

# Applying Visual Similarity Matrices to Chemical Databases

Chris Mueller  
December 1, 2005  
1571 Final Report

## Introduction

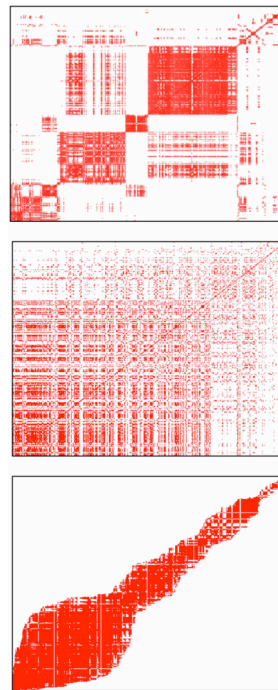
Matrix-based visualizations are a useful tool for exploring relationships between related records in a data set. They have been used for over 100 years on small and medium sized data sets in many fields. Recently, visual similarity matrices (Figure 1), a class of matrix-based visualizations, have become popular for visualizing large graphs and similarity matrices. We have been exploring their use for interactive exploratory analysis of large data sets [1].

Similarity matrices display the edges between vertices in a graph as points on a plot. Each tick on the axes is one vertex and a point is drawn if there is an edge between the vertices. For data collections, the vertices (ticks) are the data records and edges (points) are present if there is a relationship between the two vertices. Relationships can be any relation between two records, but are generally similarity or dissimilarity measures. For most data sets, only significant edges are added to the graph, determined by a threshold value. Thresholded matrices are usually fairly sparse, a property that makes it possible to apply a large class of efficient graph and sparse matrix analysis algorithms to them.

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. They have one major drawback, however, that affects their utility. The order of the vertices on the axes has a major impact on the interpretability of the resulting visualization. A poorly ordered collection of vertices leads to a plot that is indistinguishable from noise (Figure 1, middle) whereas a good ordering can reveal complex relationships and structures within the data set (Figure 1, top and bottom).

In our previous work, we have studied existing sparse matrix ordering algorithms for use in similarity matrix visualizations. These algorithms were designed to minimize the amount of memory used during sparse matrix computations by compressing the matrices into efficient storage structures. These structures generally group related matrix entries together as close to the diagonal of the matrix as possible. When translated to visualizations, the effect is a simple clustering of edges that is useful as a first step for exploratory data analysis. For past studies, we used existing data sets and artificial graphs, but never went through the entire process from generating the graph/matrix to visualizing it. Thus, we were not sure how the visualizations and their related analyses would fit into a larger analytical pipeline.

In this project, we begin to study the entire analysis process by taking a data set from collection through the initial visualization pass. Using data collected from public chemical databases, we clean the data, generate the similarity matrices and use the data in our custom tool. In this paper we detail our experience doing this and begin to elucidate the formal process, identify bottlenecks in it, and find ways to improve the existing tools and algorithms.



**Figure 1** Three views of a data set using a visual similarity matrix. The top view is the data as ordered by the curator, the middle view is a random ordering, and the bottom view was ordered using the Cuthill-McKee sparse matrix ordering algorithm. The views are symmetric around the diagonal.

## Methods

The analytical pipeline employed in this study is based on those used for cluster analysis and has the following steps:

- 1) Data collection
- 2) Data preparation
- 3) Similarity computation
- 4) Visual analysis/ordering

Steps 1 and 2 were performed once, but 3 and 4 were iterated multiple times to help understand the process. Parallel to this pipeline was a development process that added features and algorithms to our existing tools as required.

### ***Data Collection and Preparation***

The data set used was the National Cancer Institute's collection of compounds [2] and the AIDS screening annotations [3]. The SD file for the entire data set was downloaded and MolInspiration's toolkit [4] was used to generate a list of properties for each compound. These were combined with the AIDS annotations and stored in a database. From the database, three main data sets were extracted:

- 1) NCIall - the entire collection of compounds (~250k compounds)
- 2) NCA - the subset of compounds with AIDS screen results (~42k compounds)
- 3) NCAca - the subset of compounds confirmed active against AIDS (436 compounds)

Generating the properties was straightforward with MolInspiration. It required a smiles string or SD file and generated a table of properties based on command line parameters. It took 13 minutes to generate the entire property table on a 2 GHz G5, making it practical for quick, off-line property computation for large compound collections. The properties generated were: molecular weight, logP, total polar surface area (TPSA), number of atoms, number of hydrogen acceptors (nON) and donors (nOHNH), number of rule-of-5 violations, and number of rotatable bonds.

### ***Similarity Computation***

The similarity computations were performed on feature vectors derived from the computed properties. Any combination of properties could be used to compare compounds using Manhattan distance, Euclidean distance, or cosine similarity. For this study, we intended to generate three similarity matrices for each data set using each distance. The similarity matrices used the following feature vectors:

- 1) Float (mol. weight, logP, TPSA)
- 2) Rule-of-5 (nOHNH, mol. weight, logP, nON)
- 3) All (all computed properties)

For this study, no scaling or normalization was performed on the feature dimensions.

Unfortunately, setting an appropriate threshold value for each combination of (data set, measure, vector) proved challenging and the final matrices are a subset of the 27 that were possible. Much of the processing time for this project was spent generating different versions of the graphs at different threshold levels to try to determine a good setting for the threshold value for visualization. In the end, we limited the matrices to Euclidean distance and generated them for only the 'All' property vector. The rule-of-5 property vector was intriguing, but had many identical elements, making filtering difficult for the large data sets.

Once the matrices were generated, a second version of each matrix was created by shuffling the edges (Figure 1, middle). Often, the data curator orders the original data sets in a meaningful way that

leads to a good initial plot. The shuffled version of the data set helps detect biases in the original ordering of the data and show which algorithms are not susceptible to those biases.

The similarity matrices were generated by a custom C++/Python application that loaded the property vectors from a file or the database -- due to firewall restrictions, not all processing nodes had direct access to the database. The main comparison loop was written in C++ and a Python application loaded the data sets and made a call to the C++ routine to generate the results. For the NCACA data set, computations were carried out on a workstation. For the large data sets, the application was executed in parallel on a 16-node cluster of dual processor Apple Xserve servers (32 processors total). Regenerating the matrices with different parameters took about a 2 minutes for the full NCI data set on the cluster.

## Visualization

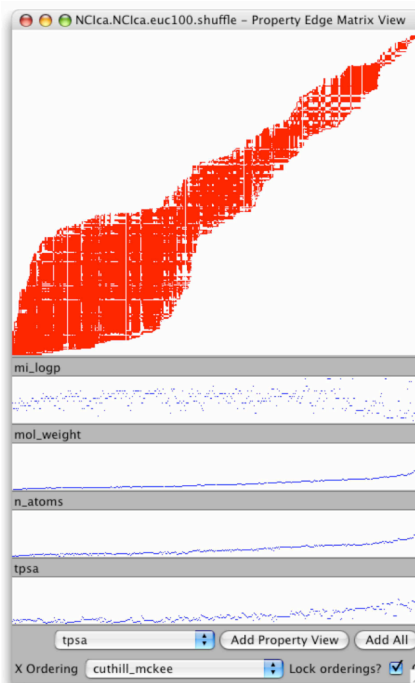
The similarity matrices were visualized with a custom matrix viewer (Figure 2) developed at the Open Systems Lab at IU (OSL) over the course of a few projects. The viewer is built in Python using the OpenGL bindings for graphics and the Boost Graph Library (BGL) Python bindings (also developed at the OSL) for high-performance data structures and algorithms. The ordering algorithms are all from the BGL, but some were modified during the course of this project to reflect new requirements.

The visualization tool itself underwent a major overhaul as a result of this project. In past projects, the matrices were generated and the underlying features that were used to compute the similarities, when applicable, were not available (some data sets are pure graphs and have no computable similarity matrix). When the first similarity matrix was generated, the first question we had was if the underlying features contribute in predictable manners under different orderings. However, the tool had no method of displaying anything other than the similarity matrix. To address this, we added a 'property plot' visualization that displays the values of any property for a node in the same ordering used by the matrix. The property plots are displayed below the similarity matrix so the x-axes of the matrix and plots align. The property plots are linked to the pan and zoom features of the matrix plot, so points from all plots align vertically based on the vertices. This additional visualization made it possible to gain insight into the relationship between the similarity measure, property, and orderings and also understand better why vertices were ordered close together. In other words, it greatly improved the interpretability of the visualization.

In addition to the property plots, we also upgraded the tool to load node and edge properties from the database. Edge properties can be used to color the dots in the plot area. For large graphs, it is not always a good idea to load all available properties. By adding flexible loading methods, we were able to load what was needed without impacting the performance of the application.

## Results

The chemistry data sets performed well with the different ordering algorithms. Figure 3 shows the results for the NCA data sets with 3 different orderings. Because the threshold levels were set low for the larger data sets, the graphs contained many components. The large number of small components makes it difficult to generate an overall good ordering, but larger components are ordered well internally.



**Figure 2** The Matrix View Tool. The property plots show that molecular weight, tpsa and the number of atoms contributed to most the ordering.

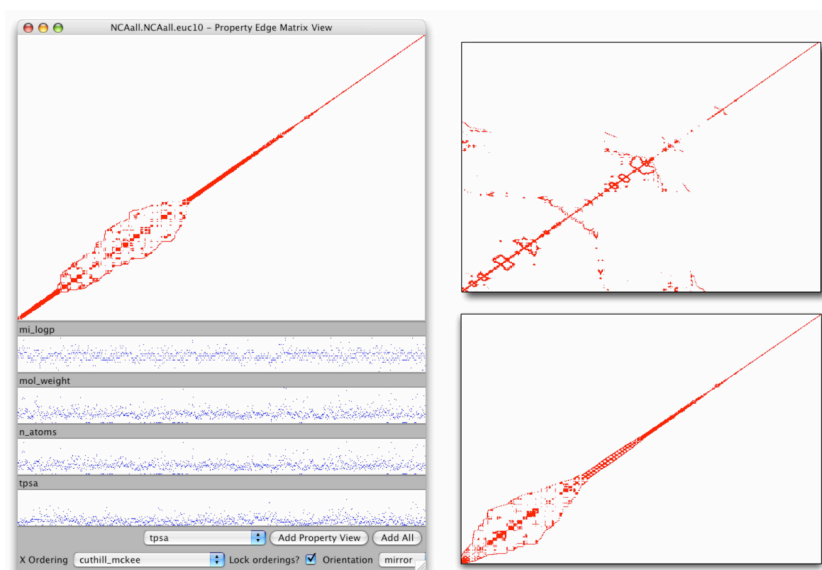
As suggested in the two previous sections, this projected illuminated many issues in the analytical pipeline. The similarity computation challenges are well known, but the visualization issues were specific to our application and not apparent in previous studies.

The two main challenges in generating similarity matrices are the computation time and the amount of space required to store the results. While a parallel implementation addresses the time problem for data sets with (low) millions of nodes, the storage problem does not

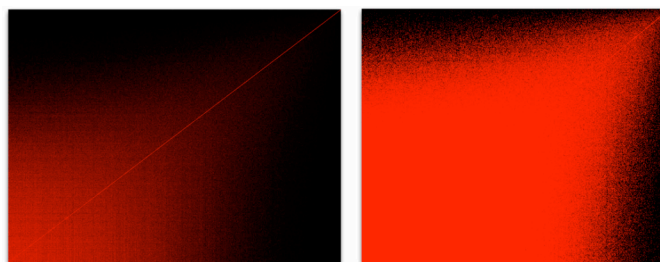
have an easy solution and leads to new set of procedural issues. For the large data set, the maximum amount of edges possible is on the order of 62.5 billion. At 8 bytes/measure, this is 500 GB of storage. Because the matrix can be thresholded, this can be significantly reduced. But, for each data set and measure, selecting an appropriate value is difficult. The resulting matrix should contain enough information to be useful for analysis but not so much as to limit the interactivity of the process (It has been stated anecdotally that many interesting avenues of analysis have been abandoned due to lack of interactivity with high performance computing resources. An important goal of my research it to maintain interactivity whenever possible and explore alternate methods when current methods become non-interactive.). In this case, we simply preformed many comparisons and looked for graphs that contained fewer than 3 million edges. A more rigorous technique is highly desirable.

In the visualizations, one issue that was identified by this study was the effect of anti-aliasing on large similarity matrices. In previous studies, we were aware that the resolution of the monitor affected the interpretability of the visual matrix - when fully zoomed out, sparse regions can appear dense due to the fact that single pixels may contain hundreds or thousands of data points, but only one actual data point may be active, coloring the entire pixel. The common solution to this problem is to enable anti-aliasing in OpenGL. However, the anti-aliasing on the stock video cards did not improve the image much (Figure 4). Recently, we acquired a new, mid-level video card and ran the visualization software on it with anti-aliasing turned on. The results were impressive. Regions that were solid blocks of colors on older cards displayed subtle variations in coloring that made it possible to discern sparse and dense regions.

As we started to visualize the larger data sets, we also found some previously unidentified issues with many of the ordering algorithms, particularly for large data sets. The matrices for these were fairly



**Figure 3** Three orderings of the NCA data set (42k nodes, 3.2M edges, 279 components). The full view shows Cuthill-McKee ordering, the upper right view is a depth-first search (DFS) ordering, and the lower right (BFS) is a breadth-first search. In DFS, similar elements are connected by wispy trails whereas BFS and Cuthill-McKee both clump relations along the diagonal.



**Figure 4** The effects of good anti-aliasing on the matrix visualization. The left image was rendered on a fairly recent nVidia 6800 graphics card and the right image on an older ATI Radeon 9600. Anti-aliasing was turned on for both images, but the 6800 produced a more interpretable image.

sparse and had between 30k and 150k components. While interpretability issues had been identified in the past for matrices with this property, the size of these data sets exposed some performance problems, particularly when bootstrapping the algorithms. For instance, Cuthill-McKee and King ordering both try to select a best starting node for each component. In the current implementation, this used a complex sequence of breadth first searches that significantly increased the runtime of the algorithms to the point where they were no longer useful for exploratory analysis. We implemented stopgap solutions as necessary, but the problems remain open.

## Conclusions

Developers and computer science researchers rarely step back and see how their work fits into the bigger process. This project gave us the opportunity to subject ourselves to the tools and techniques we are studying and gave us valuable insight into the how the matrix visualizations integrate into the analytical pipeline.

The first two components of the pipeline, data gathering and cleaning, presented no major hurdles, in large part because of the availability of a well-curated data set and good chemical property calculation tools. The third stage, generating the similarity matrix, was the main bottleneck, both computationally and procedurally. More resources address the computation problems, but storage remains an issue. Because of this, extra steps need to be added to the pipeline ensure the data set is properly filtered. The visualization and ordering tools gained the most immediate benefits from this project. Both were enhanced to increase their interpretability and scalability. The ordering algorithms generated reasonable orderings of data, but were most useful with additional information provided by the property plots.

In the context of the larger research project, this project has identified some possible avenues for future work. From a chemistry perspective, it would be good to extend the properties to include structural fingerprints and binary similarity measures (e.g., Tanimoto) and include a chemist in the process to help evaluate the results. This would also provide an opportunity to add data specific views to the interface to view chemical structures. Similarity matrix generation can be improved by studying ways to dynamically threshold the matrix as it is generated and join the resulting components. The process of running the comparison on parallel resources could also be streamlined via a carefully designed software system. The comparisons could also be ported to run on vector processors (AltiVec on Mac, SSE2 on Intel) for increased performance.

## Acknowledgements

While I (Chris Mueller) performed all the work described in this paper, Doug Gregor and Ben Martin both provided support for the Boost Graph Library and the Visualization Application. Doug Gregor is responsible for the maintaining the BGL and implemented the Python bindings for it. Ben Martin is developing components for the visualization system and helped with the architectural changes required to integrate the property plots.

## References

- [1] Mueller, Chris. *Sparse Matrix Reordering Algorithms for Cluster Identification*. For I532: Machine Learning in Bioinformatics, December 17, 2004.  
<http://www.osl.iu.edu/~chemuell/projects/bioinf/sparse-matrix-clustering-chris-mueller.pdf>
- [2] NCI Open Database Compounds, August 2000 2D file.  
<http://cactus.nci.nih.gov/ncidb2/download.html>
- [3] NCI AIDS Antiviral Screen, May 2004 release. [http://dtp.nci.nih.gov/docs/aids/aids\\_data.html](http://dtp.nci.nih.gov/docs/aids/aids_data.html)
- [4] MolInspiration Toolkit, <http://www.molinspiration.com>