DePaul University
University Libraries

DePaul Discoveries

Volume 6 | Issue 1 Article 7

2017

Signal Processing on Graphs Using Kron Reduction and Spline Interpolation

Michael Dennis
University of Berkeley, California, michael_dennis@cs.berkeley.edu

Enrico Au-Yeung
DePaul University, eauyeun1@depaul.edu

Signal Processing on Graphs Using Kron Reduction and Spline Interpolation

Acknowledgements

The first author acknowledges the financial support of an Undergraduate Research As- sistant Program (URAP) from DePaul University. The second author acknowledges the financial support of a Faculty Summer Research Grant Program from DePaul University.

Signal Processing on Graphs Using Kron Reduction and Spline Interpolation

Michael Dennis*

Department of Computer Science, University of California Berkeley

Enrico Au-Yeung, PhD

Department of Mathematical Sciences, DePaul University

ABSTRACT In applications such as image processing, the data is given in a regular pattern with a known structure, such as a grid of pixels. However, it is becoming increasingly common for large data sets to have some irregular structure. In image recognition, one of the most successful methods is wavelet analysis, also commonly known as multi-resolution analysis. Our project is to develop and explore this powerful technique in the setting where the data is not stored in the form of a rectangular table with rows and columns of pixels. While the data sets will still have a lot of structure to be exploited, we want to extend the wavelet analysis to the setting when the data structure is more like a network than a rectangular table. Networks provide a flexible generalization of the rigid structure of rectangular tables.

INTRODUCTION

The internet has made it simple to get vast quantities of data. It is now easy to generate large datasets describing social networks, road networks, the connections between web pages, geographic and atmospheric data and much, much more simply by going to the appropriate websites or writing short programs in the comfort of one's office. However, it is becoming increasingly common for these large datasets to have some irregular structure. In classical domains such as image-processing, the data is given in a very regular pattern with a known structure, in this case a grid of pixels. In general the data will not always be so well behaved. In the cases mentioned above, there are intrinsic relationships between different data points that have no simple description. In most cases, these connections should not be ignored; data about the behavior of a person will reveal much more information about the friends and family of that person than it will about complete strangers. Even in the same data-set we can have parts which are well-structured, think about the streets of downtown Chicago, and parts that seem to have little to no structure, think about the twisting streets of Washington DC. Clearly, there is a present need to be able to handle this type of data.

* Corresponding Author *michael_dennis@cs.berkeley.edu*
Research Completed in Summer 2016 while first author was a student at DePaul University

To try to better understand this problem, we look at what has worked in other classical domains.
In image recognition, one of the most successful methods has been wavelet analysis (Walnut, 2002). Over the last 25 years, the method of wavelet analysis, also commonly known as multi-resolution analysis, has been an indispensable tool for processing data and is one reason why digital image processing has become so successful. It allows a simple way to perform tasks as fundamental as detecting the edges of objects in images or image compression and are used every time your phone's camera detects your face or makes a JPEG. It is this ubiquitous success that we want to translate into the domain of irregular data.

Our project is to develop and explore this powerful technique in the setting where the data is not stored in the form of a rectangular table with rows and columns of pixels.

While the data sets will still have a lot of structure to be exploited, we want to extend the wavelet analysis to the setting when the data structure is more like a network than a rectangular table.

Networks provide a flexible generalization of the rigid structure of rectangular tables. Scientists often refer to these networks as graphs. A graph is a set of vertices (such as people, intersections, or websites) and a set of edges (such as friendships, roads, or hyperlinks). We will start by understanding the success and benefits of signal processing before formalizing the concept of a graph and moving to show how current methods can be extended to work in these new settings.

THE IMPORTANCE OF SIGNAL PROCESSSING

Crime dramas, such as CSI, Criminal Minds, Law and Order, and NCIS, often give the audience the impression that facial recognition will only take seconds for a computer to do. For example, if a little girl is kidnapped in a shopping mall, the detective can just ask the computer to compare the image of a face taken by a surveillance camera with the images of faces stored in a database. It is a dramatization of facial recognition that the computer flashes through thousands of images per second and successfully identifies the suspect within seconds. In reality, facial recognition can be a computationally intensive task, especially if tens of millions of faces need to be compared. Each image is a quarter million pixels. Unlike the situation often shown in crime drama on television shows, it can take a computer up to six hours to work through a vast number of digital images, while the little girl kidnapped in the shopping mall is still missing. When comparing digital images, features selection is more important than accurate reconstruction of images from partially stored data. Thus given two slightly blurry images, the ability to extract relevant features from each image and to compare only these features would be sufficient to tell whether these two images represent the same person. In such an application, it is desirable to have a flexible way to reconstruct a slightly blurry version of an image quickly.

A related and prominent example is the processing of images from a database of fingerprints. Often, the fingerprint found in a crime scene is not a perfect print but only a partial print. In this case, a partial reconstruction of an image is not only sufficient, but desirable. Instead of seeking a perfect match with a fingerprint, the FBI agent is more interested in the comparison with a partial reconstruction or a blurry version of a fingerprint image.

SIGNAL PROCESSING ON GRAPHS

Graphs are useful for describing the irregular structure in networks such as social, transportation, and neuronal networks. In general graphs are a set of objects called *vertices* and relationships between those objects called *edges*. For instance a graph could be people connected by friendships, intersections connected by roads, or websites connected by hyperlinks. Typically there is also a *weight* on each edge describing the strength of the connection. In this paper we will hold the convention that the higher the weight the closer the connection between the two vertices. For example, in a road network, the weight can be chosen to be inversely proportional to the length of the road or, in a social network, the weight can be the number of interactions in a typical week.

Formally, we represent a graph by the tuple $G=(V,E)$, where V is a set of N vertices and E is a set of edges. For edges, e in E we will define $w(e)$ to be the weight of the edge e. From this we can define the *weighted adjacency matrix* W as a matrix with N rows and N columns describing the weights in the graph G. For this matrix, the entry in row i and column j is the weight associated with the edge between the i-th vertex and the j-th vertex, or zero if no such edge exists. It is important to note that in this context we are only concerned about graphs whose edges have no direction. As far as the methods discussed in this paper are concerned, if I am friends with you, you are friends with me.

If there are no natural choices for weights, one can construct an *unweighted graph*, where the entries of the adjacency matrix are zeros and ones. In this case, a one indicates that two vertices are directly connected, and a zero means that the two vertices are not directly connected. The degree matrix D is a diagonal matrix with diagonal entry

$$d_i = D(i,i) = \sum_j W(i,j),$$

where the sum is over all the vertices j connected to vertex i. The graph *Laplacian* is the matrix

$$L = D - W.$$

It is well-known that many properties of the Laplacian matrix tell us about the structure of the graph G.

Finally, since we hope to use signal-processing techniques, it is important to define what we mean by a signal on a graph. A *signal* or function f: $V \mapsto \boldsymbol{R}$ defined on the vertices of a graph is often represented by a vector with N components, where the i-th component of the vector represents the value of the function at the i-th vertex. Graph signals appear in many engineering and science applications. In brain imaging, the distinct functional regions of the cerebral cortex are represented by the vertices of a weighted graph. The strength of connectivity is captured by the weights of the edges. The graph signal in this case is the data of the regions of the brain obtained from functional magnetic resonance imaging (fMRI), often a measure of the intensity of the brain activity in those regions. Sometimes, a graph and the graph signal can reveal structure that is very difficult to detect. As another example, a large collection of documents can be visualized as a graph, with one vertex for each document. Edges and weights can be the number of important words that they have in common and the signal can be the degree of relevance of the document to the task at hand. In both of these cases, the behavior of the signal is closely tied to the structure of the graph. We would expect the amount of activity in closely connected regions of the brain to be related.

VERTEX SELECTION AND GRAPH REDUCTION

Having defined what we mean by a signal on a graph, we now begin to describe the signal-processing techniques on a graph. To perform a multi-resolution analysis on graph signals, we need to specify three steps:

1. vertex selection
2. graph reduction
3. graph interpolation

The first step, vertex selection, means that we need a rule to select the vertices of a graph. In classical signal processing, where signals are often defined on a line, this step simply means that we select every other vertex and remove the remaining vertices. Since each entry of the graph signal corresponds to the value of a function at a vertex, this is the same as keeping every other component of the signal $f \in \boldsymbol{R}^N$ and discarding the remaining entries. If we try to extend this idea to the setting of graphs signal processing, it is not entirely obvious what it means to keep every other component of a signal defined on the vertices of a graph. We adopt the convention that if a graph is bipartite, which means the vertices can be coloured red or blue so that vertices of the same colour are not connected to each other, then we keep the entries of the graph signal that correspond to the red vertices. Clearly, it is equally plausible to keep the blue ones and discard the red ones. If the graph is not bipartite, then it is possible to modify this rule, based on the signs of the entries in the eigenvector of the graph Laplacian that corresponds to the largest eigenvalue, which are known to make an

approximately bipartite set. For the sake of clarity, we illustrate this rule with the following example.

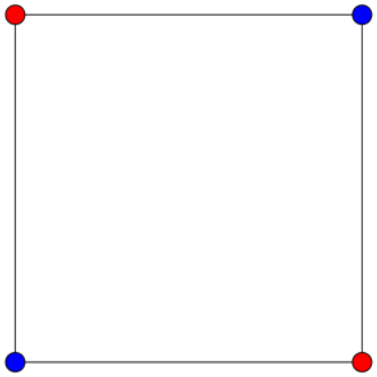

Figure 1. A small example of a bipartite graph.

Example 1 *Suppose we have the graph in Figure 1. The four corners of the square are the four vertices of the graph. This is a bipartite graph because each vertex of the graph is either red or blue, while no neighbours have the same colour. For this unweighted graph, we compute its graph Laplacian L and the eigenvector v corresponding to the largest eigenvalue* λ_{max},

$$Lv = \lambda_{max} v.$$

The eigenvalue $\lambda_{max} = 4$ *and the four entries in the eigenvector v are* $\{-0.5, 0.5, -0.5, 0.5\}$. *We note that half the entries are positive and half are negative. We select the second and fourth vertices, which correspond to the positive entries. This is our rule for vertex selection. We extend this vertex selection method to the case when the graph is not bipartite. Imagine if we were to add an edge connecting the top left corner and the bottom right corner of the square. Then the graph is no longer bipartite.*

We compute its graph Laplacian L and the eigenvector v corresponding to the largest eigenvalue. The vector v still contains two positive and two negative entries. We select those two vertices corresponding to the positive entries.

Having selected a subset of the vertices of a graph, we need a procedure to define a graph Laplacian on the subset of the chosen vertices. This is the process of graph reduction. We would like graph reduction to have the following desirable properties:

1. The resulting graph Laplacian is indeed a Laplacian matrix. In particular, it means that each row must sum to zero.

2. If the original graph is connected, then the reduced graph is also connected.

3. Some structural properties are preserved. We will discuss this below in items 3 and 4 of the subsequent list.

4. It must be computational feasible to implement. For example, if the graph contains N vertices, the algorithm to implement the graph reduction process should not take more than N^6 steps.

We emphasize that this is a wish-list and there is no theoretical reason to think that any graph reduction process can satisfy all these properties for all possible graphs. The procedure that we use for graph reduction is called Kron reduction, as investigated by Shuman, Faraji, and Vandergheynst (2016).

Suppose we have a graph $G=(V,E)$, together with the graph Laplacian L and a subset V_1 of the vertices. Our task is to form the reduced graph with the set of selected vertices V_1 and a resulting graph Laplacian. The Kron reduction of L is

$$K(L, V_1) = L_{V_1,V_1} - L_{V_1,V_1^c} \, L_{V_1,V_1^c}^{-1} \, L_{V_1^c,V_1}$$

where the notation $L_{A,B}$ denotes the sub-matrix that consists of all the entries of L whose row index is in the set A and whose column index is in the set B.

The Kron reduction enjoys several properties.

1. The graph Laplacian defined by (1) is indeed a graph Laplacian.

2. If the original graph is connected, then the resulting graph from Kron reduction is also connected.

3. The Kron reduction of a ring graph is a ring graph with half the number of vertices, and the weights of the edges are halved. A ring graph is a graph in which all the vertices are arranged around a ring, so that each vertex has one left neighbour and one right neighbour.

4. The Kron reduction of a very large 4-connected grid graph is 8-connected. In this context, we say that a graph is *k*-connected if it remains connected after the removal of any ($k-1$) edges that are not on the outer edges of a graph.

Properties (3) and (4) indicate that in some situations, some structural properties of a graph are preserved in the Kron reduction process.

One of our contributions is that we discovered a way to perform Kron reduction without explicitly inverting a matrix, when the original graph is bipartite. The Kron reduction of graph Laplacian *L* defined by (1) involves the inversion of a large matrix. By avoiding the need to explicitly compute the inverse of a large matrix during Kron reduction, the potential savings in either storage or computation time are huge. This now makes it possible to use the techniques on much larger datasets than were previously possible and it is now feasible to test our code on standard images containing millions of pixels.

GRAPH INTERPOLATION

Given the values of a graph signal on a subset V_1 of the vertices, we need a method to infer the values of the signal on the remaining verticesV_1^c. We call this process graph interpolation. There are many ways to fill in the missing values of the graph signal from the values Y that we have on V_1. Some assumptions need to be imposed on the graph signal. We assume that the graph signal to be interpolated is smooth. The criterion we use to measure the smoothness of a graph signalf is the inner product $\langle Lf, f\rangle$, where we continue to denote the graph Laplacian by L. When this inner product is small, we say that the signal is smooth. The authors in Shuman et al. (2016) recommend the spline interpolation method by Pesenson.

This method interpolates the missing values of the signal by finding the function f with the smallest value of $\langle Lf, f\rangle$. The task of this spline interpolation method is to find the smoothest signal *f* under the constraint that the values of the interpolated signal at V_1 agree with the values *Y* that we have.

It was proved by Pesenson (2009) that there is a unique function *f* defined on the vertices of the graph that will minimize the value of $\langle Lf, f\rangle$ under the constraint that the values of f agree with the values *Y* at V_1. This result implies that whichever method is used to interpolate the signal, the result is unique under this smoothness criterion. Based on this implication, we use the method that expresses the desired function *f* as a linear combination of fundamental solutions. We explain this method in the appendix, where for clarity, we illustrate the 3-step procedure with a signal defined on a graph with six vertices.

ILLUSTRATION

To check if the methods we are using have reasonable results we can fall back on the classical task of image processing. Starting with an image that contains 80×80=6400 pixels, we discard the data on half the pixels and keep only the remaining data on the other half of the pixels. Our task is to reconstruct the original image from the data that we have kept. We treat our image as a graph signal defined on a grid graph. Imagine we have a very large chessboard consisting of 6400 squares with 80 rows and 80 columns. This gives us a graph with 6400 vertices, with each vertex representing one square in the large chessboard. Except for the squares on the four boundaries of the board, each square is adjacent to exactly four other squares on the board. Analogously, each of the vertex not on the boundary of our grid graph is connected to four other vertices. Thus our graph indeed looks like a grid. You can think of the vertices that we keep as being the black squares on that chessboard.

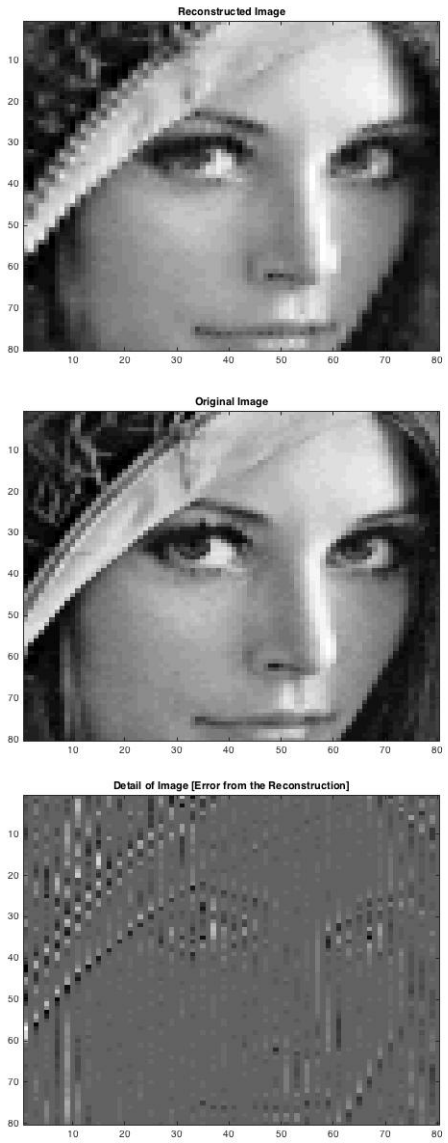

Figure 2. *On the top we have the classical Lena image, in the middle we have used our our methods used to reduce and then reconstruct the image, and on the bottom we see the error from this process.*

Using the graph interpolation method, we reconstruct the graph signal using only the data on half the vertices. In Figure 2, we have the original image, the reconstructed image, and the noise. The noise is the difference between the original image and the reconstructed image. We see that the reconstructed image is slightly blurry, as to be expected, since perfect reconstruction is not possible from only half the data. In multi-resolution analysis, we would say that:

Original Image
= Low Resolution Image + Image Detail

The detail captures the texture of the image, while the lower resolution gives us the essential features. The variance of the noise is 6 percent of the variance of the original image. This is an indication that the quality of the reconstruction falls within an acceptable range. These results show that the methods succeed at compression, since the second image is a lossy compression of the first. Additionally, looking at the error image, we can see that it also allows edge detection. Thus two of the most fundamental operations of image processing are maintained and extended to a more general context.

ACKNOWLEDGEMENTS

The first author acknowledges the financial support of an Undergraduate Research Assistant Program (URAP) from DePaul University. The second author acknowledges the financial support of a Faculty Summer Research Grant Program from DePaul University.

APPENDIX

We describe the process of graph interpolation in three steps. We illustrate this process with an example. Consider a graph with 6 vertices. The vertices are denoted by the set

$$V = \{v_1, v_2, v_3, v_4, v_5, v_6\}.$$

Assume that we have the values of the signal on only 3 vertices, denoted by $U = \{v_1, v_3, v_5\}$. These values are specified by $Y = [y_1, y_3, y_5]$.

Input:

1. the graph Laplacian *L* of the graph
2. the set $U = \{v_1, v_3, v_5\}$
3. $Y = [y_1, y_3, y_5]$ is a list of 3 numbers

Output: a function *f* defined on *V* so that

$$f(1) = y_1, f(3) = y_3, f(5) = y_5.$$

Step 1. Solve 3 systems of linear equations.

Define three vectors in $\boldsymbol{R}^6$ by

$$s_1 = [1,0,0,0,0,0]$$
$$s_3 = [0,0,1,0,0,0]$$
$$s_5 = [0,0,0,0,1,0]$$

Solve for the 3 vectors, F^1, F^3, F^5 so that they satisfy the relations:

$$LF^1 = s_1, LF^3 = s_3, LF^5 = s_5.$$

The solutions F^1, F^3, F^5 are all vectors in $\boldsymbol{R}^6$. They are called fundamental solutions.

Step 2. Solve the following 3 systems of equations.

Solve for a_1^1, a_3^1, a_5^1 so that they satisfy:

$$a_1^1 F^1(1) + a_3^1 F^3(1) + a_5^1 F^5(1) = [1,0,0].$$

Solve for a_1^3, a_3^3, a_5^3 so that they satisfy:

$$a_1^3 F^1(3) + a_3^3 F^3(3) + a_5^3 F^5(3) = [0,1,0].$$

Solve for a_1^5, a_3^5, a_5^5 so that they satisfy:

$$a_1^5 F^1(5) + a_3^5 F^3(5) + a_5^5 F^5(5) = [0,0,1].$$

In the system of equations above, $F^1(j), F^3(j), F^5(j)$ refer to entry *j* in each of the 3 vectors F^1, F^3, F^5 respectively.

Step 3. With the 9 numbers obtained in Step 2, we compute the 3 Lagrangian splines,

$$Lag^1 = a_1^1 F^1 + a_3^1 F^3 + a_5^1 F^5,$$
$$Lag^3 = a_1^3 F^1 + a_3^3 F^3 + a_5^3 F^5,$$
$$Lag^5 = a_1^5 F^1 + a_3^5 F^3 + a_5^5 F^5.$$

The 3 Lagrangian splines Lag^1, Lag^3, Lag^5 are all vectors in $\boldsymbol{R}^6$ since F^1, F^3, F^5 are all vectors in $\boldsymbol{R}^6$.

Output: $f = y_1 Lag^1 + y_2 Lag^3 + y_5 Lag^5$

The output signal f is a vector with six numbers.

We have $f(1) = y_1, f(3) = y_3, f(5) = y_5$, up to round-off error. For the sake of clarity, we have used an example to illustrate the procedure. The extension to the general case for any graph and for a signal defined on the graph is straightforward.

REFERENCES

Pesenson, I., *Variational splines and Paley-Wiener spaces on combinatorial graphs*, Constructive Approximation **29** (2009), no. 1, 1–21.

Shuman, D., M. J. Faraji, and Pierre Vandergheynst, *A multiscale pyramid transform for graph signals*, IEEE Transactions on Signal Processing **64** (2016), no. 8, 2119–2134.

Walnut, D. F., *An introduction to wavelet ananlysis*, Birkhäuser, Boston, 2002.