Reordering for Matrix Visualization with Reorder.js

Nathan van Beusekom, Jean-Daniel Fekete

Reordering for Matrix Visualization with Reorder.js

Nathan van Beusekom*
TU Eindhoven, The Netherlands

Jean-Daniel Fekete†
Université Paris-Saclay & Inria, France

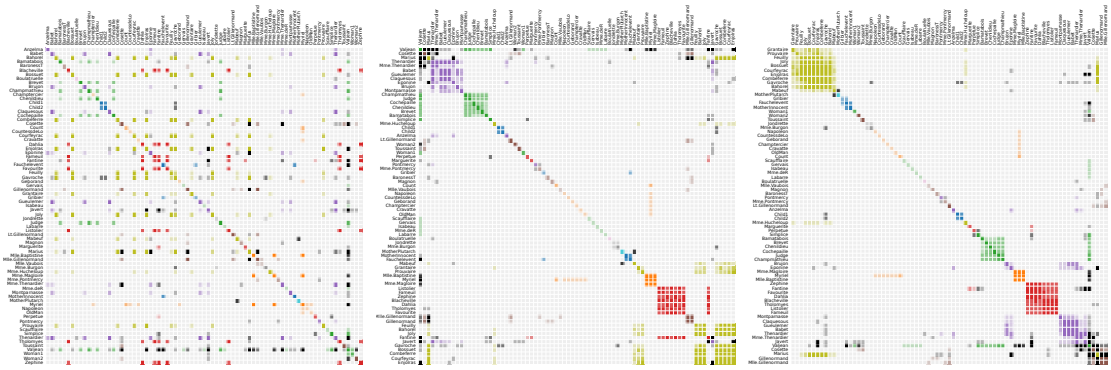

Figure 1: Several matrix visualizations of the Les Misérables co-occurrence dataset with different vertex orderings. From left to right: ordered alphabetically, ordered with Leaf Order using Euclidean distance, ordered with Leaf Order using a Moran's I-based distance.

Abstract

Effective matrix visualizations rely heavily on the order of the vertices. The patterns that appear thanks to such an ordering provide information about the structure of the graph. However, implementing reordering algorithms is not trivial. `Reorder.js` is a JavaScript library that provides several algorithms for matrix reordering. This library has now been updated with the state-of-the-art measure *Moran's I*, algorithms based on Moran's I, and *simultaneous reordering* for multiple matrices. The poster shows the results of the new algorithms applied to several visualizations. Finally, our examples aim to remind practitioners of the importance of reordering in visualization, e.g., for Parallel Coordinate Plots and tables.

Index Terms: Matrix Reordering—Simultaneous Orderings—Open Source Library—keyword

1 Introduction

Graphs are commonly used to represent network-like data, such as social networks, transportation networks, and brain connections. In order to explore this data and interact with it, the graphs must be visualized. There are two common ways to visualize a graph: node-link diagrams, where vertices (or entities) in the graph are drawn as nodes and the edges (or relations) between them are drawn as links between nodes, and matrix visualizations, where the adjacency matrix of the graph is visualized as a table, such that each row and column represents a vertex and a cell is filled if there exists an edge between the vertices in the respective row and column (see Fig. 1). For matrix visualizations, the order in which the vertices appear as rows and columns in the table is crucial. A good ordering of the vertices makes patterns appear in the visualization, which reveals information about the structure of the graph. Behrish et al. [4] identified patterns that correspond to different types of graph structures, and anti-patterns that reveal no information about the graph.

*e-mail: n.a.c.v.beusekom@tue.nl
†e-mail: Jean-Daniel.Fekete@inria.fr

In order to facilitate matrix reordering for users, the JavaScript library `Reorder.js` was released, which includes several reordering algorithms [6]. However, it only facilitated reordering methods for a single matrix. When multiple matrices are visualized, it is important to use the same ordering for all matrices, so that they can be compared to each other. For reordering multiple matrices, Bach et al. [2] compute the union of the matrices, then apply conventional reordering methods. However, this has the limitation that information about individual matrices is lost. Van Beusekom et al. [9] addressed this by doing the aggregation later on in the methods, retaining some information about individual matrices. Now, both methods have been implemented in `Reorder.js`, such that multiple matrices can be reordered to have the same, effective ordering. This implementation can directly be used by visualizations of multiple matrices such as [2] or methods that compare matrices, such as [1]. In the poster, we present examples of how to use the methods, and when to use matrix reordering in general.

2 Quality Measures and Measure-Driven Reordering

Accurately measuring the quality of a matrix visualization is essential, as a well-reordered matrix is more effective in showing the structure of the graph. The presence of patterns and the absence of anti-patterns are indicative of a matrix visualization of good quality. However, the presence of patterns is not a straightforward qualitative measure. One way to measure the quality of a matrix visualization is to measure whether vertices which are connected in the graph are also close to each other in the ordering. In the matrix visualization, this means that filled cells that are close to the diagonal are preferred over filled cells far from the diagonal. Such measures include Bandwidth, Profile, and Linear Arrangement. However, these measures do not reflect all the useful patterns. For example, an off-diagonal block pattern is sometimes desirable, but it has filled cells far from the diagonal. Instead, regarding more general desirable patterns, filled cells should be placed adjacent to each other and empty cells adjacent to each other: having filled cells together and empty cells together is desirable for matrix visualizations to visualize meaningful structures in the graph. This goal is similar to that of spatial autocorrelation. Spatial autocorrelation is a measure of similarity (or correlation) between nearby elements. In matrix visualizations we want cells near each other to have the same values. An established

spatial auto-correlation measure is Moran's I [8]. This measure can also be applied to matrix visualizations, to measure their quality [9] (see Fig. 2). A high Moran's I indicates that cells of similar values are grouped together. This implies that patterns are being formed and that structure in the graph is being visualized. A low Moran's I means that cells of similar values are not grouped. This corresponds with the noise and anti-bandwidth anti-pattern. The library has been extended to include Moran's I measure.

Furthermore, it is possible to define a distance measure between vertices based on Moran's I. This distance measure compares the two rows of the vertices and sees how many similarities there are between the cells in the same position, using the same weight values as the Moran's I measure. Minimizing such a distance would maximize Moran's I. This distance can be used by several existing algorithms, similarly to Euclidean distance, and has also been added to `Reorder.js`. One example is shown in Fig. 1, where this distance function has been used in the Optimal Leaf Order algorithm [3]. This idea of basing a distance function on a measure is very effective for maximizing said measure.

Since we have distances between the vertices and we want to minimize the distances between subsequent vertices in an ordering, the matrix reordering problem can be considered as a "Traveling Salesman" (TSP) algorithm: we want to find an ordering of the vertices, such that traversing the vertices in that order minimizes the overall distances. Hence, we can apply commonly used TSP heuristics to matrix reordering, such as NN-2OPT [5], as already indicated in the appendix of [9]. This algorithm has also been added to `Reorder.js`.

3 Simultaneous Matrix Reordering

As stated in the introduction, it is important to use the same ordering when comparing several matrices sharing the same vertices, so that they can be compared to each other. Again, we want to reorder the matrices to reveal information about the structures of the graphs, yet now we want to find an ordering which reveals structure in all the matrices simultaneously.

There are generally two methods for computing such an ordering (see Fig. 3). One is computing the union of the matrices and applying state-of-the-art reordering methods to the union, this was done by Bach et al. [2]. `Reorder.js` has been extended with a simple method for computing the union of matrices, which can then be used as an input for one of the existing algorithms. A more sophisticated method is to modify a state-of-the-art reordering algorithm to use information from all individual matrices. Van Beusekom et al. [9] modified the Optimal Leaf Order method [3] and the Barycenter method [7]. These modified (collection-aware) methods have been implemented in `Reorder.js`. For the Collection-aware Optimal Leaf Order method, only the way the distance matrix is computed has been changed. Hence, this simultaneous reordering idea can be applied to any reordering method that uses as input a distance matrix. For example, one could use the newly implemented NN-2OPT algorithm in a collection-aware manner, by simply providing it with the collection-aware distance matrix.

4 On the importance of Matrix Reordering

Often, the data that is being visualized comes with an inherent ordering. This could be the order of entry or the alphabetical order. It is easy to simply use this order for the matrix visualization, even though this might not clearly visualize the graph structure. For many visual practitioners, matrix reordering does not come to mind when using visualizations that could benefit from it. This often reduces the appeal of matrix visualizations, as they might come cluttered and hard to interpret. However, matrices and tables have the potential to show the data concisely, when reordered properly. In the poster, we aim to show examples where reordering improves

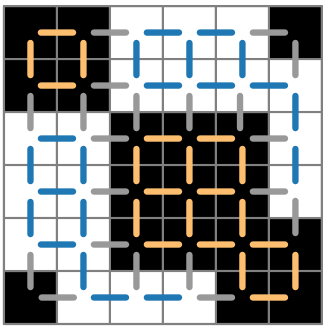

Figure 2: Figure from [9] to explain Moran's I on matrices: the yellow and blue adjacencies add to the Moran's I value, while the grey adjacencies lower the Moran's I.

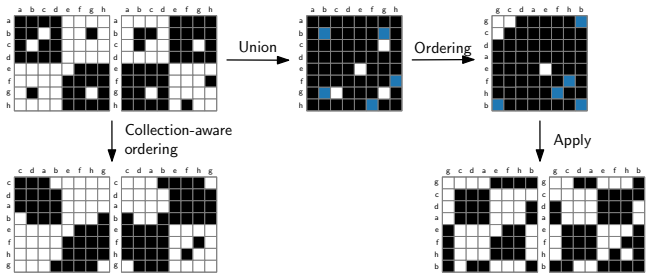

Figure 3: Figure from [9] to explain the benefit of collection-aware ordering. The information lost in the union produces a worse order for the matrices.

the interpretability of the visualization, such that practitioners will more easily recognize when matrix reordering is beneficial.

Reordering can be applied to optimize Parallel Coordinate Plots, tables, and a large number of other visualizations where consistency across rows or columns highlights structures.

References

[1] B. Alper, B. Bach, N. Henry Riche, T. Isenberg, and J.-D. Fekete. Weighted graph comparison techniques for brain connectivity analysis. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '13, p. 483–492. Association for Computing Machinery, New York, NY, USA, 2013. doi: 10.1145/2470654.2470724

[2] B. Bach, N. Henry-Riche, T. Dwyer, T. Madhayastha, J.-D. Fekete, and T. Grabowski. Small MultiPiles: Piling Time to Explore Temporal Patterns in Dynamic Networks. *Computer Graphics Forum*, 34(3):31–40, 2015. doi: 10.1111/cgf.12615

[3] Z. Bar-Joseph, D. K. Gifford, and T. S. Jaakkola. Fast optimal leaf ordering for hierarchical clustering. *Bioinformatics*, 17(suppl 1):S22–S29, 2001.

[4] M. Behrisch, B. Bach, N. Henry Riche, T. Schreck, and J.-D. Fekete. Matrix Reordering Methods for Table and Network Visualization. *Computer Graphics Forum*, 35(3):693–716, 2016. doi: 10.1111/cgf.12935

[5] G. A. Croes. A method for solving traveling-salesman problems. *Operations Research*, 6(6):791–812, 1958.

[6] J.-D. Fekete. Reorder.js: A JavaScript Library to Reorder Tables and Networks. In *Abstr. 2015 IEEE VIS posters*, 2015. Available at `https://hal.inria.fr/hal-01214274/file/reorder.pdf`.

[7] E. Mäkinen and H. Siirtola. The barycenter heuristic and the reorderable matrix. *Informatica (Slovenia)*, 29(3):357–364, 2005.

[8] P. A. P. Moran. Notes on continuous stochastic phenomena. *Biometrika*, 37(1/2):17–23, 1950.

[9] N. van Beusekom, W. Meulemans, and B. Speckmann. Simultaneous matrix orderings for graph collections. *IEEE Transactions on Visualization and Computer Graphics*, 28(1):1–10, 2022. doi: 10.1109/TVCG.2021.3114773