

Sparse Matrix Reordering Algorithms for Cluster Identification

Chris Mueller

For I532, Machine Learning in Bioinformatics

December 17, 2004

Introduction

The dot plot (Figure 1) is a technique for displaying relationships between elements in a data set. Dot plots are commonly used to visually compare genomic sequences (Gibbs 1970) and inspect the structure of sparse matrices in scientific applications (Loos 1994). They have also been applied as a navigation tool for exploring large sparse graphs (Abello 2004). To generate a dot plot, each element is represented as a row and column in the matrix and a dot is drawn for each pair of elements (i,j) when elements i and j are related. Different types of data generate different visual patterns in dot plots that can help reveal the structure of the relationships in the data.

One major drawback of this technique is that the order of the data elements significantly impacts the effectiveness of the visualization. Genomic dot plots show important relations precisely because the genomes are already ordered. The visible structure

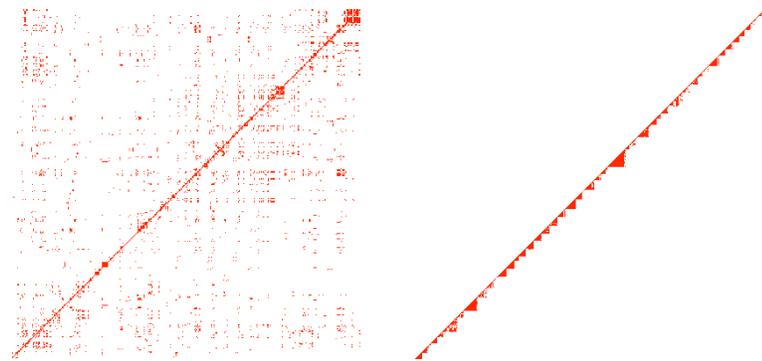


Figure 1 A dot plot with unordered and ordered vertices. The clusters in the ordered plot are clearly visible as triangles on the diagonal.

of sparse matrices and graphs in dot plots relies heavily on the order of the elements. Poorly ordered plots are indistinguishable from noise. A good ordering, however, can instantly reveal complex relationships between elements and even provide a mechanism for visually identifying clusters within the data.

Many fields have encountered the problem of reordering data to visually reveal clusters and properties of data. As long ago as the 19th century, archeologists were using *seriation* to create chronological orderings of artifacts. Some versions of seriation use dot plots to guide the process. The statistics and data analysis communities have used the dot plot, also known as the *shaded similarity/proximity matrix* for visual data analysis for at least the last 40 years (Wang 2002). More recently, shaded similarity matrices have been used in automated and interactive clustering and classification tools (Wang 2002, Strehl). Density-based clustering (Ester 1996), a technique for creating clusters based on the local density of data, has also been extended to provide orderings of data sets (Ankerst 1999) that, when used with reachability plots, display not only the smallest clusters, but also any hierarchical structure that may exist among clusters. The statistical

and density-based techniques generally rely on a valid distance metric to compute relationships between elements.

In scientific computing, the order of the elements in sparse matrix often affects the performance of numerical algorithms. Various algorithms exist for reordering the matrices to achieve certain properties that can yield better performance (George 1981). These reordering algorithms ignore the actual values of the elements and instead work on their relational structure, removing the need for a distance metric between elements. The resulting structure of the matrices, however, often produces dot plots that are visually similar to shaded similarity plots and groups similar elements together, potentially revealing clusters within the matrix.

In this paper, we explore the effectiveness of using the Reverse Cuthill-McKee, King's, and Modified Minimum Degree algorithms for sparse matrix reordering along with dot plot visualizations to reveal clusters within a data set.

Sparse Matrix Reordering

Sparse matrices are the main data structures in large-scale scientific and engineering applications for representing linear systems of equations. Many linear systems have thousands of variables, but each individual variable usually only depends on a few other variables. This leads to equations where most of the coefficients are zero. Rather than allocating space for every element in a matrix, sparse matrix data structures exploit this feature and try to minimize the amount of memory used by only allocating memory for the *non-zero* elements and elements that are used directly by an algorithm. In sparse matrix terms, non-zero elements are elements in the matrix that contain values or may be used by an algorithm, not just those elements that have non-zero values. (George 1981) provide a comprehensive survey of algorithms for operating on symmetric positive definite matrices, which are a main class of matrices used in practice. Symmetric matrices are matrices where the value at element (i, j) is the same as the value at (j, i) , which, as we discuss below, is an important feature for representing relational data as sparse matrices.

In this paper, we study the applicability of three reordering algorithms for reordering data to visually reveal its structure, two that reduce the *bandwidth* and one that reduces the *non-zero* structure of sparse matrices.

Banded representations of sparse matrices store only the diagonals of the matrix that contain non-zero elements. The *bandwidth* of a sparse matrix is the maximum distance between two elements in any row of the matrix. Often, the elements in a sparse matrix can be reordered so that the bandwidth of the new matrix is smaller than the maximum possible bandwidth. For banded storage schemes, the amount of memory required to store the matrix is directly proportional to the bandwidth, so finding orderings that minimize the bandwidth is important for reducing storage costs. Reverse Cuthill-McKee and King's algorithm (George 1981) are the bandwidth minimization algorithms that we study here.

Some sparse matrix algorithms work best if the non-zero structure of the matrix is minimized. Factoring algorithms often use elements in the matrix that were initially null (zero) elements, turning them into non-zero elements. These elements can be identified before the algorithm is executed and there exists an ordering for the elements in the matrix such that the number of additional non-zero elements is minimized. Finding the

optimal ordering is usually not possible, but finding a good ordering is. Good orderings tend to structure the matrix such that all elements fall below the main diagonal and may be useful for cluster analysis. The most common algorithm for reducing the non-zero structure of a sparse matrix is *modified minimum degree* (George 1981).

Representing Sparse Matrices as Graphs

The algorithms studied here process sparse matrices as graphs. A graph is a pair, (V, E) where V is the set of vertices in the graph and E is the set of pairs (v_i, v_j) such that $v_i, v_j \in V$ and there is an edge between (v_i, v_j) . For directed graphs, the edge goes from v_i to v_j and (v_i, v_j) and (v_j, v_i) are distinct edges, for undirected graphs, they represent the same edge. A sparse matrix is represented as an undirected graph by adding each row and column to V and adding an edge (nz_i, nz_j) to E for each non-zero element at position (i, j) in the matrix. If the matrix is symmetric, then only the rows (or columns) need to be added to V .

Representing Data as Sparse Matrices

In order to take advantage of the sparse matrix algorithms, we need to convert its relational structure to the format used by the sparse matrix algorithms. Because the algorithms we are studying use graphs as their native data structure, this is a fairly trivial task. For each element d_i in the data set D , add d_i to V . For each d_i, d_j in D , if there is a relationship between d_i and d_j and the edge (i, j) does not already exist in E , add the edge (i, j) to E . Note that this representation avoids any formal definition of the relation between d_i and d_j , it simply states that a suitable relation must exist. This is in contrast to clustering algorithms that require a valid distance metric between data elements.

Algorithms

The next few sections describe the ordering algorithms in detail.

Reverse Cuthill-McKee

Reverse Cuthill-McKee (RCM) is a bandwidth minimization algorithm that takes a graph G and produces an ordering its vertices. The algorithm is:

```
RCM(G) :
  v = ChooseStartNode(G)
  vertices = []
  for each node reachable from v:
    print(v)
    n = unvisited_neighbors(v)
    sort_by_increasing_degree(n)
    vertices.append(n)
  v = vertices.pop()
```

Starting with a suitable vertex, output that vertex as the first in the new ordering. Then, find the unvisited vertices that have edges to the current vertex and sort them based on their degree, putting the ones with the smallest degree first, and add this list to the end of the list of vertices to visit. Set the current vertex to the first vertex in the list and

repeat until all vertices have been visited. Using an adjacency list representation for the graph, RCM has a running time proportional to $O(|E| + \max_degree(V))$.

One important thing to note is that the choice of the starting node can affect the quality ordering produced by RCM (Berry 2002).

King's Algorithm

King's algorithm is a variation on RCM. Instead of ordering the vertices based on their total degree, it orders first based on the number of edges they have to *already visited* vertices. This is based on the idea that if a vertex has many edges that connect it to vertices that have already been ordered (visited), then it may be part of a local cluster and should be ordered closer to the other nodes in its cluster.

Modified Minimum Degree

Modified minimum degree (MMD) attempts to reduce the non-zero structure of a sparse matrix by ordering the edges based on their degree. Given an undirected graph G , MMD works as follows:

```
MMD(G) :
  while G has vertices:
    v = vertex_with_smallest_degree(G)
    print(v)
    G.remove(v)
```

MMD repeatedly chooses the edge with the smallest out degree, outputs it, and removes it from the graph. The complexity of MMD is $O(|V|^2|E|)$ but in practice is much better.

Experimental Methods

In order to evaluate the results of the orderings and compare the different approaches, we developed an experimental protocol and set of quantitative and qualitative measures for the results. Additionally, we implemented a visualization tool that allowed us to perform basic visual comparisons between different orders and explore in detail the structure of the sparse matrix.

Protocol

The protocol for generating new orderings for a given data set is:

1. Create a weighted edge-list graph of the data
2. Filter the edges based on some condition
3. Create a copy of the graph and randomly reorder its edges
4. For RCM and MMD, generate sub-graphs for the connected components in the original and shuffled graphs
5. Reorder the vertices of the original and shuffled graphs and sub-graphs using RCM, King, and MMD
6. Generate metrics for each ordering and graph
7. Generate metrics for a connected components ordering of the graph

The first step creates an edge list graph of the data by iterating of through the data items and adding an edge to the graph for each combination of elements, using the value of the relation between the elements as the weight. The next step prunes the graph by removing all edges that do not meet some condition. For numeric weights, this is simply a threshold that filters for significant relationships (e.g. $relationship(i, j) > 100.0$). Because all algorithms are sensitive to the initial ordering of the edges, a copy of the graph is made and the edge-list is shuffled. Comparing the results of the original and shuffled versions of the graph allows us to better identify results that are independent of the initial edge ordering. RCM and King use a vertex's neighbors to build the list of vertices to visit and as such do not span disconnected components. To make sure all vertices are visited, the graph is first grouped into its connected components. Then, RCM and King process each component. MMD processes the graph in its entirety. Once the orderings are generated, metrics are computed for the original and shuffled graphs. Additionally, an ordering based solely on connected components is processed as a control for RCM and King. If the results of RCM and King do not differ significantly from the results of connected components, then connected components is most likely responsible for the results.

Each experiment produced two sets of metrics, one for the original ordering and one for the shuffled ordering.

Data

All experiments were performed on protein comparison data for the proteins in the COG database. All proteins was compared using FASTA and resulting protein-protein comparison matrix was filtered for scores above at three levels: 30, 90, and 200. 90 was the average score in the original dataset and 30 and 200 were chosen somewhat arbitrarily to represent a noisy and clean data. The COG data in part because it is based on clustering by similarity scores, though the COG analysis pipeline includes more domain knowledge than our algorithms.

Evaluation Metrics

For each graph, the *number of edges*, *number of vertices*, and *number of connected components* was computed. These metrics are dependent on the results of the edge filtering, but do not depend on the order of the edges. It is possible that some vertices have no edges once the filter was applied and are removed from the graph. This, along with the removal of edges by the filter, has a strong impact on the number of connected components within a graph.

For each new ordering, the *bandwidth* of the resulting sparse matrix was computed. This value depends on the ordering of the edge list and the algorithm used and can be used to determine how well a given algorithm groups elements into diagonal bands.

Finally, given a new ordering of the elements, the width of previously known clusters was determined. The *cluster width* is defined as the distance from the first element in the cluster to the last element in the cluster, based on its index in the ordering. In an ordering that recovered the clusters, these distances would simply be size of the clusters. Any difference between the cluster width and the known cluster size suggests that the ordering failed to recover the known clusters.

Visualization

To visually inspect the matrices and orderings, a simple visualization tool was developed. The tool took as input a matrix, an order file, and a set of known clusters. It rendered the matrix on a grid, using the order file to assign the x and y coordinates for the vertices. For each known cluster, it drew a horizontal line from the first instance of a node in the cluster to the last instance of a node in the cluster. If the known clusters were present in the ordering, none of these lines overlap (Figure 2).

The visualizations help provide a subjective measure of the quality of the resulting clusters. As we discuss in the results, this proved to be key for determining the overall effectiveness of the techniques.

Implementation Details

All algorithms were implemented in C++ using the Boost Graph Library (BGL) from Version 1.31 of the Boost Library (Dawes 2004). The platform used to implement and test the algorithms was an Apple Dual 2 GHz PowerPC G5 with 3.5 GB DDR SDRAM running OS X 10.3.6 (Darwin Kernel Version 7.6.0). All code was compiled with g++ 3.3 (build 1671) from Xcode 1.5, with the `-O3` flag set. The graph and bandwidth metrics were generated using the `num_edges()`, `num_vertices()`, `connected_components()`, and `bandwidth()` from the BGL. The cluster width metrics were generated using a custom Python program. The visualization application was written using Python and PyOpenGL 2.0.1.04. Python 2.3 (build 1495) on Darwin was the Python interpreter.

Results and Discussion

The final results for each graph and algorithm are shown in (Table 1). As would be expected, the number of distinct connected components is inversely proportional to the filtering level. The noisiest (filter = 30) data had the fewest connected components and the cleanest (filter = 200) data had the most. For all graphs, the starting bandwidth was similar and very close to the maximum possible bandwidth (the number of vertices in the graph).

The bandwidth minimization algorithms generated graphs with bandwidths inversely proportional to the number of connected components while MMD had little effect on the bandwidth. The bandwidth for the original and shuffled data sets for each algorithm had little variation, suggesting that the initial ordering of the data has little effect on the final bandwidth for each algorithm. The connected components, RCM, and

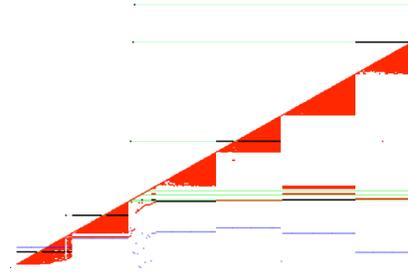


Figure 2 The sparse matrix/cluster visualization tool. The axes represent the vertices in the graph and red dots show an edge between vertices (i, j) . The green lines show the extent of a known cluster and black dots on the green lines show the members of the cluster. The blue lines can safely be ignored.

Table 1 Graph statistics and bandwidths for all orderings. CC is connected components.

Graph Statistics					Ordering Bandwidths				
Filter	Sort	Edges	Vertices	CC	Starting	CC	RCM	King	MMD
200	original	315939	70881	35105	62911	177	172	170	69251
200	shuffle	315939	70881	35105	68467	176	170	172	70464
90	original	978896	76589	18257	70244	1712	1649	1667	76045
90	shuffle	978896	76589	18257	73028	1707	1644	1653	76420
30	original	3055873	77114	161	76899	76325	61454	62711	76827
30	shuffle	3055873	77114	161	76915	76068	62074	62665	76944

King orderings all significantly reduced the bandwidth of the (filter = 200) and (filter = 90) clusters. Connected components ordering had little effect at the (filter = 30) level but RCM and King managed to improve on the connected components order by about 18% at this level.

Visually (Figure 3), the bandwidth minimization algorithms clustered the points along the diagonal, creating an image similar to those generated by ordering the vertices by the known clusters. However, as the cluster width bars suggest, the new orderings failed to effectively recover clusters from the data.

The average cluster width statistics are shown in (Table 2). The average size of a COG cluster is 21 (filter = 200) and 22 (filter = 90|30) and does not exhibit a strong dependency on the filter. The average width of the clusters under the bandwidth minimization orderings is much higher and is proportional to the filtering level. Starting with data that contains some order relative to the clusters, the bandwidth minimization algorithms actually make the clusters harder to identify. For shuffled data, the average cluster width is significantly higher. The change in the average cluster width as the data goes from ordered to random demonstrates that the algorithms are not effective at recovering COG clusters.

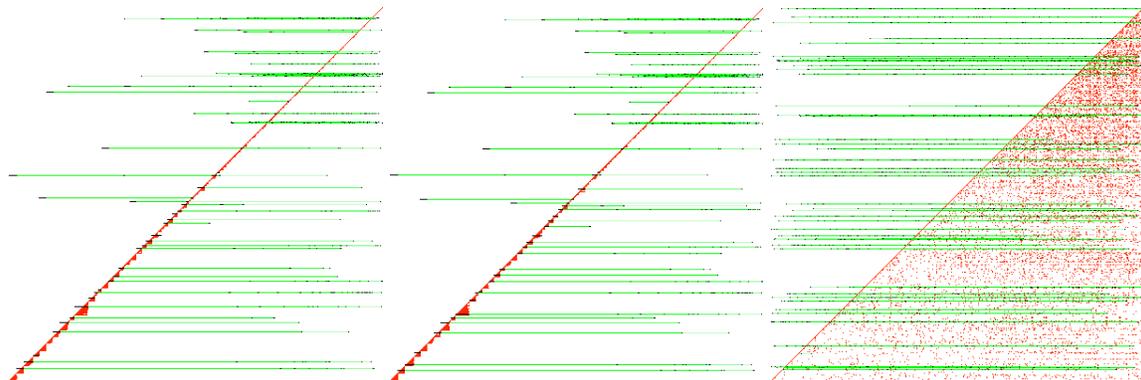


Figure 3 RCM, King, and MMD orderings for the first 49 COG families. Note that RCM and King are dominated by the connected component processing step and are very similar. In all orderings, the COG cluster widths (green lines) are much larger than the size of the COG families.

Table 2 COG cluster size and average cluster width for each ordering.

Filter	Sort	COG Size	CC	RCM	King	MMD
200	original	21	175	175	176	8562
200s	shuffle	22	33015	33015	33015	48383
90	original	22	824	823	823	16632
90s	shuffle	21	20696	20685	20685	53628
30	original	22	40528	34774	35634	54498
30s	shuffle	22	47560	34776	34728	59083

Further analysis also demonstrates that the bandwidth minimization results are dominated by the connected components preprocessing step. (Figure 4) shows the connected components and RCM cluster-width for each cluster. The solid diagonal lines for the (filter = 90|200) plots show a strong correlation between the two measures. The effect of RCM is only a slight change in cluster width; most of the change comes from connected components.

(Figure 5) shows a subset set of the connected components in the (filter = 200) data set before and after RCM and King are applied to the data. From this, it is clear that the coarse grained ordering is determined by connected components and RCM and King only have minor effects within the connected components. Additionally, RCM and King do not necessarily group related vertices together in a way that reveals any clustering structure.

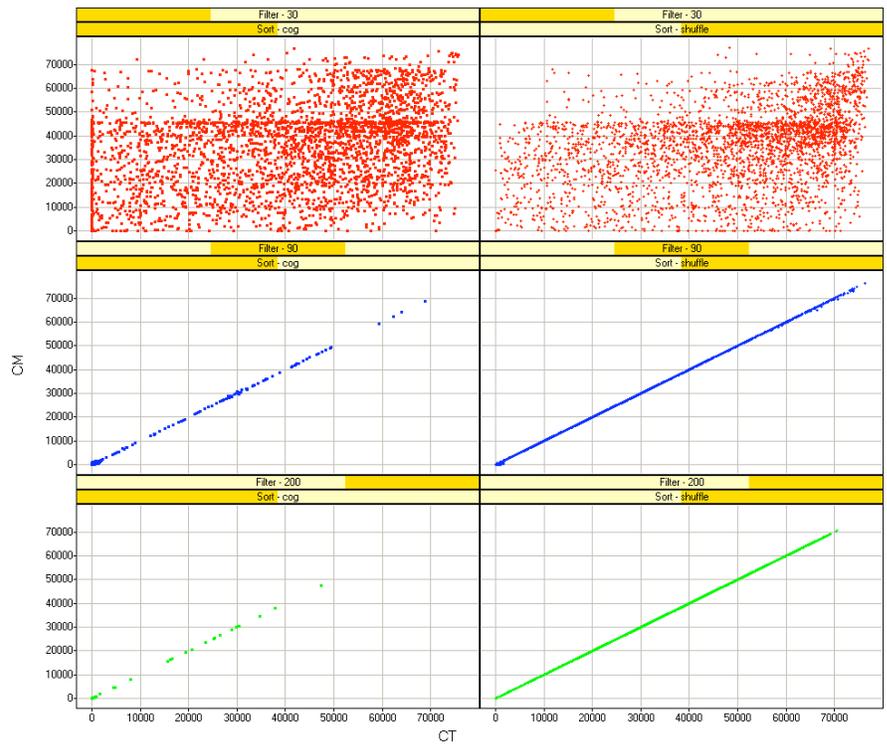


Figure 4 The relationship between connected components (CT) and RCM (CM) cluster widths. The near perfect diagonals show a strong correlation between both measures.

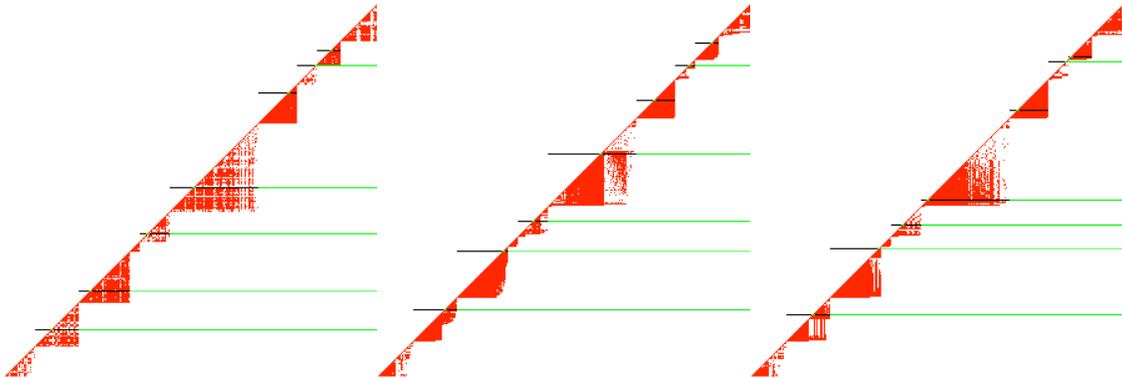


Figure 5 Close-up of the visual structure of connected component, RCM, and King components

The results of the experiments suggest that the sparse matrix reordering algorithms are ineffective at generating useful clusters based on a simple binary metric (the presence or absence of an edge between two vertices). Connected components, already well known as a rough clustering tool, dominated the results and the reordering algorithms were unable to recover clusters, even within connected components.

Future Work

This project only examined the algorithms as they are defined for sparse matrix reordering, and made no attempt to modify the algorithms to address their deficiencies. As (Figure 5) shows, both RCM and King produce predictable patterns within connected components. King, with its focus on keeping related vertices together, generated visual patterns that tend to be densest at the bottom. RCM, on the other hand, leads to multiple dense bands within a connected component. Most relational data has more information than just a binary measure, so extensions to RCM and King that adjust the order that vertices are visited based on a more detailed comparison may give more control over the final visual structure. Meaning is embedded visually in the results and if tools can be developed to produce patterns when certain relationships are present, reordering algorithms could become a powerful analytic tool.

In addition to processing the data directly, the reordering algorithms might also be applied to hyper-graphs of already ordered data. If algorithms can be developed to control the visual patterns generated, then a logical extension of the algorithms would be to have them process the graph dynamically, sometimes working on clusters and sometimes processing individual nodes. This could enable more sensitive searches in dense areas while still allowing coarse grained ordering across the whole graph.

References

- [1] B. Dawes and D. Abraham, *Boost*, www.boost.org. 2004.
- [2] K. H.-P. Ester M., Sander J., Xu X., *A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise*, *KDD*, Portland, OR, 1996, pp. pp. 226-231.

- [3] A. George and J. W.-H. Liu, *Computer Solution of Large Sparse Positive Definite Systems*, Prentice-Hall, Inc, Englewood Cliffs, NJ, 1981.
- [4] A. J. Gibbs and G. A. McIntyre, *The diagram, a method for comparing sequences. Its use with amino acid and nucleotide sequences*, Eur J Biochem, 16 (1970), pp. 1-11.
- [5] F. v. H. James Abello, *Matrix Zoom: A Visual Interface to Semi-External Graphs*, in IEEE, ed., *IEEE InfoVis*, Austin, TX, 2004.
- [6] M. M. B. Mihael Ankerst, Hans-Peter Kriegel, Jörg Sander, *OPTICS: Ordering Points To Identify the Clustering Structure*, ACM SIGMOD '99, 1999.
- [7] A. Strehl, *Relationship-based Clustering and Cluster Ensembles for High-dimensional Data Mining*, Graduate School, University of Texas at Austin, Austin, TX, 2002.
- [8] T. Loos and R. Bramley, *Emily: A visualization utility for large matrices*, Technical Report 412, Indiana University, Bloomington, IN, 1994.
- [9] J. Wang, B. Yu and L. Gasser, *Classification Visualization with Shaded Similarity Matrix*, *IEEE Visualization*, Boston, MA, 2002.