i and j as:

$$w_{ij} = e^{\frac{-\|\boldsymbol{F}(i)-\boldsymbol{F}(j)\|_2^2}{\sigma_I}} * \begin{cases} e^{\frac{-\|\boldsymbol{X}(i)-\boldsymbol{X}(j)\|_2^2}{\sigma_X}} & \text{if } \|\boldsymbol{X}(i)-\boldsymbol{X}(j)\|_2 < r \\ 0 & \text{otherwise} \end{cases} \tag{15}$$

Figure 9 shows the weight matrix $\mathbf{W}$ associated with this weighted graph.

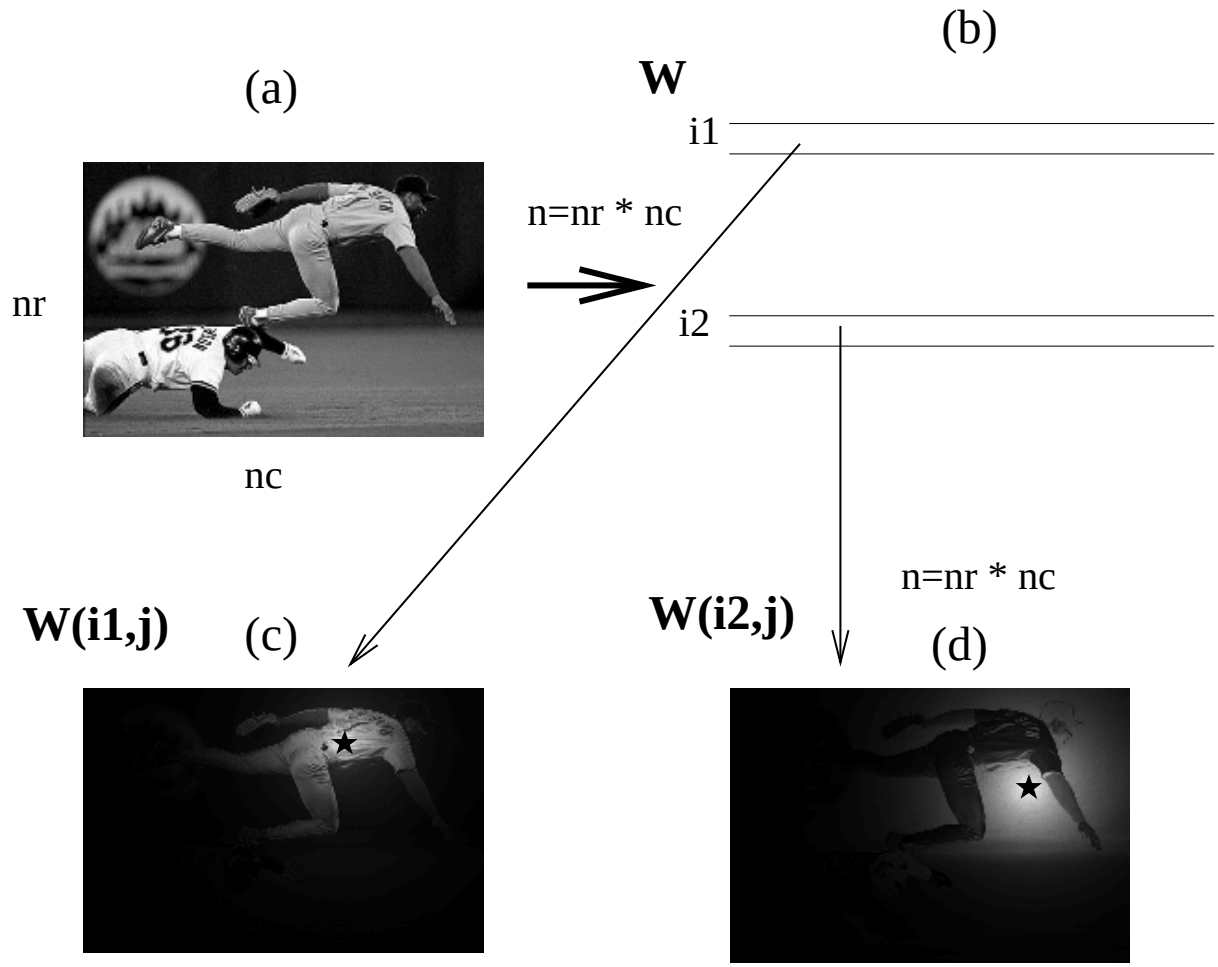

Figure 9: The similarity measure between each pair of pixels in (a) can be summarized in a $n \times n$ weight matrix $\mathbf{W}$, shown in (b), where n is the number of pixels in the image. Instead of displaying $\mathbf{W}$ itself, which is very large, two particular rows, i_1 and i_2 of $\mathbf{W}$ are shown in (c) and (d). Each of the rows, is the connection weights from a pixel to all other pixels in the image. The two rows,i_1 and i_2 are reshaped into the size of the image, and displayed. The brightness value in (c) and (d) reflects the connection weights. Note that $\mathbf{W}$ contains large number of zeros or near zeros, due to the spatial proximity factor.

2. Solve for the eigenvectors with the smallest eigenvalues of the system

$$(\mathbf{D} - \mathbf{W})\boldsymbol{y} = \lambda \mathbf{D}\boldsymbol{y}, \tag{16}$$

As we saw above, the generalized eigensystem in (16) can be transformed into a standard eigenvalue problem of

$$\mathbf{D}^{-\frac{1}{2}}(\mathbf{D} - \mathbf{W})\mathbf{D}^{-\frac{1}{2}}\boldsymbol{x} = \lambda \boldsymbol{x}, \tag{17}$$

Solving a standard eigenvalue problem for all eigenvectors takes $O(n^3)$ operations, where n is the number of nodes in the graph. This becomes impractical for image segmentation applications where n is the number of pixels in an image. Fortunately, our graph partitioning has the following properties: 1) the graphs often are only locally connected and the resulting eigensystem are very sparse, 2) only the top few eigenvectors are needed for graph partitioning, and 3) the precision requirement for the eigenvectors is low, often only the right sign bit is required. These special properties of our problem can be fully exploited by an eigensolver called the Lanczos method. The running time of a Lanczos algorithm is $O(mn) + O(mM(n))$[11], where m is the maximum number of matrix-vector computations required, and $M(n)$ is the cost of a matrix-vector computation of $\mathbf{A}\boldsymbol{x}$, where $\mathbf{A} = \mathbf{D}^{-\frac{1}{2}}(\mathbf{D} - \mathbf{W})\mathbf{D}^{-\frac{1}{2}}$. Note that sparse structure in $\mathbf{A}$ is identical to that of the weight matrix $\mathbf{W}$. Due to the sparse structure in the weight matrix $\mathbf{W}$, and therefore $\mathbf{A}$, the matrix- vector computation is only of $O(n)$, where n is the number of the nodes.

To see why this is the case, we will look at the cost of the inner product of one row of $\boldsymbol{A}$ with a vector $\mathbf{x}$. Let $\mathbf{y}_i = \boldsymbol{A}_i \cdot \mathbf{x} = \sum_j \boldsymbol{A}_{ij}\mathbf{x}_j$. For a fixed i, $\boldsymbol{A}_{ij}$ is only nonzero if node j is in a spatial neighborhood of i. Hence there are only a fixed number of operations required for each $\boldsymbol{A}_i \cdot \mathbf{x}$, and the total cost of computing $\boldsymbol{A}\mathbf{x}$ is $O(n)$. Figure 10 is graphical illustration of this special inner product operation for the case of the image segmentation.

Furthermore, it turns out that we can substantially cut down additional connections from each node to its neighbors by randomly selecting the connections within the neighborhood for the weighted graph as shown in figure 11. Empirically, we have found that one can remove up to 90% of the total connections with each of the neighborhoods when the neighborhoods are large, without effecting the eigenvector solution to the system.

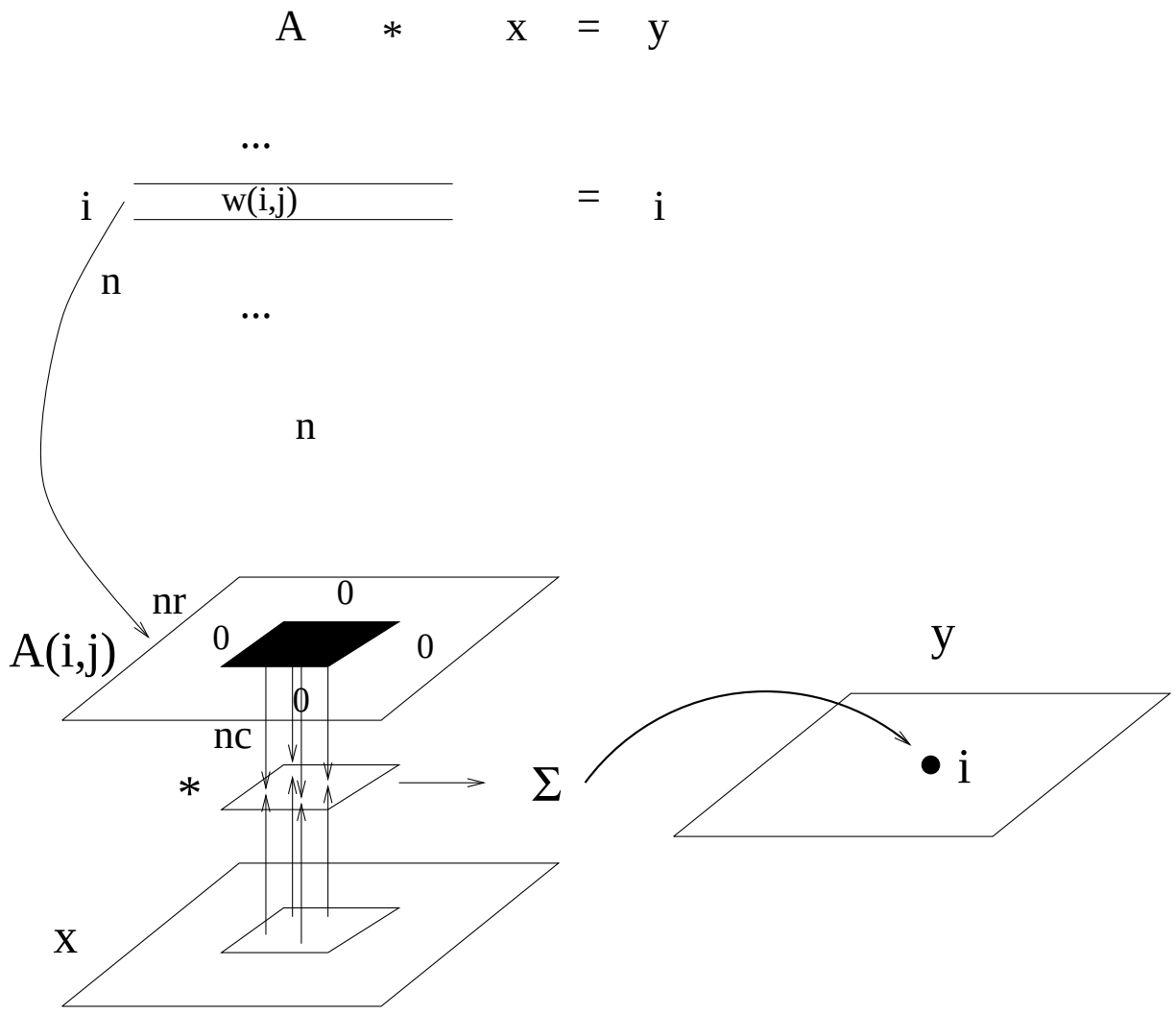

Figure 10: The inner product operation between $A(i,j)$ and x, of dimension n, is a convolution operation in the case of image segmentation.

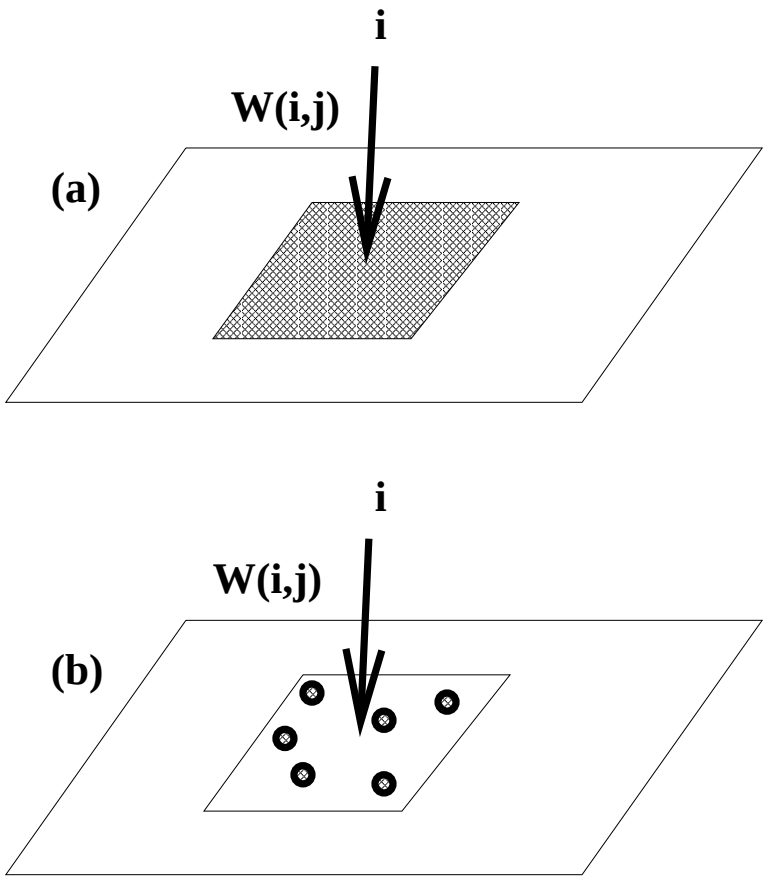

Figure 11: Instead of connecting a node i to all the nodes in its neighborhood (indicated by the shaded area in (a)), we will only connect i to randomly selected nodes(indicated by the open circles in (b)).

Putting everything together, each of the matrix-vector computations cost $O(n)$ operations with a small constant factor. The number m depends on many factors[11]. In our experiments on image segmentation, we observed that m is typically less than $O(n^{\frac{1}{2}})$.

Figure 12 shows the smallest eigenvectors computed for the generalized eigensystem with the weight matrix defined above.

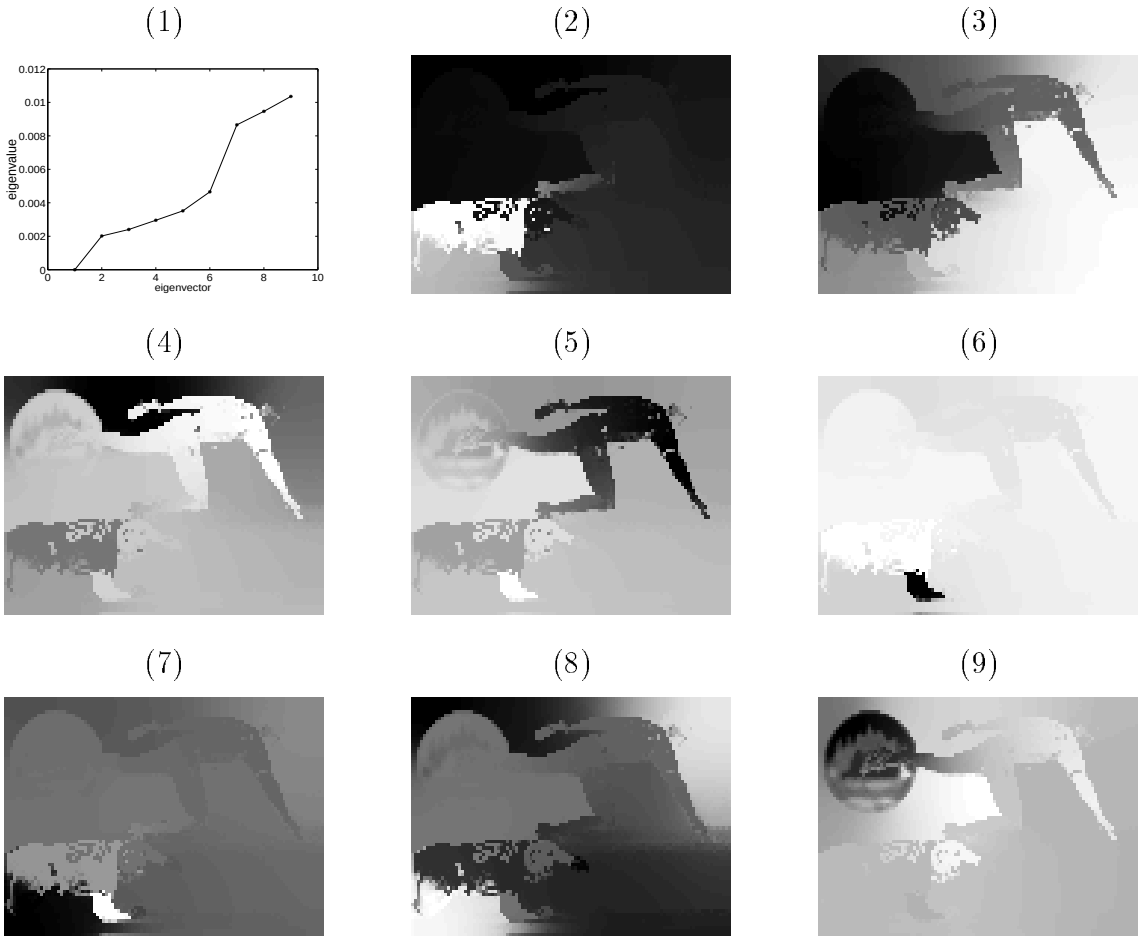

Figure 12: Subplot (1) plots the smallest eigenvectors of the generalized eigenvalue system (11). Subplot (2) - (9) shows the eigenvectors corresponding the 2nd smallest to the 9th smallest eigenvalues of the system. The eigenvectors are reshaped to be the size of the image.

3. Once the eigenvectors are computed, we can partition the graph into two pieces using the second smallest eigenvector. In the ideal case, the eigenvector should only take on two discrete values, and the signs of the values can tell us exactly how to partition the graph.

However, our eigenvectors can take on continuous values, and we need to choose a splitting point to partition it into two parts. There are many different ways of choosing such splitting point. One can take 0 or the median value as the splitting point, or one can search for the splitting point such that the resulting partition has the best $Ncut(A, B)$ value. We take the latter approach in our work. Currently, the search is done by checking l evenly spaced possible splitting points, and computing the best $Ncut$ among them. In our experiments, the values in the eigenvectors are usually well separated, and this method of choosing a splitting point is very reliable even with a small l. Figure 13 shows this process.

4. After the graph is broken into two pieces, we can recursively run our algorithm on the two partitioned parts. Or equivalently, we could take advantage of the special properties of the other top eigenvectors as explained in previous section to subdivide the graph based on those eigenvectors. The recursion stops once the $Ncut$ value exceeds certain limit.

We also impose a stability criterion on the partition. As we saw earlier, and as we see in the eigenvectors with the 7-9th smallest eigenvalues(figure(12.7-9)), sometimes an eigenvector can take on the shape of a continuous function rather that the discrete indicator function that we seek. From the view of segmentation, such an eigenvector is attempting to subdivide an image region where there is no sure way of breaking it. In fact, if we are forced to partition the image based on this eigenvector, we will see there are many different splitting points which have similar $Ncut$ values. Hence the partition will be highly uncertain and unstable. In our current segmentation scheme, we simply choose to ignore all those eigenvectors which have smoothly varying eigenvector values. We achieve this by imposing a stability criterion which measures the degree of smoothness in the eigenvector values. The simplest measure is based on first computing the histogram of the eigenvector values, and then computing the ratio between the minimum and maximum values in the bins. When the eigenvector values are continuously varying, the values in the histogram bins will stay relatively the same, and the ratio will be relatively high. In our experiments, we find that simple thresholding on the ratio described above can be used to exclude unstable eigenvectors. We have set that value to be 0.06 in all our experiments.

Figure 14 shows the final segmentation for the image shown in figure 8.

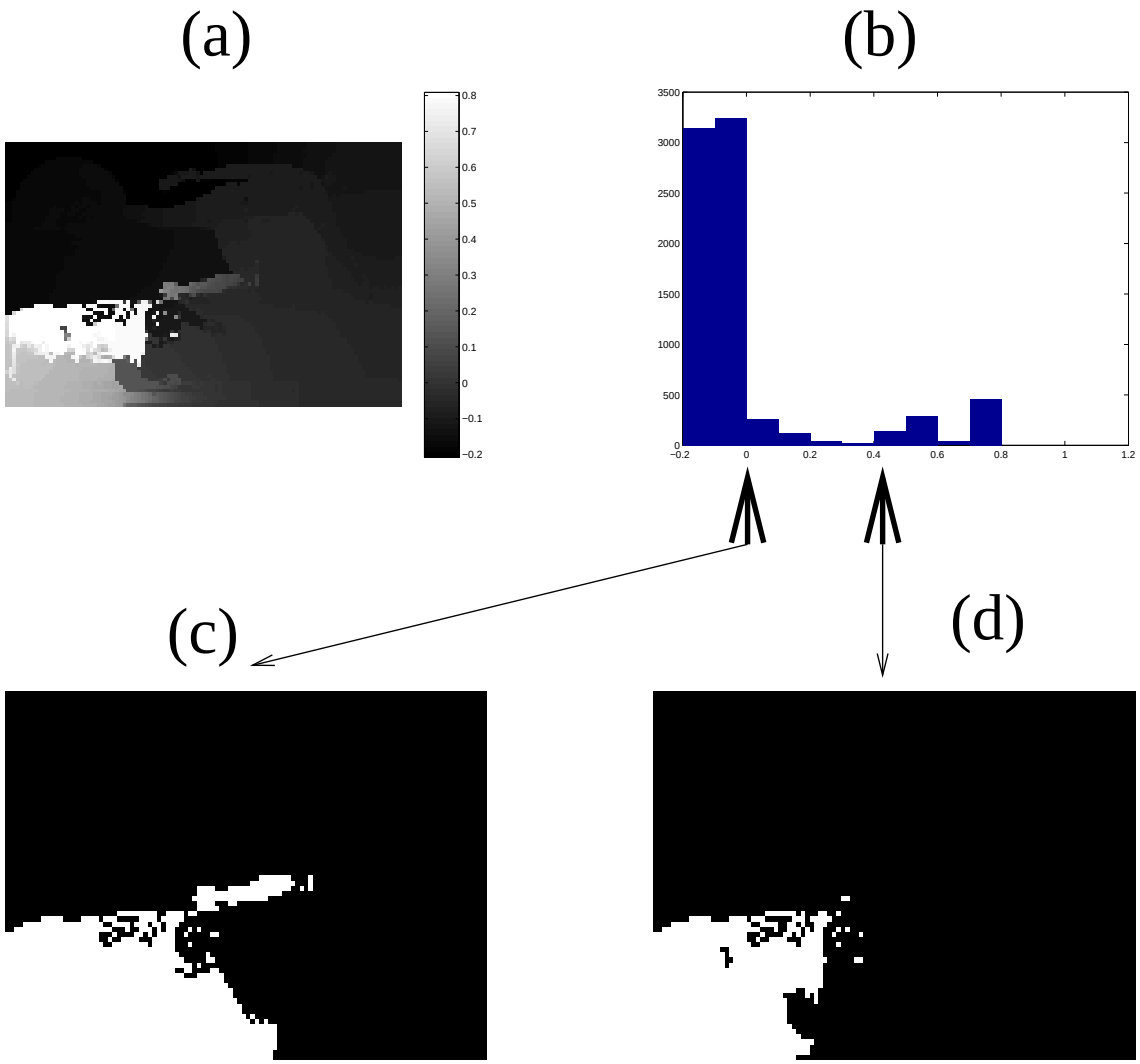

Figure 13: The eigenvector in (a) is a close approximation to a discrete partitioning indicator vector. Its histogram, shown in (b), indicates that the values in the eigenvector cluster around two extreme values. (c) and (d) shows the partitioning results with different splitting points indicated by the arrows in (b). The partition with the best normalized cut value is chosen.

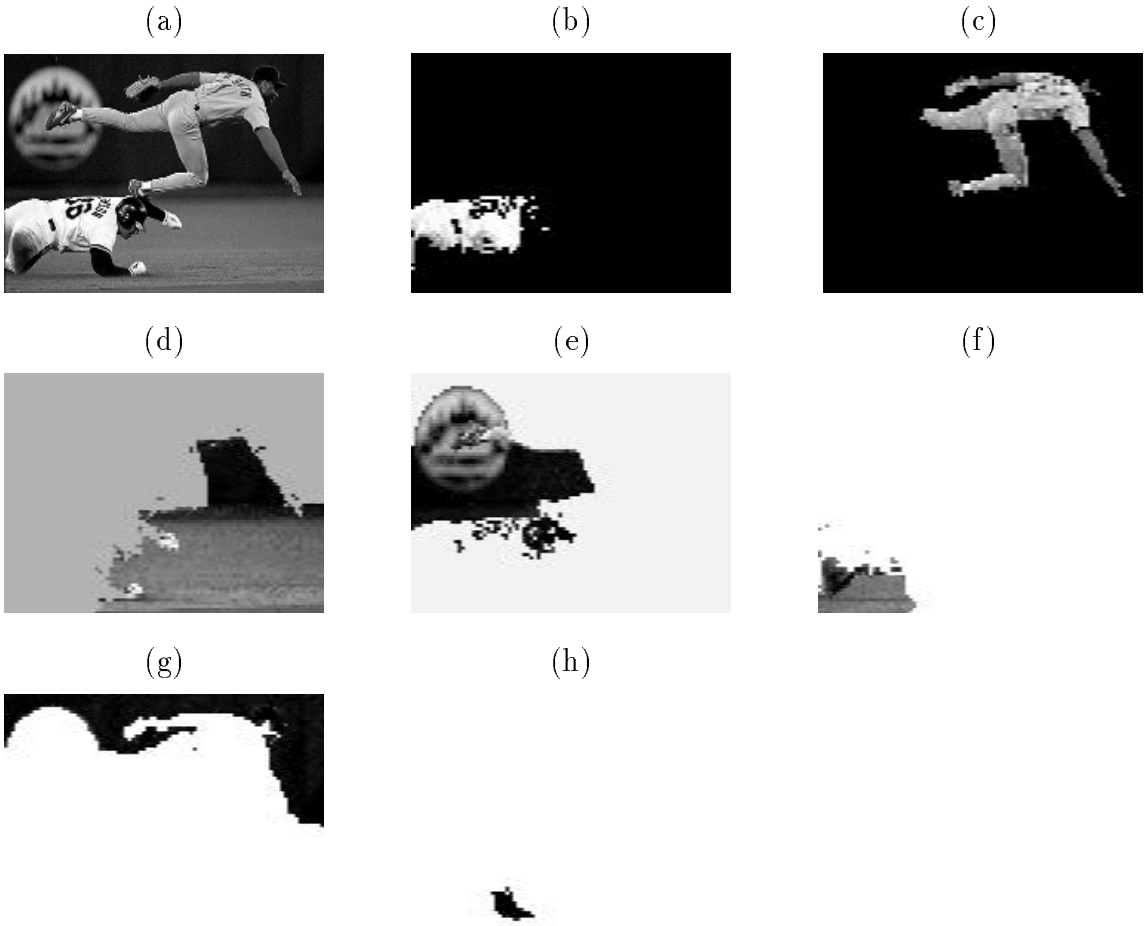

Figure 14: (a) shows the original image of size 80×100. Image intensity is normalized to lie within 0 and 1. Subplot (b) - (h) shows the components of the partition with $Ncut$ value less than 0.04. Parameter setting: $\sigma_I = 0.1$, $\sigma_X = 10.0$, $r = 10$.

3.3 Recursive 2-way Ncut

In summary, our grouping algorithm consists of the following steps:

1. Given a set of features, set up a weighted graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$, compute the weight on each edge, and summarize the information into $\mathbf{W}$, and $\mathbf{D}$.

2. Solve $(\mathbf{D} - \mathbf{W})\boldsymbol{x} = \lambda \mathbf{D}\boldsymbol{x}$ for eigenvectors with the smallest eigenvalues.

3. Use the eigenvector with second smallest eigenvalue to bipartition the graph by finding the splitting point such that $Ncut$ is maximized,

4. Decide if the current partition should be sub-divided by checking the stability of the cut, and make sure $Ncut$ is below pre-specified value. Recursively repartition the segmented parts if necessary.

The number of groups segmented by this method is controlled directly by the maximum allowed $Ncut$.

3.4 Simultanous K-way cut with multiple eigenvectors

One drawback of the recursive 2-way cut is its treatment of the oscillatory eigenvectors. The stability criteria provides us from cutting oscillatory eigenvectors, but it also prevents us cutting the subsequent eigenvectors which might be perfect partitioning vectors. Also the approach is computationally wasteful; only the second eigenvector is used whereas the next few small eigenvectors also contain useful partitioning information.

Instead of finding the partition using recursive 2-way cut as described above, one can use the all the top eigenvectors to simultanously obtain a K-way partition. In this method, the n top eigenvectors are used as n dimensional indicator vectors for each pixel. In the first step, a simple clustering algorithm, such as the k-means algorithm, is used to obtain an over-segmentation of the image into k' groups. No attempt is made to identify and exclude oscillatory eigenvectors–they exarcabete the oversegmentation, but that will be dealt with subsequently.

In the second step, one can proceed in the following two ways:

1. Greedy pruning: iteratively merge two segments at a time until only k segments are left. At each merge step, those two segments are merged that minimize the k-way $Ncut$ criterion defined as:

$$Ncut_k = \frac{cut(\mathbf{A}_1, \mathbf{V} - \mathbf{A}_1)}{assoc(\mathbf{A}_1, \mathbf{V})} + \frac{cut(\mathbf{A}_2, \mathbf{V} - \mathbf{A}_2)}{assoc(\mathbf{A}_2, \mathbf{V})} + ... + \frac{cut(\mathbf{A}_k, \mathbf{A} - \mathbf{A}_k)}{assoc(\mathbf{A}_k, \mathbf{V})}, \qquad (18)$$

   where $\mathbf{A}_i$ is the ith subset of whole set $\mathbf{V}$.

   This computation can be efficiently carried out by iteratively updating the compacted weight matrix $\mathbf{W}^c$, with $\mathbf{W}^c(i,j) = assoc(\mathbf{A}_i, \mathbf{A}_j)$.

2. Global recursive cut. From the initial k' segments we can build a condensed graph $\mathbf{G}^c = (\mathbf{V}^c, \mathbf{E}^c)$, where each segment $\mathbf{A}_i$ corresponds to a node $\mathbf{V}_i^c$ of the graph. The weight on each graph edge $\mathbf{W}^c(i,j)$ is defined to be $assoc(\mathbf{A}_i, \mathbf{A}_j)$, the total edge weights from elements in $\mathbf{A}_i$ to elements in $\mathbf{A}_j$. From this condensed graph, we then recursively bi-partition the graph according the $Ncut$ criteria. This can be carried out either with the generalized eigenvalue system as in section 3.3, or with exhaustive search in the discrete domain. Exhaustive search is possible in this case since k' is small, typically $k' \leq 100$.

We have experimented with this simultanous k-way cut method on our recent test images. However, the results presented in this paper are all based on the recursive 2-way partitioning algorithm outlined in the previous subsection 3.3.

4 Experiments

We have applied our grouping algorithm to image segmentation based on brightness, color, texture, or motion information. In the monocular case, we construct the graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ by taking each pixel as a node, and define the edge weight w_{ij} between node i and j as the product of a feature similarity term and spatial proximity term:

$$w_{ij} = e^{\frac{-\|\boldsymbol{F}(i) - \boldsymbol{F}(j)\|_2^2}{\sigma_I}} * \begin{cases} e^{\frac{-\|\boldsymbol{X}(i) - \boldsymbol{X}(j)\|_2^2}{\sigma_X}} & \text{if } \|\boldsymbol{X}(i) - \boldsymbol{X}(j)\|_2 < r \\ 0 & \text{otherwise} \end{cases}$$

where $\boldsymbol{X}(i)$ is the spatial location of node i, and $\boldsymbol{F}(i)$ is a feature vector based on intensity, color, or texture information at that node defined as:

- $\boldsymbol{F}(i) = 1$, in the case of segmenting point sets,
- $\boldsymbol{F}(i) = \boldsymbol{I}(i)$, the intensity value, for segmenting brightness images,
- $\boldsymbol{F}(i) = [v, v \cdot s \cdot sin(h), v \cdot s \cdot cos(h)](i)$, where h, s, v are the HSV values, for color segmentation,
- $\boldsymbol{F}(i) = [|\boldsymbol{I} * f_1|, ..., |\boldsymbol{I} * f_n|](i)$, where the f_i are DOOG filters at various scales and orientations as used in[16], in the case of texture segmentation.

Note that the weight $w_{ij} = 0$ for any pair of nodes i and j that are more than r pixels apart.

We first tested our grouping algorithm on spatial point sets similar to the one shown in figure (2). Figure (15) shows a point set and the segmentation result. As we can see from the figure, the normalized cut criterion is indeed able to partition the point set in a desirable way as we have argued in section (2).

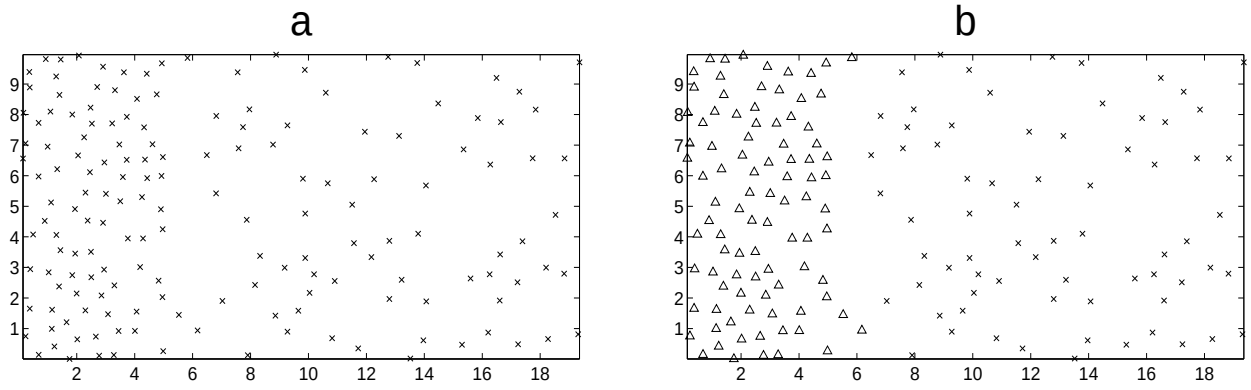

Figure 15: (a) Point set generated by two Poisson processes, with densities of 2.5 and 1.0 on the left and right clusters respectively, (b) $\triangle$ and $\times$ indicates the partition of point set in (a). Parameter settings: $\sigma_X = 5, r = 3$.

Figures (16), (17), (18), and (19) shows the result of our segmentation algorithm on various brightness images. Figure (16), (17) are synthetic images with added noise. Figure (18) and (19) are natural images. Note that the "objects" in figure (19) have rather ill-defined

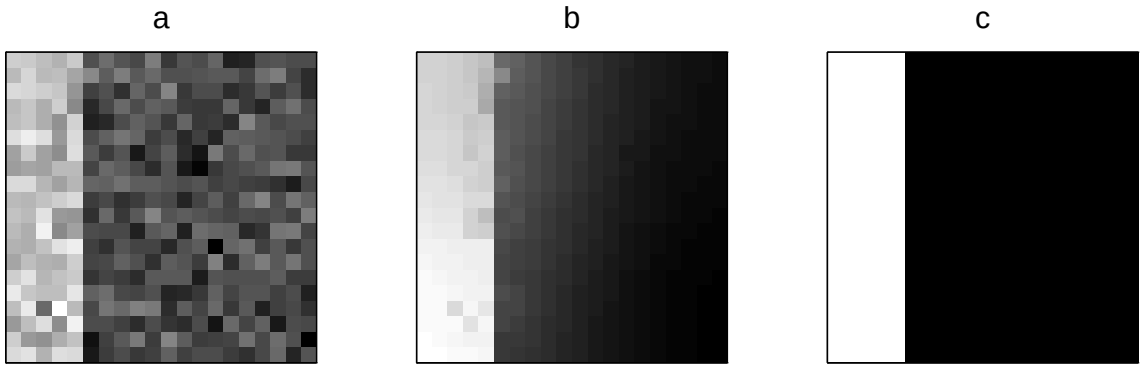

Figure 16: A synthetic image showing a noisy "step" image. Intensity varies from 0 to 1, and Gaussian noise with $\sigma = 0.2$ is added. Subplot (b) shows the eigenvector with the second smallest eigenvalue, and subplot (c) shows the resulting partition.

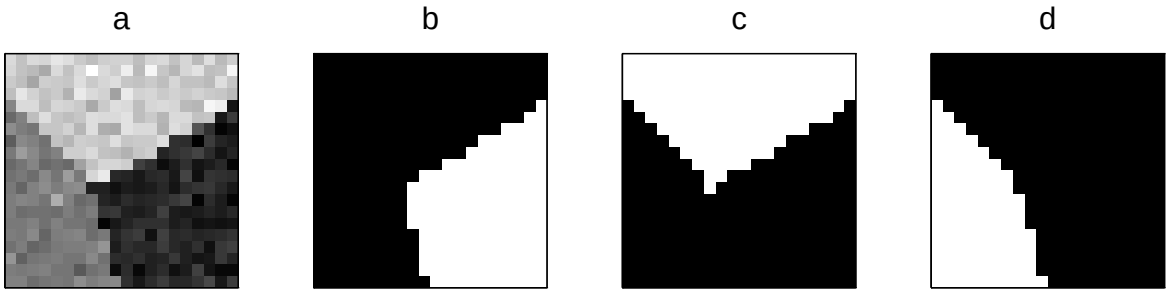

Figure 17: (a) A synthetic image showing three image patches forming a junction. Image intensity varies from 0 to 1, and Gaussian noise with $\sigma = 0.1$ is added. (b)-(d) shows the top three components of the partition.

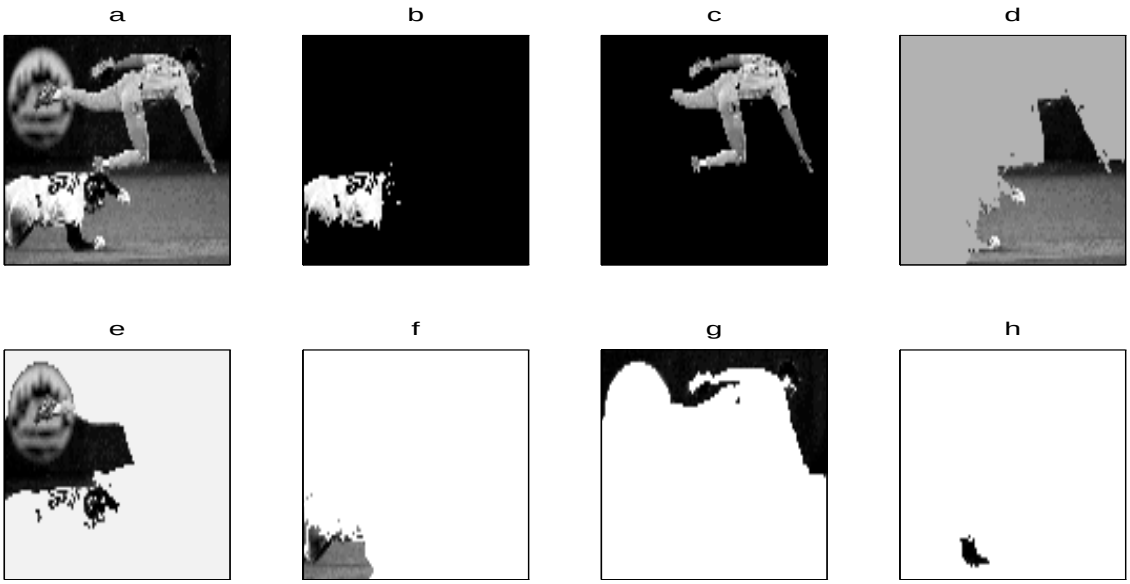

Figure 18: (a) shows a 80x100 baseball scene, image intensity is normalized to lie within 0 and 1. (b)-(h) shows the components of the partition with $Ncut$ value less than 0.04. Parameter setting: $\sigma_I = 0.01$, $\sigma_X = 4.0$, $r = 5$.

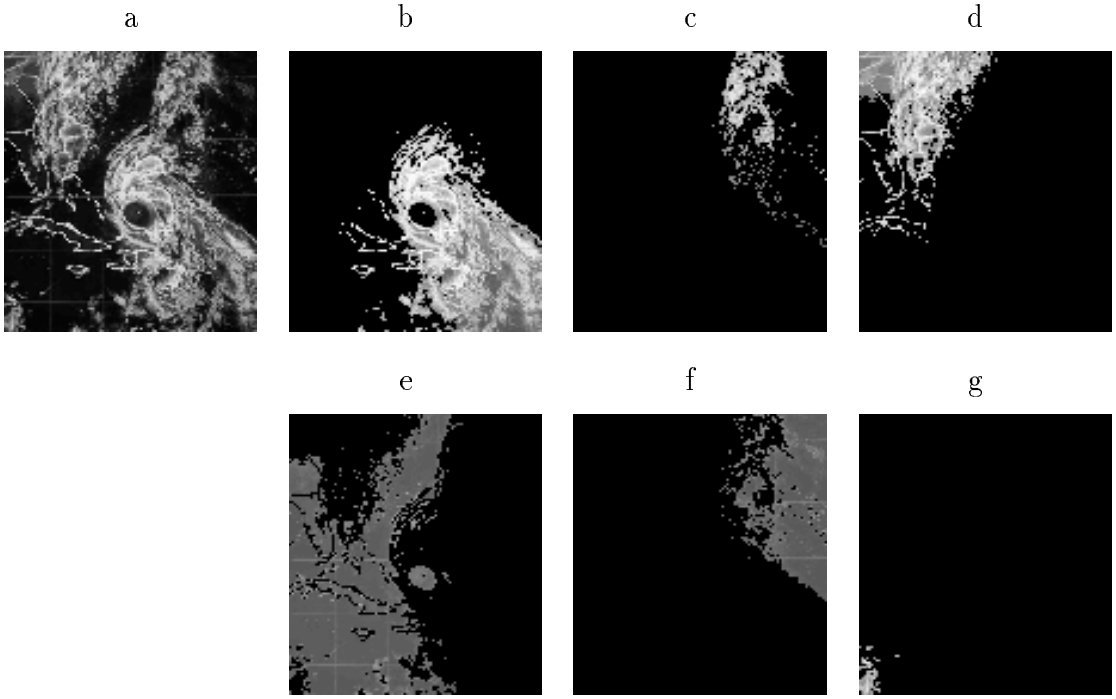

Figure 19: (a) shows a 126x106 weather radar image. (b)-(g) show the components of the partition with $Ncut$ value less than 0.08. Parameter setting: $\sigma_I = 0.005$, $\sigma_x = 15.0$, $r = 10$

boundaries which would make edge detection perform poorly. Figure (20) shows the segmentation on a color image, reproduced in gray scale in these transactions. The original image and many other examples can be found at web site *http://www.cs.berkeley.edu/~jshi/Grouping.*

Note that in all these examples the algorithm is able to extract the major components of scene, while ignoring small intra-component variations. As desired, recursive partitioning can be used to further decompose each piece.

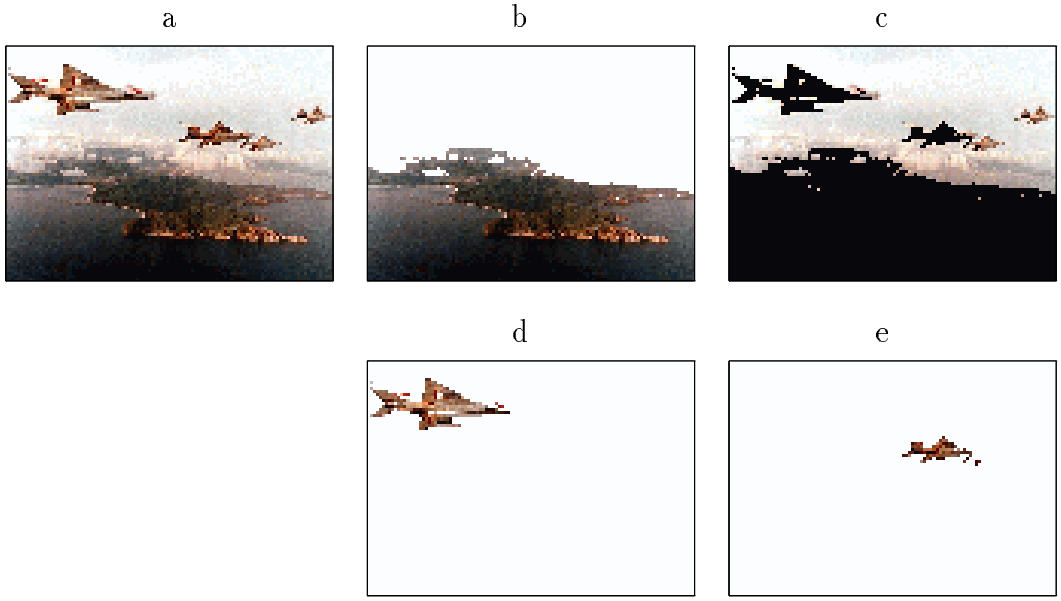

Figure 20: (a) shows a 77x107 color image. (b)-(e) show the components of the partition with $Ncut$ value less than 0.04. Parameter settings: $\sigma_I = 0.01$, $\sigma_X = 4.0$, $r = 5$.

Figure (21) shows preliminary results on texture segmentation for a natural image of a zebra against a background. Note that the measure we have used is orientation-variant, and therefore parts of the zebra skin with different stripe orientation should be marked as separate regions.

In the motion case, we will treat the image sequence as a spatiotemporal data set. Given an image sequence, a weighted graph is constructed by taking each pixel in the image sequence as a node, and connecting pixels that are in the spatiotemporal neighborhood of each other. The weight on each graph edge is defined as:

$$w_{ij} = \begin{cases} e^{\frac{-\mathbf{d}_m(i,j)^2}{\sigma_m^2}} & \text{if } \|\boldsymbol{X}(i) - \boldsymbol{X}(j)\|_2 < r \\ 0 & \text{otherwise} \end{cases}$$

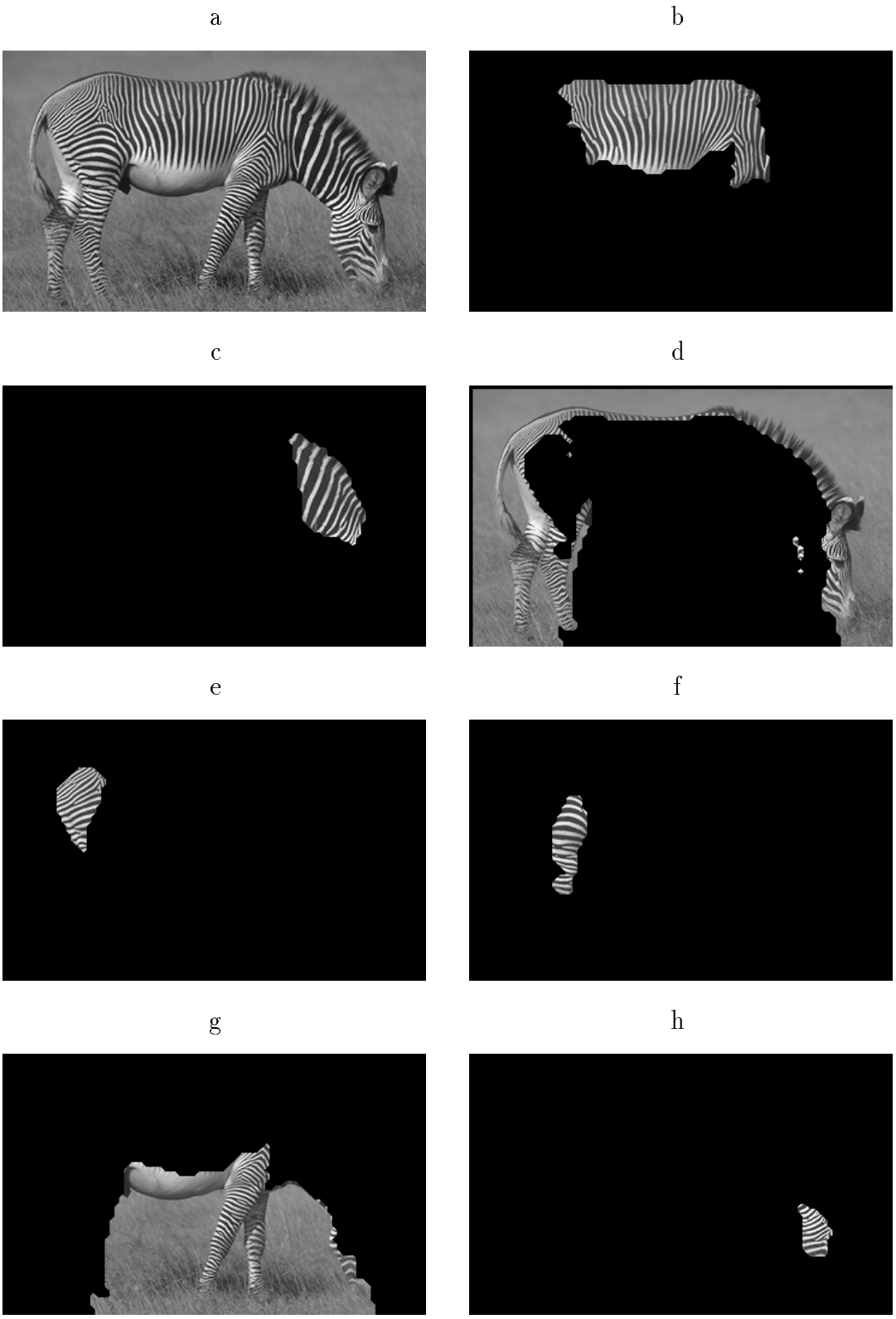

Figure 21: (a) shows an image of a zebra. The remaining images show the major components of the partition. The texture features used correspond to convolutions with DOOG filters[16] at 6 orientations and 5 scales.

where $\mathbf{d}(i,j)$ is the "motion distance" between two pixels i and j. Note that $\boldsymbol{X}_i$ in this case represents the spatial-temporal position of pixel i.

To compute this "motion distance", we will use a motion feature called *motion profile*. By *motion profile* we seek to estimate the probability distribution of image velocity at each pixel. Let $\boldsymbol{I}^t(\mathbf{X})$ denote a image window centered at the pixel at location $\mathbf{X} \in R^2$ at time t. We denote by $P_i(\mathbf{dx})$ the *motion profile* of an image patch at node i, $\boldsymbol{I}^t(\mathbf{X}_i)$, at time t corresponding to another image patch $\boldsymbol{I}^{t+1}(\mathbf{X}_i + \mathbf{dx})$ at time $t+1$. $P_i(\mathbf{dx})$ can be estimated by first computing the similarity $S_i(\mathbf{dx})$ between $\boldsymbol{I}^t(\mathbf{X}_i)$ and $\boldsymbol{I}^{t+1}(\mathbf{X}_i + \mathbf{dx})$, and normalizing it to get a probability distribution:

$$P_i(\mathbf{dx}) = \frac{S_i(\mathbf{dx})}{\sum_{\mathbf{dx}} S_i(\mathbf{dx})}. \tag{19}$$

There are many ways one can compute similarity between two image patches; we will use a measure that is based on the sum of squared differences(SSD):

$$S_i(\mathbf{dx}) = exp(-\sum_{\mathbf{w}} (\boldsymbol{I}^t(\mathbf{X}_i + \mathbf{w}) - \boldsymbol{I}^{t+1}(\mathbf{X}_i + \mathbf{dx} + \mathbf{w}))^2 / \sigma^2_{ssd}), \tag{20}$$

where $\mathbf{w} \in R^2$ is within a local neighborhood of image patch $\boldsymbol{I}^t(\mathbf{X}_i)$. The "motion distance" between two image pixels is then defined as one minus the cross-correlation of the motion profiles:

$$\mathbf{d}(i,j) = 1 - \sum_{\mathbf{dx}} P_i(\mathbf{dx}) P_j(\mathbf{dx}). \tag{21}$$

In figure (22) and (23) we show results of the normalized cut algorithm on a synthetic random dot motion sequence and a indoor motion sequence respectively. For more elaborate discussion on motion segmentation using normalized cut, as well as how to segment and track over long image sequences, readers might want to refer to our paper[21].

4.1 Computation time

As we saw from section 3.2, the running time of the normalized cut algorithm is $O(mn)$, where n is the number of pixels, and m is the number of steps Lanczos takes to converge. On the 100×120 test images shown here, the normalized cut algorithm takes about 2 minutes on a Intel Pentium 200MHz machines.

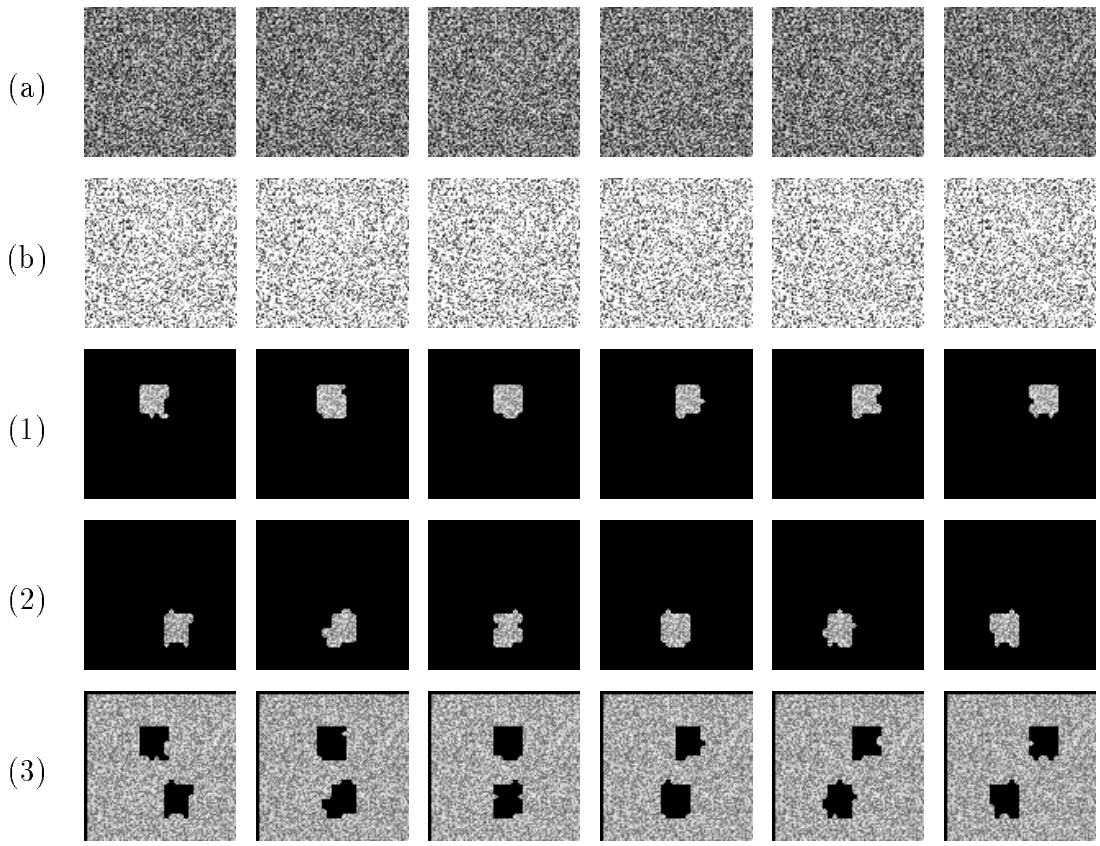

Figure 22: Row (a) of this plot shows the six frames of a synthetic random dot image sequence. Row (b) shows the outlines of the two moving patches in this image sequence. The outlines shown here are for illustration purposes only. Row (1)-(3) shows the top three partitions of this image sequence that have *Ncut* values less than 0.05. The segmentation algorithm produces 3D space-time partitions of the image sequence. Cross-sections of those partitions are shown here. The original image size is 100×100, and the segmentation is computed using image patches(superpixels) of size 3×3. Each image patch is connected to other image patches that are less than 5 superpixels away in spatial distance, and 3 frames away in temporal distance.

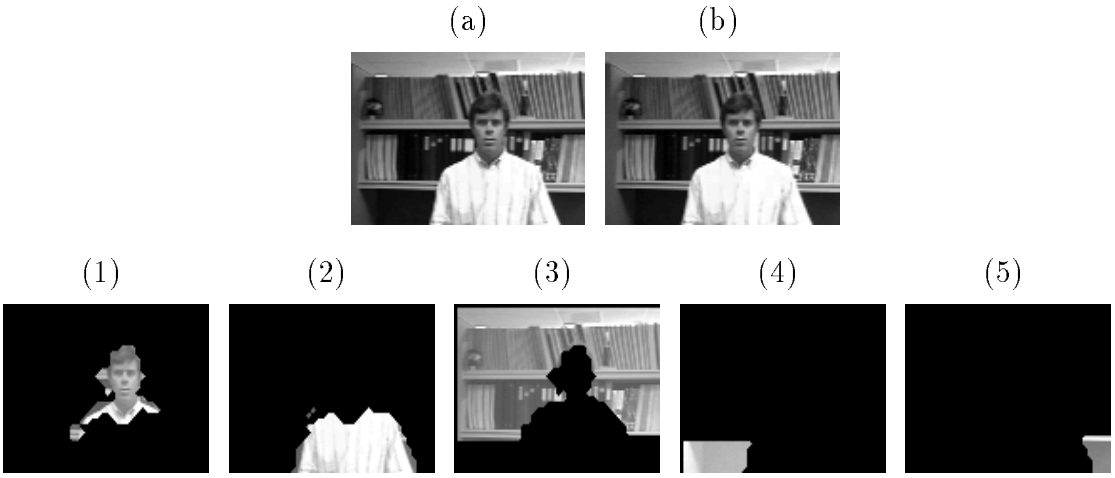

Figure 23: Subimage (a) and (b) shows two frames of an image sequence. Segmentation results on this two frame image sequence are shown in subimage (1) to (5). Segments in (1) and (2) correspond to the person in the foreground, and segments in (3) to (5) correspond to the background. The reason that the head of the person is segmented away from the body is that although they have similar motion, their motion profiles are different. The head region contains 2D textures and the motion profiles are more peaked, while in the body region the motion profiles are more spread out. Segment (3) is broken away from (4) and (5) for the same reason.

An multi-resolution implementation can be used to reduce this running time further on larger images. In our current experiments, with this implementation, the running time on a 300×400 image can be reduced to about 20 seconds on Intel Pentium 300MHz machines. Furthermore, the bottle neck of the computation, a sparse matrix-vector multiplication step, can be easily parallelized taking advantage of future computer chip designs.

In our current implementation, the sparse eigenvalue decomposition is computed using the LASO2 numerical package developed by D. Scott.

4.2 Choice of graph edge weight

In the examples shown here, we used an exponential function of the form of $w(x) = e^{-(d(x)/\sigma)^2}$, on the weighted graph edge with feature similarity of $d(x)$. The value of σ is typically set to $10-20\%$ of the total range of the feature distance function $d(x)$. The exponential weighting function is chosen here for its relatively simplicity as well as neutrality, since the focus of this paper is on developing a general segmentation procedure, given a feature similarity measure. We found this choice of weight function is quite adequate for typical image and feature spaces. Section 6.1 shows the effect of using different weighting functions and parameters on the output of the normalized cut algorithm.

However, the general problem of defining feature similarity incorporating a variety of cues is not a trivial one. The grouping cues could be of different abstraction levels and types, and they could be in conflict with each other. Furthermore, the weighting function could vary from image region to image region, particularly in a textured image. Some of these issues are addressed in [15].

5 Relationship to Spectral Graph Theory

The computational approach that we have developed for image segmentation is based on concepts from spectral graph theory. The core idea to use matrix theory and linear algebra to study properties of the incidence matrix, $\mathbf{W}$ and the Laplacian matrix, $\mathbf{D}-\mathbf{W}$, of the graph and relate them back to various properties of the original graph. This is a rich area

of mathematics, and the idea of using eigenvectors of the Laplacian for finding partitions of graphs can be traced back to Cheeger[4], Donath & Hoffman[7], and Fiedler[9]. This area has also seen contributions by theoretical computer scientists[1, 3, 22, 23]. It can be shown that our notion of *normalized cut* is related by a constant factor to the concept of conductance in[22].

For a tutorial introduction to spectral graph theory, we recommend the recent monograph by Fan Chung[5]. In this monograph, Chung[5] proposes a "normalized" definition of the Laplacian, as $\mathbf{D}^{-\frac{1}{2}}(\mathbf{D}-\mathbf{W})\mathbf{D}^{-\frac{1}{2}}$. The eigenvectors for this "normalized" Laplacian, when multiplied by $\mathbf{D}^{-\frac{1}{2}}$, are exactly the generalized eigenvectors we used to compute normalized cut. Chung points out that the eigenvalues of this "normalized" Laplacian relate well to graph invariants for general graph in ways that eigenvalues of the standard Laplacian has failed to do.

Spectral graph theory provides us some guidance on the goodness of the approximation to the normalized cut provided by the second eigenvalue of the normalized Laplacian. One way is through bounds on the normalized Cheeger constant[5] which in our terminology can be defined as

$$h_G = inf \frac{cut(A,B)}{min(assoc(A,V), assoc(B,V))}. \tag{22}$$

The eigenvalues of (6) are related to the Cheeger constant by the inequality[5]:

$$2h_G \geq \lambda_1 > \frac{h_G^2}{2}. \tag{23}$$

Earlier work on spectral partitioning used the second eigenvectors of the Laplacian of the graph defined as $\mathbf{D}-\mathbf{W}$ to partition a graph. The second smallest eigenvalue of $\mathbf{D}-\mathbf{W}$ is sometimes known as the Fiedler value. Several results have been derived relating the *ratio cut*, and the Fiedler value. A *ratio cut* of a partition of V, $P=(A, V-A)$, which in fact is the standard definition of the Cheeger constant, is defined as $\frac{cut(A,V-A)}{min(|A|,|V-A|)}$. It was shown that if the Fiedler value is small, partitioning graph based on the Fiedler vector will lead to good *ratio cut*[1][23]. Our derivation in section 2.1 can be adapted (by replacing the matrix $\mathbf{D}$ in the denominators by the identity matrix $\mathbf{I}$) to show that the Fiedler vector is a real valued solution to the problem of $min_{A \subset V} \frac{cut(A,V-A)}{|A|} + \frac{cut(V-A,A)}{|V-A|}$, which we can call the *average cut*.

Although *average cut* looks similar to the *normalized cut*, *average cut* does not have the important property of having a simple relationship to the *average association*, which can be analogously defined as $\frac{assoc(A,A)}{|A|} + \frac{assoc(V-A,V-A)}{|V-A|}$. Consequently, one can not simultaneously minimize the disassociation across the partitions, while maximizing the association within the groups. When we applied both techniques to the image segmentation problem, we found that the *normalized cut* produces better results in practice. There are also other explanations why the *normalized cut* has better behavior from graph theoretical point of view, as pointed out by Chung[5].

As far as we are aware, our work, first presented in[20], represents the first application of spectral partitioning to computer vision or image analysis. There is however one application area that has seen substantial application of spectral partitioning–the area of parallel scientific computing. The problem there is to balance the workload over multiple processors taking into account communication needs. One of the early papers is [18]. The generalized eigenvalue approach was first applied to graph partitioning by [8] for dynamically balancing computational load in a parallel computer. Their algorithm is motivated by [13]'s paper on representing a hypergraph in a Euclidean Space.

5.1 A physical interpretation

As one might expect, a physical analogy can be set up for the generalized eigenvalue system (6) that we used to approximate the solution of normalized cut. We can construct a spring-mass system from the weighted graph by taking graph nodes as physical nodes and graph edges as springs connecting each pair of nodes. Furthermore, we will define the graph edge weight as the spring stiffness, and the total edge weights connecting to a node as its mass.

Imagine what would happen if we were to give a hard shake to this spring-mass system forcing the nodes to oscillate in the direction perpendicular to the image plane. Nodes that have stronger spring connections among them will likely oscillate together. As the shaking become more violent, weaker springs connecting to this group of node will be over-stretched. Eventually the group will "pop" off from the image plane. The overall steady state behavior of the nodes can be described by its fundamental mode of oscillation. In fact, it can be

shown that the fundamental modes of oscillation of this spring mass system are exactly the generalized eigenvectors of (6).

Let k_{ij} be the spring stiffness connecting nodes i and j. Define $\mathbf{K}$ to be the $n \times n$ stiffness matrix, with $\mathbf{K}(i,i) = \sum_i k_{ij}$, and $\mathbf{K}(i,j) = -k_{ij}$. Define the diagonal $n \times n$ mass matrix $\mathbf{M}$ as $\mathbf{M}(i,i) = \sum_i k_{ij}$. Let $\mathbf{x}(t)$ be the $n \times 1$ vector describing the motion of each node. This spring-mass dynamic system can be described by:

$$\mathbf{K}\mathbf{x}(t) = -\mathbf{M}\ddot{(\mathbf{x})}(t). \tag{24}$$

Assuming the solution take the form of $\mathbf{x}(t) = \mathbf{v}_k cos(\omega_k t + \theta)$, the steady state solutions of this spring-mass system satisfy:

$$\mathbf{K}\mathbf{v}_k = \omega_k^2 \mathbf{M}\mathbf{v}_k, \tag{25}$$

analogous to equation (6) for *normalized cut*.

Each solution pair $(\omega_k, \mathbf{v}_k)$ of equation 25 describes a *fundamental mode* of the spring-mass system. The eigenvectors $\mathbf{v}_k$ give the steady state displacement of the oscillation in each mode, and the eigenvalues ω_k^2 give the energy required to sustain each mode of oscillation. Therefore, finding graph partitions that have small normalized cut values is, in effect, the same as finding a way to "pop" off image regions with minimal effort.

6 Relationship to other graph theoretic approaches to image segmentation

In the computer vision community, there has been some been previous work on image segmentation formulated as a graph partition problem. Wu&Leahy[25] use the *minimum cut* criterion for their segmentation. As mentioned earlier, our criticism of this criterion is that it tends to favor cutting off small regions which is undesirable in the context of image segmentation. In an attempt to get more balanced partitions, Cox *et.al.* [6] seek to minimize the ratio $\frac{cut(A,V-A)}{weight(A)}, A \subset V$, where $weight(A)$ is some function of the set A. When $weight(A)$ is taken to the be the sum of the elements in A, we see that this criterion becomes one of the

terms in the definition of *average cut* above. Cox *et. al.* use an efficient discrete algorithm to solve their optimization problem assuming the graph is planar.

Sarkar & Boyer[19] use the eigenvector with the largest eigenvalue of the system $\mathbf{W}\boldsymbol{x} = \lambda\boldsymbol{x}$ for finding the most coherent region in an edge map. Using a similar derivation as in section (2.1), we can see that the first largest eigenvector of their system approximates $min_{A\subset V}\frac{assoc(A,A)}{|A|}$, and the second largest eigenvector approximates $min_{A\subset V, B\subset V}\frac{assoc(A,A)}{|A|} + \frac{assoc(B,B)}{|B|}$. However, the approximation is not tight, and there is no guarantee that $A+B=V$. As we should see later in the section , this situation can happen quite often in practice. Since this algorithm is essentially looking for clusters that have tight within-grouping similarity, we will call this criteria *average association*.

6.1 Comparison with related eigenvector based methods

The *normalized cut* formulation has certain resemblance to the *average cut*, the standard spectral graph partitioning, as well as *average association* formulation. All these three algorithms can be reduced to solving certain eigenvalue systems. How are they related to each other?

Figure 24 summarizes the relationship between these three algorithms. On one hand, both the *normalized cut* and the *average cut* algorithm are trying to find a "balanced partition" of a weighted graph, while on the other hand, the *normalized association* and the *average association* are trying to find "tight" clusters in the graph. Since the *normalized association* is exactly $2 - ncut$, the *normalized cut* value, the *normalized cut* formulation seeks a balance between the goal of clustering and segmentation. It is, therefore, not too surprising to see that the normalized cut vector can be approximated with the generalized eigenvector of $(\mathbf{D}-\mathbf{W})\mathbf{x} = \lambda\mathbf{D}\mathbf{x}$, as well as that of $\mathbf{W}\mathbf{x} = \lambda\mathbf{D}\mathbf{x}$.

Judging from the discrete formulations of these three grouping criterion, it can be seen that the *average association*, $\frac{assoc(A,A)}{|A|} + \frac{assoc(B,B)}{|B|}$, has a bias for finding tight clusters. Therefore it runs the risk of becoming too greedy in finding small but tight clusters in the data. This might be perfect for data that are Gaussian distributed. However for typical data in the real world that are more likely to be made up of a mixture of various different types of

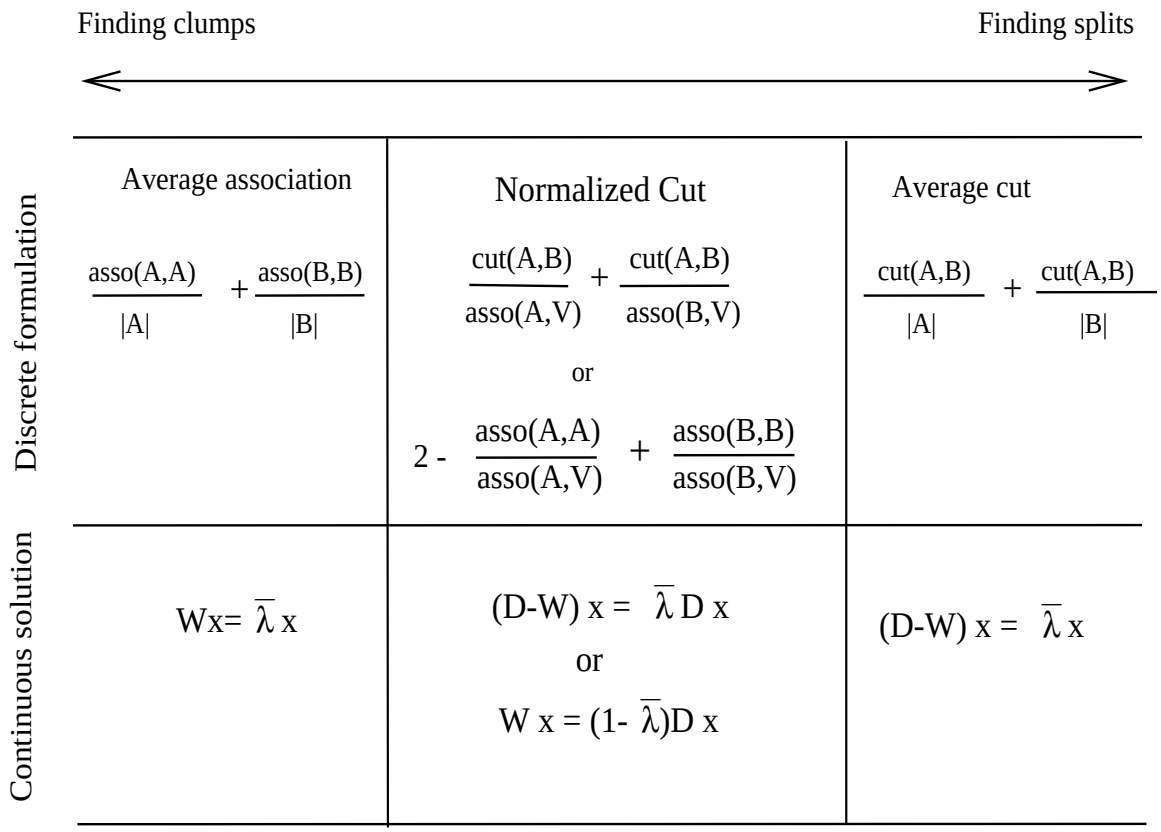

Figure 24: Relationship between *normalized cut* and other eigenvector based partitioning techniques. Compared to the *average cut* and *average association* formulation, *normalized cut* seeks a balance between the goal of finding clumps and finding splits.

distributions, this bias in grouping will have undesired consequences, as we shall illustrate in the examples below.

For *average cut*, $\frac{cut(A,B)}{|A|} + \frac{cut(A,B)}{|B|}$, the opposite problem arises – one can not ensure the two partitions computed will have tight within-group similarity. This becomes particularly problematic if the dissimilarity among the different groups varies from one to another, or if there are several possible partitions all with similar *average cut* values.

To illustrate these points, let us first consider a set of randomly distributed data in 1D shown in figure 25. The 1D data is made up by two subsets of points, one randomly distributed from 0 to 0.5, and the other from 0.65 to 1.0. Each data point is taken as a node in the graph, and the weighted graph edge connecting two points is defined to be inversely proportional to the distance between two nodes. We will use three monotonically decreasing weighting functions, $w(x) = f(d(x))$, defined on the distance function, $d(x)$, with different rate of fall-off. The three weighting functions are plotted in figure 26(a), 27(a), and 28(a).

The first function, $w(x) = e^{-(\frac{d(x)}{0.1})^2}$, plotted in figure 26(a), has the fastest decreasing rate among the three. With this weighting function, only close-by points are connected, as shown in the graph weight matrix $\mathbf{W}$ plotted in figure 26(b). In this case, *average association* fails to find the right partition. Instead it focuses on finding small clusters in each of the two main subgroups.

The second function, $w(x) = 1 - d(x)$, plotted in figure 27(a), has the slowest decreasing rate among the three. With this weighting function, most points have some non-trivial connections to the rest. To find a cut of the graph, a number of edges with heavy weights have to be removed. In addition, the cluster on the right has less within-group similarity comparing with the cluster on the left. In this case, *average cut* has trouble deciding on where to cut.

The third function, $w(x) = e^{-\frac{d(x)}{0.2}}$, plotted in figure 28(a), has a moderate decreasing rate. With this weighting function, the nearby point connections are balanced against far-away point connections. In this case, all three algorithms performs well with *normalized cut* producing a clearer solution than the two other methods.

These problems illustrated in figure 26, 27 and 28, in fact are quite typical in segmenting real natural images. This is particularly true in the case of texture segmentation. Different

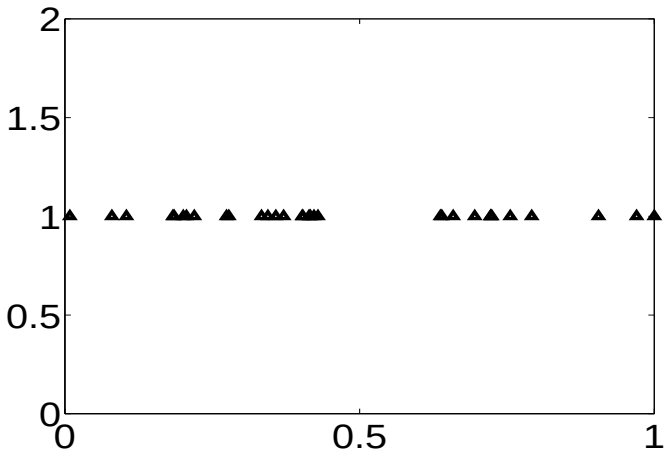

Figure 25: A set of randomly distributed points in 1D. The first 20 points are randomly distributed from 0.0 to 0.5, and the remaining 12 points are randomly distributed from 0.65 to 1.0. Segmentation result of these points with different weighting functions are show in figure 26, 27, and 28.

texture regions often have very different within-group similarity, or coherence. It is very difficult to pre-determine the right weighting function on each image region. Therefore it is important to design a grouping algorithm that is more tolerant to a wide range of weighting functions. The advantage of using *normalized cut* becomes more evident in this case. Figure 29 illustrates this point on a natural texture image shown previously in figure 21.

7 Conclusion

In this paper, we developed a grouping algorithm based on the view that perceptual grouping should be a process that aims to extract global impressions of a scene and provides a hierarchical description of it. By treating the grouping problem as a graph partitioning problem, we proposed the normalized cut criteria for segmenting the graph. Normalized cut is an unbiased measure of disassociation between sub-groups of a graph, and it has the nice property that minimizing normalized cut leads directly to maximizing the normalized association which is an unbiased measure for total association within the sub-groups. In finding an efficient algorithm for computing the minimum normalized cut, we showed that a generalized eigenvalue system provides a real valued solution to our problem.

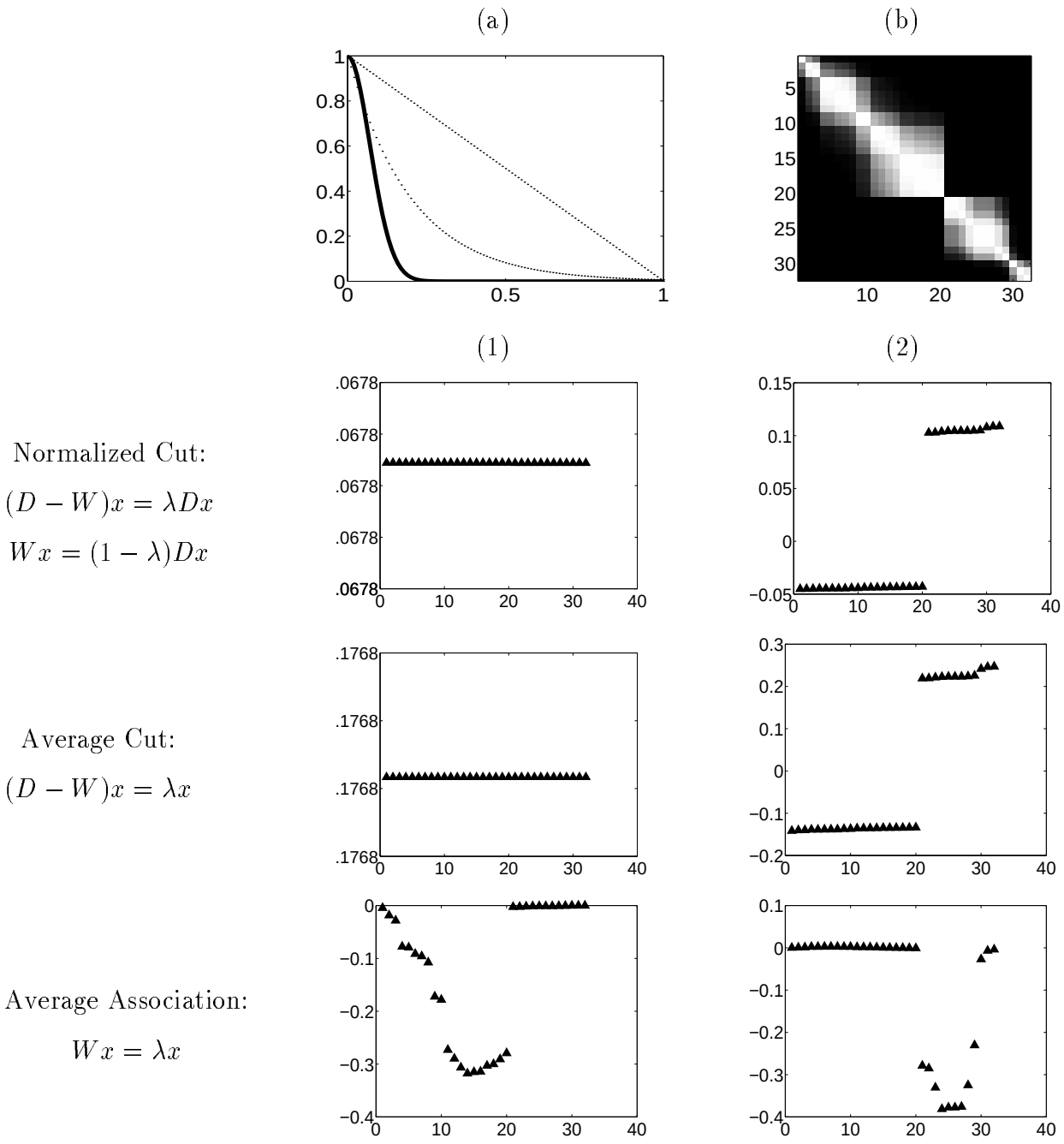

Figure 26: A weighting function with fast rate of fall-off: $w(x) = e^{-(\frac{d(x)}{0.1})^2}$, shown in subplot (a) in solid line. The dotted lines show the two alternative weighting functions used in figure 27 and 28. Subplot (b) shows the corresponding graph weight matrix $\boldsymbol{W}$. The two columns (1) and (2) below show the first, and second extreme eigenvectors for the Normalized cut(row 1), Average cut(row 2), and Average association(row 3). For both *normalized cut*, and *average cut*, the smallest eigenvector is a constant vector as predicted. In this case, both *normalized cut* and *average cut* perform well, while the *average association* fails to do the right thing. Instead it tries to pick out isolated small clusters.

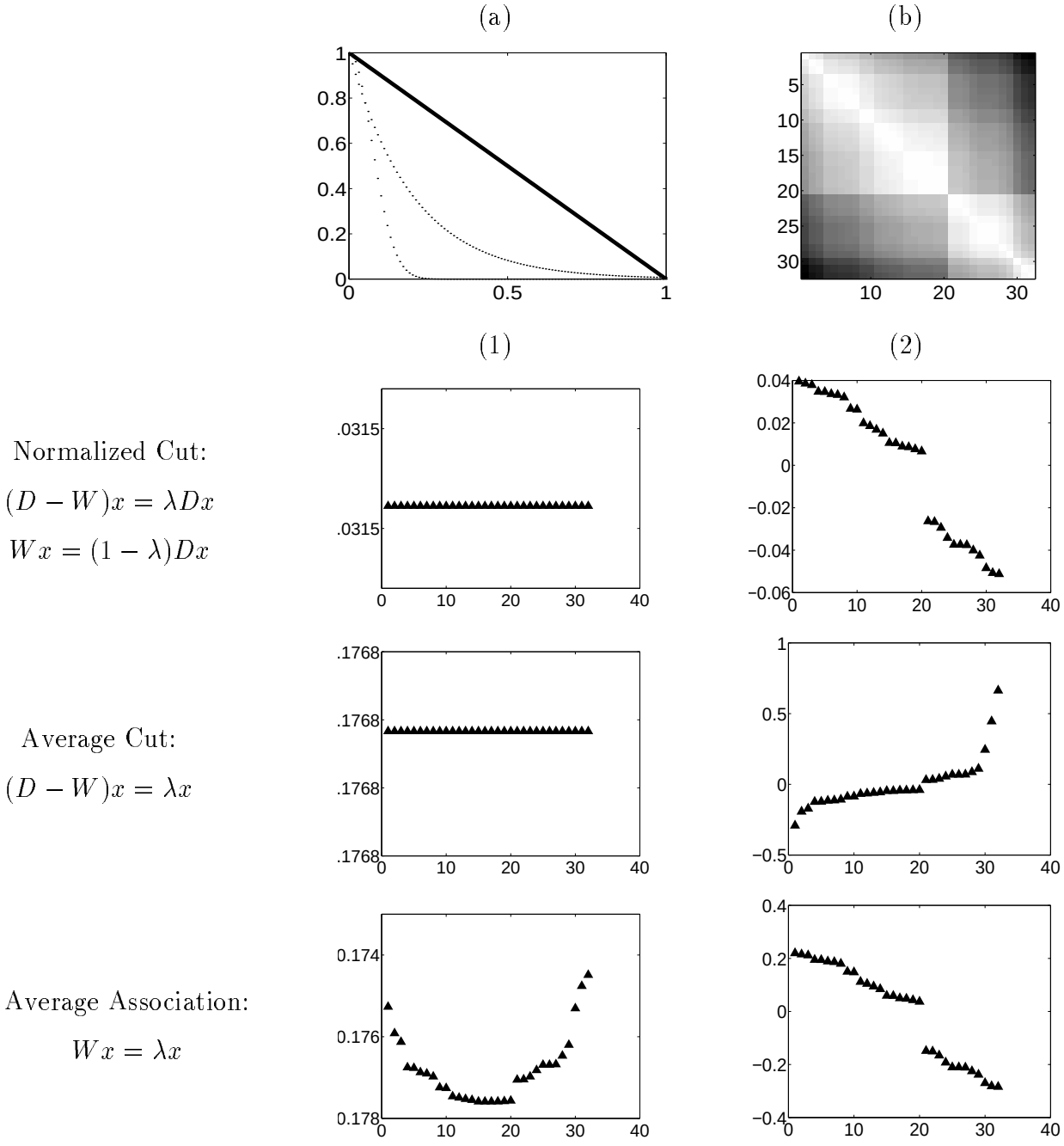

Figure 27: A weighting function with slow rate of fall-off: $w(x) = 1 - d(x)$, shown in subplot (a) in solid line. The dotted lines show the two alternative weighting functions used in figure 26 and 28. Subplot (b) shows the corresponding graph weight matrix $\boldsymbol{W}$. The two columns (1) and (2) below show the first, and second extreme eigenvectors for the Normalized cut(row 1), Average cut(row 2), and Average association(row 3). In this case, both *normalized cut* and *average association* give the right partition, while the *average cut* has trouble deciding on where to cut.

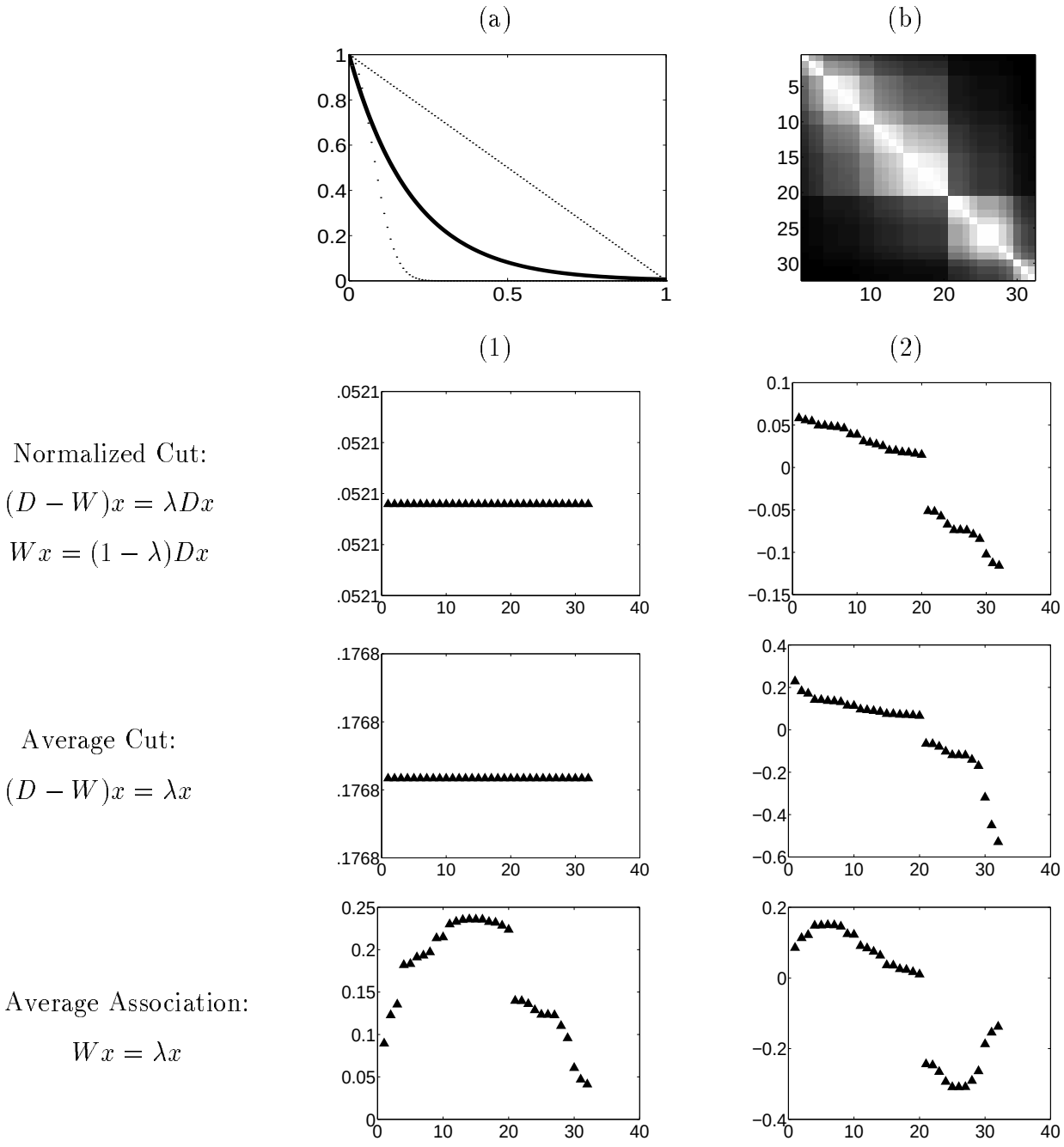

Figure 28: A weighting function with medium rate of fall-off: $w(x) = e^{-\frac{d(x)}{0.2}}$, shown in subplot (a) in solid line. The dotted lines show the two alternative weighting functions used in figure 26 and 28. Subplot (b) shows the corresponding graph weight matrix $\boldsymbol{W}$. The two columns (1) and (2) below show the first, and second extreme eigenvectors for the Normalized cut(row 1), Average cut(row 2), and average association(row 3). All these three algorithms perform satisfactorily in this case, with *normalized cut* producing a clearer solution than the other two cuts.

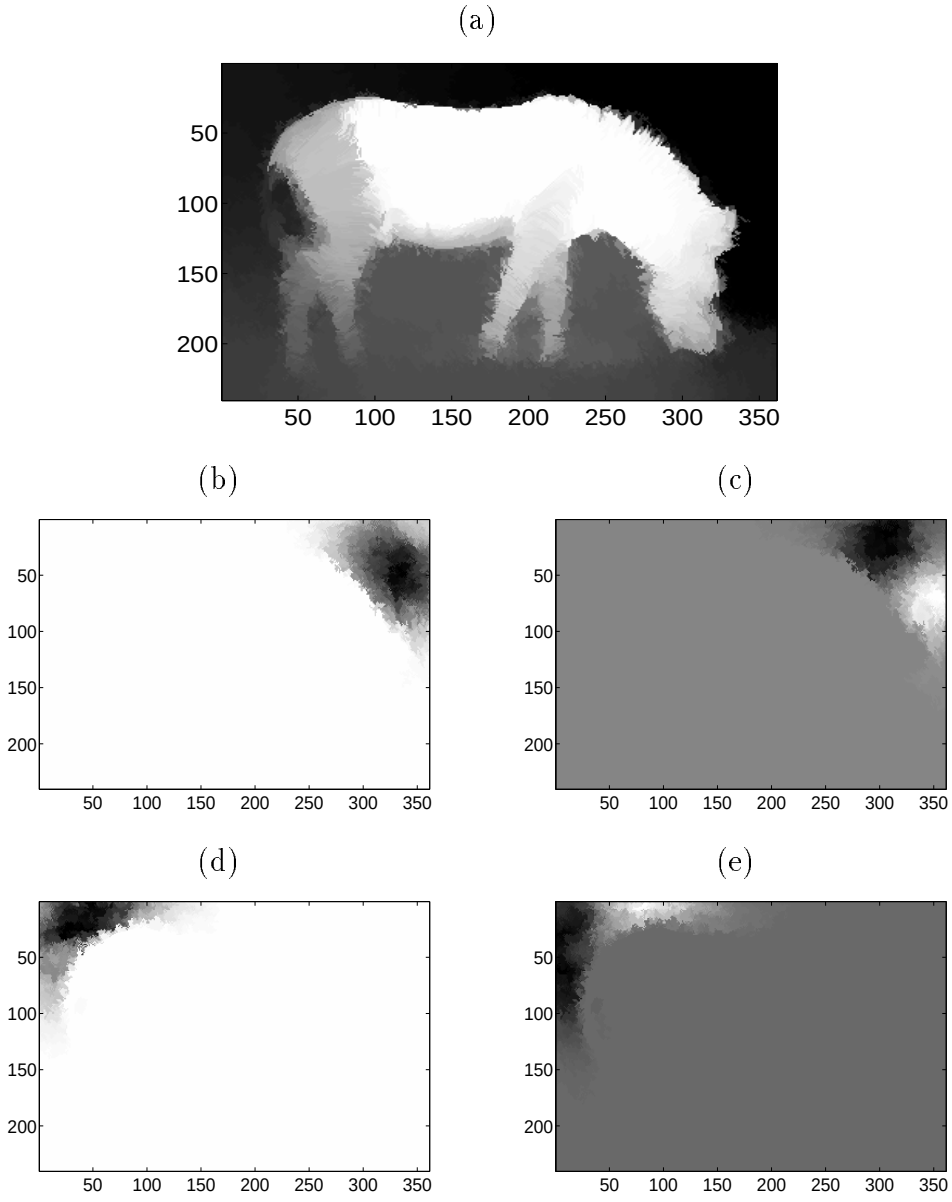

Figure 29: Normalized cut and average association result on the zebra image in figure 21. Subplot (a) shows the second largest eigenvector of $W\mathbf{x} = \lambda D\mathbf{x}$, approximating the normalized cut vector. Subplot (b) - (e) shows the first to fourth largest eigenvectors of $W\mathbf{x} = \lambda\mathbf{x}$, approximating the average association vector, using the same graph weight matrix. In this image, pixels on the zebra body have, on average, lower degree of coherence than the pixels in the background. The average association, with its tendency to find tight clusters, partitions out only small clusters in the background. The normalized cut algorithm, having to balance the goal of clustering and segmentation, finds the better partition in this case.

A computational method based on this idea has been developed, and applied to segmentation of brightness, color, and texture images. Results of experiments on real and synthetic images are very encouraging, and illustrate that the normalized cut criterion does indeed satisfy our initial goal of extracting the "big picture" of a scene.

8 Acknowledgment

This research was supported by (ARO)DAAH04-96-1-0341, and an NSF Graduate Fellowship to J. Shi. We thank Christos Papadimitriou for supplying the proof of NP-completeness for *normalized cuts* on a grid. In addition, we wish to acknowledge Umesh Vazirani and Alistair Sinclair for discussions on graph theoretic algorithms and Inderjit Dhillon and Mark Adams for useful pointers to numerical packages. Thomas Leung, Serge Belongie, Yeir Weiss and other members of the computer vision group at U.C. Berkeley provided much useful feedback on our algorithm.

References

[1] N. Alon. Eigenvalues and expanders. *Combinatorica*, 6(2):83–96, 1986.

[2] A. Blake and A. Zisserman. *Visual Reconstruction.* The MIT Press, 1987.

[3] R.B. Boppana. Eigenvalues and graph bisection: an average-case analysis. In *28nd Symposium on Foundations of Computer Science*, pages 280–285, 1987.

[4] J. Cheeger. A lower bound for the smallest eigenvalue of the laplacian. In R. C. Gunning, editor, *Problems in Analysis*, pages 195–199. Princeton Univ. Press, 1970.

[5] Fan R. K. Chung. *Spectral Graph Theory.* American Methematical Society, 1997.

[6] I.J. Cox, S.B. Rao, and Y. Zhong. Ratio regions: a technique for image segmentation. In *13th International Conference on Pattern Recognition*, 1996.

[7] W.E. Donath and A.J. Hoffman. Lower bounds for the partitioning of graphs. *IBM Journal of Research and Development*, pages 420–425, 1973.

[8] R. Van Driessche and D. Roose. An improved spectral bisection algorithm and its application to dynamic load balancing. *Parallel Computing*, 21:29–48, 1995.

[9] M. Fiedler. A property of eigenvectors of nonnegative symmetric matrices and its applications to graph theory. *Czech. Mathematical Journal*, 25(100):619–633, 1975.

[10] S. Geman and D. Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *PAMI*, 6:721–741, November 1984.

[11] G. H. Golub and C. F. Van Loan. *Matrix computations*. John Hopkins Press, 1989.

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

[13] K.Fukunaga, S. Yamada, H.S. Stone, and T. Kasai. A representation of hypergraphs in the euclidean space. *IEEE Trans. Comput.*, C-33:364–367, April 1984.

[14] Y.G. Leclerc. Constructing simple stable descriptions for image partitioning. *International Journal of Computer Vision*, 3:73–102, 1989.

[15] J. Malik, S. Belongie, J. Shi, and T. Leung. Textons, contours and regions: Cue integration in image segmentation. In *International Conf. on Computer Vision*, pages 918–925, 1999.

[16] J. Malik and P. Perona. Preattentive texture discrimination with early vision mechanisms. *Journal of Optical Society of America*, 7(2):923–932, May 1990.

[17] D. Mumford and J. Shah. Optimal approximations by piecewise smooth functions, and associated variational problems. *Comm. Pure Math.*, pages 577–684, 1989.

[18] A. Pothen, H.D. Simon, and K.P. Liou. Partitioning sparse matrices with eigenvectors of graphs. *SIAM Journal of Matrix Anal. Appl.*, 11:430–452, 1990.

[19] S. Sarkar and K.L. Boyer. Quantitative measures of change based on feature organization: Eigenvalues and eigenvectors. In *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, 1996.

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

[21] J. Shi and J. Malik. Motion segmentation and tracking using normalized cuts. In *International Conf. on Computer Vision*, pages 1154–1160, 1998.

[22] A.J. Sinclair and M.R. Jerrum. Approximative counting, uniform generation and rapidly mixing markov chains. *Information and Computation*, 82:93–133, 1989.

[23] D.A. Spielman and S.H. Teng. Disk packings and planar separators. In *Proceedings of 12th ACM Symposium on Computational Geometry*, May 1996.

[24] M. Wertheimer. Laws of organization in perceptual forms(partial translation). In W.B. Ellis, editor, *A Sourcebook of Gestalt Psycychology*, pages 71–88. Harcourt, Brace and Company, 1938.

[25] Z. Wu and R. Leahy. An optimal graph theoretic approach to data clustering: Theory and its application to image segmentation. *PAMI*, 11:1101–1113, November 1993.

A NP-Completeness Proof for Normalized Cut

Proposition 1 [Papadimitriou 97] *Normalized Cut(NCUT) for a graph on regular grids is NP-complete.*

Proof: We shall reduce NCUT on regular grids from PARTITION:

- Given integers $x_1, x_2, ..., x_n$ adding to $2k$, is there a subset adding to k?

We construct a weighted graph on a regular grid that has the property that it will have a small enough normalized cut if and only if we can find a subset from $x_1, x_2, ..., x_n$ adding to k. Figure 30(a) shows the graph and 30(b) shows the form that a partition that minimizes the normalized cut must take.

In comparison to the integers $x_1, x_2, ..., x_n$, M is much larger, $M > 2k^2$, and a is much smaller, $0 < a < 1/n$. We ask the question

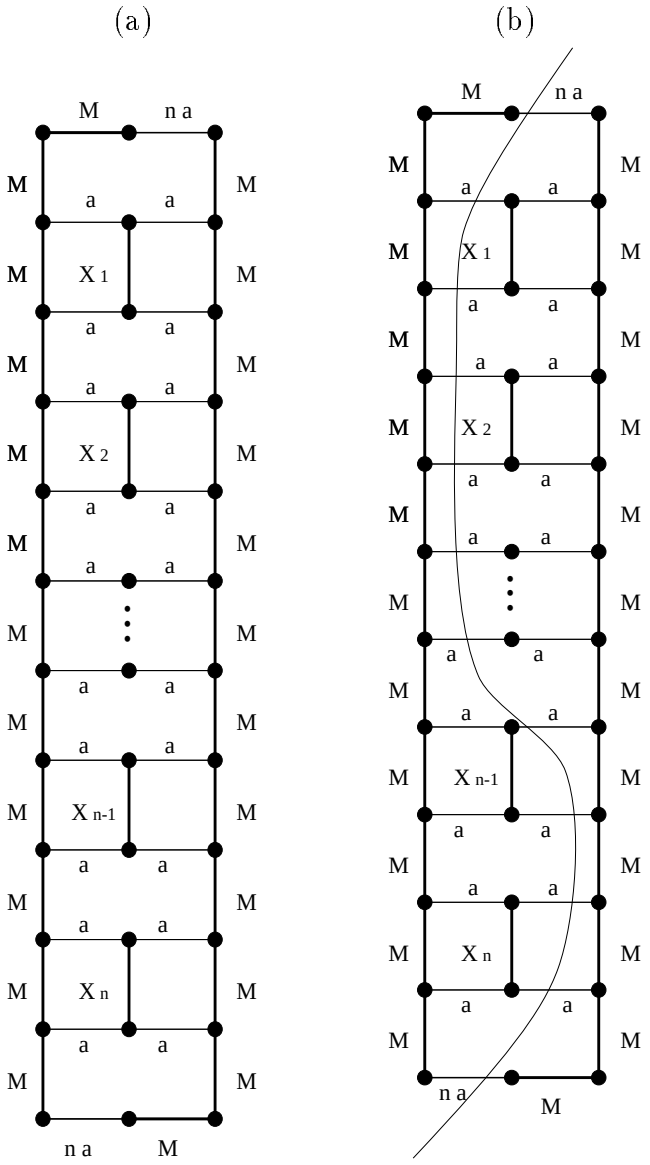

Figure 30: (a) shows a weighted graph on a regular grid. The missing edges on the grids have weights of 0. In comparison to the integers $x_1, x_2, ..., x_n$, M is a large number($M > 2k^2$), and a is very small number ($0 < a < 1/n$). (b) shows a cut that has a $Ncut$ value less than $\frac{4an}{c-1/c}$. This cut, which only goes through edges with weight equal to a or na, has the property that the x_i's on each side of the partition sum up to k.

- Is there a partition with $Ncut$ value less than $\frac{4an}{c-1/c}$, where c is half the sum of edge weights in the graph, $c = 2M(n+1) + k + 3a\,n$.

We shall see that a good $Ncut$ partition of the graph must separate the left and right columns. In particular, if and only if there is a subset $S_1 = \{x_1, ..., x_m\}$ adding to k, by taking the corresponding edges in the middle column to be in one side of the partition, as illustrated in figure 30(b), we achieve a $Ncut$ value less than $\frac{4an}{c-1/c}$. For all other partitions, the $Ncut$ value will be bounded below by $\frac{4an}{c-1/c}$.

First, let us show that the cut illustrated in 30(b), where each side has a subset of middle column edges $x_1, x_2, ..., x_n$ that add upto k, does have $Ncut$ value less than $\frac{4a\,n}{c-1/c}$ Let the $ncut^*$ be the $Ncut$ value for this cut. By using the formula for $Ncut$(equation 2.2), we can see that $ncut^* = \frac{4an}{2c+2a\,n(2k_1-1)} + \frac{4a\,n}{2c-2a\,n(2k_1-1)}$, where c is half the total edge weights in the graph, $c = 2M(n+1) + k + 3a\,n$, and $k_1 n$ and $(1-k_1)n$ are the number of edges from the middle column on the two sides of the graph partition, $0 < k_1 < 1$. The term $a\,n(2k_1 - 1)$ can be interpreted as the amount of imbalance between the denominators in the two terms in the $Ncut$ formula and lies between -1 and $+1$(since $0 < a\,n < 1$). Simplifying, we see that $ncut^* = \frac{4a\,n\,c}{c^2-(a\,n(2k_1-1))^2} < \frac{4a\,n\,c}{c^2-1} = \frac{4a\,n}{c-1/c}$. as was to be shown.

To complete the proof we must show that all other partitions result in a $Ncut$ greater than or equal to $\frac{4a\,n}{c-1/c}$. Informally speaking, what will happen is that either the numerators of the terms in the $Ncut$ formula–the cut become too large, or the denominators become significantly imbalanced, again increasing the $Ncut$ value. We need to consider three cases:

1. A cut that deviates from the cut in 1(b) slightly by re-shuffling some of the x_i edges, so that the sums of the x_i in each subset of the graph partition are no longer equal. For such cuts, the resulting $Ncut$ values are at best $ncut_1 = \frac{2a\,n}{c+x} + \frac{2a\,n}{c-x} = \frac{4a\,n\,c}{c^2-x^2}$. But since $x \geq 1$, we have $ncut_1 \geq \frac{4a\,n\,c}{c^2-1} = \frac{4a\,n}{c-1/c}$.

2. A cut that goes through any of the edges with weight M. Even with the denominators on both sides completely balanced, the $Ncut$ value $ncut_2 = \frac{2M}{c}$ is going to be larger than $\frac{4a\,n}{c-1/c}$. This is ensured by our choice in the construction that $M > 2k^2$. We have

to show that

$$\begin{aligned} \frac{2M}{c} &\geq \frac{4a\,n}{c-1/c}, \text{ or} \\ M &\geq 2a\,n\frac{c^2}{c^2-1} \end{aligned}$$

This is direct, since $a\,n < 1$ by construction, $\frac{c^2}{c^2-1} \leq \frac{81}{80}$ (using $k \geq 1$, $M \geq 2$, $c \geq 9$).

3. A cut that partitions out some of the nodes in the middle as one group. We see that any cut that goes through one of the x_i's can improve its $Ncut$ value by going through the edges with weight a instead. So we will focus on the case where the cut only goes through the weight a edges. Suppose that m edges of x_i's are grouped into one set, with total weight adding to x, where $1 < x < 2k$. The corresponding ncut value, $ncut_3(m) = \frac{4a\,m}{4a\,m+2x} + \frac{4a\,m}{8M(n+1)+4k+12a\,n-4a\,m-2x} = \frac{2a\,m}{c-d_m} + \frac{2a\,m}{c+d_m}$, where $d_m = 2M(n+1)+k+3a\,n-2a\,m-x > 2M(n+1)-k+3a\,n-2a\,m = x_l$. The lower bound on $ncut_3(m) = \frac{4a\,m\,c}{c^2-d_m^2}$ is then $ncut_l(m) = \frac{4a\,m\,c}{c^2-x_l^2}$. Further expansion of the $ncut_l(m)$ yields

$$\begin{aligned} ncut_l(m) &= \frac{4a\,m\,c}{c^2-x_l^2} \\ &= \frac{4a\,m\,c}{c^2-(B-2a\,m)^2}, \text{ where } B = 2M(n+1)-k+3a\,n \\ &= \frac{4a\,c}{\frac{c^2-B^2}{m} - 4a^2\,m + 4a\,B} \end{aligned}$$

One can check to see that $ncut_l(m)$ is a non-decreasing function, and has its minimum at $\frac{4a\,c}{(c^2-B^2)+4a\,B-4a^2}$ when $m = 1$.

In order to prove that $ncut_l(m) > \frac{4a\,n}{c-1/c}$, we need to establish the inequality

$$\begin{aligned} \frac{4a\,c}{(c^2-B^2)+4a\,B-4a^2} &\geq \frac{4a\,n\,c}{c^2-1} \text{ or} \\ \frac{1}{(c^2-B^2)+4a\,B-4a^2} &\geq \frac{n}{c^2-1} \text{ or} \\ ((c^2-B^2)+4a\,B-4a^2)n &\leq c^2-1 \text{ or} \\ (4c\,k-4k^2)n+4a\,n(c-2k-a)+1 &\leq c^2 \end{aligned}$$

using the fact that $c = B + 2k$. To continue, note that since $an < 1$, this will be true if

$$\begin{aligned} (4c\,k - 4k^2)n + 4(c - 2k) - 4a + 1 &\leq c^2 \text{ or if} \\ 4c\,k^2 + 4c - (4k^3 + 8k + 4a - 1) &\leq c^2, \end{aligned}$$

since $n < k$. Since $4k^3 + 8k + 4a - 1 > 0$, we only need to show that $4c\,k^2 + 4c < c^2$, or that $c > 4(k^2 + 1)$. This is so because $c = 2M(n + 1) + k + 3a\,n$ and $M > 2k^2$.

■