Hyperspectral image segmentation using spatial-spectral graphs

David B. Gillis* and Jeffrey H. Bowles
Naval Research Laboratory, Remote Sensing Division, Washington, DC 20375

ABSTRACT

Spectral graph theory has proven to be a useful tool in the analysis of high-dimensional data sets. Recall that, mathematically, a graph is a collection of objects (nodes) and connections between them (edges); a weighted graph additionally assigns numerical values (weights) to the edges. Graphs are represented by their adjacency whose elements are the weights between the nodes. Spectral graph theory uses the eigendecomposition of the adjacency matrix (or, more generally, the Laplacian of the graph) to derive information about the underlying graph. In this paper, we develop a spectral method based on the 'normalized cuts' algorithm to segment hyperspectral image data (HSI). In particular, we model an image as a weighted graph whose nodes are the image pixels, and edges defined as connecting spatial neighbors; the edge weights are given by a weighted combination of the spatial and spectral distances between nodes. We then use the Laplacian of the graph to recursively segment the image. The advantages of our approach are that, first, the graph structure naturally incorporates both the spatial and spectral information present in HSI; also, by using only spatial neighbors, the adjacency matrix is highly sparse; as a result, it is possible to apply our technique to much larger images than previous techniques. In the paper, we present the details of our algorithm, and include experimental results from a variety of hyperspectral images.

Keywords: Hyperspectral, segmentation, classification, spatial-spectral, graph, nonlinear

1. INTRODUCTION

Hyperspectral image (HSI) is fairly unique in that it contains both spectral (pixel spectra) and spatial (grayscale band images) components. To make full use of the data, it would appear reasonable that analytical algorithms should exploit both types of information in the scene. Unfortunately, it is far from obvious how to do this; the majority of current algorithms work mainly in either spectral space (by modeling the image spectra as points in high-dimensional band space) or spatial space (by applying grayscale image processing routines to the individual bands).

In this paper, we present a model that we feel can be used to jointly model both the spatial and spectral information. Our model uses mathematical graph theory to connect the (spatially contiguous) pixels in an image; the spectral image is encoded by weighing the edges via spectral similarity. Once the underlying graph has been constructed, various existing spectral graph algorithms (such as Laplacian eigenmaps [5] and normalized cuts [6]) can be applied to implement dimensionality reduction and/or image segmentation.

In mathematics, a *graph* is simply a collection of abstract objects (*nodes* or *vertices*) and pair-wise connections or relations between them (*edges*). A *weighted* graph additionally includes a notion of the 'strength' of a connection by assigning a number (*weight*) to each edge. A weighted graph can be naturally represented as an n-by-n matrix A (where n is the number of nodes); the entries of which are simply the edge weights between the corresponding nodes (or zero, if no connection exists). Recall that the set of all eigenvalues of a given matrix is known as the matrix *spectrum* (not to be confused with an image spectrum); spectral graph theory uses the eigenvalues and eigenvectors of the associated graph matrix in order to analyze the graph [13].

Given their abstract nature and ease of computer implementation, graphs are ubiquitous in many fields of computer science and discrete mathematics. As such, there is a long history of research and extremely optimized numerical algorithms that can be applied to a given graph. The 'trick' is to develop a graph model for a given data set that can then be exploited to derive desired information.

In recent years there have been a number of new spectral graph algorithms introduced to analyze high-dimensional data, particularly in nonlinear dimensionality reduction [3,4] and image segmentation [6]. In addition, recent authors have been applying graph theoretical techniques explicitly to HSI, including methods for dimensionality reduction [2], anomaly detection [8], and classification [7,9].

* Please address any correspondence to: David.Gillis@nrl.navy.mil; (202) 767-5248

Algorithms and Technologies for Multispectral, Hyperspectral, and Ultraspectral Imagery XVIII,
edited by Sylvia S. Shen, Paul E. Lewis, Proc. of SPIE Vol. 8390, 83901Q · © 2012 SPIE
CCC code: 0277-786X/12/$18 · doi: 10.1117/12.919743

The model presented in this paper builds on this work, by introducing a graph structure (based on the normalized cut and Laplacian eigenmap algorithms) that encapsulates the spatial and spectral information in HSI. We then use this graph structure to solve a generalized eigenvalue problem for the graph Laplacian that can be used for either dimensionality reduction or segmentation. We also note that the graph structure is highly sparse (that is, each node in the graph has only a few edges); the advantage of this is that the eigenvalue problem is also highly sparse; as a result we are able to handle realistic size images (currently up to 1,000,000 pixels) on a standard desktop PC.

The rest of the paper is structured as follows: in the next section, we present a relatively large amount of background material, summarizing the basic graph theory we need, as well as a brief discussion of several algorithms we use in our model. In Sec. 3 we present the details of our model, and present experiment results of the algorithm on real-world data in Sec. 4.

2. BACKGROUND

2.1 Graph Theory

Mathematically, graph theory is concerned with the study of objects and the pair-wise relationships between them. More formally, a graph $G = (V, E)$ consists of a set of abstract objects called *vertices* (or sometimes *nodes*) $V = (v_1, \ldots, v_n)$ and *edges* $E = \{e_{i,j}\}$ that represent connections between vertices; $e_{i,j} \in E$ implies that nodes v_i and v_j are connected. A graph is *undirected* if the order of the nodes connected by an edge is irrelevant; this implies that edges are symmetric, that is $e_{i,j} \in E$ implies that $e_{j,i} \in E$. A graph is *weighted* if each edge $e_{i,j}$ has an associated number (weight) $w_{i,j}$. In undirected graphs, we always have $w_{i,j} = w_{j,i}$. For convenience, nodes that are not connected are assigned a weight of zero; $w_{i,j} = 0$ if and only if $e_{i,j} \notin E$. In this paper, all graphs will be assumed to be weighted and undirected.

For a given graph, a *path* is an ordered list of vertices such that each pair of consecutive vertices has an edge between them. Two nodes are *connected* if there exists a path between them. The *connected component* of a given node is the list all of nodes that are connected to it; a graph is said to be connected if the connected component for each node is equal to the entire graph; in other words, there exists a path between every pair of nodes in the graph. Finally, a graph is said to be *complete* if there exists an edge between every pair of nodes; note that the number of edges $|E|$ in a complete graph is quadratic in the number of vertices: $|E| = n \times (n-1)$ where *n* is the number of nodes in the graph.

To represent a graph, we can assume without less of generality that the nodes are represented by the numbers 1,2,…,n. There are two traditional methods for storing the edges: the first is to associate with each node a list of all other nodes that are connected via an edge to the given node. An alternative approach is to store the edge information in an $n \times n$ matrix M. Depending on the application, different matrices can be used to store the edge information.

The most common matrix used is the *adjacency* matrix A, whose entries are simply the weights of the associated edge; $a_{i,j} = w_{i,j}$. Note that we assume that nodes are not connected to themselves (that is, the graph has no loops) and so the diagonals are all zero ($a_{i,i} = 0$). The *degree* $\deg(v_i)$ of a node is defined to be the sum of all of the edge weights incident to the node.

$$\deg(v_i) = \sum_j w_{i,j}$$

The degree matrix D is the diagonal matrix whose entries are the degrees of the nodes: $d_{i,j} = \deg(v_i)$ for $i = j$ and zero elsewhere. Finally, the *Laplacian* matrix L is defined to be the difference of the diagonal and adjacency matrices; $L = D - A$.

As an example, the adjacency matrix for the graph in Fig. 1 is

$$A = \begin{bmatrix} 0 & 26.2 & 0 & 20.12 & 0 \\ 26.2 & 0 & 8.13 & 6.4 & 0 \\ 0 & 8.13 & 0 & 0 & 2.24 \\ 20.12 & 6.4 & 0 & 0 & 11.29 \\ 0 & 0 & 2.24 & 11.29 & 0 \end{bmatrix}$$

The degree matrix is

$$D = \begin{bmatrix} 46.32 & 0 & 0 & 0 & 0 \\ 0 & 40.73 & 0 & 0 & 0 \\ 0 & 0 & 10.37 & 0 & 0 \\ 0 & 0 & 0 & 37.81 & 0 \\ 0 & 0 & 0 & 0 & 13.53 \end{bmatrix}$$

Finally, the Laplacian is

$$L = \begin{bmatrix} 46.32 & -26.2 & 0 & -20.12 & 0 \\ -26.2 & 40.73 & -8.13 & -6.4 & 0 \\ 0 & 8.13 & 10.37 & 0 & -2.24 \\ -20.12 & -6.4 & 0 & 37.81 & -11.29 \\ 0 & 0 & -2.24 & -11.29 & 13.53 \end{bmatrix}$$

In the most general of terms, the aim of spectral graph theory is to use the eigenvalues and eigenvectors of (one of) the matrix graph representations to develop intuition about the underlying graph (and, more generally, the original objects of study). In the next few subsections we show two particular applications that are particularly suited to hyperspectral image analysis; namely, dimensionality reduction and image segmentation.

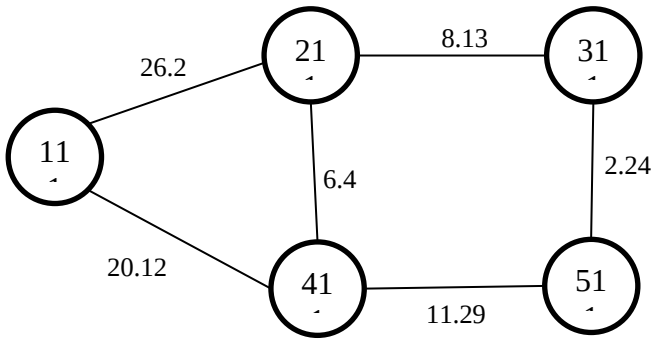


Figure 1. An example of a (weighted and undirected) graph. The large circles represent the (five) nodes, the lines represent the edges, and the numbers represent the weights. For example, the weight of the edge between nodes 2 and 3 is $w_{2,3} = 8.13$.

2.2 Dimensionality Reduction

In this subsection we discuss three related methods for dimensionality reduction that can be thought of as spectral graph approaches, namely, multidimensional scaling (MDS), ISOMAP, and Laplacian eigenmaps (LEM).

2.2.1 Multidimensional Scaling

The aim of MDS is to construct an isometric embedding from points in some high-dimensional space $\Re^m$to a lower-dimensional set of points in $\Re^k$, where $k < m$. More precisely, given points $y_1, \ldots, y_n \in R^m$, we would like to find a set of points $x_1, \ldots, x_n \in \Re^k$ such that the interpoint distances are preserved:

$$\|y_i - y_j\| = \|x_i - x_j\| \quad \forall i, j \tag{1}$$

where here $\|\cdot\|$ is the standard Euclidean norm. To create the embedding, we define the matrix A as

$$a_{i,j} = -\frac{1}{2}\|y_i - y_j\|. \tag{2}$$

Let $H = I - \frac{1}{n}1$ be the centering matrix, where I is the n-by-n identity matrix, and 1 is the n-by-n matrix with all entries equal to 1, and let $B = HAH$. By a very nice theorem of Householder and Young, it can be shown [1] that there exists a set of points $x_i \in \Re^k$ satisfying Eq. 1 if and only if B is positive semidefinite of rank *k*. Assuming B is so, let $\lambda_1 > \cdots >$

$\lambda_k > 0$ be the nonzero (real) eigenvalues of B with corresponding eigenvectors $v_1, \ldots, v_k$, normalized so that $\lambda_i^2 = \|v_i\|$, and let X by the n-by-k matrix whose columns are the eigenvectors. Then the *rows* of X, $x_1, \ldots, x_n$ define the embedding $y_i \in \Re^m \mapsto x_i \in \Re^k$.

In the context of graph theory, our original points y_i define the nodes of a graph with weights given the by the interpoint distances $w_{i,j} = -\frac{1}{2}\|y_i - y_j\|$ and an adjacency matrix defined by Eq. 2 (or, more generally, after centering by H). The eigendecomposition of this matrix thus allows us to construct a smaller dimensionality representation of the data that preserves the inter-point distances of the original data.

2.2.2. ISOMAP

In MDS, the strength of the connections between the original data points is given by the Euclidean distance; however, there is no *a priori* reason for using only this distance. For example, if the original data points lay on a curved surface (e.g. a manifold), such as the surface of the earth, then we wish way to measure distances along that surface (or the *geodesic* distance). The celebrated ISOMAP algorithm [4] is one attempt to do this. In particular, ISOMAP replaces the Euclidean distance (Eq. 2) by an approximate value of the geodesic distance $d(y_i, y_j)$ to define a new distance matrix A. Once this has been done, ISOMAP works exactly as MDS: the distance matrix is centered and eigenvalues / eigenvectors are used to define an embedding. The 'trick' is to find a way to estimate the geodesic distance. In ISOMAP, this is done by first defining an auxiliary graph that connects to each node some number of its neighbors in the original space. Once that is done, the distance between *any* two points (not necessarily connected in this graph) can be found by finding the shortest path between them. This shortest-path distance defines the geodesic distance.

It is very important to note that both MDS and ISOMAP are *complete* graphs; that is, every node is connected to every other node and given an appropriate weight. As a result, the corresponding adjacency or distance matrix is *full*; that is, nearly every element is non-zero. As we have noted, the number of edges in a complete graph scales quadratically with the number of nodes. As a consequence, the computational and storage complexity of the eigendecomposition can quickly become overwhelming, especially in the case of image analysis (in which every pixel is a node in the graph). For example, a 100-by-100 image (n = 10,000) would have an adjacency matrix of size ~ 400 megabytes in the case of a complete graph (here we assume all weights are stored as 32-bit floating point numbers). A 'megapixel' 1000-by-1000 image would require ~ 4,000,000 megabytes – or 4 *terabytes* – simply to store the matrix. It is clear that, for modern sized images, complete graphs are simply too big to be handled.

To get around this limitation, there are a couple of possibilities. The first is to simply choose a small, representative subset of the data (known as *landmarks* or *exemplars*) and perform the eigenanalysis using only these points; the remaining points can be projected using various methods (often called out-of-sample extensions). An alternative approach would be to limit the number of edges in the graph. In this case, the resulting adjacency matrix would be *sparse*; that is, most of the entries would be zero. In this case, highly optimized sparse matrix eigensolvers are available, allowing us to work with a relatively large number of nodes. This latter approach forms the basis of the next subsections, as well as our own spatial-spectral method presented in Section 3.

2.2.3 Laplacian Eigenmaps

Laplacian eigenmaps (LEM) are another method for dimensionality reduction that, like ISOMAP, does not assume the original data lies within a linear subspace, but instead on a low-dimensional surface (manifold) within the high-dimensional object space. Unlike ISOMAP, the underlying matrix representation of the object graph is sparse, allowing LLE to be run against relatively large graphs (e.g. large number of nodes).

As above, we begin with a set of points $y_1, \ldots, y_n \in R^m$, and wish to find to a new set of lower-dimensional points $x_1, \ldots, x_n \in \Re^k$ that 'preserve' distance between the points. Intuitively, LLE does this by forcing points that are 'close' in the original space to remain 'close' in the projected space.

The first step in the LLE algorithm is to construct a weighted graph to model the original data points. Unlike MDS and ISOMAP, LLE only connects a given node to those nodes that are 'close' (in the Euclidean sense) to it; in practice, this means connecting a given node to its N nearest neighbors (for some choice of N), or connecting any node that lies within ϵ of the given node. Next, edges are weighted using a heat (or radial basis) kernel:

$$w_{i,j} = e^{-\frac{\|y_i - y_j\|}{\sigma}}$$

for some user defined parameter σ. Note that, unlike the previous two approaches we have discussed, the weights in LLE are stronger when the points are close together; as the distance between two nodes increases, the weights quickly go to zero.

The next step in LLE is to define a suitable eigenvalue problem using one of the matrix representations of the graph in order to define the projection. Recall that LLE seeks to keep 'close' points together after the embedding; it follows that a reasonable definition of a 'good' mapping $y_i \mapsto x_i$ is one that minimizes

$$\sum_{i,j} \|x_i - x_j\| w_{i,j} \tag{3}$$

It can be shown that the minimization problem Eq. 3 is equivalent to minimizing

$$tr(X'LX)$$

where $tr(M)$ is the matrix trace, X is the n-by-k matrix whose rows are the projected points, and L is the Laplacian matrix defined by the weights. To prevent trivial solutions, we enforce the constraint $X'DX = I$, where D is the degree matrix, and I is the identity. Under these conditions, it can be shown that the solution X is given by the eigenvectors corresponding to the *smallest* eigenvalues of the generalized eigenvalue problem

$$Lx = \lambda Dx. \tag{4}$$

It is straightforward to show that 0 is always an eigenvalue of Eq. 4; the multiplicity of 0 is exactly the number of connected components in the graph (in particular, 0 will be a simple eigenvalue for connected graphs), with a constant eigenvector. Since the 0 eigenvector may be interpreted as a simple translation in the projected space, this eigenvector (e.g. the first column of X) is generally ignored.

2.3 Segmentation and Normalized Cuts

We now consider the problem of segmenting or partitioning a graph into homogeneous clusters. Intuitively, we think of edges as defining a 'neighborhood' relationship on our objects; the edge weights define how 'close' the neighbors are. For example, if our objects are pages on the internet, than an edge may be used to define a hyperlink between two pages, while the weights define how similar the two pages are in content (although incidental to this discussion, it is worth mentioning that Google's page rank algorithm is in fact a spectral graph technique). The goal of segmentation is to provide two (or more) groups of nodes such that neighbors with large weights are in the same cluster, while nodes that are not related (e.g. small or zero weights) are placed in a different cluster. For simplicity, we assume that the graph will be split into two distinct groups; note that each group can then recursively subdivided, leading to a 'top-down' into as many groups as desired.

To separate a graph into two disconnected components, one simply removes any edges that connect nodes in the separate groups. Intuitively, a 'good' partition will minimize the weights of the removed edges, since they should represent nodes that are not similar. More formally, if we denote the two components as A and B, we define the *cut* as the sum of all weights between the two:

$$cut(A,B) = \sum_{i \in A, j \in B} w_{i,j} \tag{5}$$

Here we assume that A, B form a cover of the vertices; that is, $A \cup B = V$ and $A \cap B = \emptyset$.

In theory, minimizing Eq. 5 would produce an optimal segmentation of the graph into two segments; in practice, however, the minimal cut tends to small groups of outliers that lie far from the remaining nodes.

To avoid this situation, Shi and Malik [6] proposed normalizing the cut measure by the sum of all weights in each group to the entire graph. In particular, they define the *association* of a subset $A \subset V$ as

$$assoc(A) = cut(A,V) = \sum_{i \in A, v \in V} w_{i,v}$$

The *normalized cut* is then

$$Ncut(A,B) = \frac{cut(A,B)}{assoc(A)} + \frac{cut(A,B)}{assoc(B)} \ . \tag{6}$$

To create a partition, one aims to choose a partitioning A,B that minimizes Eq. 6. After some algebraic manipulation, this can be reformulated as

$$\min_v \frac{v'Lv}{v'Dv} \tag{7}$$

Here, L and D are the Laplacian and degree matrices, respectively, and $v \in \Re^n$ is an *indicator* vector whose entries are either 0 or –k for some constant k, and subject to the constraint $v'D1 = 0$ where 1 is the n-dimensional vector whose entries are all equal to 1. Unfortunately, minimizing Eq. 7 can be shown to be NP-hard. However, if the indicator constraint is relaxed, and v is allowed to take on any real values, then the minimization can be solved via the generalized eigenvalue problem:

$$Lv = \lambda Dv.$$

Note that this is exactly the same eigenvalue problem as in LEM (Eq. 4).

In order to run the normalized cuts algorithm, one needs to define an appropriate graph structure. As mentioned, the nodes should be connected in such a way that only 'neighboring' nodes share an edge, with a weight that encapsulates how 'similar' the nodes are.

In the particular case of image segmentation, there is fortunately very intuitive methods for each step; in this case, the pixels are the nodes of the graph. Nodes are connected if and only if they are spatially contiguous; the weights represent how similar the pixels are. More formally, Shi and Malik defined the following scheme for grayscale images:

$$w_{i,j} = e^{\frac{-\omega(p_i,p_j)}{\sigma_I^2}} \times \begin{cases} e^{\frac{-d(p_i,p_j)}{\sigma_X^2}} & if\ d(p_i,p_j) < r \\ 0 & else \end{cases}$$

Here, $\omega(p_i,p_j)$ is the absolute value of the difference in image intensities between pixels p_i and p_j, while $d(p_i,p_j)$ represent the spatial distance (in pixel coordinates). The parameters $\sigma_I^2, \sigma_X^2, r$ are user defined; in [4], the authors recommend using $\sigma_I = 0.1, \sigma_X = 4.0, r = 5.0$ for image with intensities normalized to the range (0,1). As with Laplacian eigenmap, the constraint $d(p_i,p_j) < r$ ensures that only a limited number of nodes are connected to a given node. As a result, the adjacency and Laplacian matrices are sparse, allowing for relatively large images as compared to MDS and ISOMAP.

3. SPATIAL SPECTRAL GRAPHS FOR HSI

In this section we present our main algorithm, which uses the Laplacian eigenmap / normalized cuts algorithm to segment and/or perform dimensionality reduction on a hyperspectral image. In particular, we present a graph coupling and weighting structure that naturally encapsulates both the spatial and spectral information present in HSI data. Once the graph has been defined, we solve the generalized eigenvalue problem

$$Lv = \lambda Dv \tag{8}$$

for the smallest nonzero eigenvalues and eigenvectors.

Once the eigenvalue problem has been defined, the analysis can proceed in one of two ways: in the first case, the first k eigenvectors can be used to define a (nonlinear) dimensionality reduction, exactly as in Laplacian eigenmaps. In particular, if we let X by the n-by-k matrix whose columns are the eigenvectors, then the *rows* of X define our projected data (here, n is the total number of pixels in the image, and k is the reduced dimension).

Alternatively, one can use the single smallest eigenvector to partition the scene; ideally, the eigenvector will be bimodal, with some subset of the entries clustered close to a single (positive) value, with the remaining entries clustered around a second (negative) value. By choosing an appropriate threshold, the eigenvector entries are partitioned into two groups; the corresponding pixels form the clusters in image space.

Once an initial split has been made, we simply recurse on each of the partitions. Note that the original graph contains all the information we need, and only needs to be calculated once. Subsequent iterations simply require us to select the

rows and columns from the original adjacency matrix to create a new (smaller) reduced matrix. From there, the new degree and Laplacian matrices are easily formed, and the next eigendecomposition calculated.

Some care must be taken as the recursion progresses, since the graph may cease to be connected (when clusters are no longer spatially contiguous), and/or no 'good' partition is possible. The former case can be handled by simply checking to see if the graph is connected before running the eigenanalysis; if not, the graph is split into its individual connected components, and the recursion is continued on each component separately.

The latter case is not as easy to handle, especially in automated fashion. In essence, the problem reduces to deciding if a given distribution (in this case, the values of the eigenvector) are bimodal or not. Currently, we use a statistical test known as Hartigan's Dip Test [12] to guide the user; the final determination of whether or not to further partition the given cluster is made by the user. An alternative approach, suggested by Shi and Malik, is to histogram the eigenvector and calculate the ratio between the maximal and minimal bin counts.

To define the spatial-spectral graph, we follow the normalized cuts algorithm: image pixels define the nodes, and each node is coupled to all other pixels that are spatially contiguous – that is, in some r-by-r window centered at the given pixel. The weights are given by heat kernel using a spectral distance measure, scaled by a 'spatial dampening' that is also a heat kernel (in pixel coordinates). In particular, our weights are defined as

$$w_{i,j} = e^{-\omega(p_i,p_j)} \times \begin{cases} e^{\frac{-d(p_i,p_j)}{\sigma}} & if\ d(p_i,p_j) < r \\ 0 & else \end{cases} \tag{9}$$

Here, $\omega(p_i,p_j)$ is the spectral angle (in degrees) between pixels p_i and p_j

$$\omega(p_i,p_j) = \cos^{-1}\left(\frac{< p_i,p_j >}{\|p_i\| \cdot \|p_j\|}\right)$$

and $d(p_i,p_j)$ is the (squared) spatial distance

$$d(p_{r,s},p_{m,n}) = (r-m)^2 + (s-n)^2.$$

The parameters r and σ are variable; we generally use neighborhood sizes in the range 2 – 7, and fix $\sigma = 50$ for radiance images.

4. EXPERIMENTAL RESULTS

In this example we present experimental results on real-world hyperspectral images. Our focus is on segmentation, but we include examples of dimensionality reduction for completeness. Table 1 lists the details of each image used in our experiments, as well as the run times for both the graph construction and the first eigenvector calculation. Note that subsequent eigenvector calculations are run on progressively smaller matrices; the calculation time quickly decreases.

A few notes on our implementation: all graph construction is done via a C# program that reads in an image, calculates the edges and weights, and writes this information to a file. The graph file is then imported into MATLAB (64-bit), where all subsequent processing is done. The segmentation is done via a MATLAB program that finds the smallest non-zero eigenvector (using the built-in eigs routine) and presents to the user a suggested threshold: in some cases, the threshold is tweaked by the user. Processing finishes when the user manually decides no further partitioning is possible. All processing was done on a hyperthreaded, 6-core Intel i7 CPU with 24 Gb of RAM, running 64-bit Windows 7.

Our first example is a small 'chip' from the HYDICE Forest Radiance data collection. The original image is shown in Fig. 2, and contains a large grassy field bordered on the left by a dirt road (in shadow) and a forest. Lying within the field are 5 rows of three panels; the panels on each row are made of identical material but get progressively smaller.

To run this experiment, we constructed the graph using the weights in Eq. 8, with a 5-by-5 window and $\sigma = 50$. We then used this graph to perform both dimensionality reduction and segmentation. For the dimensionality reduction, we used the entire graph as input into the generalized eigenvalue problem (Eq. 8) and calculated the first 10 eigenvectors. A pseudo-color projection using the first three eigenvectors is shown in Fig. 2. It is clear that the projected data does a very nice job of separating the image into its major constituents (grass, forest, shadow). The forest part is further split by

a large shadow that separates the top and bottom components; note this is reasonable, since we enforce spatial connectiveness. A larger neighborhood size in the graph construction would 'merge' this clusters, if desired.

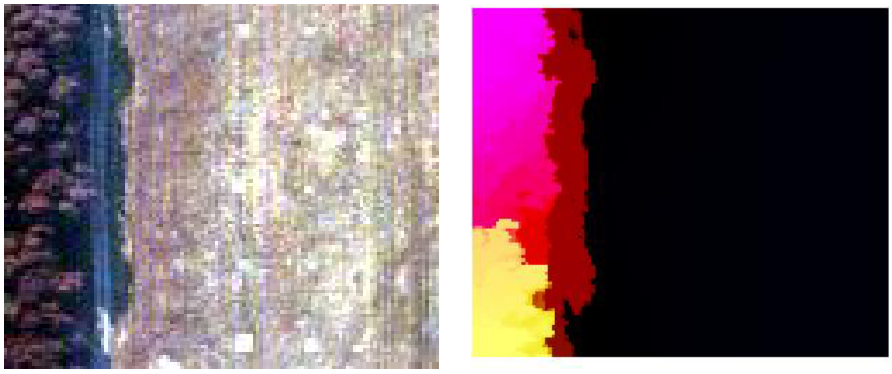
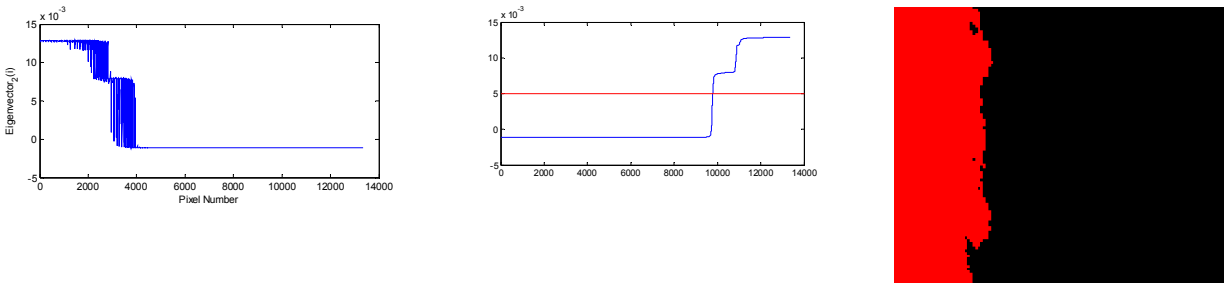


Figure 2. Forest Radiance (small) cube. Left: the original image. Right: Pseudo-color image of the projected data using the first three eigenvectors.

We also used the same original graph to run a segmentation procedure on the image. In this case, we begin by solving Eq. 8 for the first (smallest) non-zero eigenvector. Three views of that eigenvector are shown in Fig. 3. Recall that the eigenvector solutions are *n*-dimensional, where here *n* is the *total* number of pixels in the scene. Viewing the eigenvector in its original form (as in Fig. 3 (a)) loses all spatial information, but does allow one to see the 'indicator' nature of the entries. Sorting the eigenvector by the value of the entries (Fig. 3 (b)) makes this even more explicit, and allows one to easily pick a threshold value that will divide the entries into distinct groups. Finally, reshaping the eigenvector back to the image size (Fig. 3 (c)) allows one to recapture the spatial information in the eigenvector.

Once the first eigenvector has been found, and a suitable threshold calculated, the scene is split into two groups (Fig. 3 (c)). Next, the reduced graph(s) corresponding to this split are extracted from the original graph (note that simply means choosing the correct rows and columns from the original adjacency matrix; no further calculations are needed), the new degree / Laplacian matrices are formed, and the process repeats. Fig. 4 (b) and (c) show the results of the next step of the partitioning. We note that the eigenvectors in this case are no longer *n*-dimensional, since the graphs no longer have all of the original nodes; however, it is easy to embed this information back into an *n*-vector by setting the 'missing' pixel entries to zero; this enables us to reshape the vector back to the original image, as shown in the figure.

The recursive partitioning continues until no further 'cuts' are possible; in this case, the first eigenvector will be essentially unimodal, indicating that the current cluster is homogeneous. In our example, the right hand side of the original split is partitioned three more times; each partition picks up one or two of the larger panels. The final results are shown in Fig. 4 (d).



(a) (b) (c)

Figure 3. Three views of the first eigenvector for Forest Radiance. (a) The original eigenvector. (b) Sorted by the entries. (c) After reshaping back to the original size and binary thresholding.

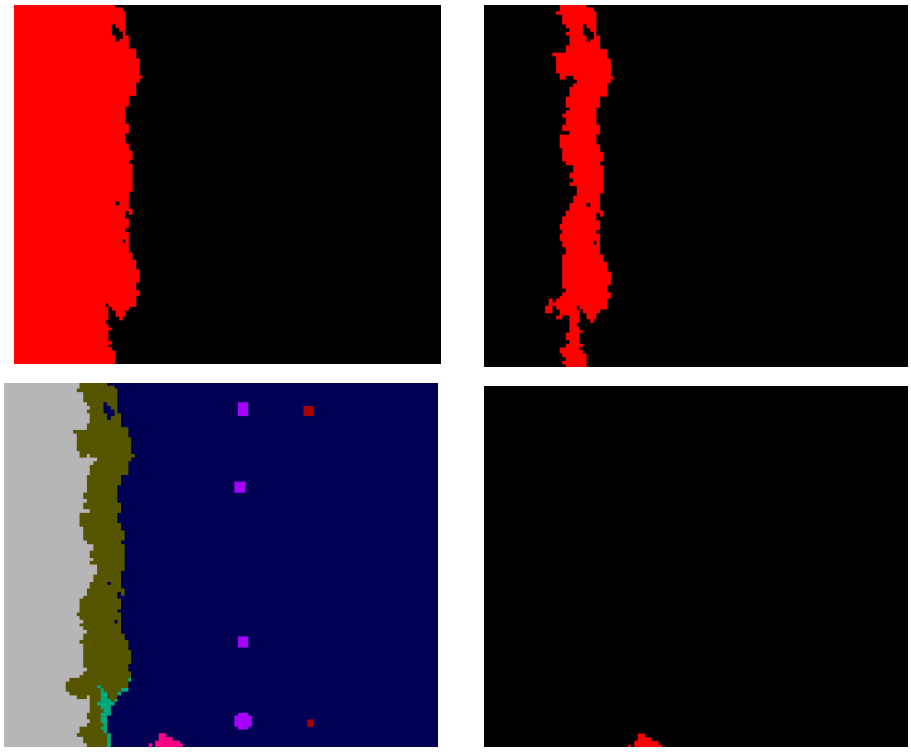


Figure 4. Recursive partitioning. Clockwise from top left: (a) After the first eigenvector split (b) and (c) Splitting the left / right parts of (a), respectively. (d) The final result

Our next two experiments were on more realistic images; the (nearly) full Forest Radiance Run 5 scene, and the HYDICE 'urban' scene, which is publicly available online from the Army Corps of Engineers [10]. In each example the neighborhood size was r=3 and the heat parameter $\sigma = 50$. The results for Forest Radiance are shown in Fig. 5, which shows both projected data (using the first three eigenvectors), as well as the results of running the semi-automated partitioning algorithm as described above. The final result contains 20 classes; as can be seen, the partitioning subdivides the image nicely, and also manages to find a few of the more obvious objects in the field.

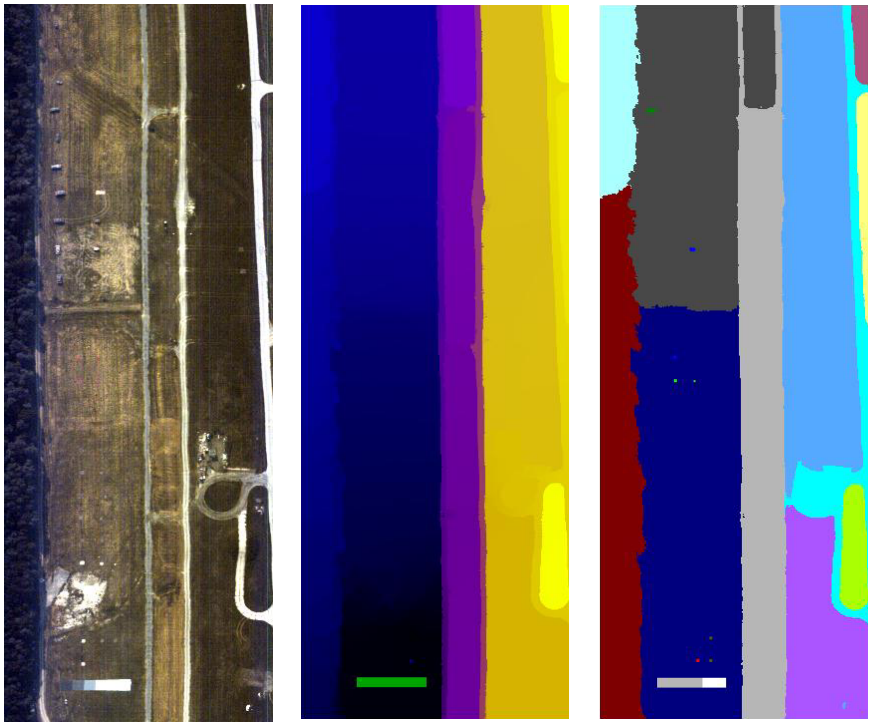


Figure 5. The full Forest Radiance image. Left: the original image. Center: pseudo-color of the projected data onto the first three eigenvectors. Right: the class map from the partitioning algorithm (20 classes).

In Fig. 6, we show the results of the well-known 'urban' data set. In our first experiment we ran in a semi-automated mode; the final results are shown in Fig. 6 (b) for 20 classes. Next, we ran our algorithm in 'fully automated' mode, which automatically chooses the threshold and decides when to stop the partitioning process, with no user input. The results, with 956 separate classes, are shown in Fig. 6 (c).

Finally, we note for completeness that we are currently able to run on our desktop setup images from the HICO sensor [11] that contain approx. 1,000,000 pixels in about 2 minutes, 1 minute for the graph construction and 2 minutes for the first eigenvector. For reasons of space, we do not include these results in this paper.

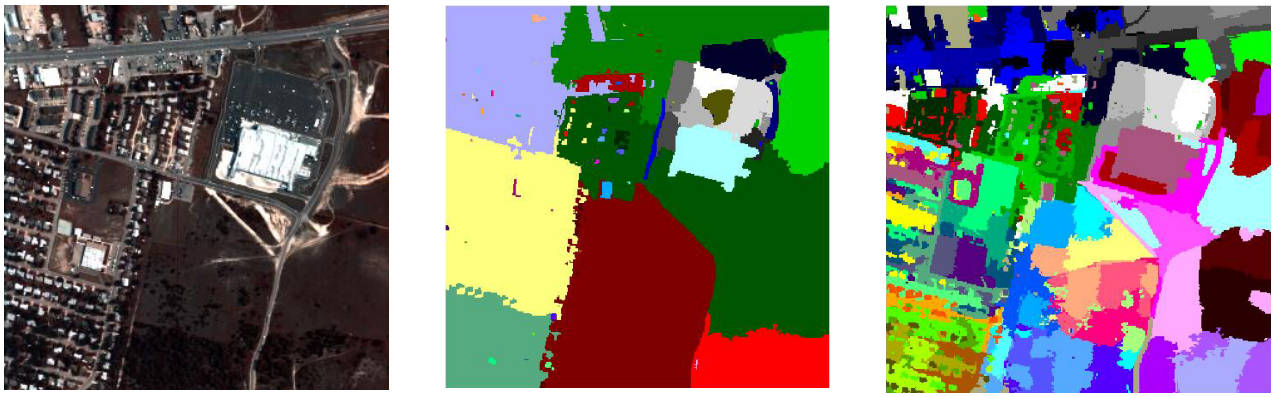


Figure 6. The urban image. Left: the original image. Center: the class map from the semi-automated partitioning algorithm (20 classes). Right: the class map from the fully automated partitioning algorithm (956 classes).

Table 1. Image details and run times.

Image	Size	Run Time (sec)

	Total Pixels	Lines	Samples	Bands	graph	Eigenvector (first)
Forest Radiance (small)	13,356	106	126	170	3	2
Forest Radiance (large)	194,400	720	270	170	30	26
Urban	94,249	307	307	170	6	5
HICO	1,024,000	2000	512	128	40	122

5. SUMMARY

By its very nature, hyperspectral data contains both spatial and spectral information. In order to extract all the information present in a given scene, algorithms should be able to exploit both of these modalities. In this paper, we have presented a graph-based model that naturally encapsulates both the spatial and spectral information. Once the graph has been constructed, we then use spectral graph-based approaches (based on Laplacian eigenmaps and normalized cuts) to perform dimensionality reduction and segmentation. Moreover, the naturally sparse structure of the graph allows us to use specialized, sparse linear algebra routines for the eigen-analysis, allowing us to apply the model on realistic sized imagery.

ACKNOWLEDGEMENTS

This research was funded in part by the US Office of Naval Research and the US Department of Defense.

REFERENCES

[1] Mardia K.V, Kent J.T., and Bibby J.M., [Multivariate Analysis], Academic Press, San Diego, CA, 397-404 (2003).
[2] Bachmann, C.M., Ainsworth, T.L., Fusina, R.A., "Exploiting manifold geometry in hyperspectral imagery," IEEE Transactions on Geoscience and Remote Sensing, 43 (3), 441-454 (2005).
[3] Roweis, S.T., Saul, L.K., "Nonlinear dimensionality reduction by locally linear embedding," Science, 290 (5500), pp. 2323-2326 (2000).
[4] Tenenbaum, J.B., De Silva, V., Langford, J.C., "A global geometric framework for nonlinear dimensionality reduction," Science, 290 (5500), pp. 2319-2323 (2000).
[5] Belkin, M., Niyogi, P., "Laplacian eigenmaps for dimensionality reduction and data representation," Neural Computation, 15 (6), pp. 1373-1396 (2003).
[6] Shi, J., Malik, J., "Normalized cuts and image segmentation," IEEE Transactions on Pattern Analysis and Machine Intelligence, 22 (8), pp. 888-905 (2000).
[7] Camps-Valls, G., Bandos Marsheva, T.V., Zhou, D., "Semi-supervised graph-based hyperspectral image classification," IEEE Transactions on Geoscience and Remote Sensing, 45 (10), pp. 3044-3054 (2007).
[8] Basener, B., Ientilucci, E.J., Messinger, D.W., "Anomaly detection using topology," Proc. SPIE 6565 (2007).
[9] Camps-Valls, G., Shervashidze, N., Borgwardt, K.M. "Spatio-spectral remote sensing image classification with graph kernels" IEEE Geoscience and Remote Sensing Letters, 7 (4), pp. 741-745 (2010).
[10] http://www.agc.army.mil/hypercube/
[11] Lucke, R.L., *et al.*, "Hyperspectral Imager for the Coastal Ocean: Instrument description and first images," Applied Optics, 50 (11), pp. 1501-1516 (2011).
[12] Hartigan, J. A. and Hartigan P. M., "The Dip Test of Unimodality," Annals of Statistics, 13 (1) pp. 70-84 (1985)
[13] Chung, F. [Spectral Graph Theory], American Mathematical Society, 1996.