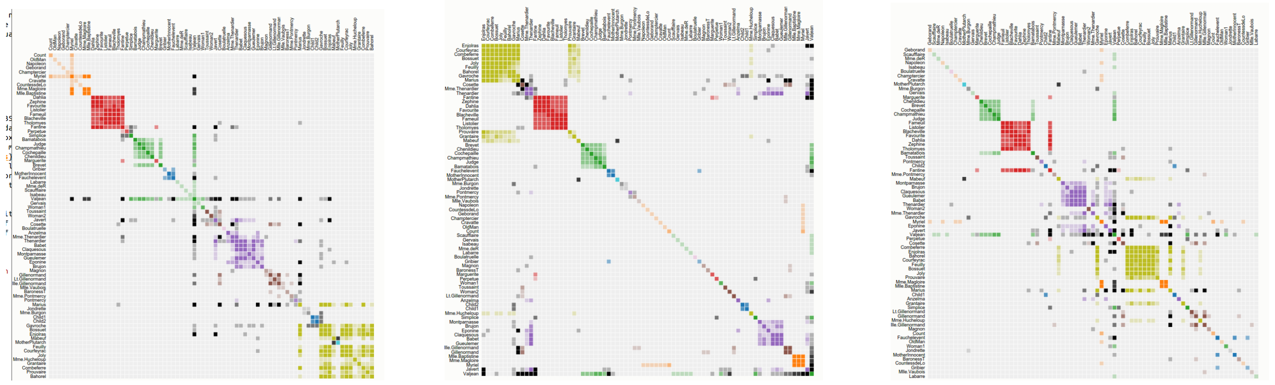
Reorder.js: A JavaScript Library to Reorder Tables and Networks

Jean-Daniel Fekete

Reorder.js: A JavaScript Library to Reorder Tables and Networks

Jean-Daniel Fekete*
INRIA



(a) Barycenter heuristic [9]. (b) Optimal Leaf Ordering [2]. (c) Spectral Ordering.

Figure 1: Reorderings of the character-occurrences network of Victor Hugo's "Les Misérables".

Abstract

Reordering, also known as *seriation*, and *linear ordering*, consists in assigning an order to rows/columns of a table or to vertices of a network to reveal structures. When visualizing a table as a heatmap or when visualizing a network as an adjacency matrix, finding a good order is essential to revealing high-level patterns in the data. The literature on reordering algorithms is dense but few implementations are easily available. `Reorder.js` is a novel JavaScript library, provided in source code with a BSD license, which provides several of these algorithms and variants to facilitate the use, adoption, and experimentation of these algorithms.

`Reorder.js` is used in several existing Web-based applications such as *Bertifier* [11], *NodeTrix* [7], and *Multipiles* [1]. It can be used with e.g. `D3.js` for adjacency matrices, Heatmaps, and Parallel-coordinates.

Index Terms: G.2.1 [Combinatorics]: Permutations and combinations—; D.2.13 [Reusable Software]: Reusable libraries—; G.2.2 [Graph Theory]: Graph labeling—.

1 Introduction

Reordering methods are essential in visualization, in particular for tables visualized as *Heatmaps* [3, 13], for networks visualized as adjacency matrices, but also for ordering axes in parallel coordinates plots and its variants. Although there has been surveys on the reordering techniques, also known as *seriation*, and *linear ordering* [4, 13, 8], the literature remains dispersed, actual algorithm implementations are hard to find. As a consequence, very few visualization systems use any reordering at all.

To address this issue, we have implemented the library `Reorder.js` that provides an extensive list of algorithms and utility functions to streamline the use of the algorithms, but also to help experiment with existing ones, adapt them to particular cases, or design new ones and compare them with baseline implementations.

2 Related Work

Finding useful orders for table rows/columns and network vertices is not a new problem. Liiv, and Wilkinson and Friendly [8, 13] have published well-documented historical overviews of methods for reordering tables. However, they are restricted to tabular data. Ordering vertices of a network is also an important problem in the graph community, with a survey by Dìaz et al. [4]. It turns out a table can be modeled as a bipartite network: creating vertices for all the rows and for all the columns, and connecting them with edges that are weighted by the contents of the cells. Assuming the table is numerical, the graph is simply weighted. Edges can be removed when related cells are empty (e.g. contain zero). On the other direction, a graph can be turned into a table by using its adjacency matrix. When the graph is weighted, the adjacency matrix cells contain the weight values. If the graph is undirected, the adjacency matrix is a symmetrical table. For directed graphs, the adjacency matrix is a general table.

Therefore, algorithms used to order a table can be used to order the vertices of a network and vice versa, although just using the bipartite graph and ordering it will usually not work since there should be an order for the rows and another for the columns. A more effective way of using graph algorithms for reordering tables consists in computing a distance-matrix (or similarity matrix) for all the rows, and another one for all the columns. Each of these similarity matrices are symmetric, positive-definite, and can be seen as the adjacency matrices of weighted graphs: graph ordering algorithms can then be applied to them directly.

Although the references on reordering algorithms are numerous and almost centralized, there are very few library that implement them all. The extensive R package "seriation" [6] provides 30 algorithms for tables only. Various libraries in varied languages implement some ordering algorithms but without the goal of being complete: most graph libraries provide some ordering algorithms; the Boost Graph Library (BGL) [12] in C++ provides 4 algorithms that

*e-mail: jean-Daniel.Fekete@inria.fr

have been used to visualize graphs [10]. While the source code of many of the reordering algorithms is sometimes available, it comes in a wide variety of languages with library dependencies, hampering their reuse in visualization.

This is why we have designed and implemented our `Reorder.js` library: to facilitate the use, experimentation, and extension of reordering algorithms in JavaScript.

3 Concepts and Implementation

`Reorder.js` provides a few concepts: *table*, *graph*, *order* or *permutation*, and multiple utility functions to convert from graph to table, compute a *distance-matrix*, and obviously *reorder*. Additionally, `Reorder.js` provides multiple ways to measure ordering qualities; this is needed because, unfortunately, there is not one universally accepted measure of order quality.

Table is a JavaScript 2D array, made of a list (the columns) containing lists of identical sizes (the rows).

Graph is a JavaScript object, with an interface mostly borrowed from D3 graphs but with a richer interface to handle undirected as well as directed graphs.

Permutation or *order* is a list containing indexes or columns, rows, or vertices. It can be a proper permutation when all the indices are contained in the list, but it can also be an *ordered selection* when it contains only a subset of the indices. For example, computing the connected components of a graph returns a list of *ordered selections*, one for each component.

Distance-matrix is a table containing distance measures for all the elements of a table (distance between rows or columns) or vertices in a graph. Most table-based reordering algorithms first compute the row distance-matrix and column distance-matrix, reorder them independently. Note that a distance-matrix can also be computed from the vertices of a graph.

Quality measure Although there is no universal quality measure capable of assessing the optimality of an order, there are many measures that can be used to compare two orderings for a particular feature. For example, the Traveling Salesman algorithm tries to find an order that minimizes the sum of distances of consecutive vertices in a graph; whereas it is not clear whether this measure is the best over all the possible measures, it can still be used to compare two or more orders.

4 Algorithms

Currently, 8 basic algorithms are available, with multiple variants, the most interesting being:

Barycenter that implements the "Barycenter heuristics" [9], but using optimizations from [5], i.e. the median instead of the barycenter, and a post-optimization phase that swaps adjacent vertices when it further improves the ordering (Fig. 1a).

PCA and Correspondence Analysis are two methods that are fast and effective for some particular tables. The speed comes from an iterative computation of the main eigenvector of a square matrix using a power-iteration.

Cuthill McKee and its inverse are popular methods to reduce the bandwidth of a matrix; actually, bandwidth optimization alone does not produce good results, but it is fast.

Optimal Leaf Ordering, a very effective algorithm to order a matrix based on a hierarchical clustering [2]. Unfortunately, its complexity is high, but using the classical "memoization" optimization, it usually becomes quadratic. This algorithm has been used for the Bertifier system [11] because it consistently generates excellent results visually (Fig. 1b).

Spectral Order uses the Laplacian of the graph or distance-matrix and computes an order based on its smallest eigenvector with a non-zero eigenvalue (the Fiedler vector). This order is relatively fast to compute since it also uses a power-iteration to find a good approximation (Fig. 1c).

All these algorithms and utility functions are programmed with readability and performance in mind. JavaScript is certainly not the best language for optimal code speed, but it is simple and readable, and `Reorder.js` parts can easily be adapted or translated in other languages.

For example, the Bertifier system [11] relies on the optimal leaf ordering algorithm, but with multiple adaptations to allow finer user-control, such as blocking consecutive rows/columns while still allowing the reordering of rows/columns around it. This higher-level API is provided in `Reorder.js`, and boils-down to manipulating the distance-matrix prior to calling the main reordering algorithm, with some pre- and post- processing to take care of special cases such as identical rows/columns (distance 0) that are not well handled by the hierarchical clustering algorithm.

5 Conclusion

`Reorder.js` is provided in source form at `https://github.com/jdfekete/reorder.js` with a liberal BSD license. It can be used "as is" for matrix visualization, or adapted to particular situations since it does not provide only black box algorithms but lower-level primitives that can be adapted to multiple uses. Additionally, the implementation of `Reorder.js` is meant to be simple yet efficient; it can be adapted to other languages and running environments than a web browser. Many of the reordering algorithms are non-trivial and can be further improved using tricks that are not always described in research papers.

`Reorder.js` will continue to evolve with novel algorithms and quality measures that will hopefully facilitate the spread of matrix-based visualizations. Feel free to contribute to it if you have interesting reordering methods.

References

[1] B. Bach, N. Henry-Riche, T. Dwyer, T. Madhyastha, 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.

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

[3] J. Bertin. *Sémiologie graphique : Les diagrammes-Les réseaux-Les cartes*. Mouton/Gauthier-Villars, 1967.

[4] J. Díaz, J. Petit, and M. Serna. A survey of graph layout problems. *ACM Comput. Surv.*, 34(3):313–356, Sept. 2002.

[5] E. Gansner, E. Koutsofios, S. North, and K.-P. Vo. A technique for drawing directed graphs. *Software Engineering, IEEE Transactions on*, 19(3):214–230, Mar 1993.

[6] M. Hahsler and K. Hornik. Dissimilarity plots: A visual exploration tool for partitional clustering. *Journal of Computational and Graphical Statistics*, 20(2):335–354, 2011.

[7] N. Henry, J.-D. Fekete, and M. J. McGuffin. Nodetrix: a hybrid visualization of social networks. *Visualization and Computer Graphics, IEEE Transactions on*, 13(6):1302–1309, Nov 2007.

[8] I. Liiv. Seriation and matrix reordering methods: An historical overview. *Statistical analysis and data mining*, 3(2):70–91, 2010.

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

[10] C. Mueller, B. Martin, and A. Lumsdaine. A comparison of vertex ordering algorithms for large graph visualization. In *Visualization, 2007. APVIS'07. 2007 6th International Asia-Pacific Symposium on*, pages 141–148. IEEE, 2007.

[11] C. Perin, P. Dragicevic, and J.-D. Fekete. Revisiting Bertin matrices: New Interactions for Crafting Tabular Visualizations. *IEEE Transactions on Visualization and Computer Graphics*, Nov. 2014.

[12] J. G. Siek, L.-Q. Lee, and A. Lumsdaine. *Boost Graph Library, The: User Guide and Reference Manual*. Addison-Wesley Professional, 2001.

[13] L. Wilkinson and M. Friendly. The history of the cluster heat map. *The American Statistician*, 63(2), 2009.