A Comparison of Vertex Ordering Algorithms for Large Graph Visualization

Christopher Mueller, Benjamin Martin, and Andrew Lumsdaine
Open Systems Laboratory
Indiana University
Bloomington, IN 47405
{chemuell,benjmart,lums}@osl.iu.edu

ABSTRACT

In this study, we examine the use of graph ordering algorithms for visual analysis of data sets using visual similarity matrices. Visual similarity matrices display the relationships between data items in a dot-matrix plot format, with the axes labeled with the data items and points drawn if there is a relationship between two data items. The biggest challenge for displaying data using this representation is finding an ordering of the data items that reveals the internal structure of the data set. Poor orderings are indistinguishable from noise whereas a good ordering can reveal complex and subtle features of the data.

We consider three general classes of algorithms for generating orderings: simple graph theoretic algorithms, symbolic sparse matrix reordering algorithms, and spectral decomposition algorithms. We apply each algorithm to synthetic and real world data sets and evaluate each algorithm for interpretability (i.e., does the algorithm lead to images with usable visual features?) and stability (i.e., does the algorithm consistently produce similar results?). We also provide a detailed discussion of the results for each algorithm across the different graph types and include a discussion of some strategies for using ordering algorithms for data analysis based on these results.


1 INTRODUCTION

Visual similarity matrices (Figure 1), or similarity matrices, have emerged as effective tools for visually examining relational data sets represented as graphs [13]. While various forms of similarity matrices have been used for over a century for analyzing small data sets, very little literature exists that explores techniques for generating usable images and scaling the visualization to support larger data sets. In this paper, we examine the use of graph ordering algorithms as similarity matrix layout algorithms. We apply a diverse collection of algorithms to synthetic and real world data sets to study the utility of the ordering algorithms for generating layouts for similarity matrices and we explore the limits of what the orderings can expose about the relational structure of the data.

A graph, in the mathematical sense, is a collection of vertices with edges between them. To represent data as a graph, each data item is treated as a vertex in the graph and an edge connects two vertices if there is a relationship between the items corresponding to the vertices. This relationship can be any measure that relates two items but it is most often either the presence of some shared attribute or one of the many similarity or dissimilarity metrics used for data clustering. If the measure used to create the edges has a range of values, the value for a particular relation may be stored with the edge as an edge weight. These values are also used to filter graphs to reduce the size and remove ‘noisy’ edges. Representing data as graphs opens the possibility of using the many graph theoretic algorithms that have been developed for analyzing graphs.

Visual similarity matrices, also known as shaded similarity matrices, distance matrices, or proximity matrices, label their axes with each data item and display a point in the plot area at \((i,j)\) when there is a relationship between the items \((x_i,x_j)\) (Figure 2). When most items are related, the visualization is a dense collection of points. However, when the number of relations is smaller (e.g., in a filtered data set), similarity matrices can reveal data clusters and other structural patterns.

Visual similarity matrices scale to hundreds of thousands of vertices and millions of edges at interactive speeds on workstation-class hardware, providing an effective way to visually explore a large amount of data with a modest amount of resources. Because each vertex only needs one point on the axis, modern displays can interactively display up to 1500 vertices with each vertex assigned to one column of pixels. If pixel-level resolution for individual vertices is not required, anti-aliasing techniques can be used to display millions of vertices interactively. Large-format displays (e.g., display walls) and high-resolution displays (e.g., printers) also extend the size of data that can be displayed. Thus, visual similarity matrices are an attractive solution for large graph visualization.

However, visual similarity matrices suffer from one serious flaw that can have a negative impact on their utility. The ordering of the
1.1 Vertex Ordering Algorithms for Visualization

A set of n vertices can be ordered n! different ways. In graph theory, the problem of ordering the vertices to optimize a cost function is known as minimum linear arrangement, or MINLA, and is a known NP-hard problem [11]. For visualization, the cost function can be thought of as finding the best layout for the data. Since NP-hard problems cannot be solved in a reasonable amount of time, many heuristic algorithms have been developed for ordering vertices that generate orderings by meeting domain-specific optimization criteria. These solutions are not perfect and have limitations, but are usually a large improvement over random orderings.

To make algorithms practical for visualization, they need to run in a reasonable amount of time. For interactive applications, this would ideally be linear or better based on the number of vertices in the graph. Attribute orderings can potentially achieve this runtime behavior, but structural orderings generally visit most of the edges in the graph. Thus, for structural orderings, a run that is linear in the number of edges is preferred. In the worst case, this may be quadratic in the number of vertices (one relationship between each pair of vertices), but for filtered graphs it can be much closer to the number of vertices times a small constant. The algorithms should also not require memory beyond a few extra properties per edge. Some algorithms derived from linear algebra use multiple copies of the dense matrix and are impractical for interactive use on workstations.

An important heuristic in most ordering algorithms is the choice of the starting vertex. For some of the algorithms considered here, auxiliary algorithms exist for selecting a good starting vertex. In addition to the starting vertices, the layout of the graph in memory can affect the result. Most algorithms process lists of edges for each vertex and do not impose any order requirements on the edges. If the edges were stored in memory using a good default ordering, they may be returned to the algorithm in a good order and have a positive effect on the result. While this is a good feature to exploit in practice, it is dangerous for evaluating ordering algorithms. To control for the influence of the initial ordering, it is important to evaluate algorithms against a default initial ordering and a shuffled initial ordering [32].

For this paper, we considered three general classes of ordering algorithms: graph theoretic, sparse matrix, and spectral ordering.

**Graph Theoretic Algorithms** The simplest way to generate an ordering of vertices in a graph is to simply number the vertices as a they are visited by a graph theoretic algorithm, such as a search or a partitioning algorithm. We use breadth-first search (BFS), depth-first search (DFS), the separator tree partitioning algorithm, and an ordering based on the degree (number of edges) of the vertices in the graph.

**Sparse Matrix** Symbolic sparse matrix ordering algorithms attempt to produce orderings that minimize fill-in during subsequent numerical factorization [12]. Some such algorithms attempt to minimize the envelope occupied by non-zero (non-null) values in the matrix. For many sparse matrices, it is possible to pack the non-zero elements around the diagonal. By pulling in the non-zeros this way, the space between the diagonal and the most distant elements can be reduced. Interestingly, the visual representations of these matrices are often similar to good default orderings for similarity matrices. Here, we employ Reverse Cuthill McKee (RCM), King’s algorithm, and Sloan ordering.

**Spectral** The final algorithm we consider in this paper is an ordering based on the spectral decomposition of graphs. The algorithm uses the spectral properties of the Laplacian of a graph to generate a suitable ordering of the vertices. While more expensive than other methods, spectral decomposition is stable with respect to the initial ordering as long as certain numerical constraints and uniqueness conditions are met.

2 Related Work

General matrix visualizations and their underlying similarity matrices are important tools in cluster analysis. All classic texts on data clustering include a section on similarity matrices (e.g., [9, 35]) and their associated visualizations.

**Feature-based** matrix visualizations are closely related to visual simulation matrices and have a long history, having been originally used in the late 1800s by archaeologists for dating artifacts. With the feature-based approach, data items are displayed along one matrix axis and item attributes along the other. Using a process called seriation, artifacts can be visually clustered using feature vectors to form groups with similar features, with the final order implying a timeline [23]. The seriation problem has been studied rigorously as an optimization problem ([23], [25]) and feature-based matrix visualizations have been applied heavily in archaeology, psychology, operations research and, more recently, for comparing gene expression profiles [8].

Visual similarity matrices themselves appear to have their roots in operations research in the 1960s [18] as an alternative to feature-based visualizations for studying scheduling problems. In the 1970s, the dotplot was introduced for comparing genomic sequences [14]. Two genomes are placed on the axes of the plot and dots are placed where they contain the same nucleotide. The genomes have a default ordering that allows dots in the matrix to reveal structural similarities between them. Helfman used a variation on the dotplot for studying texts and source code and also introduced a basic vocabulary for discussing features in the plot [16]. We have recently extended this work by providing interpretations for features in visual similarity matrices where there is no natural default ordering [27]. In 1985, Murtagh phrased the matrix reordering problem as a general method for data clustering and included the notion of thresholding the similarity matrix to improve the results [28]. Matrix-based visualizations have also been used to visualize sparse matrices used in numerical computations, as a tool for debugging and developing new algorithms [2, 22] and as a general tool for software visualization [37].

Numerous small studies have been performed that combine visual similarity matrices with clustering algorithms on real-world data sets. Gale describes a procedure for using hierarchical clustering and seriation algorithms along with visual similarity matrices for data analysis [10]. In a similar vein, Wang uses nearest neighbor clustering and an ordering based on the decision tree algorithm to visualize a similarity matrix [38]. Strehl introduces OPPOSITE [36], a modified version of the graph partitioning algorithm METIS [19], to generate orderings and the CLUSION visualization toolkit for visualization. Abello takes a slightly different approach.
with his graph sketches by ordering the data with BFS and displaying a sketch of the similarity matrix that shows summary information on the local density of the plot [1]. In his dissertation [32], Schumaker used BFS, DFS, and a custom optimization algorithm to order similarity matrices. He also introduced the idea of using both default and shuffled initial orderings for evaluation.

Sparse matrix ordering algorithms are described in detail in [12] and can be found in many applications and libraries (Matlab [24], Pajek [7], BGL [33]). Gibbs performed an early comparison of ordering algorithms using sparse-matrix bandwidth and profile reduction as the measurement [15]. Spectral methods [39] are a more recent development and have found widespread use not only in matrix ordering applications, but also in gene sequencing and text mining applications. In [3], Berry describes the use of RCM and two spectral methods for browsing term-document feature matrices for hypertext collections. The images in his tutorial suggested their utility for general use in visualization and helped lead to this study. Kumfert [21] and Hu [17] propose hybrid methods using of Sloan’s algorithm and spectral methods for wavefront reduction in sparse matrices. Images in their papers also suggest that similar approaches may be useful for visualizations purposes.

Recently, two user studies have validated the use of similarity matrices for identifying relationships and structural features in data. In his dissertation [32], Schumaker performs a user study comparing node-link diagrams and structure-based matrix visualizations using attribute orderings, DFS and BFS derived orderings, and a new algorithm based on the dynamics of social networks. The work in [13] also contributed a user study that evaluated matrix-based visualizations for common graph-based tasks, such as identifying related vertices. Both studies conclude that similarity matrices are effective tools for visually analyzing graphs.

As a testament to their general utility, many data analysis tools include matrix visualizations (e.g., SPY in Matlab [24] and TV in IDL [31]). Pajek [7] also includes numerous algorithms for partitioning and ordering the data (including BFS and DFS) and has recently begun to include sparse matrix algorithms, inspired by an earlier, unpublished version of this paper.

In this work, we extend [3] and [32] by combining and extending the set of algorithms studied, introducing the use of multiple synthetic data sets, and defining a method for evaluating orderings. We also characterize the results to provide suggestions for further refinements of existing algorithms and directions for new algorithms.

3 Methods

To evaluate the ordering algorithms for use in visual similarity matrices, each algorithm was tested against a collection of synthetic and real graphs. Two versions of each graph were used: one built using the default ordering of the vertices and the other built using a random ordering of the vertices (Figure 1). Using randomly ordered vertices allowed us to determine the influence of the default ordering on the final image. For the synthetic graphs, multiple graphs were generated, varying the size and generator specific parameters to get a representative collection of graphs of each type.

All data, parameters, images, and code used in this study are available as supplementary material online [26].

3.1 Data

To test the algorithms in with different graph types, we used five types of synthetic graphs and three graphs derived from real data. The synthetic graph types are as follows:

- **Erdos-Renyi.** So-called “random graphs” connect vertices based on a Gaussian probably distribution. If the probability is 1.0, the graph is a clique.
- **Small World.** Vertices are connected to \( k \) neighbors in a linear fashion. A re-wiring operation randomly rewires some edges to add noise to the graph.
- **Power-law.** Edges are added to the graph following a power-law distribution.
- **K-partite.** \( K \) sets of vertices have no intra-set edges and are randomly connected to vertices in disjoint sets based on a probability parameter
- **K-linear-partite.** Similar to K-partite graphs, but sets are only connected to two adjacent sets, forming a “chain” of k-partite graphs.

Graphs of each type were generated with 100, 500, and 1000 vertices and a small parameter sweep was performed on the parameters for each graph type. For instance, the small world graphs were generated with 5 and 10 neighbors and re-wiring probabilities of 1% and 10%, leading to 12 variations on the small world graph (or 24 variations when considering the initial ordering). The number of edges is dependent on the graph type and is a small constant multiplier for small world graphs and a percentage of all possible edges for the other graph types.

The real graphs were derived from two life science data sets. The first two graphs were generated from proteins listed in the COG database [30] and similarity scores acquired from the PLATCOM database [6]. The proteins comparison scores were filtered for dissimilarities greater than 2.0 to create the full COG graph. Two sub-graphs were extracted from the full graph. The COGsimilar graph consisted of a set of COG families that had a large number of similarities between proteins in different families and was highly connected. The COGdissimilar graph consisted of proteins from families that had few relations. The graph was highly connected within families but had few connections between families. The COGsimilar graph contained 1770 vertices and 290,583 edges and the COGdissimilar had 2030 vertices and 158,724 edges.

The final two real graphs were built from the National Cancer Institute’s collection of chemical compounds and their AIDS screening annotations [29]. MolInspiration’s toolkit [5] was used to generate a list of properties for each compound, which were combined with the AIDS annotations to form the feature vector for each data item. The properties generated were: molecular weight, logP, total polar surface area (TPSA), number of atoms, number of hydrogen acceptors (nON) and donors (nOBNH), number of rule-of-5 violations, and number of rotatable bonds. Two graphs were generated from the data set, one containing the compounds that had AIDS screening results, NCAAll, and the other containing the 436 compounds that were confirmed active against AIDS (CA in the annotation data), NCIca. A similarity matrix was created for each data set using Euclidean distance and a feature vector containing all computed properties. The NCAAll matrix was thresholded at 10.0, resulting in a graph with 42,750 vertices and 3,292,778 edges. The NCIca graph was thresholded at 100.0 and had 436 nodes and 18,683 edges.

3.2 Algorithms

The next few paragraphs provide brief descriptions of the ordering algorithms and their complexity. Complexity is defined using big-O notation in terms of the number of vertices \( V \), the number of edges \( E \) in the graph, and \( m = \max\{\text{degree}(v) | v \in V\} \). More details are available from the main reference or the Boost Graph Library documentation [33].

Breadth-first search (BFS) and depth-first search (DFS) [39] are both foundational graph theoretic algorithms that visit each vertex in the graph in a predictable manner. When BFS visits a new vertex, it adds each vertex connected to the current vertex to a buffer and visits the vertices in the order they were added. It visits all nodes...
connected to the starting vertex first, and then proceeds to nodes discovered during those visits. DFS, on the other hand, visits the first vertex from each node in the order they are discovered before moving on to the second vertex in each node. Both DFS and BFS have \(O(V + E)\) time complexity.

Degree ordering is an attribute ordering that orders the vertices in the graphs by their degree, i.e. the number of edges connected to a vertex. For vertices with the same degree, the order is undefined. The running time for degree ordering is dominated by the cost of acquiring the degrees or by the cost of the sort operation, depending on what data structures are used. In our implementation, the adjacency graph data structure provided \(O(1)\) degree access, so the time complexity was determined by the sort algorithm at \(O(V \log V)\).

Reverse Cuthill-McKee (RCM) [12] and King’s [20] algorithm are variations on breadth-first search that use a local priority queue to select the next vertex to visit, based on the vertices that have been discovered but not already visited in the search. In RCM, the vertex to select the next vertex to visit, based on the vertices that have been discovered but not already visited in the search. In RCM, the vertex with the highest degree is visited next. Instead of ordering the vertices based on their total degree, King orders the vertices in its priority queue based on the number of edges each vertex has to already visited vertices. The time complexity for RCM is \(O(m \log(m) |V|)\) and the time complexity for King is \(O(m^2 \log(m) |E|)\).

In Sloan’s algorithm [34], start and end vertices are chosen to maximize distance between the two vertices and all other vertices are initially prioritized by their degree and their distance from an end vertex. Low degree, distant vertices have highest priority. Then one by one the algorithm goes through the vertices, increasing the priority of the children and grandchildren of the vertex. After ordering one vertex, the highest priority of the remaining vertices is chosen. The time complexity for Sloan is \(O(\log(m) |E|)\).

The separator tree ordering is based on the recursive partitioning algorithm in [4]. It uses graph separators, sets of vertices that can be removed to partition the graph into subgraphs with a fixed number of vertices, to partition the graph into subgraphs. Like the sparse matrix ordering algorithms, it was designed to reduce the storage requirements for large graphs.

Spectral ordering uses an order based on a constrained energy-minimal linear representation of the graph (in which the sum of all edge lengths squared is minimized). This is accomplished by finding the eigenvector corresponding to the second-smallest eigenvalue (the “Fiedler vector”) of the graph’s Laplacian matrix, and sorting the vertices according to their corresponding entries in the eigenvector [39]. If the vertices have unique values, spectral methods are stable with respect to the initial ordering. That is, spectral methods will return the same ordering for default ordered and shuffled graphs. Spectral methods are known to be able to expose some structures (Figure 3). These measures were designed to measure consistency of the visualizations for each ordering and how useful the visual features were. The term structure is used strictly to identify visual structures imparted by the ordering algorithm, and implies no underlying structure in the data. Detailed exploration of the interpretation of visual structures for individual algorithms is beyond the scope of this work.

Stability was determined by comparing the thumbnails (Figure 3). Images were determined to have structure if the overall visual density of the image was not uniform. Structures were deemed similar if, in the thumbnail view, the overall images looked similar. Interpretability measures the amount and type of structure present in the image and is assigned to all images, not just image pairs. Five interpretability categories were used:

- **Stable.** The ordered and shuffled images have structure and are similar in appearance.
- **Structure.** The ordered and shuffled images have structure but are not similar.
- **Ordered.** The ordered image has structure, while the shuffled image has no structure.
- **No structure.** Neither image contained structure.

Stability was determined by comparing the thumbnails (Figure 3). Images were determined to have structure if the overall visual density of the image was not uniform. Structures were deemed similar if, in the thumbnail view, the overall images looked similar. Interpretability measures the amount and type of structure present in the image and is assigned to all images, not just image pairs. Five interpretability categories were used:

- **Coarse and fine structure.** Structure is present at all levels of detail.
- **Coarse structure only.** Large structural features are present, but no features are discernible within the structures.
- **Fine structure only.** Small features, such as horizontal or vertical lines are visible in the large image, but there is no obvious coarse structure.
- **Minimal structure.** A minimal amount of algorithm-specific coarse structure is present, but the image otherwise has no structure.

run on an Apple dual 1.8 GHz G5 Mac with an nVidia 6800 graphics card and 1 GB of RAM. The BGL code was compiled using g++ 3.3 (build 1671) and the Python version was Apple framework build of Python 2.3. The versions of the Python libraries used are: OpenGL 2.0.1.04, Numeric 23.8, PIL 1.1.4. These libraries are all thin wrappers around optimized C and C++ libraries and have minimal calling overhead.

### 3.3 Implementation Details

Except for separator tree and spectral ordering, all algorithms were from version 1.33 of the Boost Graph Library (BGL) [33] and accessed using BGL/Python. All graphs and algorithms used the boost.graph.Graph class. The separator tree and spectral ordering algorithms were implemented in Python, using the BGL graph classes, with the latter using Numeric Python. For BFS and DFS, the starting vertices were the first vertices stored in the graph data structure. For RCM and King, the starting vertices were selected using the algorithms provided by the BGL. The experiments were...
• No structure. No structure is present in the image

Coarse structures are large features in the images that are composed of most or all of the edges in the graph and are often thick diagonals or blocky regions. Fine structures are composed of a few local edges and multiple instances of the same structure needed to be present for an image to get this rating. Some algorithms added a small amount of structure to otherwise random images. The minimal structure category distinguishes these cases from those where there truly is no visible structure. Interpretability assignments were also modified with a diagonal qualifier when the structure was only present around the diagonal.

These ratings were coarse gained enough that they almost every image and pairing was unambiguously assigned a stability and interpretability category. In the few cases where there was not an obvious assignment, the value closer to no structure was used.

3.5 Visualization

The visual similarity matrices were generated using a custom matrix viewer developed by the Open Systems Lab. The viewer is built in Python and uses the Python OpenGL bindings to render the matrix. Each dot in the matrix was drawn as a GL_POINT. The images were rendered using an nVidia 6800 graphics card at a resolution of 1000x1000 with point anti-aliasing (GL_POINT_SMOOTH) enabled. 1000x1000 images represent a realistic size for interactive, on-screen visualization. Each image was read directly off the card and saved as a PNG using the Python Imaging Library (PIL). The thumbnails were generated using the BICUBIC sampling algorithm from PIL and saved as 100x100 PNG files. For the paper, the images were processed to change the background from black to white using Photoshop 7.0 on OS X. The images were inverted then converted to grayscale and back to RGB to remove color information.

4 Results

The overall stability and interpretability results are listed in tables 1 and 2. All images used to generate these results are available online [26]. In the next few sections, we discuss the results from different perspectives. First, we present the stability and interpretability results. Then, we break down the results by ordering algorithm and graph type. We conclude with observations about scalability and the effects of the graph structures.

Table 1: Stability Results Stability measures the similarity in appearance between the default ordered images and the shuffled images. Stable, have both structure and are similar, Structure present, but not similar, Ordered only has structure, No structure

<table>
<thead>
<tr>
<th>Graph</th>
<th>Unit</th>
<th>BFS</th>
<th>DFS</th>
<th>King</th>
<th>RCM</th>
<th>Deg</th>
<th>Spec</th>
<th>Sr</th>
<th>Minor</th>
</tr>
</thead>
<tbody>
<tr>
<td>sw</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>S</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>pl</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>S</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>er</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>N</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>kp2</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>N</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>kp5</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>N</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>kp8</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>N</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>k12</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>N</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>k15</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>N</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>COnca</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>S</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>COGdis</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>S</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>NCaca</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>S</td>
<td>S</td>
<td>S</td>
</tr>
</tbody>
</table>

4.1 Stability

All algorithms except degree were stable or contained structure for both the default and shuffled orderings for a majority of the graph types. The control pairing (i.e., the default ordered image and the shuffled image) consistently generated good images for the default ordering and ‘noisy’ images for the shuffled ordering.

Table 2: Interpretability Results Interpretability measures the quality of the structure. Coarse grained structure, Fine grained structure, Minimal structure, Diagonal structure, No structure

<table>
<thead>
<tr>
<th>Graph</th>
<th>Unit</th>
<th>BFS</th>
<th>DFS</th>
<th>King</th>
<th>RCM</th>
<th>Deg</th>
<th>Spec</th>
<th>Sr</th>
<th>Minor</th>
</tr>
</thead>
<tbody>
<tr>
<td>sw</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>pl</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>er</td>
<td>N</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>N</td>
<td>M</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>kp2</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>N</td>
<td>C</td>
<td>N</td>
<td>N</td>
<td>C</td>
</tr>
<tr>
<td>kp5</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>N</td>
<td>C</td>
<td>N</td>
<td>N</td>
<td>C</td>
</tr>
<tr>
<td>kp8</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>N</td>
<td>C</td>
<td>N</td>
<td>N</td>
<td>C</td>
</tr>
<tr>
<td>k12</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>N</td>
<td>C</td>
<td>N</td>
<td>N</td>
<td>C</td>
</tr>
<tr>
<td>k15</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>N</td>
<td>C</td>
<td>N</td>
<td>N</td>
<td>C</td>
</tr>
<tr>
<td>COnca</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
</tr>
<tr>
<td>COGdis</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
</tr>
<tr>
<td>NCaca</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
</tr>
</tbody>
</table>

Figure 4: Three different orderings (left to right: BFS, DFS, degree) of a 100 vertex power law graph show envelope, horizon, and galaxy footprints, respectively.

On synthetic data, the orderings were almost always stable regardless of shuffling. The only exceptions were DFS and degree. For graphs with more fine grained structure, DFS generated different, but usable images depending on the initial ordering. Degree typically generated images that were indistinguishable from each other. However, when the degree distribution was meaningful to the graph type (e.g., power law), the degree orderings exposed this.

On real data, all algorithms generated images with some structure. BFS, DFS, King, RCM, and separator tree generated different, but usable structures for the default graph and shuffled initial orderings (the structure category using our ranking system). While degree and spectral were stable, the type of structure present in the images was difficult to use to explore the underlying data.

4.2 Interpretability

Coarse grained structure was present in almost all images and varied with the type of graph, size of graph, and algorithm. If there was no coarse grained structure in the graph, (e.g. bi-partite graphs), all algorithms except degree were able to generate consistent patterns. The patterns did not always reveal the intrinsic structure of the graph, but did provide clues to the presence of such a structure.

The algorithms resolved coarse structures better as the size of the graph increased. Smaller synthetic graphs contained too few nodes and too many random connections to reveal their structure except under the default ordering. However, as the size of the graph increased, consistent patterns developed in the images.

Some algorithms had a characteristic footprint (Figure 4) that was evident across all graph types. The graphs with the least amount of coarse grained structure (e.g., ER and 5-partite graphs) led to images with patterns dominated by the algorithm’s footprint. As the amount of internal structure increased in the graph, the footprint became less noticeable and the graph structure became more evident. The footprints fell into three main categories: envelopes, galaxies, and horizons. Envelopes form a bubble centered on the diagonal that contains all the edge dots and are characteristic of BFS-based algorithms. Galaxies are dense regions of dots centered on the diagonal. These are generated by the spectral and degree orderings. Horizon footprints are tight lines of dots that follow the main diagonal and sometimes branch at oblique angles off the main diagonal and are indicative of DFS and separator tree. Both envelopes and
horizons highlight the path the algorithm took through the graph. The edges that make up the feature are the edges that were directly used by the algorithm.

The small world and real graphs both contain fine-grained structure and all algorithms generated more detailed patterns that could be used to trace relations between vertices and groups of vertices in these graphs. This was most apparent with BFS, King, RCM, DFS, and Sloan, though separator tree also exposed some detail in the small world graphs. These fine-grained patterns were not dependent on the size of the graph and were present in the 100 vertex small world graphs. While the characteristic footprint for each algorithm was visible in these cases, it only affected the overall structure of the image at the boundaries of the more detailed sections. RCM and King both introduced a secondary footprint to dense regions of the images that added some order compared to BFS.

4.3 Algorithms

**BFS** BFS orderings are characterized by an envelope footprint. The BFS envelope is generally solid and the interior of the envelope can reveal some of the graph structure. In cases where there was coarse structure and the graph was not too dense (e.g., bipartite, 5-linear-partite, real data), the interior contained overall patterns of connectedness but could not separate them further. For instance, in the 5-linear-partite graph, BFS created a simple checkerboard pattern with two dark and two light squares that appear to wrap around the image. This reveals the partite nature of the graph but fails to provide more insight into how many sets make up the graph. Dense graphs, such as ER and 5-paritite, proved challenging and the interior of the envelope contained no structure. Small world graphs, on the other hand, had two consistent features: a solid envelope and a diagonal composed of many tight clusters. Between the diagonal and the envelope, a few small clusters and outliers occupied an otherwise empty expanse. In the real world graphs, BFS produced tight clusters with little internal structure connected by envelopes.

**DFS** DFS orderings varied the most between default and shuffled initial orderings, but always left a horizon footprint that exposes the path it took through the graph. If there is fine-grained internal structure in the graph, DFS generated images with fine-grained features. Small world graphs resembled Rorschach ink-blots that varied in appearance depending on the initial ordering of the graphs. A strong diagonal was always present and clusters either trailed off at oblique angles or sat away from the diagonal in the plot area. On real world graphs, the diagonal tied together components, similar to those found in BFS images. However, unlike BFS, DFS generated some internal structure in the components, leaving a distorted checkerboard with rectangular squares of various aspect ratios connected by stocky, angular clusters.

For shuffled power-law graphs, DFS consistently generated a secondary footprint. In power law graphs, the diagonal eventually trailed off, but before it did, a strong horizon-like front emerged orthogonally from it. The region preceding the horizon contained the majority of the edges and the area outside this region was void save for a few small clusters clinging to the diagonal. The horizon effect was also evident in the 5-paritite and 5-linear-partite graphs where it was strong enough that not even the diagonal passed through. In the latter case, the empty region past the horizon serves the same purpose as the empty squares generated by BFS and hints at the partite nature of the graph.

**Degree** Degree orderings were the poorest overall. Degree ordering is an attribute ordering on the vertices, and in most graphs the degree distribution is fairly small. Thus, large numbers of vertices have the same degree, and within those sets the initial ordering dominates. Power-law graphs were an exception and degree ordering immediately revealed the power-law nature of the graph. The nodes with the highest degree are generally interconnected and formed a dense galaxy footprint around the origin. Real world graphs also showed some structure under degree ordering. Often, cluster within the data were cliques or near cliques. Because all vertices in the cluster had the same degree and the sizes of the cluster tended to be unique, clusters appeared using degree ordering.

**Separator Tree** Separator tree orderings were characterized by ‘fat’ diagonals and variations in the density of remaining dots when structure was present in the graph. Separator tree was one of the algorithms that nearly reproduced the default ordering for a graph type. The diagonal produced by the algorithm recovered the overall visual structure of the default small world graph ordering for the shuffled small world graphs. The effect improved as the number of vertices increased. In power-law graphs, a dense cluster sat on the diagonal that decreased in density further out from its center, similar to the power-law structure at the origin in degree orderings. On real world graphs, separator tree separated the graph into clusters and generated localized patterns similar to those of the synthetic graph in each cluster.

**RCM and King** RCM and King both refined the images produced by BFS (Figure 5). The envelope was still evident, but when there was more structure to the graph, both algorithms generated regular patterns in the regions within the envelope. The 5-linear-partite graphs demonstrated this effect most vividly but it was also visible in the real world graphs. RCM adds long, wedge shaped striations to the clusters and King imparts a crosshatched structure, with short wedges making up each hatch.

**Sloan** Sloan ordering revealed structure in all graphs that contained structure and was the only algorithm without a characteristic footprint. Rather, Sloan generated images reminiscent of the default orderings and enough features were present to identify the overall structural features in the graphs. Small-world graphs had strong diagonals and k-paritite and k-linear-partite graphs all had defined block structures. Of the synthetic graphs, power-law graphs resolved the least. A single dense region formed, but it was not as tightly packed as in other orderings. On the real data, Sloan packed the edges along the diagonal in block structures that all contained some internal structure. As with the Sloan orderings for synthetic data, the Sloan orderings for real data were visually similar to the ‘expert’ default orderings.

**Spectral Decomposition** Spectral decomposition performed well on structured graphs and in the case of 5-linear-partite graphs with greater than 10% connectivity, completely recovered the visual structure of the default ordering. For other graphs and less connected 5-linear-partite graphs, it generally produced a galaxy footprint with the shape of the galaxy indicative of the graph structure. Small world graphs with few nodes appeared as fat diagonals that spread out into edge-on galaxy views as the number of nodes and rewiring probabilities increased. Power-law graphs were square galaxies. The various k-paritite graphs all contained distinct holes that were indicative of the number of sets in the graph. As the size...
of and connectivity of the graph increased, these holes resolved into the gaps in the default ordering. It is known that eigenvalues, the basis for spectral ordering, can reveal properties about regular and bipartite graphs [39], so the results for small world and k-partite graphs are not unexpected.

While spectral ordering generated good images for the synthetic data, it did not work as well on real data. If the real data was densely connected and had small world properties (e.g., NCACA), it gave orderings similar to BFS. However, when the relationships between vertices were more complex, the data resolved to a few clusters with little internal structure.

4.4 Graph Types

Among the graph types, small world and 5-linear-partite graphs almost always had useful structures visible. For small world graphs, most orderings generated strong diagonals and showed the lack of relations among most of the vertices. 5-linear-partite graphs, especially with a large number of nodes and high connectivity were easily ordered and resolved into images that contained important clues to their structure. On the other hand, ER and 5-partite graphs almost never contained useful structures. For ER graphs, this is expected. 5-partite graphs, with lots of connections between the sets, are very close to ER graphs and thus display similar behavior. Power law graphs almost always had some structure that showed a power-law distribution in the connections but, except for the curious horizon pattern generated by DFS, rarely had any other consistent structure.

Real world graphs (Figure 6) always contained some structure, though in some cases it was only at the level of the densely connected regions within the graph and little revealing structure. For the large real world graphs, many of the final images looked the same, regardless of the ordering algorithm. This is because, at the resolution used, only the connected components appear as small clusters. In fact, the components are large and zooming in on them reveals the structure more characteristic of each algorithm.

5 Discussion and Conclusions

Visual similarity matrices have a much higher information density than most information visualization techniques. For graphs, traditional node-link graph diagrams can only display tens to a few hundred nodes with a small number of edges before they become unreadable. Scatter plots can display a large number of points, but require multiple visual dimensions (e.g., size, color) to display complex relations. Similarity matrices, when ordered properly, can easily display thousands of vertices and millions of edges with detailed structures between them. In this study, we examined a core set of algorithms for ordering vertices to generate usable images. Most algorithms were capable of coaxing some visual information out of the graphs, but none consistently generated images good enough that it could be blindly applied to any data set and reveal its structure. Despite this, there are some important conclusions, directions for future research, and practical advice for using similarity matrices that can be derived from this study.

First, and most importantly, if structure is present in the data, the ordering algorithms will consistently provide clues to it. The characteristic footprints of many of the algorithms provide a basic framework for exploring the data. Once the algorithmic artifacts are identified, they can be used to trace relationships (e.g., blots in DFS, striations and crosshatches in King and RCM) or identify broad trends in the data (e.g. galaxies in separator tree and spectral orderings).

The connectedness and amount of randomness in the graph is directly related to the quality of the ordering. Dense graphs with lots of random connections rarely resolved into usable images. Sparse graphs, especially those with structure, tended to resolve to good images. This suggests that care should be taken when filtering data to produce the graph and multiple versions may be necessary to find the correct filtering level. While locally dense regions will still suffer, the overall structure can be determined and local regions processed independently.

The algorithms that looked at the local and global properties tended to produce the most detail. Whereas breadth-first search showed coarse structure, the modifications made by King and RCM to take into account local features improved the patterns in the images. Sloan generated very good images by considering the starting and ending vertex from the beginning. One caveat is that when algorithms worked off a local priority queue, such as those used in King and RCM, the resolving power was limited to the average size of the queue, which was often quite small. New algorithms should look to exploit and extend these techniques by identifying other methods of including more global information in the algorithm beyond starting and ending nodes and better ways of managing the priority queues.

When used for data analysis, these ordering algorithms are most appropriate at the exploratory stage. As part of a pipeline for examining data, different algorithms can be applied with different initial orderings. In the same way that many different attributes are used as axes for scatter-plots in scatter-plot matrices, different vertex orderings can enable a user to quickly see the structure of the data from many different perspectives. Combined and linked with other interactive visualization tools, the user can use the maps provided by the orderings to identify and explore regions of interest within the data. Our results with augmenting similarity matrices with other views of the data suggest that this is a powerful method for studying large data sets [27].

Because the use of automatic ordering algorithms is relatively new for similarity matrices, the types of patterns and their meanings are not fully understood in analytical contexts. Typically, attribute orderings are scanned and those that contain block structures are accepted. As these algorithms show, structural orderings contain other types of patterns that are repeatable and can reveal more information about the data. As these techniques get applied to more real data sets, a common vocabulary for describing the features could emerge that gives analysts another tool for discussing data [27].

For this study, we looked at algorithms that relied only on the structure of the data. However, more information is present in the graphs, often in the form of edge weights or attributes on the vertices. Most of these algorithms could be modified to use this additional information and possibly produce better orderings.

This study systematically examined a number of vertex ordering algorithms against a collection of real and synthetic data and described and evaluated the results. Visual similarity matrices have the potential to be a powerful visual analytical tool, but still have some hurdles to overcome. Our results suggest that algorithms designed for graph and sparse matrix analysis are useful for visually discerning coarse and fine grained features in data sets. Further research will help refine these algorithms for regular use on real data and to develop algorithms designed specifically for visualization.

6 Acknowledgments

This work was funded by a grant from the Lily Endowment. Doug Gregor provided support for the Boost Graph Library and its Python bindings. Jeremiah Wilcock supplied the separator tree implementation and many useful insights through the course of this project. This work was originally suggested by Sun Kim, who maintains the PLATCOM database and provided important feedback early on in the project. Katy Börner’s group provided feedback and end-user perspectives that helped drive this study.
Figure 6: All orderings for the COGsimilar data set. The top row contains orderings using the default initial ordering from the COG database and the bottom row used the shuffled initial ordering.

REFERENCES


[29] NCBI. Clusters of Orthologous Groups of proteins (COGs).


