EuroVis 2016
R. Maciejewski, T. Ropinski, and A. Vilanova
(Guest Editors)

Volume 35 (2016), Number 3
STAR – State of The Art Report

Matrix Reordering Methods for Table and Network Visualization

Michael Behrisch[1], Benjamin Bach[2], Nathalie Henry Riche[3], Tobias Schreck[4], Jean-Daniel Fekete[5]

[1]Universität Konstanz, Germany
[2]Microsoft Research-Inria Joint Centre, France
[3]Microsoft Research, USA
[4]University of Technology Graz, Austria
[5]Inria, France

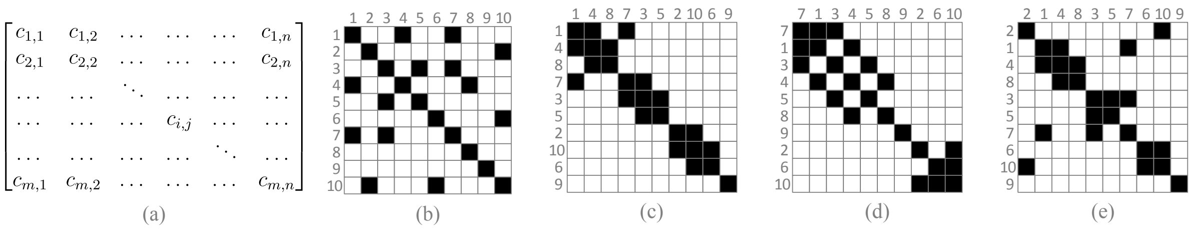


Figure 1: Visual matrix of numerical data (a) ordered randomly (b) and with three algorithms (c-e) revealing different patterns.

Abstract

This survey provides a description of algorithms to reorder visual matrices of tabular data and adjacency matrix of networks. The goal of this survey is to provide a comprehensive list of reordering algorithms published in different fields such as statistics, bioinformatics, or graph theory. While several of these algorithms are described in publications and others are available in software libraries and programs, there is little awareness of what is done across all fields. Our survey aims at describing these reordering algorithms in a unified manner to enable a wide audience to understand their differences and subtleties. We organize this corpus in a consistent manner, independently of the application or research field. We also provide practical guidance on how to select appropriate algorithms depending on the structure and size of the matrix to reorder, and point to implementations when available.

Categories and Subject Descriptors (according to ACM CCS): Visualization [Human-centered computing]: Visualization Techniques—

1. Introduction

A *Visual Matrix* or *Matrix Plot* is a visual representation of tabular data also used to depict graphs and networks by encoding visually an adjacency matrix. This representation has been used for almost a century in many domains: biology, neurology, social science, supply management, transportation and artificial intelligence. While most network exploration tools today leverage node-link diagrams, several studies demonstrated that matrix representations are more effective for several low-level tasks [GFC04] (e.g., finding if two vertices are connected or estimating the network density) and graph comparison [ABHR*13], especially when networks become larger and denser.

Since matrices represent connections between vertices by a cell at the intersection of a row (source vertex) and a column (target vertex), they do not suffer from occlusion and link crossings as node-link diagrams do. While low-level tasks do not necessarily require to reorder rows or columns [Ber73, GFC04], higher-level tasks, such as identifying groups or highly-connected vertices, do require a reordering of rows, respectively columns, to reveal higher order patterns. Figure 1 illustrates how a *good* ordering can reveal blocks along the diagonal of the matrix, indicating densely connected groups in the network. The choice of the ordering highly influences which visual patterns are visible, thus proving a critical factor for exploration and analysis.

Historically, matrix reordering was done *manually*, with *"the eye of the artist"* [Ber73] and prove to be an extremely tedious task [Lii10]. The exploration, as well as evaluation of the results, was implicitly done in the process [PDF14]. Today, numerous *automatic* reordering algorithms are described in the literature [Lii10, MML07a]. Such automated matrix reordering approaches provide much faster results, the ability to order large matrices, and sometimes can even seek to reveal specific patterns in the data [WTC08]. In most cases, respective papers report on benchmarks of calculation complexity and run-time. Some of them including pictures of resulting matrices for individual algorithms or subsets. However the problem of assessing the quality of an ordering remains the same; similar to layout algorithms for node-link representations of graphs, knowing which patterns are artifacts of the algorithms and which patterns represent structure in the data is crucial [MBK97]. To the best of our knowledge, there is no coherent survey of reordering algorithms for matrices available today. Moreover, there exist no visual comparison or methodology that allows for evaluating the patterns and artifacts each reordering algorithm produces.

In this survey, we describe the main algorithms published for the purpose of finding a proper order for the rows and columns composing a visual matrix. For each of the six groups of algorithms we could establish, we illustrate the underlying ordering principles and discuss pictures of matrices for a set of different graphs. The problem of finding a *good* ordering for the vertices of an adjacency matrix is known with multiple names: *seriation*, *reordering*, *linear layout*, *layout*, *linear ordering*, *linear arrangement*, *numbering*, *labeling*, and *ordering*. In this document, we simply call it *reordering*, but these terms can be used interchangeably when the context is not ambiguous since they are mostly an artifact of their history. However, the multiplicity of terms testifies of the wide variety of domains that have studied the problem, and this survey tries to summarize and organize this wide diversity.

While we focus entirely to network data (symmetric matrices), we found that several ordering methods can similarly be applied to table data. However, table data can produce visual patterns with different interpretations and meaning; thus their description and visual analysis remains future work.

1.1. Related Surveys

There has been several prior surveys describing matrix reordering approaches. Liiv's overview [Lii10] summarizes the most common reordering methods, highlighting the historical developments in various affected domains. Similarly, Wilkinson and Friendly present an abstract of the history of cluster heatmaps [WF09]. Also, Wilkinson describes in "Grammar of Graphics" [Wil05] the process of creating a range of visual base patterns (cf. Section 2.3) and compares four reordering approaches with regards to their retrieval performance. Influential for our evaluation approach is the work of Mueller et al., which focuses on interpreting and describing structures in matrix plots [Mue04, MML07b, MML07a]. Although related, none of these works provide empirical evidence of the individual reordering performances with respect to (i) specific user tasks and (ii) the data set characteristics.

1.2. Methodology

In contrast to the historically inspired matrix reordering surveys we are developing our categorization from an algorithm-agnostic point of view. Specifically, we are contributing considerations about the expected visual patterns that can result from the algorithm's inherent design and the problem space from which a solution is derived.

For our empirical comparison, we chose graphs from three different categories: (i) a set of graphs commonly used to benchmark matrix reordering algorithms [Pet03], (ii) a set of systematically generated graphs in order to test for specific graph characteristics, such as density or number of clusters, and (iii) a set of real world graphs (social networks, brain connectivity networks). For any of our 150 graphs, we generated an ordered matrix plot which we used as input for our performance measures. This let us with a total of 4348 matrix plots as basis for our comparison. Graphs and matrices shown in this survey represent a purposeful selection. The complete browsable collection can be found online at http://matrixreordering.dbvis.de.

1.3. Outline

In the reminder of this article, we start with a background on matrix reordering methods and challenges. We then provide an extensive survey on existing algorithms. We review and divide 35 exemplary matrix reordering *implementations* into six major categories, as discussed in Section 3, such as Robinsonian-, Spectral-, Dimension Reduction-, Heuristic-, Graph-Theoretic-, and Bi-Clustering. Then, we report on our comparative analysis and discuss the results. We conclude with a list of guidelines, which algorithm to choose for specific data sets and characteristics, as well as suggestions for future studies.

2. Background

This section introduces definitions and concepts that we rely upon to describe matrix reordering algorithms. The third subsection also presents 4 visual patterns and 2 anti-patterns one can identify in visual matrices. We use these patterns to illustrate and compare the algorithms.

2.1. Definitions

A *graph* G is a couple (V,E) where V is a set of vertices, and E is a set of edges where:

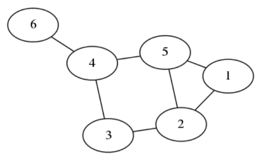
$$\begin{aligned} V &= \{v_0, \cdots, v_n\}, \\ E &= \{e_0, \cdots, e_m\}, e \in V^2 \end{aligned} \tag{1}$$

A *directed graph* is a graph where the two vertices associated with an edge are considered ordered. An *undirected graph* is a graph where the vertices associated with an edge are not ordered.

We use the term *network* to describe the graph topology as well as attributes associated with vertices (e.g., labels), and attributes associated with edges (e.g., weights). Most networks used in this survey have names associated with vertices, and positive weights associated with edges. A weighted graph G_W adds a weight function $w(e)$ to G so that:

$$w(e_i) = w_i, \text{with } w_i \in \mathbb{R}^+ \tag{2}$$

An *ordering* or *order* is a bijection $\varphi(v) \to i$ from $v \in V$ to $i \in N = \{1, \cdots, n\}$ that associates a unique index to each vertex. We use φ^* to describe one specific ordering from the set of all possible orderings. A network usually comes with an arbitrary ordering reminiscent of its construction or storage. We call that order the *initial order* noted $\varphi_0(v)$ to distinguish it from a computed order. A



$$\begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Figure 2: A Simple Labeled Graph and its Adjacency Matrix

transformation from one ordering to another is called a *permutation* π. Formally, a permutation is a bijection $\pi(x) \rightarrow y$ such that:

$$\pi(x_i) = y_i, (x,y) \in N^2 \text{ where } y_i = y_j \Rightarrow i = j \tag{3}$$

It is usually implemented as a vector containing n distinct indices in N. We call S the set of the $n!$ possible permutations for n vertices. A permutation can also be represented as a $n \times n$ matrix P with all entries are 0 except that in row i, the entry $\pi(i)$ equals 1.

An *adjacency matrix* of a graph G, as depicted in Figure 2, is a square matrix M where the cell $m_{i,j}$ represents the edge (or lack of) for the vertices v_i and v_j. It is equal to 1 if there is an edge $e = (v_i, v_j)$ and 0 otherwise. When the graph is weighted, $m_{i,j}$ represents the weight (for clarity purposes, we restricted weights to be strictly positive in Equation 2).

$$M = \begin{bmatrix} m_{1,1} & \cdots & m_{1,n} \\ \vdots & \ddots & \vdots \\ m_{n,1} & \cdots & m_{n,n} \end{bmatrix} \tag{4}$$

A *bipartite-graph* or *bi-graph* is a graph $G = (V_1, V_2, E)$ where the vertices are divided into two disjoint sets V_1, V_2, and each edge e connects a vertex in V_1 to a vertex in V_2:

$$V = V_1 \cup V_2, V_1 \cap V_2 = \emptyset \text{ such that } e \in E = V_1 \times V_2 \tag{5}$$

The adjacency matrix of a bi-graph is generally rectangular, composed of V_1 in rows and V_2 in columns to limit empty cells. We consider a general *data table*, such as data presented in spreadsheet form, a valued bi-graph. A classic example of a bi-graph is a document-author network with a single relation *is-author* connecting authors to documents. The adjacency matrix of such bi-graph includes authors in rows (respectively in columns) and documents in columns (respectively in rows), a value of 1 marking the authoring relationship, and a value of 0 otherwise.

2.2. Related Concepts

In addition to the previous definitions and notations, this survey frequently bridges graph concepts with linear algebra concepts. Since readers might not be familiar with these relationships, we summarize them here, introducing concepts often used in this article.

Adjacency matrices typically bridge graph theory and linear algebra, allowing the interpretation of a graph as a multidimensional (n-dimensional for n vertices) linear system, and vice-versa. We list below several interrelated properties of networks when considered as adjacency matrices.

- When encoded as an adjacency matrix, a vertex becomes an n-dimensional **vector of edges** (or edge weights). When the network is undirected, the matrix is **symmetric** and the vectors can be read horizontally or vertically. Otherwise, two vectors can be considered: the vector of *incoming edges*, and the vector of *outgoing edges*.
- Since vertices are vectors, a **distance** measure $d(x,y)$ can be computed between two vertices $(x,y) \in V^2$ (or a *similarity* or *dissimilarity* measure $s(x,y)$). For example, the Euclidean distance L_2: $d(x,y)$ between vertices x and y is:

$$L_2(x,y) = \sqrt{\sum_{k \in [1,n]} (x_k - y_k)^2} \tag{6}$$

- Several reordering algorithms use a **distance matrix** (or *similarity matrix*) as input, which is a square matrix D containing the pairwise distances between multiple vectors. From the $n \times n$ adjacency matrix of an undirected graph, one symmetric distance matrix can be computed of size $n \times n$. From a general n-rows $\times$ m-columns matrix, two distance matrices can be computed: one of size $n \times n$ for the rows (m-dimensional vectors we will call A), and one of size $m \times m$ for the columns (n-dimensional vectors we will call B). A distance matrix is always symmetric and positive (it is *positive-definite* mathematically speaking).
- A particularly important distance matrix is the **graph distance matrix**, which contains the length of the shortest path between every pair of vertices for an undirected graph. Note that a distance matrix or more generally a positive-definite matrix can also be interpreted as an adjacency matrix of a weighted undirected graph. Note also that any symmetric matrix can be interpreted as an adjacency matrix of a valued undirected graph (a graph where each edge has an associated value).
- From any undirected graph, or positive-definite matrix, many graph measures can be computed. These can serve as objective functions to minimize or as quality measures of reordering algorithms. We describe three key measures below: *bandwidth*, *profile*, and *linear arrangement*.

Let us call $\lambda(u,v)$ the length between two vertices in G, given a one-dimensional alignment of the vertices φ: $\lambda((u,v),\varphi,G) = |\varphi(u) - \varphi(v)|$.

Bandwidth BW is the maximum distance between two vertices given an order φ.

$$\mathrm{BW}(\varphi,G) = \max_{(u,v)\in E} \lambda((u,v),\varphi,G) \tag{7}$$

Intuitively and visually, when looking at the adjacency matrix of an undirected graph (a symmetric matrix), the bandwidth is the minimum width of a diagonal band that can enclose all the non-zero cells of the matrix. A small bandwidth means that all the non-zero cells are close to the diagonal. Therefore, a quality measure is MINBW, the *minimum bandwidth* of a graph $\mathrm{MINBW}(G) = \arg\min_{\varphi^*}(\mathrm{BW}(\varphi^*,G))$.

Profile PR is:

$$\mathrm{PR}(\varphi,G) = \sum_{u \in V} \left(\varphi(u) - \min_{v \in \Gamma(u)} \varphi(v) \right) \tag{8}$$

where $\Gamma(u) = \{u\} \cup \{v \in V : (u,v) \in E\}$. Intuitively and visually, the profile is the sum, for each column i of the matrix, which can be intuitively understood as the “raggedness”: the distance from the diagonal (with coordinates (i,i)) to the farthest-away non-zero cell for that column (with coordinates (i,j)). It is a more subtle measure than the bandwidth because it takes into account all the vertices and not only the vertex with the largest length. The *minimum profile* is $\textsc{MinPR}(G) = \arg\min_{\varphi^*}(\text{PR}(\varphi^*,G))$.

Linear arrangement LA is the sum of the distances between the vertices of the edges of a graph:

$$\text{LA}(\varphi,G) = \sum_{(u,v)\in E} \lambda((u,v),\varphi,G). \tag{9}$$

It is an even more subtle measure than the profile since it takes into account all the edges. The *minimum linear arrangement* is $\textsc{MinLA}(G) = \arg\min_{\lambda^*}(\text{LA}(\varphi^*,G))$. MinLA is an established performance measure for matrix reordering algorithms [Pet03], since it targets a block-diagonal form of the matrix (cf. Section 10).

2.3. Patterns and Anti-Patterns in Matrices

Formally, reordering an undirected network G consists in computing one permutation $\pi \in S$ that maximizes or minimizes an objective function $q(\pi,G)$, such that:

$$\arg\min_{\pi\in S} q(\pi,G) \tag{10}$$

For example, q can compute the sum of distances $d(x,y)$ between vertices according to the order π; Equation 10 would find $\pi \in S$ that minimizes this sum.

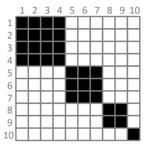
A brute-force approach to return an optimal solution for a symmetric matrix would require $n!$ computations, which renders impractical when n gets large. Even for $n = 10$, computing a global optimum would take 18.29 hours on a computer that evaluates 10^4 permutations per millisecond. For an 11×11 matrix, it would take 92.21 days, while a 12×12 matrix would take 36.38 years. Since a directed network requires two permutations, one π_r for the rows and one π_c for the columns, a brute-force approach would actually require $n! \times m!$ computations.

In addition, there is no consensus of an objective function q in the reordering literature. Therefore, we cannot understand the reordering problem as a pure optimization problem, and need to consider reordering algorithms according to the structures they reveal visually. Therefore, the goal of matrix reordering is to make *visual patterns* emerge, which represent data properties of the underlying network. To understand why this is possible, it is essential to realize that the order of matrix rows and columns can be freely changed without changing the data in the matrix.

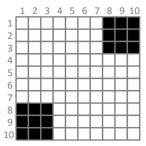
Bertin [Ber73, Ber81] developed several important ideas about the distinct levels of information contained in data displays and the user tasks–he uses the term questions–that refer to the respective levels [Ber73, p. 141]. He mentions (i) an elementary level, comprised of individual graphic elements and the task to understand their specificities; (ii) an intermediate level, for the comparisons among subsets of graphic elements and the discovery of homogeneous information parts; and (iii) an overall level, comprised of overall trends and relations. As a result, the analysis of visual patterns in matrices is important, since these patterns can be interpreted in the user’s analysis context, i.e., they relate to an analysis question and task at hand, and second, since they constitute the core information of a matrix plot, they allow the analyst to interpret and reason about their presence or salience.

Ghoniem et al. [GFC04] found that a range of overview tasks, such as estimating the amount of nodes or edges, or lower level tasks such as finding the most connected node, can be answered with matrices independent of the matrix ordering. On the other hand, higher level tasks about the specific topology of the network, require an appropriate reordering of rows and columns. The graph task taxonomy of Lee [LPP*06] states eight specific tasks which can be particularly well facilitated with matrices: retrieving (specific) nodes, edges; finding connected components and clusters; assessing distributions and cluster memberships, retrieving of (adjacency) relationships and general topology estimations.

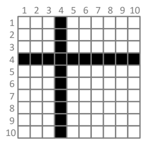
Extending Wilkinson’s [Wil05] and Mueller et al. [MML07b] work, we list below main *visual patterns* in visual matrices, along with their graph-theoretic interpretations and associated user tasks.



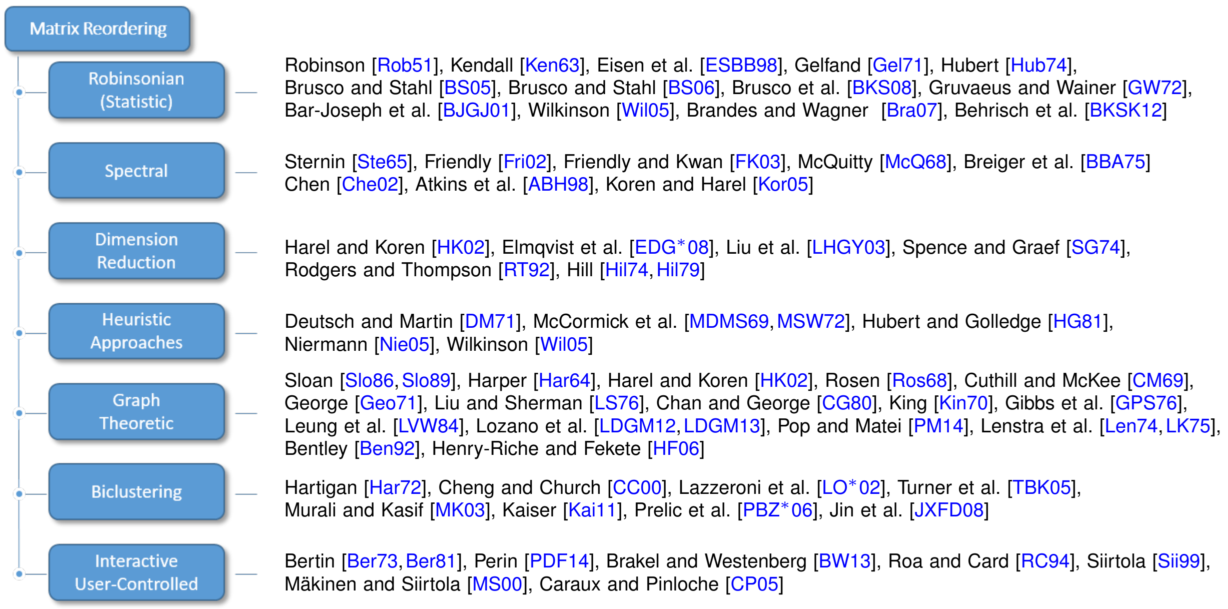
Block Pattern (*P1*): Coherent rectangular areas appear in ordered matrix plots whenever strongly connected components or cliques are present in the underlying topology. The figure shows 4 disconnected cliques (complete sub-graphs) containing 4, 3, 2, and 1 vertices. Mathematically, these matrices are called *block-diagonal matrices*.
Block-diagonal forms are central in the analysis of matrices, since they directly relate to *partitioning* and *grouping* tasks of the data. Blocks visually represent that their contained vertices share a similar connection characteristic. In a network analysis scenarios these blocks would be referred to as cohesive groups or clusters. Clear block patterns help counting clusters, estimate cluster overlap and identify larger and smaller clusters. Furthermore, many networks show block patterns with missing cells, meaning that clusters have missing connections (i.e., holes) or being connected to other clusters (i.e., off-diagonal dots).



Off-diagonal Block Pattern (*P2*): Off-diagonal coherent areas correspond to either sub-patterns of a *block pattern* or relations in a bi-graph. In the first case, the off-diagonal pattern would be visible in addition to the previous block pattern, and show connections between cliques.
Off-Diagonal blocks map to the user task of understanding how groups/entities are connected. In the graph task taxonomy of Lee [LPP*06], this pattern would allow approaching *adjacency assessment* and *overview* tasks. In the case of a bi-graph, the off-diagonal pattern would show consistent mappings from e.g., a set of authors to a set of documents. Just like the diagonal block pattern, off-diagonal blocks can contain missing connections.



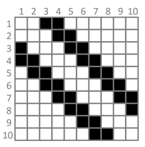
Line/Star Pattern (*P3*): Continuous horizontal and vertical lines are present in matrix plots if a vertex is strongly connected to several distinct other vertices.
This pattern helps the analysts to understand and



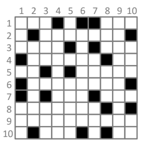
Robinsonian (Statistic)	Robinson [Rob51], Kendall [Ken63], Eisen et al. [ESBB98], Gelfand [Gel71], Hubert [Hub74], Brusco and Stahl [BS05], Brusco and Stahl [BS06], Brusco et al. [BKS08], Gruvaeus and Wainer [GW72], Bar-Joseph et al. [BJGJ01], Wilkinson [Wil05], Brandes and Wagner [Bra07], Behrisch et al. [BKSK12]
Spectral	Sternin [Ste65], Friendly [Fri02], Friendly and Kwan [FK03], McQuitty [McQ68], Breiger et al. [BBA75] Chen [Che02], Atkins et al. [ABH98], Koren and Harel [Kor05]
Dimension Reduction	Harel and Koren [HK02], Elmqvist et al. [EDG*08], Liu et al. [LHGY03], Spence and Graef [SG74], Rodgers and Thompson [RT92], Hill [Hil74, Hil79]
Heuristic Approaches	Deutsch and Martin [DM71], McCormick et al. [MDMS69, MSW72], Hubert and Golledge [HG81], Niermann [Nie05], Wilkinson [Wil05]
Graph Theoretic	Sloan [Slo86, Slo89], Harper [Har64], Harel and Koren [HK02], Rosen [Ros68], Cuthill and McKee [CM69], George [Geo71], Liu and Sherman [LS76], Chan and George [CG80], King [Kin70], Gibbs et al. [GPS76], Leung et al. [LVW84], Lozano et al. [LDGM12, LDGM13], Pop and Matei [PM14], Lenstra et al. [Len74, LK75], Bentley [Ben92], Henry-Riche and Fekete [HF06]
Biclustering	Hartigan [Har72], Cheng and Church [CC00], Lazzeroni et al. [LO*02], Turner et al. [TBK05], Murali and Kasif [MK03], Kaiser [Kai11], Prelic et al. [PBZ*06], Jin et al. [JXFD08]
Interactive User-Controlled	Bertin [Ber73, Ber81], Perin [PDF14], Brakel and Westenberg [BW13], Roa and Card [RC94], Siirtola [Sii99], Mäkinen and Siirtola [MS00], Caraux and Pinloche [CP05]

Figure 3: The taxonomy of the reviewed algorithms. For each algorithm the taxonomy reports first author, the year and the corresponding bibliographic reference is given.

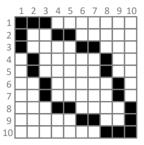
reason on the general *connectivity* aspects within the network. In a network analysis scenario lines would refer to hubs, i.e., nodes with many connections. The length of a line thereby indicates the number of connections (node degree).



Bands Pattern (*P4*⧅): Off-diagonal continuous lines refer to paths and cycles, or meshes in a network. They represent a set of vertices with a few connections to other vertices. Visually, this pattern can be regarded as a special case of the line pattern and is useful whenever (adjacency) relationships and *connectivity* aspects are in the user focus [LPP*06]. In a network analysis scenario bands would refer connection paths and transition chains, where the width of the band visually depicts how many distinct paths could be taken through the network.



Noise Anti-Pattern (*A1*▩): Noise (also called *salt-and-pepper*) is the classic anti-pattern for a matrix plot. It can be found whenever the row-/column ordering is not able to reveal the underlying graph topology or if simply no structure exists. However, submatrices can occur to be noisy, even if other submatrices show structure. Moreover, a matrix can be noisy or show structure on different levels: locally, i.e. for subgraphs (submatrices), and globally, i.e. the entire graph (matrix).
The distinction between anti-patterns and the mentioned (interpretable) visual patterns helps the analyst to develop an *overview* about the topological aspects of the network at hand.



Bandwidth Anti-Pattern (*A2*⧅): Bandwidth- or sparsity patterns visually group non-zero elements (connections) within an enclosure around the diagonal. This pattern adds little interpretation asset to the matrix plot if the inner part of the bandwidth enclosure reveals no structure. Bandwidth patterns are typical for breadth-first search algorithms where the outer border depicts the stop criterion of the enumeration (cf. Section 8).
However, similarly to the noise anti-pattern, bandwidth patterns allow to reason on the absence of (expected) topological aspects and facilitate thus *overview* and *exploration* tasks [LPP*06].

Any graph motif has a corresponding visual pattern in a visual matrix, and we only described the most important ones above. Real world graphs exhibit a mixture of overlapping patterns appearing at different scales. Hence, the visual patterns we describe are not always clearly discernible (Figure 1) and may appear merged together. Reordering algorithms take into consideration different aspects of the topology, inducing different patterns. Others directly optimize for specific patterns. Note that several of these algorithms however fail to reveal any pattern or introduce visual artifacts.

2.4. Implementations

Most of the reordering algorithms are available in public libraries, although no library implements them all yet. The `Reorder.js` [Fek15] library provides many algorithms in JavaScript. The R packages *'corrplot'* [Wei13], *'biclust'* [KL08] and *seriation* [HHB08] provide a large number of seriation algorithms for tabular data. Several graph algorithms are available in the C++ Boost library [SLL01].

3. Classification of Algorithms

We now survey algorithms for matrix reordering. Our coverage is not exhaustive but biased towards impact publications in the respective sub-domains. There is also a large number of methods to speed-up or otherwise improve some algorithmic approaches, but we try to capture the most important concepts and algorithms here; the details can be found in the original articles that are cited.

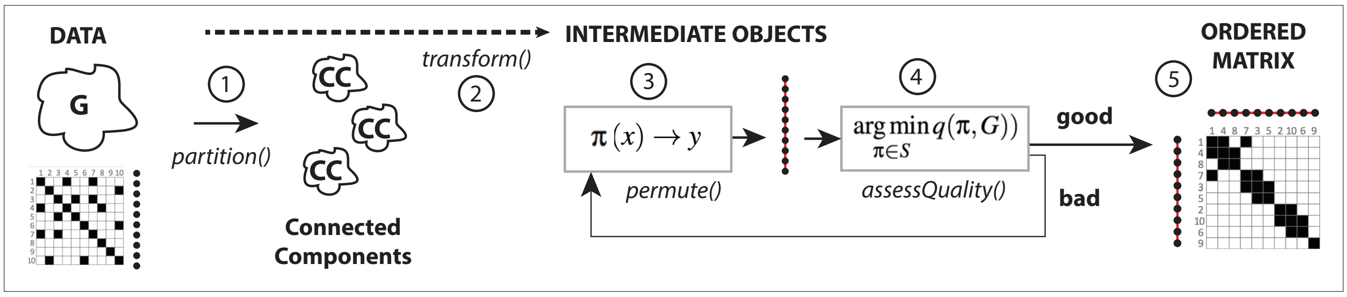


Figure 4: General process of reordering matrices: After an optional partitioning phase of the input data (1), the data is transformed into a problem space related representation (e.g., graphs) (2). Within this problem space, the reordering approaches determine a valid permutation (3) and assess the reordering based on an algorithm-specific quality criterion (4). If a sufficiently good ordering is found the matrix is reordered accordingly (5).

For the purpose of simplicity and understandability, we decided to group the algorithms into seven algorithmic families that arise from the *inherently shared reordering concept*, as depicted in our taxonomy in Figure 3. While all algorithms share the same objective of deriving an appropriate matrix reordering, every algorithm itself comprises its own design considerations and decisions.

Our classification of algorithms is tailored to the central goal of providing guidance on what algorithm to use depending on the dataset characteristics (e.g., size and structure). During our research we examined different taxonomies and orthogonal dimensions to describe algorithms in a comprehensive way; the domain they were developed in, the mathematical background, and the kind of information used to determine the distances between vertices (rows and columns). However, we found that none of those taxonomies was expressive enough while remaining simple to classify algorithms. Overall, we derived orthogonal taxonomy families/groups wherever possible, but also use the concept of overlapping and meta families to stress the importance of shared concepts or to emphasize particular features.

3.1. Multiple Ways of Reordering

We classify the algorithms for computing these permutations, depending on the stages and intermediate objects required to perform the computation. Figure 4 outlines the steps involved in reordering:

1. **Partition** the network into connected components and apply the reordering in each component separately. For the final matrix, the components are usually concatenated by decreasing sizes.
2. **Transform** the data into intermediate objects, such as distance matrices, Laplacian matrices, or Eigenvector spaces.
3. **Create a permutation** from those intermediate objects,
4. **Assess the quality** of the obtained permutation. If unsatisfactory, create a permutations, otherwise
5. **Apply permutation** to the matrix, i.e. reorder rows and columns in the original visual matrix.

The following sections will explain which intermediate objects as well as permutation and quality assessment methods each algorithm group employs.

4. Robinsonian (Statistic) Approaches

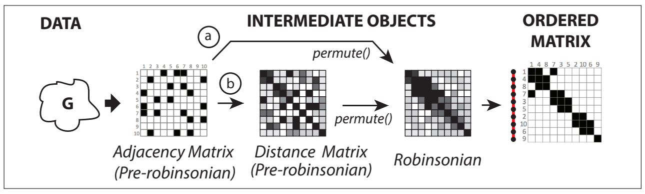


Figure 5: Robinsonian Matrix Reordering.

The fundamental idea of all Robinsonian approaches is to reorder a matrix so that similar rows (resp. columns) are arranged close and dissimilar rows (res. columns) are placed farther apart.

Robinsonian methods compute a similarity matrix from the A vectors (resp. B) of the (adjacency) matrix M. They then try to compute a permutation π_r (resp. π_c) to transform these similarity matrices into a Robinson matrix or R-matrix. An $n \times n$ symmetric matrix R is called a Robinson similarity matrix if its entries decrease monotonically in the rows and columns when moving away from the diagonal, *i.e.,* if:

$$\begin{cases} R_{i,j} \le R_{i,k} & \text{for } j < k < i \\ R_{i,j} \ge R_{i,k} & \text{for } i < j < k \end{cases} \tag{11}$$

If instead of computing a similarity we compute a distance, then the entries should *increase* monotonically, but the principle remains identical. This property means that similar vertices are as close as possible in a consistent way.

When a similarity matrix can be permuted to become an R-matrix, it is called a *Pre-robinsonian matrix* (Pre-R) (see Figure 5 (middle)). The challenge is to find the permutation. When found, this permutation can be applied to the similarity matrix as shown in the Figure 5 (middle), but also to the original matrix M.

However, there are similarity matrices that are not Pre-R; in other words, not all the similarity matrices can be permuted to become an R-matrix. For real world cases, very few matrices are in fact pre-R. Therefore, two problems arise:

1. When a matrix is Pre-R, how to compute the permutation that transforms it into an R-matrix form?
2. When a matrix is not Pre-R, what is a good approximation of an R-matrix and how to compute it?

It turns out that there is a solution to the first question that has been ignored until recently by the statistical community [ABH98]. However, it does not address the second question at all. Therefore, the heuristics developed to approximate the general Robinsonian problem are still useful since they provide many solutions potentially applicable to the second question.

Distance/Similarity Measures

The first step to all the traditional Robinsonian algorithms consists in computing the similarity matrices from the (adjacency) matrix. Computing a distance—alike the similarity—matrix is always quadratic in time and space, and it implies choosing a measure. For distances, classical measures include the well-known norms L_p:

$$L_p(x,y) = (\sum_{k=1}^{n}(x_k - y_k)^p)^{1/p}, p > 0 \qquad (12)$$

The most used are L_1, L_2 (see Equation 6), and L_∞ that simplify as $L_1(x,y) = \sum_k |x_k - y_k|$ and $L_\infty(x,y) = \max_k(|x_k - y_k|)$.

The choice of the measure may not be considered arbitrarily and has *significant impact* on the visual appearance of the matrix to be calculated. Since this aspect relates also to other matrix reordering families we will discuss distance metrics and parameterizing algorithms in Section 11.2.

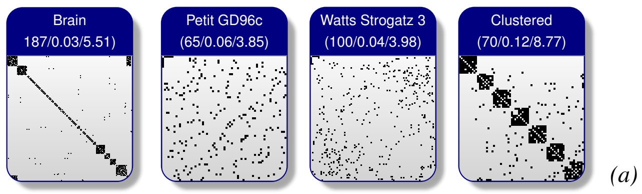
Algorithms and Variations

We now survey the three main approaches to compute a good permutation for the Robinsonian problem: greedy algorithms, clustering, and optimal-leaf ordering. Almost all algorithms in this group have to deal with the problem of potentially retrieving a *local optimal solution*, since a full enumeration of all permutations in the problem space is mostly infeasible. Few algorithms exist that are able to retrieve perfect anti-Robinson structures. These methods are not practical, due to their runtime, but provide an upper bound to this family of algorithmic methods. For example, the Branch-and-Bound algorithm or the Heuristic Simulated Annealing approach from Brusco and Stahl [BS05, BKS08] are able to retrieve global optimum solutions for up to 35 nodes and 35×35 connections with CPU times ranging from 20 to 40 min [BS05, p. 263].

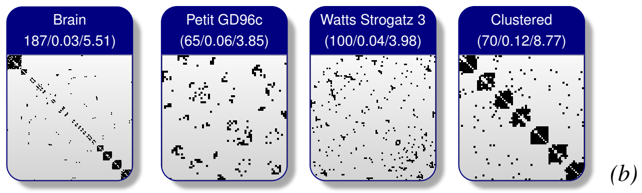
(a) Greedy Algorithms: For large graphs with hundreds or thousands of nodes, one can enumerate permutations in a greedy fashion. For example, the *Bipolarization* algorithm [Hub74] alternates between rows and columns to reorganize a distance matrix, placing the highest dissimilarities in the remotest cells from the diagonal (Algorithm 1). Like most other greedy algorithms, Bipolarization (depicted in Figure 6(a)) is fast and yields good results whenever the underlying data structure contains a bi-polar organization (e.g., patterns *P1*, *P2*, *A1*). However, as already mentioned, greedy algorithms do not guarantee to find an optimal solution, but quickly yield a "reasonable good" solution. These solutions can then be improved further by seeking to optimize locally. Figure 6(c) shows examples of locally optimized matrices.

Algorithm 1 Greedy suboptimal enumeration of matrix permutations [Hub74, CP05].

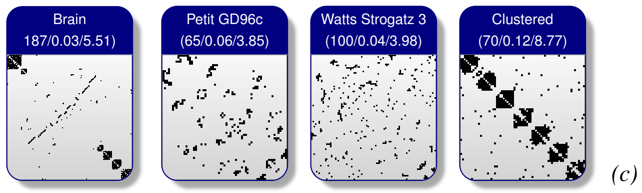
```
procedure BIPOLARIZATION ALGORITHM
    $D \leftarrow distMatrix(M)$.                                  ▷ Distance Matrix
    $D_{max} \leftarrow \max(upperTriangle(D))$.
    $\pi_{col}[1] \leftarrow colIndex(D_{max})$.
    $\pi_{row}[1] \leftarrow rowIndex(D_{max})$.
    $D \leftarrow applyPermutation(\pi_{col,row}, D)$.   ▷ Reorganize Distance Matrix
    $D_{max} \leftarrow maxFromIndex(row_i, col_i, i)$.
    if $maxFromIndex(row_i, i) > maxFromIndex(col_i, i)$ then
        $rowDirection \leftarrow$ true.
        $\pi_{row}[2] \leftarrow rowIndex(D_{max})$.
    else
        $rowDirection \leftarrow$ false.
        $\pi_{col}[2] \leftarrow colIndex(D_{max})$.
    end if
    $D \leftarrow applyPermutation(\pi_{col,row}, D)$.   ▷ Reorganize Distance Matrix
    $rIndex, cIndex \leftarrow 3$.
    repeat
        if $rowDirection$ then
            $D_{max} \leftarrow maxFromIndex(row_i, i)$.
            $\pi_{row}[rIndex] \leftarrow rowIndex(D_{max})$.
            $rIndex+ = 1$.
        else
            $D_{max} \leftarrow maxFromIndex(col_i, i)$.
            $\pi_{col}[cIndex] \leftarrow colIndex(D_{max})$.
            $cIndex+ = 1$.
        end if
        $D \leftarrow applyPermutation(\pi_{col,row}, D)$.
        $rowDirection \leftarrow$ not$rowDirection$.
    until $rIndex = cIndex = |M|$.
    $M \leftarrow applyPermutation(\pi_{col,row}, M)$.      ▷ Reorganize Data Matrix
end procedure
```



Bipolarization [Hub74]



Hierarchical Clustering [GW72]



Optimal-Leaf-Ordering [BJGJ01]

Figure 6: Examples for Robinsonian Matrix Reorderings.

(b) Clustering Algorithms: Clustering algorithms, in the context of matrix reordering, are based on deriving clusters of “similar” data elements (e.g., nodes) and ordering each cluster individually. Building on this, Gruvaeus and Wainer [GW72] suggested to order clusters at different levels using an *hierarchical* clustering (dendrogram). Elements at the margin of each cluster, i.e. the first and last element in the obtained order for the respective clusters, should also be similar to the first (or last) element in the adjacent cluster. Figure 6(b) shows the result of the hierarchical clustering algorithm by Gruvaeus and Wainer (*RSeriationGW*).

Hierarchical clustering algorithms aim at producing grouping patterns (*P1*), however, groups are not necessarily placed along the matrix diagonal (Figure 6(b), *P2*). In biology, Eisen et al. [ESBB98] used agglomerative hierarchical clustering with average linkage to reveal interesting phenomena in gene expression data matrices. As also discussed by Wilkinson in “The Grammar of Graphics” [Wil05, p. 526-527] the choice of the average linkage method often yields visually good results, but represents a middle-ground between two extremes: single and complete linkage. While complete linkage tends to produce spherical clusters, single linkage tends to produce snakelike clusters.

(c) Optimal-Leaf-Ordering Algorithms: In addition to the aforementioned clustering approaches, smoothing the clusters by ordering the vertices according to their neighborhood similarities reveals structures more clearly than walking the leaf of the hierarchical clusters in an arbitrary order. Finding an ordering consistent with the hierarchical binary tree is known as the *Optimal-Leaf-Ordering* problem. An optimal ordering is computed globally, so as to minimize the sum of distances between successive rows (columns) while traversing the clustering tree in depth-first order. For any two nodes p and q in the binary clustering tree and that share the same parent, two orders are possible: (p,q) or (q,p).

Bar-Joseph et al. [BJGJ01] describe an exact solution that has a time complexity of $\mathcal{O}(n^4)$ and a memory complexity of $\mathcal{O}(n^2)$.Though this can be improved at the expense of more memory, using *memoization* techniques. Brandes [Bra07] was able to present a solution with time complexity of $\mathcal{O}(n^2 \log(n))$ and memory complexity of $\mathcal{O}(n)$, making it practical for larger matrices. Figure 6(c) depicts exemplified matrix reordering results with visually coherent block patterns (*P1*) derived from the *RSeriationOLO* algorithm.

Discussion

The quality of Robinsonian approaches is essentially influenced by two choices: (i) the measure of distance (or similarity) and (ii) the enumeration approach. The goal of every R-matrix reordering approach is to optimize similarity between neighboring rows and columns. The direct outcome is that block patterns (*P1*) are made visible. Hence, Robinsonian approaches should be preferred if a dataset partitioning is expected for yet undetermined data subgroups. On the other hand, even if reordering is less strict than clustering, “*analysis via clustering makes several a-priori assumptions that may not be perfectly adequate in all circumstances. First, clustering [...] implicitly directs the analysis to a particular aspect of the system under study (e.g., groups of patients or groups of co-regulated genes). Second, clustering algorithms usually seek a disjoint cover of the set of elements, requiring that no gene or sample belongs to more than one cluster*” [TSS05, adapted, p. 3-4].

Yet, even if the data contains inherent groupings, the similarity function has to be selected with caution, since an inappropriate choice will disturb grouping patterns. Generally, when clusters exists, (hierarchical) clustering approaches show visually promising results, for all linkage functions.
In turn, Robinsonian approaches—in comparison to the other reordering approaches—allow incorporating more domain-specific knowledge into the analysis process, e.g., by applying domain-specific similarity considerations. The Bertifier [PDF14] system uses extensively this flexibility to allow interactively specified preferences to influence the reordering algorithm.

5. Spectral Methods

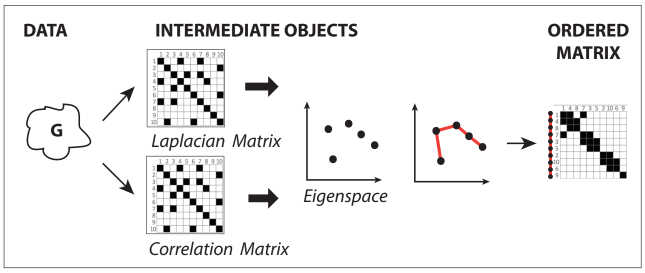


Figure 7: Spectral Matrix Reordering.

Spectral methods relate to linear algebra and use eigenvalues and eigenvectors to calculate a reordering, i.e. each row (or column) is projected into the eigenspace where distances between eigenvectors are used to calculate a reordering (Figure 7). Given a symmetric matrix M of dimension $n \times n$, we say that $\boldsymbol{\lambda}$ is an *eigenvalue* of M if $Mx = \boldsymbol{\lambda}x$ for some vector $x \neq 0$. The corresponding vector x is an *eigenvector*. An $n \times n$ symmetric matrix has n eigenvectors that can be constructed to be pairwise orthogonal, and its eigenvalues are all real. We refer to the eigenvalues in increasing numbers $\boldsymbol{\lambda}_1 \geq \boldsymbol{\lambda}_2 \ldots \geq \boldsymbol{\lambda}_n$.

Computing the eigendecomposition of a matrix is an expensive operation. However, since the reordering algorithms will use only the few first or last eigenvectors, iterative methods can efficiently be used, in particular *Power-Iteration* [HK02] (see Algorithm 2).

The Power-iteration method is efficient when the largest eigenvalues have different magnitudes. Otherwise, the system is *ill-conditioned* and convergence will be slower or not existent at all. Efficient *preconditioning* methods exist to address this problem, such as the `LOBPCG` method [Kny01], with implementations in many popular languages.

To compute the eigenvectors with the smallest eigenvalues, a simple transformation is required. The matrix $M' = g \cdot I - M$ has the same eigenvectors as a symmetric matrix M but with the eigenvalues in reverse order. The constant g is called the Gershgorin bound [Wat91], which is a theoretical upper bound for (the absolute value of) the largest eigenvalue of a matrix:

$$g = \max_i \left(M_{i,i} + \sum_{j \neq i} |M_{i,}| \right). \qquad (13)$$

Algorithm 2 Power-Iteration to compute the first k eigenvalues and eigenvectors [HK02].

```
function PowerIterations($M$)            ▷ This function computes
    $x_1, x_2, \dots, x_k$, the first k eigenvectors of M.
    const $\varepsilon \leftarrow 0.001$
    for $i \leftarrow 1, k$ do
        $\hat{x}_i \leftarrow$ random
        $\hat{x}_i \leftarrow \frac{\hat{x}_i}{\|\hat{x}_i\|}$
        do
            $x_i \leftarrow \hat{x}_i$                    ▷ orthogonalize
            for $j \leftarrow 1, i-1$ do
                $x_i \leftarrow x_i - (x_i^T x_j) x_j$
            end for
            $\hat{x}_i \leftarrow M x_i$
            $\hat{x}_i \leftarrow \frac{\hat{x}_i}{\|\hat{x}_i\|}$
        while $\hat{x}_i^T x_i < 1 - \varepsilon$
        $x_i \leftarrow \hat{x}_i$
    end for
return $x_1, x_2, \dots, x_k$
end function
```

Algorithms and Variations

Spectral methods are used in two ways: (i) using the properties of the eigenvectors with the largest eigenvalue(s), or (ii) using the eigenvector with the smallest non-null eigenvalue due to its structural properties.

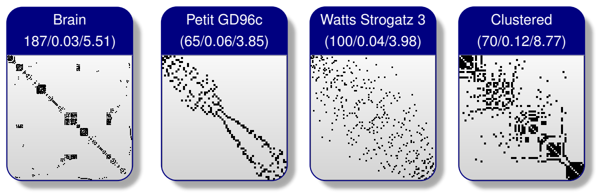
Sternin commented already in 1965 on the useful rank-order properties of the two eigenvectors corresponding to the two largest eigenvalues $\boldsymbol{\lambda}_1$ and $\boldsymbol{\lambda}_2$ [Ste65] In his divide-and-conquer approach, he splits the row/column indices into three distinct classes according to their index position and the value of the integral abscissa values of the *second* principal component at that specific index position. Each class is then permuted by the value of the integral abscissa of the *first* principal component at the specific index position.

Friendly [Fri02] developed this concept further in 2002. He was using *correlation matrices* as opposed to the raw data (Figure 7) to position similar variables adjacently, facilitating perception. Moreover, rather than sticking to the integral abscissa values of the first two principal components, he arranges the row/columns based in the angular order of the eigenvectors:

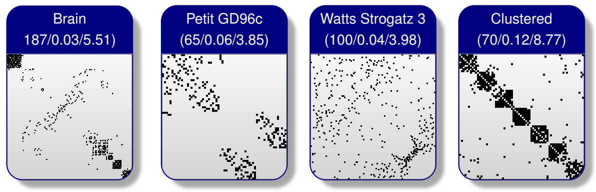
$$\alpha_i = \begin{cases} \tan^{-1}(e_{i2}/e_{i1}) & \text{for } e_{i1} > 0 \\ \tan^{-1}(e_{i2}/e_{i1}) + \pi & \text{otherwise} \end{cases} \qquad (14)$$

The achieved circular order for the row/column vectors is unfolded into a linear arrangement by splitting at the largest gap between adjacent vectors. Figure 8(a) shows example results for the *RCorrplotSortingAOE* algorithm, depicting visually pleasing global structures, but also bandwidth patterns (*P4*▧) without recognizable structure within the band.

A related technique was introduced by McQuitty in 1968 [McQ68], who observed the convergence of recursively formed (Pearson) correlation matrices into a matrix with the only elements being -1 and $+1$. Starting from $R^{(1)}$ (the correlation matrix of the original distance matrix) the sequence $(R^{(1)}, R^{(2)}, \dots)$ is formed



Angular Order of Eigenvectors [Fri02]



Rank-two Ellipse Seriation [Che02]

Figure 8: Examples for Spectral Matrix Reorderings.

by $R^{(i)} = corr(R^{(i-1)})$ and eventually derives $R^{(\text{inf})}$ with the mentioned properties. In 1975 Breiger, Boorman and Arabie [BBA75] found that the resultant block form of the correlation matrix $R^{(\text{inf})}$ (if correctly ordered) represents a valid matrix reordering. Later, in 2002 Chen [Che02] developed these ideas further and explored a rank reduction property with an elliptical structure, even before the convergence, as Algorithm 3 showcases. Figure 8(b) shows exemplified results for Chen's rank-two ellipse seriation with noticeable (even though sparse) off-diagonal pattern tendencies (*P2*▚).

Algorithm 3 Rank-two Ellipse Reordering with recursively built Pearson correlation matrices [Che02].

```
procedure Rank-Two Ellipse Reordering
    $D \leftarrow distMatrix(M)$.                        ▷ Distance Matrix
    $R^{(0)} \leftarrow PearsonCorr(D)$.
    $i \leftarrow 1$.
    repeat
        $R^{(i)} \leftarrow PearsonCorr(R^{(i-1)})$.
        $i \leftarrow i + 1$.
    until $rank(R^{(i)}) = 2$.          ▷ Recursive Pearson Corr. Matrices
    Get first two Principal Components (PCs) of $R^{(i)}$.
    Project rows/columns onto the 2D plane of these PCs.
    Cut ellipse between the two converged groups.
    $\pi \leftarrow$ 1D rank approximation.
    $applyPermutation(\pi, M)$.               ▷ Final Matrix Permutation
end procedure
```

Solution to the Robinsonian Problem Atkins et al. [ABH98] have solved the Robinsonian problem using the eigenvectors of the *Laplacian matrix*. The Laplacian matrix is defined as $L = D - M$, where D is the degree matrix and M is the adjacency matrix of the graph G:

$$D_{i,j} := \begin{cases} \sum_{k=1}^{n} M_{k,i} & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} \qquad (15)$$

A general graph G with c connected components has c eigenvectors with an eigenvalue of 0. If the graph has only one connected component, then $\boldsymbol{\lambda}_n = 0$ and the associated eigenvector is a vector filled with 1 and noted **1**. The smallest non-null eigenvalue $\boldsymbol{\lambda}_{n-1}$ is called the *Fiedler value* and the corresponding eigenvector x_{n-1} is called the *Fiedler vector* [ABH98,Fie73]. The Fiedler vector can be used to order a graph by ordering the vertices according to the order of the vector values. This order is a solution to the Robinsonian problem for a Pre-R matrix.

Discussion

Spectral approaches using the first eigenvectors build on the assumption that the core matrix structure can be extracted from only a few dominant dimensions. Unfortunately, the eigenvectors are very sensitive to data corrupted with outliers, missing values, and non-normal distributions [LHGY03]. In cases where the underlying correlation-dimension is not uni-dimensional (e.g. multi-dimensional, or cyclic), such as Wilkinson's *circumplex* form [Wil05, cf. p. 521], these approaches will fail inherently, producing salt-and-pepper visual patterns (*A1*).

In contrast, spectral approaches using the Fiedler vector seem robust to noise and tend to generate good results consistently.

6. Dimension Reduction Techniques

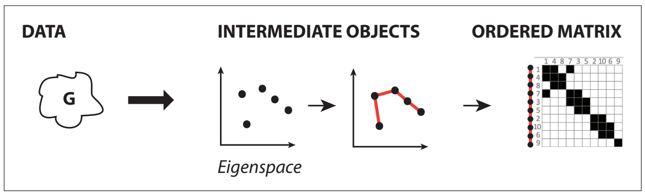


Figure 9: Dimension Reduction Matrix Reordering.

Dimension reduction techniques constitute a rather small stream to the matrix reordering landscape. While the actual mathematical ideas have been developed centuries ago, their applicability to meaningful problem instances in terms of size was hindered by the calculation performance.

The central goal of dimension reduction techniques is to retrieve a one-dimensional layout/ordering of the rows/columns that reflects the *(non-)linear relationships* between them (Figure 9).

Algorithms and Variations

The main methods in this field (*Principal Components Analysis* (PCA) and variants, and *Multidimensional Scaling* (MDS)) share the commonality that they decompose varying intermediate objects (e.g., covariance matrix, normalized data matrix) with the help of a *Singular Value Decomposition* (SVD) step to derive a permutation.

(a) Principal Component Analysis: One of the most popular techniques for dimension reduction is Principal Component Analysis (PCA). PCA computes a projection of multidimensional data into an n-dimensional space that preserves the variance between elements in the data. In the context of matrix reordering, the 1^{st} principal component of a *covariance matrix* must project the data. This 1^{st} principal component represents the most variance and accordingly the most expressiveness of the data. Figure 10(a) shows matrix plots resulting from the *RSeriationPCA* matrix reordering.

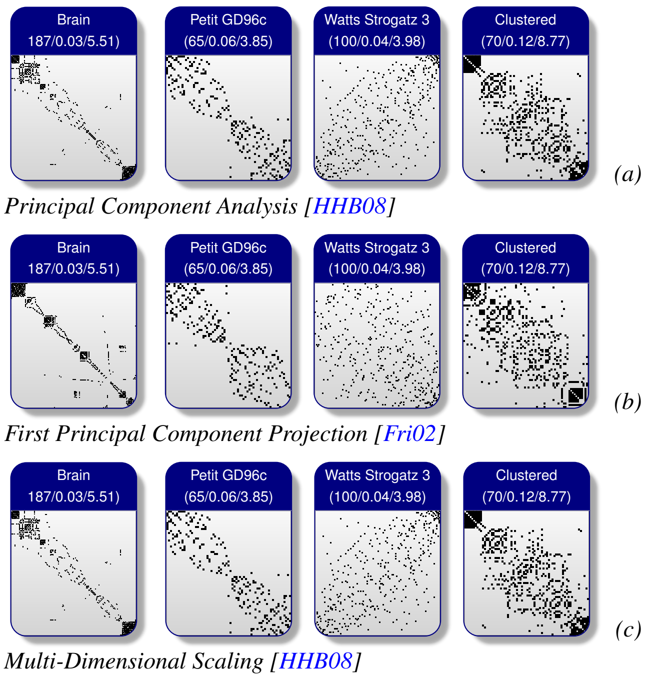


Figure 10: Examples for Dimension Reduction Reorderings.

(b) First Principal Component Projection Alternatively, the 1^{st} principal component can be directly derived from the raw matrix data; without the intermediate step of building a covariance matrix. Figure 10(b) shows this approach on exemplified matrix plots.

As one can see, Figure 10(a) (PCA) and Figure 10(b) (First PCP) resemble each other. However, PCA is able to show off-diagonal patterns, such as *P2*, more clearly and tends to produce patterns along the anti-diagonal of a matrix.

(c) High Dimensional Embedding: Computing the eigenvectors was long time only possible for small matrices. However, iterative approaches (see Algorithm 2) and the ability to parallelize the calculations (even GPU implementations are available [And09, KY05]) allow computing the PCA in real-time for large graphs. A modified PCA method, called "High Dimensional Embedding", is described by Harel and Koren [HK02]. Their method uses the first two/three components of the PCA for laying out node-link diagrams with up to one million nodes in a few seconds. This method was adapted by Elmqvist et al. for reordering matrices for graphs of a million edges [EDG*08]. Elmqvist et al.'s results show that High Dimensional Embedding results in a visually pleasing overview, while the local ordering is poor *A1*.

(d) Single Value Decomposition: Liu et al. [LHGY03] follow the idea that data is inherently corrupted with outliers, missing values, and non-normal distributions that cover up the matrix patterns. Thus, a row vector approximation with bilinear forms $x_{ij} = r_i m_j + e_{ij}$ is used, where r_i is a parameter corresponding to the ith row, m_j corresponds to the jth column and e_{ij} is a residual/error value. Rows are ordered iteratively by their regression coefficients r_i, respectively m_j for the columns, with the assumption that similar regression coefficients group visually similar rows/columns. The equation can be solved using SVD, which decomposes a rectangular matrix D_{mn} into the product of 3 matrices $U_{mm}\Sigma_{mn}V_{nn}^T$ where $U^TU = I$ and $V^TV = I$. Although the method is robust, it is expensive since its complexity is higher than quadratic in time.

(e) Multi-Dimensional Scaling: Similar to PCA, another possibility to discover structure in matrices is multi-dimensional scaling (MDS) [BL12]—also denoted as Similarity Structure Analysis. In 1974, Spence and Graef recognized this interrelation and applied MDS to the matrix reordering problem [SG74]. MDS assigns rows/columns to a specific index in a conceptual one-dimensional space, such that the distances between the respective vectors in the space match the given dissimilarities as closely as possible. The cost function to be minimized is an overall distortion of the positions. With this approach MDS can derive *non-linear relationships* among the matrix rows/columns.

MDS techniques can be distinguished into two types: (i) non-metric MDS, which involves data that is not necessarily all numeric was applied by Rodgers and Thompson [RT92] for matrix reordering, and (ii) classical MDS which involves numeric data (preferably variables in the same scale) was applied by Spence and Graef in [SG74]. Classical MDS algorithm is based on the fact that the permutation indices X— or one dimensional coordinate matrices—can be derived by eigenvalue decomposition from the scalar product matrix $M = XX^T$. To achieve this, each value in the distance matrix must be squared and "double centered", such that the columns and rows both have a zero mean. Subsequently the SVD of this (normalized) matrix is calculated and the index positions are retrieved from the factors returned by the SVD. The steps in Algorithm 4 summarize the algorithm of classical MDS. Figure 10(c) shows matrix plots for the same *RSeriationMDS* algorithm, resulting in almost identical plots than when using PCA.

Algorithm 4 Double Centering and Singular Value Decomposition in the MDS Matrix Reordering.

```
 1: procedure MDS MATRIX REORDERING
 2:     $D^2(i,j) \leftarrow -\frac{1}{2}d(x_i,x_j)^2$.                ▷ Squared Distance Matrix
 3:     $rowMean = mean(M)$.
 4:     $colMean = mean(transpose(D^2))$.
 5:     $totalMean = mean(rowMeans)$.
 6:     for $i = 0, |D^2|$ do
 7:         for $j = 0, |D^2|$ do
 8:             $D^2(i,j) += totalMean - rowMean_i - colMean_j$;
 9:         end for
10:     end for                                                        ▷ Double Centering
11:     $U\Sigma V^\top = \mathrm{SVD}(D^2)$                          ▷ Singular Value Decomposition
12:     $eigenValues \leftarrow \sqrt{\Sigma}$
13:     $U' \leftarrow \sqrt{\Sigma}$
14:     for all $row \in U$ do
15:         $row \times eigenValues$.
16:     end for                                                        ▷ Eigenvector Normalization
17:     $\pi \leftarrow U_1$.
18:     $M \leftarrow applyPermutation(\pi, M)$.                      ▷ Final Matrix Permutation
19: end procedure
```

Alike PCA methods, classical MDS can be heuristically adapted to allow for larger problem instances. Brandes and Pich [BP07, Pic09] propose *PivotMDS*, a sampling-based approximation technique for classical MDS, which is able to determine a layout of node-link diagrams in linear calculation time and with linear memory consumption.

Discussion

The central idea of dimension reduction techniques is to take advantage of the inherent and potentially hidden (non-)linear structures in the data. This has direct consequences on the matrix plot to be expected: Normally these approaches favor high-level/coarse-grained structures (*P1*) over fine matrix patterns (e.g., lines (*P3*)). While PCA is only able to retrieve linear structures, MDS also allows determining non-linear data relationships. On the other hand, there are only rare cases where a non-linear data structure should be examined in a matrix form. Other visualizations, i.e., the raw two-dimensional MDS projection, is better suited for these purposes. In general, Wilkinson notes that SVD and MDS methods are performing best in terms of the Spearman correlation between the known row indices (after constructing the matrix) and the permuted indices [Wil05, p.532].

7. Heuristic Approaches

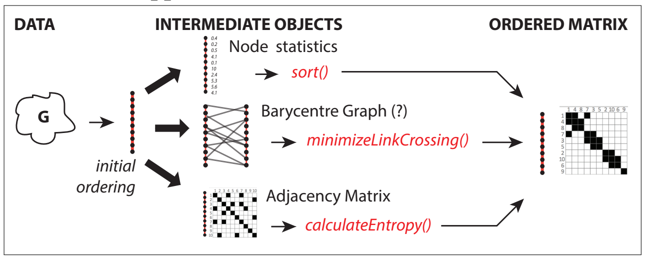


Figure 11: Heuristic Approaches for Matrix Reordering.

Heuristics are methods that transform the problem of finding an ordering into another problem space that abstracts the problem appropriately and allows for computationally efficient problem solving. Heuristic approaches can be separated into *problem simplification* and *problem space transformation* approaches. Problem simplification methods try to use row/column approximations to iterate through the permutation possibilities (or a subset thereof), while problem space transformation methods are transforming rows and/or columns into other meaningful representations (e.g., the nodes of a bi-graph).

Algorithms and Variations

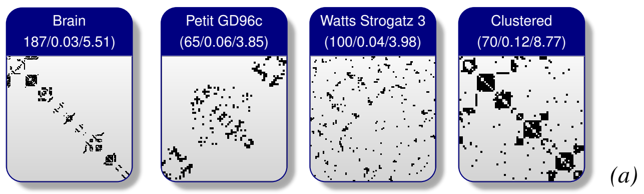
(a) Numerical Abstractions: One classical instance for simplification methods is to neglect the row dimensionality and use numerical abstractions for each row and/or column instead. Deutsch and Martin [DM71] proposed to use the *mean row moments* to find the principal axis and thus a single dominant relationship of the data. Mean row/column moments are defined as follows:

$$x_i = \frac{\sum_{j=1}^{N} j m_{ij}}{\sum_{j=1}^{N} m_{ij}} \qquad (16)$$

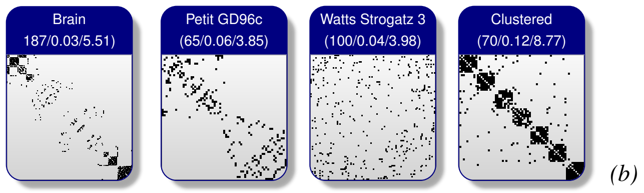
This heuristic approximation is iteratively applied *separately* on the rows and columns until the simple vector mean quality measure stabilizes, as Algorithm 5 depicts.

Algorithm 5 Separately applied row/column ordering with the Mean Row Moments quality criterion [DM71].

```
1:  procedure MEAN ROW MOMENTS HEURISTIC REORDERING
2:      repeat
3:          for all $row_i \in M$ do
4:              $x_i \leftarrow meanRowMoment(row_i)$.
5:          end for
6:          $\pi \leftarrow orderPermutation(x)$.
7:          $M \leftarrow permute(\pi, M)$.                  ▷ Row Reordering
8:          for all $col_i \in M$ do
9:              $y_i \leftarrow meanColMoment(col_i)$.
10:         end for
11:         $\pi \leftarrow orderPermutation(y)$.
12:         $M \leftarrow applyPermutation(\pi, M)$.        ▷ Column Reordering
13:     until Row and column reordering is stable.
14: end procedure
```



Bond Energy Algorithm [MDMS69]



Anti-Robinson Simulated Annealing [BKS08]

Figure 12: Example for Heuristic Matrix Reorderings.

McCormick et al. contrast in [MDMS69, MSW72] three different heuristics for establishing a matrix reordering: (i) Moment Ordering; by means of the mean row moments, (ii) Moment Compression Ordering; by means of the sum of second moments and (iii) the *Bond Energy Algorithm*, short BEA. BEA uses a "measure of effectiveness" as a quality criterion, and tries to maximize the so called *bond energy* over all row- and column permutations. Figure 12(a) shows matrix plots for the *RSeriationBEA* algorithm, which shows tendencies to produce Block-Diagonal matrix forms (*P1*) in combination with off-diagonal groupings (*P2*) and star patterns (*P3*). As seen from the results in Figure 12(a), BEA tries to maximize contiguous chunks, forming more or less concise *groupings (P1, P2)* in the matrix plot.

Three further heuristics are suggested by Hubert and Golledge: "Different Counting Rule", "Gradient Within Rows", "Szczotka's Criterion" [HG81, p. 436-439]. Mäkinen and Siirtola [MS00] propose to reorder the rows/columns iteratively and separately (such as in Algorithm 5) by their weighted row/column sum. Further heuristics with varying goals are described in [MS14, p. 199].

(b) Barycenter Heuristic: Mäkinen and Siirtola propose to use the *barycenter heuristic* [MS05] to layout a bipartite graph, in

Algorithm 6 Barycenter Heuristics [MS05].

```
1:  procedure BARYCENTER HEURISTIC(graph)
2:      $layer1 \leftarrow [v \in V$ where $outDegree(v) \neq 0]$
3:      $\pi_1 \leftarrow identityPermutation(|layer1|)$
4:      $layer2 \leftarrow [v \in V$ where $inDegree(v) \neq 0]$
5:      $\pi_2 \leftarrow identityPermutation(|layer2|)$
6:      repeat
7:          for $v \in layer1$ do
8:              $n \leftarrow outNeighbors(v)$
9:              $position[v] \leftarrow barycenter(\pi_2, n)$
10:             $\pi_1 \leftarrow orderPermutation(position)$
11:         end for
12:         for $v \in layer2$ do
13:             $n \leftarrow inNeighbors(v)$
14:             $position[v] \leftarrow barycenter(\pi_1, n)$
15:             $\pi_2 \leftarrow orderPermutation(position)$
16:         end for
17:     until Row and column reordering is stable.
18: end procedure
19: function BARYCENTER($\pi, n$) return $\frac{\sum_{v \in n} \pi(v)}{|n|}$
20: end function
```

which the matrix rows correspond to the first graph partitioning and the columns to the other partitioning. An adaption of the *Sugiyama* algorithm is applied to minimize the edge crossings (Figure 13). Algorithm 6 operates on the two "layers" (the two partitions). It is repeated until the number of crossings does not decrease any more. Two small changes improve the algorithm results substantially: replacing the barycenter by the *median* [EW94], and applying a post-processing, repeatedly swapping consecutive vertices on the two layers as long as it lowers the number of crossings [GKNV93]. This algorithm has the tendency to arrange matrix plots, such that *clusters* stick out in the top left and bottom right corners.

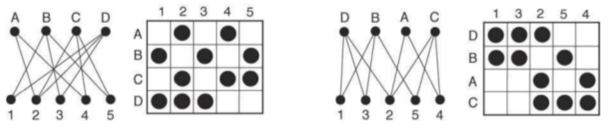


Figure 13: Reordering using the Barycenter Heuristic [MS05].

(c) Entropy Optimization and Genetic Algorithms: Niermann [Nie05] uses a genetic algorithm to assess the "fitness" of a matrix permutation by means of an entropy-based objective function over all matrix plot pixels. The intuitive idea is that a well-ordered matrix plot—containing locally coherent regions of similar values (e.g., black or white)—requires a minimum number of bits for encoding. In other words, the better a matrix plot can be compressed, the better is its reordering (under the assumption the data contains clusters, groupings, or other partitionings). For the Niermann's genetic algorithm he models permutations, the individuals of the algorithm, as arrays of ordering indices. Child individuals are created from consistently rearranging permutation subsequences in the parents (crossover) and mutations are implemented by reversing arbitrary subsequences in the permutation. After each round, the fitness of every offspring is evaluated. Less fit individuals are discarded, while more fit offsprings—in our case permutations—survive and can reproduce.

Brusco et al. [BKS08] and Wilkinson [Wil05] propose to use simulated annealing (depicted in Figure 12(b)), a technique similar to genetic algorithms and used for finding (local) optima in large search spaces. In each annealing step two rows or two columns are exchanged and the performance is measured in terms of anti-robinsion events, respectively residual variance.

Discussion

Heuristic approaches transform the matrix reordering problem, such that specific assumptions are met. While problem simplification algorithms are usually fast, they suffer inherently from this restriction. If a dataset is not of the expected form, the results will be inappropriate for an analysis (*A1*▩). One other problem seems to be specific for problem space transformation: The algorithms are reported to converge slowly and are sensitive to parameter settings. Also it is questionable whether general settings can be derived or inherently depend on the structure and size of the data sets. Particularly, in these cases, it might be beneficial to (pre-)assess the matrix in terms of density, clusteredness, etc.

8. Graph-Theoretic Approaches

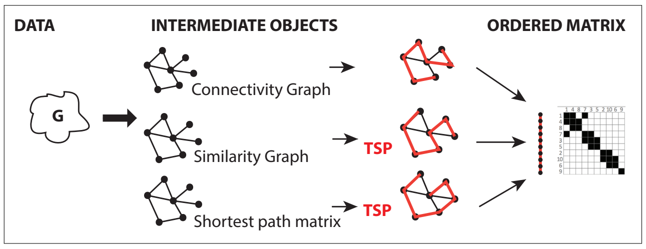


Figure 14: Graph-Theoretic Approaches for Matrix Reordering.

Graph-based approaches share a commonality with heuristic methods: They transform the permutation problem into a related problem space, in this case graph enumeration. The central idea of graph-theoretic approaches is to exploit the graph structure for computing a linear order that optimizes a graph-theoretic layout cost function. Díaz et al. [DPS02] compiled a list of nine layout cost functions from which three objectives have been applied in the context of matrix reordering. We detailed these layout cost functions, along with their visual interpretation in Section 2.2.

Algorithms and Variations

(a) Bandwidth Minimization: In an early approach (1968) Rosen presented an *iterative* method to reduce the bandwidth in sparse matrices [Ros68]. The same central objective is shared by the well-established *(Reverse) Cuthill-McKee* matrix reordering algorithms [CM69, Geo71]. Cuthill and McKee exploit a direct correspondence between the structure of the coefficient matrix and the structure of the adjacency matrix to be ordered. Algorithm 7 shows the pseudo code for the Reverse Cuthill-McKee algorithm. Starting from a graph vertex with low degree, it enumerates, in a *breadth-first search* manner, all neighboring vertices sorted by their neighborhood degree (number of common neighbors with the initial vertex). Figure 16(a) shows exemplified matrix plots, which depict that this inherently fast approach tends to produce strong bandwidth anti-patterns (*A2*▧). However, it can also lead to good results (*P1*◪) within the bandwidth if the graph structure allows for it and—more crucial—the initial input permutation is appropriate.

Algorithm 7 Bandwidth Minimization with Breadth-First Search in the Cuthill-McKee Matrix Reordering Algorithm [CM69].

```
procedure CUTHILL-MCKEE MATRIX REORDERING
    $G(V,E) \leftarrow adjacencyMatrix(M)$.
    $v_{start} \leftarrow minDegree(V)$.
    $\pi \leftarrow \emptyset \cup v_{start}$.
    $i \leftarrow 1$.                                   ▷ Initialization
    repeat
        $neighbors \leftarrow adjacent(v_i)$.
        $sortByDegree(neighbors)$.
        for all $v_n \in neighbors$ do
            $\pi \leftarrow \pi \cup v_n$.
        end for
        $i \leftarrow i+1$.
    until $i = |V|$.                                    ▷ Breadth-first enumeration
    $M \leftarrow applyPermutation(\pi, M)$.            ▷ Final Matrix Permutation
end procedure
```

An improved version of the Cuthill-McKee algorithm, known as Reverse Cuthill-McKee algorithm, is proposed by George [Geo71]. It reverses the degree ordering of the neighboring vertices in the breadth-first search. A comparative analysis of the two variants shows that this –marginal– change leads to better reordering results [LS76, p. 207]. Figure 16(a)(b) show exemplified matrix plots for both algorithm variants However, in our implementations the algorithms produce visually dissimilar results. Chan and George show in [CG80] a linear time implementation of the Reverse Cuthill-McKee algorithm.

The memory consumption for the Cuthill-McKee algorithm was improved by King [Kin70]. King uses a local priority queue to select the next vertex to visit, based on the number of vertices that will be added to the neighborhood list in the subsequent iteration. Later in 1976, Gibbs, Poole and Stockmeyer focused on runtime improvements in their popular GPS algorithm [GPS76]. GPS decreases the search space by starting with a vertex that has a maximal distance to another vertex (pseudo-diameter path) and a level minimization step to reduce the number of vertex enumeration cycles. The GPS algorithm is reported to work up to eight times faster than the Reverse Cuthill-McKee algorithm.

(b) Anti-bandwidth Maximization: A related and visually interesting adaption of bandwidth minimization was introduced by Leung in 1984: the matrix *anti-bandwidth maximization* problem [LVW84]. It says that after a matrix's row/column permutation all nonzero entries should be located as far as possible to the main diagonal. Figure 15 shows the exemplified result of a matrix reordering with respect to anti-bandwidth optimization.

Anti-bandwidth maximization is able to show off-diagonal line patterns (a sub-form of the bands pattern, *P4*▧, describing paths) spreading over the matrix plot. Similar line patterns can be found in the analysis of high performance computing clusters [vRHB*15].

Alike bandwidth minimization, also anti-bandwidth maximization, is in the class of $\mathcal{NP}$-complete problems [LVW84]. Accordingly, heuristic approaches were developed to solve the problem. Lozano et al. [LDGM12] propose a heuristic algorithm based on variable neighborhood search. Also Lozano et al. [LDGM13] proposed a hybrid approach combining the artificial bee colony methodology with tabu search to obtain appropriate results in short computational times. A genetic algorithm for bandwidth reduction, anti-bandwidth maximization and linear ordering is proposed in [PM14], where an exchangeable cost function guides to the expected result. Further discussion on anti-bandwidth maximization is given by Raspaud et al. in [RSS*09].

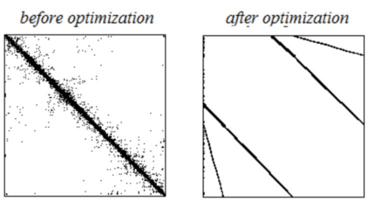


Figure 15: An example for the antibandwidth optimization [MS14, image courtesy].

(c) Profile Minimization: Sloan's algorithm [Slo86, Slo89] has the goal to reduce the profile and the wavefront of a graph by reordering the indices assigned to each vertex. Similarly to the GPS algorithm pseudo-peripheral nodes are chosen as a start and end vertices. All other vertices are prioritized by a weighted sum of the distance of the vector to the end vertex (global criterion). Additionally, a local criterion is incorporated with the current vertex degree. It reflects the status of the renumbering in the neighborhood of a vertex. Therefore, the Sloan algorithm not only takes into account the global criterion, but also incorporates local criteria for the reordering process. Figure 16(c) shows exemplified matrix plots for the *Sloan* reordering algorithm.

(d) Minimum Linear Arrangement: Koren and Harel propose a multi-scale approach to deal with the MinLA problem [KCH02]. In their multi-level algorithm (depicted in Figure 16(d)) the entire graph is progressively divided into lower dimensional problems (reordering of a segment graph). This process is referred to as the coarsening. The coarsening of the graph is based on restricting the consecutive vertex pairs of the current arrangement. In the coarsest level exact solutions for the problem can be calculated. These sub-solutions are projected back to the higher level problem in the subsequent refinement process until the initial problem is reconstructed. This refinement step iterates over all local permutations and selects the one that minimizes the MinLA (in a dynamic programming fashion). Multi-level approaches in general have one significant advantage: They allow fast exploration of properties related to the global structure, while the local structures can be refined iteratively if necessary. Further multi-scale graph coarsening approaches are described in [OKLS15].

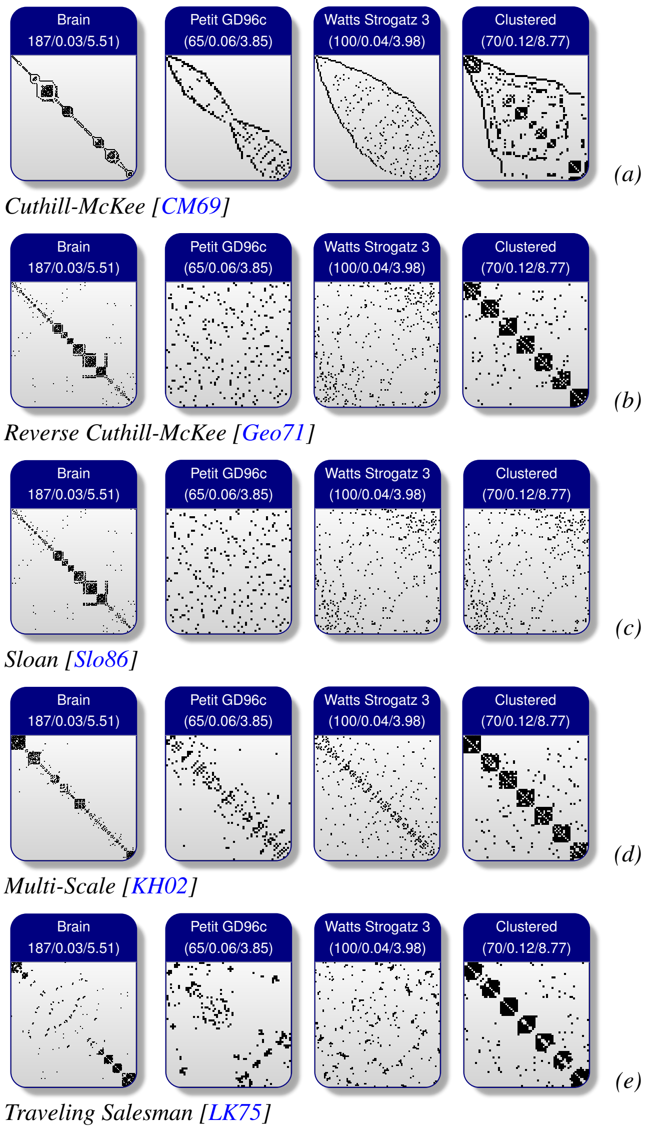


Figure 16: Examples for Graph-based Reordering Approaches.

(e) Traveling Salesman Problem: In a technical note from 1974 Lenstra pointed out that the Bond Energy Algorithm (cf. Section 7) could be modeled as two subsequent Traveling Salesman Problems (TSP) for the rows and the columns: "*It performs well for this type of problem*" [Len74, p. 414]. Shortly after, Lenstra and Kan [LK75, p. 724] showed that the bond energy heuristic is a simple suboptimal TSP variant and compared it to the optimal TSP solution.

TSP matrix reordering approaches model each row, respectively column, as a city and translate the row/column-wise similarity into virtual distances. Computing an optimal TSP path (a minimum distance TSP tour with a virtual city that is used for breaking the cycle) corresponds then in finding an optimal matrix permutation, such that the pairwise similarity is maximized (wrt. the chosen similarity function).

Figure 16(e) shows matrix plots for the *RSeriationTSP* algorithm. Internally the *Concorde* TSP solver [ACR03] and the 2-OPT edge exchange improvement procedure [Cro58] is used to minimize the Hamiltonian path length. A different approach is pursued by the *Multiple Fragment* algorithm, such as described by Bentley [Ben92] or Steiglitz and Weiner [SW68]. It starts by considering every node in the graph as one independent fragment. Repeatedly, any edge that will not make it impossible to complete a tour is added to the closest fragment. Fragments can be merged by a connecting edge.

More recently, Henry-Riche and Fekete [HF06] incorporated in their MatrixExplorer system the consideration that vertices with *similar connection patterns* should be positioned next to each other. This has the advantage that not only the coarse matrix plot structure (groups/cluster patterns) is focused, but also that the local density of the occurring clusters is optimized. In their approach the authors use—instead of the adjacency matrix—the shortest path matrix for each connected component of the graph and reorder each component with a TSP solver; alternatively a hierarchical clustering can be applied (see also Section 4).

Discussion

The idea to explore the graph structure for computing a linear ordering is self-evident and obvious. But, in analogy to our question *"What is a good matrix reordering?"* the graph community is posing the question *"What is a good 2D graph layout?"*. These questions are yet unanswered in both domains. However, they share the common ground that a good result allows perceiving interpretable visual patterns.

Related to this challenge, several of the mentioned approaches, such as the Multi-Scale or TSP, have the interesting characteristic that a consistent and *intermediate* reordering result can be shown to the analyst on-the-fly, while the actual matrix reordering takes place. This idea follows the "*results-first-refine-on-demand*" mantra. On the other hand, when it comes to the optimization of graph-theoretic layout functions, such as bandwidth or profile, these algorithms are solely governed by the assumption that the data can be brought into this form. This assumption implicitly neglects all other potential patterns in a matrix plot. Though, a matrix plot that represents patterns along the diagonal well (e.g., clusters) will be perceived as interpretable and effective; a goal also expressed by Bertin [Ber73].

The efficiency aspect has to be regarded, as well. Sloan notes, that bandwidth and profile reduction "schemes may be inefficient for sparse matrices which contain a significant number of zeros inside the bandwidth envelope" [Slo86, p. 240]. Recently, Wong also noted that "algorithms such as Fiedler and Sloan consistently require more time to compute than the others when the graphs grow to tens of thousands of nodes" [WMFM13, p. 95]. While this is certainly true, a heuristic implementation for most approaches can be found in the vast literature of the graph-theoric domain.

9. Biclustering Approaches

Recently, the concept of *Biclustering*, also called co- or two-mode clustering or generalized blockmodeling, gained importance for

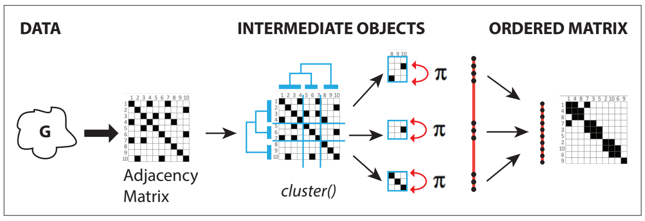


Figure 17: Biclustering Approaches for Matrix Reordering.

matrix reordering. Biclustering is related to clustering, but comprises a central difference: While clustering can be applied separately to either the rows or columns of the matrix, biclustering performs clustering in these two dimensions *simultaneously* (Figure 17). In other words, clustering derives *global data models* and biclustering algorithms allow identifying subsets of rows and columns that reveal a similar activity pattern, and thus have to be seen as *local data models*.

Biclustering approaches can be subdivided by the clustering structure they are able to reveal: Single bicluster, exclusive row and column biclusters, checkerboard structure, exclusive rows biclusters, exclusive columns biclusters, nonoverlapping biclusters with tree structure, nonoverlapping nonexclusive biclusters, overlapping biclusters with hierarchical structure, and arbitrarily positioned overlapping biclusters [MO04, p. 34]. For a matrix reordering task the most general subgroup with arbitrarily positioned overlapping biclusters is of the highest interest, since it enables the user to see overlapping block patterns in the matrix plot. A repetitive execution of algorithms that reveals a single bicluster allows finding arbitrarily positioned submatrices, too.

Algorithms and Variations

(a) Single Bicluster Approaches: Cheng and Church [CC00] present a greedy iterative search algorithm that models the biclustering problem as an optimization problem. The algorithm finds one $\delta-$bicluster (or submatrix) at a time with a potentially local optimal mean squared residue score not higher than a user-specified parameter δ.

The pseudo code for the Cheng and Church algorithm is given in Algorithm 8, depicting two greedy row/column removal/addition steps that are executed subsequently to find one $\delta-$bicluster: Starting from the full matrix, (i) rows and columns with a score higher than the mean squared residue score are deleted; (ii) removed rows or columns are added if they do not increase the actual mean squared residue score of the bicluster. This approach converges with low mean residue and locally maximal size for one $\delta-$cluster. In order to find distinct biclusters, an already identified bicluster can be artificially masked with random noise so that a subsequent invocation of the algorithm finds a different $\delta-$bicluster.

(b) Arbitrarily Positioned Overlapping Biclusters Several algorithms to retrieve arbitrarily positioned overlapping biclusters exist. For example, the *Plaid* model biclustering algorithm of Lazzeroni and Owen [LO*02] assumes that a matrix can be described as a linear function of possibly overlapping constant layers that do not necessarily have to cover the whole matrix. Iteratively new layers

Algorithm 8 Cheng-and-Church Biclustering algorithm [CC00].

```
1: function CHENG AND CHURCH BICLUSTERING
2:     $A \leftarrow rows(M)$.                                        ▷ Definitions.
3:     $B \leftarrow columns(M)$.
4:     $e_{Aj} \leftarrow \frac{\sum_{i \in A} m_{ij}}{|A|}$.
5:     $e_{iB} \leftarrow \frac{\sum_{j \in B} m_{ij}}{|B|}$.
6:     $e_{AB} \leftarrow \frac{\sum_{i \in A, j \in B} m_{ij}}{|A||B|}$.
7:     $RS_{AB}(i,j) \leftarrow m_{ij} - e_{Aj} - e_{iB} + e_{AB}$.
8:     $H(I,J) \leftarrow \sum_{i \in A, j \in B} \frac{RS_{ij}^2}{|A||B|}$.
9:     Initialize $bicluster(I,J)$ with $I = A, J = B$.               ▷ Algorithm Start.
10:    while $H(I,J) > \delta$ do                                     ▷ Deletion Phase.
11:        $d(i) \leftarrow \frac{1}{|J|} \sum_{j \in J} RS_{IJ}(i,j)$ for all $i \in I$.
12:        $e(j) \leftarrow \frac{1}{|I|} \sum_{i \in I} RS_{IJ}(i,j)$ for all $j \in J$.
13:        if $max_{i \in I} d(i) > max_{j \in J} e(j)$ then
14:            $I \leftarrow I \setminus \{\arg\max_i d(i)\}$.
15:        else
16:            $J \leftarrow J \setminus \{\arg\max_j e(j)\}$.
17:        end if
18:    end while
19:    $I' \leftarrow I, J' \leftarrow J$.
20:    while $H(I',J') < \delta$ do                                   ▷ Addition Phase.
21:        $I \leftarrow I', J \leftarrow J'$
22:        $d(i) \leftarrow \frac{1}{|J|} \sum_{j \in J} RS_{IJ}(i,j)$ for all $i \in A \setminus I$.
23:        $e(j) \leftarrow \frac{1}{|I|} \sum_{i \in I} RS_{IJ}(i,j)$ for all $j \in B \setminus J$.
24:        if $max_{i \in I} d(i) > max_{j \in J} e(j)$ then
25:            $I' \leftarrow I \cup \{\arg\max_i d(i)\}$.
26:        else
27:            $J' \leftarrow J \cup \{\arg\max_j e(j)\}$.
28:        end if
29:    end while                                                      ▷ Initialization
30:    return $bicluster\ I, J$.
31: end function
```

are added to the model, so that the new layer minimizes the sum of squared errors. The Plaid model biclustering algorithm was improved by Turner et al. [TBK05], using a binary least squares algorithm to update the cluster membership parameters. This takes advantage of the binary constraints on these parameters and allows to simplify the other parameter updates.

(c) Biclusters with similar Visual Patterns: Another approach, called *xMotifs*, is presented by Murali and Kasif [MK03] for the biological gene expression analysis domain. The authors state that "*a conserved gene expression motif or xMotif is a subset of genes whose expression is simultaneously conserved for a subset of samples.* [...] *If we map each gene to a dimension, each sample to a point, and each expression value to a coordinate value, an xMotif is identical to a multi-dimensional hyperrectangle that is bounded in the dimensions corresponding to the conserved genes in the motif and unbounded in the other dimensions.*" [MK03, p. 2]. In other words, the xMotif algorithm searches for biclusters that share a common visual pattern or motif. To achieve this goal, the data is discretized into statistically significant intervals—sometimes even binarized. In a probabilistic approach, randomly chosen "seed" columns are iteratively compared against growing sets of "discriminant" columns. For these discriminant columns, all rows are incorporated into the bicluster if they share a common state with the seed column at the specific position. Motifs with a low overlap coefficient are discarded. The algorithm is adapted for ordinal and nominal scales in the algorithms *Quest* and *Questmet* [Kai11]. A similar approach, called *BiMax* was presented by Prelić et al. [PBZ*06].

(d) A Priori Submatrices: An interesting biclustering variant is presented by Jin et al. [JXFD08]. Based on the assumption that a set of submatrices of interest is known a priori, the authors try to find a row/column permutation that allows showing these relationships best. For this purpose, the authors generalize the minimum linear arrangement problem into the *hypergraph vertex ordering* problem and propose to solve the submatrix pattern visualization problem in this problem domain. In their suggested algorithm existing graph ordering algorithms are incorporated to solve the optimization problem.

Several other biclustering algorithms exist and are discussed in [MO04, PBZ*06, TSS05]

Discussion

Biclustering focuses on finding subsets of rows and columns that allow perceiving coherent activity patterns which cannot be seen with a global scope. Biclustering approaches are therefore operating on *local models*. By definition the retrieved biclusters, or submatrices, should form nearly uniformly colored coherent visual block patterns (*P1*◪, *P2*◩) that stand out from the neutral background color. This ideal corresponds to the existence of k mutually exclusive and exhaustive clusters, and a corresponding k-way data partitioning [LO*02, p. 62].

Unlike standard cluster matrix reordering approaches (see also Section 4), biclustering approaches are not necessarily depending on a similarity model. In contrast, these approaches even doubt the rationale of an equal weighting of rows and/or columns. Cheng and Church state that any such similarity formula leads to the discovery of some similarity groups at the expense of obscuring some other similarity groups [CC00, p. 1].

Another central difference to the other reordering approaches is the notion of data partitioning. While standard approaches mostly facilitate a data separation into exclusive groups, biclustering approaches generalize this assumption: data items may be contained in several overlapping clusters. This understanding is based on the circumstances of the gene expression domain, where co-regulatory gene patterns are represented by (multiple) subsets of conditions.

Generally, it has to be noted that the approaches in this field are not restricted to a-priori constraints on the organization of biclusters, which allows for more freedom, but consequently leads to a higher vulnerability to overfitting. This is especially obvious in the *high parameter sensitivity*: In many cases only slight modifications of the input parameters lead to empty biclustering result sets and even erroneous matrix permutations. However, this problem is mitigated by the fact that most algorithms are greedy implementations, which allows a fast but potentially locally optimal result.

10. Performance Comparison

The previous sections grouped algorithms into six categories, described the underlying methods, and showed examples of resulting matrices. In this section we look at two measures to quantify "performance" of an algorithm: (i) algorithm runtime and (ii) Linear Arrangement score (LA), a measure for the compactness of blocks in matrices. A matrix, ordered by an algorithm, results in a *low* LA score if it reveals coherent blocks in the matrix (Figure 18 (left, center left)). In the opposite case, a matrix results in a *high* LA score if it is noisy (Figure 18 (center right, right)). Comparing both measures can inform the tradeoff between *fast* algorithms and *visually pleasing* reordering results.

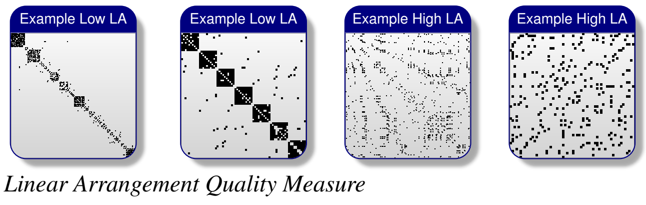


Figure 18: Examples of low/high scores for the Linear Arrangement quality criterion.

For our analysis, we obtained implementations of 35 algorithms, representative for the groups in Section 3. We used these algorithms to reorder matrices for 150 graphs, resulting in 4348 of 5250 total reordered matrices (35 ×150). The missing ones are erroneous results (e.g., not all row/column indices were returned or indexes occur multiple times), which we attribute to issues with parameters, especially problematic for Biclustering approaches (cf. Section 9).

For each trial (i.e., reordered matrix) we measured runtime and LA score, as well as captured the visual matrix in a picture. We provide online the browsable collection of all matrices and associated measures at `http://matrixreordering.dbvis.de`.

10.1. Design and Setup

Algorithms We selected 35 algorithms satisfying the following criteria: (i) well-known and used in practice, (ii) available and accessible implementation, and (iii) runtime complexity is reasonable given the tested datasets.

For example, we tested the two Branch-and-Bound algorithms `BBURCG` and `BBWRCG`, (R *seriation* package [BS05]), but opted to remove them due to their long runtime. Our experiments reveal them being impractical for graphs larger than 30 nodes. Table 1 gives an overview of the selected algorithms, the group they belong to (Section 3), and the source of their implementation.

Graphs To obtain a large and heterogeneous collection for graphs, we selected 150 graphs from three different sources and with varying characteristics (e.g., size, density, cluster coefficient). Interested readers can explore the relations and distributions within this multidimensional feature space of graph measures in an interactive dashboard on our website. In the following we present only a higher level overview of the selected graphs:

- 20 **real-world graphs** from the Pajek graph collection [BM98].
- 23 **test graphs** from the *Petit Testsuite* [Pet03], one of the primary benchmark suites used in the literature for comparing matrix reordering algorithms.

Family	**Name**	**Implementation**
Robinsonian	Hierarchical Cluster	Java [ESBB98]
(R)	Bipolarization	Java [Hub74]
	RSeriationGW	R *seriation* [HBH14]
	RSeriationBEA	R *seriation* [HBH14]
	RSeriationOLO	R *seriation* [HBH14]
	RSeriationHC	R *seriation* [HBH14]
Spectral	RCorrplotSortingAOE	R *corrplot* [Wei13]
(S)	RCorrplotSortingFPC	R *corrplot* [Wei13]
	RSeriationCHEN	R *seriation* [HBH14]
Heuristic	Row Sum (Asc)	
(H)	Row Sum (Desc)	
	Median Iteration	
	Mean Iteration	
	V-Cycle	
Dimension	RSeriationMDS	R *seriation* [HBH14]
Reduction (D)	RSeriationPCA	R *seriation* [HBH14]
Graph	Multiple-Fragment	Java [Ben92]
(G)	Multi Heuristic	
	Multi-Scale	Java [KH02]
	Cuthill-McKee	Java [CM69]
	Reverse Cuthill-McKee	Java [CM69]
	Degree (Ascending)	Java
	Local Refine	Java
	RSeriationTSP	R *seriation* [HBH14]
	RSeriationBEATSP	R *seriation* [HBH14]
	Sloan	C++ Boost [boo]
	King	C++ Boost [boo]
Bi-Clustering	RBiclusteringBCPlaid	R *biclust* [KL08]
(B)	RBiclusteringBCBimax	R *biclust* [KL08]
	RBiclusteringBCQuest	R *biclust* [KL08]
	RBiclusteringBCQuestmet	R *biclust* [KL08]
	RBiclusteringBCQuestord	R *biclust* [KL08]
	RBiclusteringBCBimax	R *biclust* [KL08]
	RBiclusteringBCrepBimax	R *biclust* [KL08]
	RBiclusteringBCSpectral	R *biclust* [KL08]

Table 1: Overview of tested matrix reordering implementations. The table shows (i) the algorithm group according to our taxonomy, (ii) the internal identifier and (iii) the implementation source or respective publication for our Java implementations.

- 107 **random graphs** generated to control for graph characteristics such as size, density, and number of clusters. We generated graphs using the random graph generators in NetworkX [Net]. We tested the following types of graphs: bipartite graphs, clustered graphs, graphs with small-world graphs (Watts-Strogatz), and graphs with power-law degree distribution (Albert-Barabasi).

For this analysis, we categorized graphs according to two measures: *size*: (i) *small* (25-100 nodes), (ii) *large* (100-1500 nodes) and *density*: (i) *sparse* (density of 0.05 - 0.28), (ii) *dense* (density of 0.28-0.6).

Setup We generated all trials on an Intel Core i7-2600 Quadcore (3.4 GHz) PC with 16 GB RAM (DDR3-PC3-10600) and a 120 GB SSD. The PC is operated by Windows 7 Enterprise Edition and runs R in the version 3.1.2, Java SE 7. We excluded transfer and preprocessing times from the measured execution time.

We conducted the computation with a Java program. We included 19 implementations from the R packages *'corrplot'* [Wei13], *'biclust'* [KL08] and *'seriation'* [HHB08]. For these R packages we used the R-Proxy implementation from Nuiton[†] to send and receive information from Java to and from an R server. We invoked two algorithms, taken from the C++ Boost library [SLL01], via the Java Native Interface (JNA). We implemented in Java several algorithms for which we could not find any reference implementation.

10.2. Results and Findings

Runtime Figure 19 shows runtime statistics in milliseconds for each of the four graph groups: *small-sparse, small-dense, large-sparse*, and *large-dense*.

Graph-theoretic algorithms (e.g., *Cuthill-McKee*, *King*, *Sloan*, *Multi-Scale*) and some Robinsonian algorithms (e.g., *Bipolarization*, *Multi-Heuristic*, *Hierarchical Clustering*) are returning reordering results mostly below 1000 *msec*. More interestingly, these algorithms are nearly independent from the graph topology. For example, it appears that the runtime is not influenced by variation in the degree of clustering of the graph.

R Seriation and *Biclustering* algorithms, independent from their algorithm family, tend to perform slower than graph-theoretic algorithms. This could be due to (i) particular sophisticated implementations, and/or (ii) the data structures used. However, the R *corrplot* package is as fast as the fastest algorithms, which makes it unlikely that the used data structure (access times on the row- or column vectors and random cell access) has a significant impact on the overall calculation time.

Runtime for large graphs (*large*) are still < 3000 *msec*. An exception are *RSeriationBEA* and *RSeriationARSA* with a runtime about $144{,}000$ *msec* and $24{,}000$ *msec* respectively, on the "c4y" real-world Integrated Circuit (IC) network with 1366 nodes and 2915 edges.

Linear Arrangement Linear Arrangement (LA) is a loss function that refers to the minimum linear arrangement problem (e.g., described in [KCH02]). As Figure 18 depicts, it allows assessing the visual quality of a matrix plot: The exemplified low LA scores refer to the block pattern (*P1*◪ and *P2*◩), while high scores prove to be valid indicators for noisy plots (*A1*▨).

Figure 20 depicts boxplots for our Linear Arrangement experiments under varying topology aspects: We can see that sparse graphs lead to a consistent high median LA score; however the mean scores (dotted line) and the prominent Whisker lines indicate the strong variance within the data. Noteworthy algorithms are the Reverse Cuthill-McKee and the Sloan algorithm (both graph-related algorithms), which tend to produce consistently either low scores or end up with noisy visual matrices. In the taxonomy group of large and dense graphs (Figure 20(d)) we can derive similar tendencies: graph-related measures often outperform Robinsonian, Spectral and BiClustering methods.

[†] `https://nuiton.org/projects/nuiton-j2r/`

11. Discussion and Research Directions

The question: "What is a good matrix reordering?" does not have a unique answer. However, we consider a "bad" ordering, one that fails to reveal patters such as those described in Section 2.3, when they actually are present in the data (e.g., clusters, hubs, bi-graphs).

Our empirical experience tends to indicate that a higher visual quality requires more sophisticated approaches and thus, more time to calculate. This may prove problematic in scenarios where providing rapid feedback is more crucial than displaying the highest quality result. In fact, there are many trade offs pertaining to the selection of a reordering algorithm. In this section, we discuss strategies to select an algorithm, explain how several of them can be parameterized and briefly discuss interactive approaches to introduce the human in the loop. We conclude by discussing the limitations of our survey and outline directions for future work.

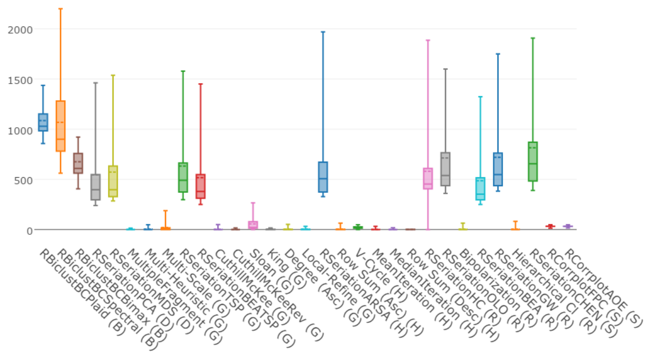
11.1. Selecting a Matrix Reordering Algorithm

While ideally one would provide specific guidance on which algorithm to select and which parameter settings to use with respect to data and tasks, there are too many open research questions remaining to provide formal and robust guidelines at this point. Instead, we provide several insights on which specific matrix reordering algorithm and parameter setting proves to be effective, based on our observations in Section 3, the analysis in Section 10 and our own empirical knowledge gathered by applying reordering algorithms in domain-specific applications.

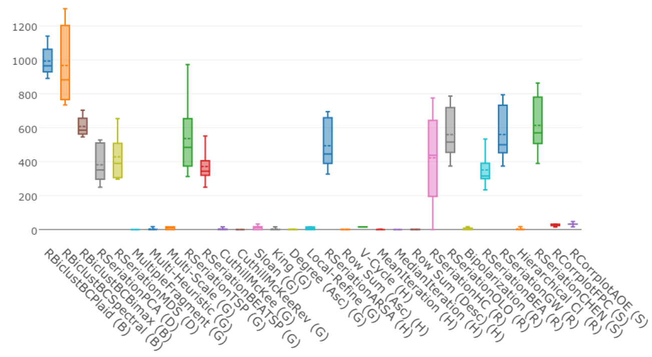
Fast algorithms first—Fast reordering approaches can produce results in sub-second runtime when the data structure matches characteristics that this algorithm is optimizing for. Others are robust to certain properties of the data, making them practical to at least try first. For example, *Cuthill-McKee* (see: Section 8) or *RCorrplot-SortingAOE* (see: Section 5) algorithms produce results at near-interactive processing rates, almost independently of the matrix density. However, results are tailored for a specific data structure and, when not present, these algorithms often produce anti-patterns (*A1*▨) or depict calculation artifacts (*A2*⧅), such as described in Section 2.3. From our empirical experimentations, we note that if a fast algorithm reveals desired patterns, a more sophisticated one is unlikely to improve on its quality significantly.

Heuristic algorithms offer tradeoffs—Heuristic approaches offer a good tradeoff between runtime and quality. They often produce more patterns than very fast algorithms, and improve them via multiple iterations. The freedom to interrupting them after a certain number of iterations enables to strike the balance between runtime and (potentially measurable) quality. For example, the *Barycenter* or *RSeriationARSA* algorithm (cf. Section 7) can be stopped in early iterations, but generally require more to produce higher quality results. These algorithms tend to first reveal data partitioning (e.g., connected components, sub-networks), but patterns within these partitions require more iterations.

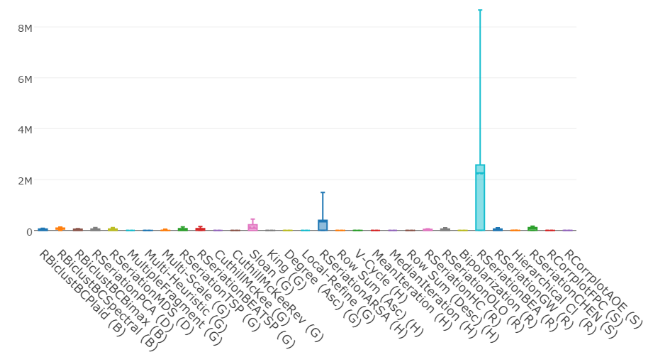
Optimizing for cluster identification—For scenarios where identifying clusters is essential, we recommend selecting (hierarchical) clustering approaches, since they explicitly detect clusters and order each one individually, placing them at the matrix diagonal. A good example is *Hierarchical Clustering* (cf. Section 4).



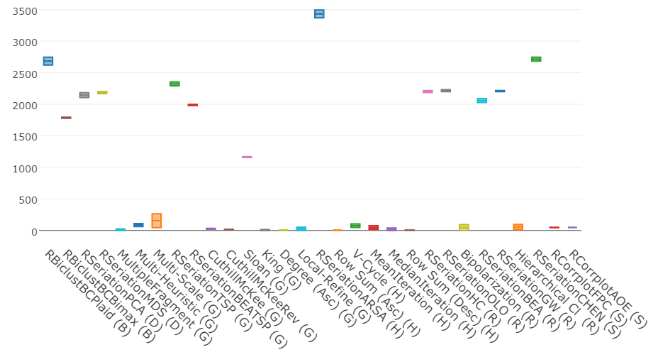
(a) Small Sparse Graphs



(b) Small Dense Graphs

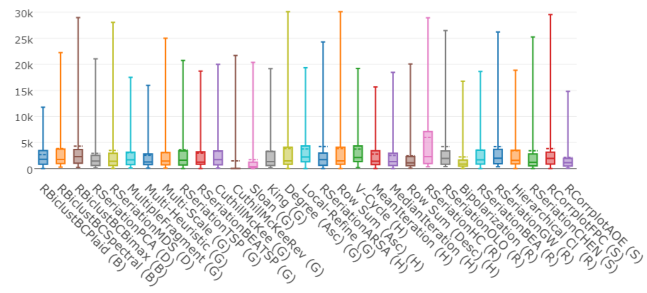


(c) Large Sparse Graphs

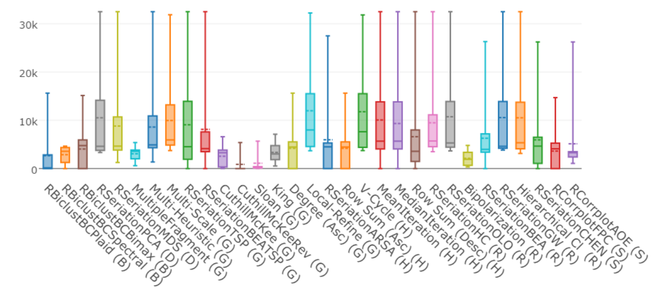


(d) Large Dense Graphs

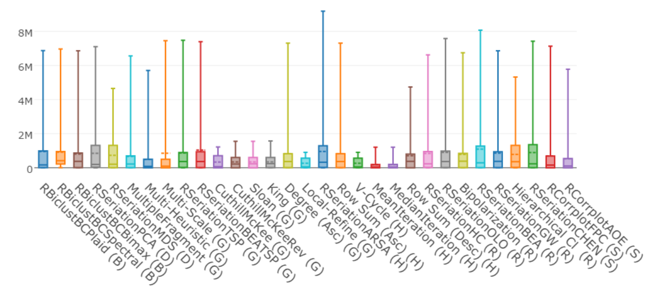
Figure 19: Calculation time (in msec) for each graph category (small/large versus sparse/dense).



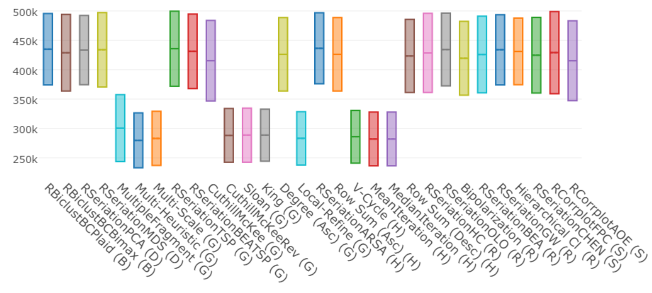
(a) Small Sparse Graphs



(b) Small Dense Graphs



(c) Large Sparse Graphs



(d) Large Dense Graphs

Figure 20: Linear arrangement Scores for each graph category (small/large versus sparse/dense).

Alternatively spectral algorithms also highlight clusters in the data set since similarly connected nodes are located close in Eigenspace. A good example is *RSeriationChen* (cf. Section 5).

Quality takes time—If the previous algorithms fail to reveal patterns, or if the patterns produced are not matching the tasks and scenario, one may need to compromise on runtime and opt instead for remaining algorithms. Through our experimentations, we observed that Optimal-Leaf-Ordering (cf. Section 4) tends to produce visually coherent and well organized block-diagonal forms. Note that we consider higher visual quality when local (sub-)structures and patterns are discernable in the matrix plot. As a direct consequence, algorithms attempting to optimize functions on the entire data may not be able to grasp these phenomena. For example algorithms such as Biclustering approaches often produce visually interpretable (sub-)structures if, and only if, appropriate parameters pertaining to the clusters are set. Another example are the Traveling Salesmen algorithms (cf. Section 8), considering distances between each pair of vertices, which often reveal local patterns (e.g., cliques or hubs) but may fail to optimize the general matrix bandwidth.

11.2. Opportunities and Future Directions

While this document describes existing solutions to reorder matrices, there are still many opportunities to improve these solutions and provide better ones. We list here some possible future work and pitfalls of our approach.

Global vs Local Algorithms vary on their strategy to explore the search space: top-down or bottom-up. Top-down approaches focus on optimizing a global structure metric (e.g., *Multi-Scale*, Section 8), while bottom-up approaches may retrieve local graph structures (e.g., *Bipolarization*, Section 4). The strategy has a direct impact on the visual results. The majority of algorithms proposed in this article are bottom-up approaches.

Hybrid approaches are an interesting future direction, where retrieving global structures is in the focus of the first iterations, and further (potentially different) algorithms can be applied to subnetworks in later iterations. Another interesting research direction relates to multi-scale approaches which could allow a user to retrieve results at different scales of interest (e.g., entire data, connected component, sub-network).

Similarity/Distance Calculation An important parameter for reordering algorithms, especially crucial for Robinsonian algorithms (cf. Section 4), is the choice of the distance metric between nodes (rows and columns). However, there is no simple method to choose a good measure given a specific graph or task. Gelfand [Gel71] notes the importance of these considerations and describes three exemplary similarity functions, which should be applied with respect to the different data domains $[0,1]$, $[true, false]$, $(-\infty,\infty)$.

Alternatively, domain-specific considerations can be included into the distance calculations, such as in Eisen et al. [ESBB98]: gene offset shifts over a log-scaled correlation coefficient are applied to analyze gene similarity. Behrisch et al. [BKSK12] calculate text similarity on news articles and present the pair-wise comparisons in a matrix format.

More sophisticated techniques, such as the earth movers distance, or statistically inspired distance considerations (i.e., Jenson-Shannon Divergence or $\mathcal{X}^2$) cannot be found in the current literature for matrix reordering. Gregor et al. [GLS*15] recently made an attempt at empirically deriving the impact of the respective distance functions for visual tasks. They found that for feature retrieval tasks, the Manhattan Distance is a robust choice and outperforms Jensen-Shannon Divergence and even the Euclidean Distance.

Visual Pattern Assessment Several approaches are tailored to produce block-diagonal patterns (*P1*◪). Unfortunately, if the data does not contain such patterns, these algorithms mostly fail to reveal any pattern. More work is required to design algorithms that focus on different other patterns. A crucial research direction is to develop quantitative measures to evaluate the quality of these patterns and thus craft objective functions to optimize or assess the algorithms' performance.

Human-Assisted Reordering While automatic reordering algorithms ideally take the burden off the user while producing high quality results, it may not happen often in practice. To address shortcomings of certain algorithms and enable the user to steer algorithms by making decisions at critical points, interactive reordering techniques started to appear. We point to several examples in this section, however, note that a complete review of these techniques is out of scope of our survey.

We can broadly categorize interactive reordering techniques in two categories: interactive and semi-assisted (steering). Bertifier [PDF14], TableLens [RC94] and InfoZoom [SBB96] are examples of interactive techniques, providing a user interface in which users can manually reorder rows and columns. Compare to Bertin's initial physical device that enable users to move a single row or column at a time [Ber81], strategies used in these pieces of software provide grouping and aggregation mechanisms that enable to move sets or rows and columns at a time, decreasing the labor to reorder a matrix. In addition, Bertifier provides primitives to let users change the visual style of the resulting visual matrices, an important step towards communication of the results to an audience.

On the other hand, MatrixExplorer [HF06], Select Objective Measures [BW13], or PermutMatrix [CP05] provide semi-automatic approaches and enable the user to steer or guide algorithms when reordering a matrix. For example, MatrixExplorer enable the user to select subsections of the matrix to reorder in addition to interactive reordering of individual rows or columns. PermutMatrix provides a suite of features to enable users to apply different reordering algorithms and a set of primitives to identify, select and manipulate substructures of the matrix (e.g., clusters, cliques or sub-trees).

While these techniques can address shortcomings of certain algorithms and leverage human and computer skills, these systems are still in their infancy and rather rare in practice. We believe some of the most exciting advances for matrix reordering will occur in this space, as our research community crafts visual analytics systems that leverage user interaction and automatic algorithms.

11.3. Limitations

This survey presents a categorization of matrix reordering approaches into six reordering families, sharing similar reordering concepts. Our goal is to provide a central document where concepts from multiple disciplines are defined and related, algorithms grouped in categories and discussed in contrast to each other, and, finally, examples of visual results provided systematically.

While we discussed at length several alternatives to our present taxonomy, possibly able to better capture nuances between different approaches, we finally opted to provide a categorization of reordering algorithms as simple as possible. We believe matrix reordering algorithms are a fundamental barrier for the use of matrices in practice today. By providing a straightforward classification and formulating mechanisms and approaches in simple terms, we hope to help a wide audience better understand these algorithms and integrate them in future systems and libraries.

While we gave insights in discussion on how to select algorithms for certain data characteristics or specific tasks (e.g., identifying clusters), matching systematically algorithms and tasks (for example the tasks described by Lee et al. in [LPP*06]) is extremely challenging. We decided against attempting to describe this matching formally as there are still many unknowns and doing so would require a substantial amount of future work. In particular, we do not think this is possible without developing measures to assess and quantify visual patterns produced reliably.

12. Conclusion

Matrix reordering algorithms are essential to make patterns in matrices visible. While previous surveys on algorithms centered on historical- and domain-related aspects, this present work aims at providing a comprehensive overview and guidance for selecting algorithms according to their performance and ability to reveal visual patterns. We collected 35 reordering algorithms from different disciplines and organized them in six families. We explained and related major concepts from different disciplines to enable a wider audience to understand the approaches and underlying mechanisms of these reordering algorithms.

From our observations and experiments with these algorithms, we created an online repository of over 4500 visual matrices, and included practical guidance for selecting and comparing them. Our general goal with this survey is to lower the barrier for using matrices in visual exploration systems and libraries across many disciplines. By gathering the knowledge in a central document, we also hope to inspire more research to develop novel strategies to reorder matrices, novel approaches to assess the quality of their results, as well as to develop high-quality libraries to reorder matrices for improving their visualization and exploration.

References

[ABH98] Atkins J., Boman E., Hendrickson B.: A spectral algorithm for seriation and the consecutive ones problem. *SIAM Journal on Computing 28*, 1 (1998), 297–310. 5, 7, 9, 10

[ABHR*13] Alper B., Bach B., Henry Riche N., Isenberg T., Fekete J.-D.: Weighted graph comparison techniques for brain connectivity analysis. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (New York, NY, USA, 2013), CHI '13, ACM, pp. 483–492. 1

[ACR03] Applegate D., Cook W., Rohe A.: Chained lin-kernighan for large traveling salesman problems. *INFORMS Journal on Computing 15*, 1 (2003), 82–92. 15

[And09] Andrecut M.: Parallel gpu implementation of iterative pca algorithms. *Journal of Computational Biology 16*, 11 (2009), 1593–1599. 10

[BBA75] Breiger R. L., Boorman S. A., Arabie P.: An algorithm for clustering relational data with applications to social network analysis and comparison with multidimensional scaling. *Journal of mathematical psychology 12*, 3 (1975), 328–383. 5, 9

[Ben92] Bentley J. J.: Fast algorithms for geometric traveling salesman problems. *ORSA Journal on computing 4*, 4 (1992), 387–411. 5, 15, 17

[Ber73] Bertin J.: *Sémiologie Graphique - Les diagrammes, les reseaux, les cartes*. Éditions Gauthier-Villars, Paris, 1973. 1, 4, 5, 15

[Ber81] Bertin J.: Théorie matricielle de la graphique. *Communication et langages 48*, 1 (1981), 62–74. 4, 5, 20

[BJGJ01] Bar-Joseph Z., Gifford D. K., Jaakkola T. S.: Fast optimal leaf ordering for hierarchical clustering. *Bioinformatics 17*, suppl 1 (2001), S22–S29. 5, 7, 8

[BKS08] Brusco M. J., Köhn H.-F., Stahl S.: Heuristic implementation of dynamic programming for matrix permutation problems in combinatorial data analysis. *Psychometrika 73*, 3 (2008), 503–522. 5, 7, 12, 13

[BKSK12] Behrisch M., Krstajic M., Schreck T., Keim D. A.: The News Auditor: Visual Exploration of Clusters of Stories. In *Proc. EuroVA International Workshop on Visual Analytics* (2012), Eurographics, pp. 61–65. 5, 20

[BL12] Borg I., Lingoes J.: *Multidimensional similarity structure analysis*. Springer Science & Business Media, 2012. 11

[BM98] Batagelj V., Mrvar A.: Pajek-program for large network analysis. *Connections 21*, 2 (1998), 47–57. 17

[boo] : *BOOST C++ Library:*. `http://www.boost.org`. 17

[BP07] Brandes U., Pich C.: Eigensolver methods for progressive multidimensional scaling of large data. In *Graph Drawing* (2007), Springer, pp. 42–53. 11

[Bra07] Brandes U.: Optimal leaf ordering of complete binary trees. *Journ. of Discrete Algorithms 5*, 3 (2007), 546 – 552. 5, 8

[BS05] Brusco M. J., Stahl S.: Optimal least-squares unidimensional scaling: Improved branch-and-bound procedures and comparison to dynamic programming. *Psychometrika 70*, 2 (2005), 253–270. 5, 7, 17

[BS06] Brusco M., Stahl S.: *Branch-and-Bound Applications in Combinatorial Data Analysis*. Statistics and Computing. Springer, 2006. 5

[BW13] Brakel R. B. J. V., Westenberg M. A.: COMBat : Visualizing Co-Occurrence of Annotation Terms. 17–24. 5, 20

[CC00] Cheng Y., Church G. M.: Biclustering of expression data. In *Ismb* (2000), vol. 8, pp. 93–103. 5, 15, 16

[CG80] Chan W.-M., George A.: A linear time implementation of the reverse cuthill-mckee algorithm. *BIT Numerical Mathematics 20*, 1 (1980), 8–14. 5, 13

[Che02] Chen C. H.: Generalized association plots: information visualization via iteratively generated correlation matrices. *Statistica Sinica 12* (2002), 7–29. 5, 9

[CM69] Cuthill E., McKee J.: Reducing the bandwidth of sparse symmetric matrices. In *Proceedings of the 1969 24th National Conference* (New York, NY, USA, 1969), ACM '69, ACM, pp. 157–172. 5, 13, 14, 17

[CP05] Caraux G., Pinloche S.: Permutmatrix: a graphical environment to arrange gene expression profiles in optimal linear order. *Bioinformatics 21*, 7 (2005), 1280–1281. 5, 7, 20

[Cro58] Croes G. A.: A method for solving traveling-salesman problems. *Operations research 6*, 6 (1958), 791–812. 15

[DM71] Deutsch S. B., Martin J. J.: An ordering algorithm for analysis of data arrays. *Operations Research 19*, 6 (1971), 1350–1362. 5, 11, 12

[DPS02] Díaz J., Petit J., Serna M.: A survey of graph layout problems. *ACM Comput. Surv. 34*, 3 (Sept. 2002), 313–356. 13

[EDG*08] Elmqvist N., Do T.-N., Goodell H., Henry N., Fekete J.-D.: ZAME: Interactive Large-Scale Graph Visualization. *IEEE Pacific Visualization Symposium* (Mar. 2008), 215–222. 5, 10

[ESBB98] Eisen M. B., Spellman P. T., Brown P. O., Botstein D.: Cluster analysis and display of genome wide expression patterns. *Proceedings of the National Academy of Sciences 95*, 25 (1998), 14863–14868. 5, 8, 17, 20

[EW94] Eades P., Wormald N. C.: Edge crossings in drawings of bipartite graphs. *Algorithmica 11*, 4 (1994), 379–403. URL: http://dx.doi.org/10.1007/BF01187020, doi:10.1007/BF01187020. 12

[Fek15] Fekete J.-D.: Reorder.js: A JavaScript Library to Reorder Tables and Networks. IEEE VIS 2015, Oct. 2015. Poster. URL: https://hal.inria.fr/hal-01214274. 5

[Fie73] Fiedler M.: Algebraic connectivity of graphs. *Czechoslovak mathematical journal 23*, 2 (1973), 298–305. 10

[FK03] Friendly M., Kwan E.: Effect ordering for data displays. *Computational statistics & data analysis 43*, 4 (2003), 509–539. 5

[Fri02] Friendly M.: Corrgrams: Exploratory displays for correlation matrices. *The American Statistician 56*, 4 (2002), 316–324. 5, 9, 10

[Gel71] Gelfand A. E.: Rapid seriation methods with archaeological applications. *Hodson, FR, Kendall, DG and Tautu (eds), Mathematics in the Archaeological and Historical Sciences. Edinburgh University Press, Edinburgh* (1971), 186–201. 5, 20

[Geo71] George J. A.: *Computer implementation of the finite element method*. PhD thesis, Stanford University, 1971. 5, 13, 14

[GFC04] Ghoniem M., Fekete J., Castagliola P.: A comparison of the readability of graphs using node-link and matrix-based representations. In *Information Visualization, 2004. INFOVIS 2004. IEEE Symposium on* (2004), IEEE, pp. 17–24. 1, 4

[GKNV93] Gansner E. R., Koutsofios E., North S. C., Vo K.-P.: A technique for drawing directed graphs. *IEEE Trans. Softw. Eng. 19*, 3 (Mar. 1993), 214–230. URL: http://dx.doi.org/10.1109/32.221135, doi:10.1109/32.221135. 12

[GLS*15] Gregor R., Lamprecht A., Sipiran I., Schreck T., Bustos B.: Empirical evaluation of dissimilarity measures for 3d object retrieval with application to multi-feature retrieval. In *Content-Based Multimedia Indexing (CBMI), 2015 13th International Workshop on* (2015), IEEE, pp. 1–6. 20

[GPS76] Gibbs N. E., Poole Jr W. G., Stockmeyer P. K.: An algorithm for reducing the bandwidth and profile of a sparse matrix. *SIAM Journal on Numerical Analysis 13*, 2 (1976), 236–250. 5, 13

[GW72] Gruvaeus G., Wainer H.: Two additions to hierarchical cluster analysis. *British Journal of Mathematical and Statistical Psychology 25*, 2 (1972), 200–206. 5, 7, 8

[Har64] Harper L. H.: Optimal assignments of numbers to vertices. *Journal of the Society for Industrial and Applied Mathematics* (1964), 131–135. 5

[Har72] Hartigan J. A.: Direct clustering of a data matrix. *Journal of the american statistical association 67*, 337 (1972), 123–129. 5

[HBH14] Hahsler M., Buchta C., Hornik K.: *Infrastructure for seriation*, r package version 1.0-14. ed., 2014. 17

[HF06] Henry N., Fekete J.-D.: MatrixExplorer: a dual-representation system to explore social networks. *IEEE transactions on visualization and computer graphics 12*, 5 (2006), 677–84. 5, 15, 20

[HG81] Hubert L. J., Golledge R. G.: Matrix reorganization and dynamic programming: Applications to paired comparisons and unidimensional seriation. *Psychometrika 46*, 4 (1981), 429–441. 5, 12

[HHB08] Hahsler M., Hornik K., Buchta C.: Getting things in order: An introduction to the r package seriation. *Journal of Statistical Software 25*, 3 (March 2008), 1–34. 5, 10, 18

[Hil74] Hill M. O.: Correspondence analysis: A neglected multivariate method. *Journal of the Royal Statistical Society. Series C (Applied Statistics) 23*, 3 (1974), pp. 340–354. 5

[Hil79] Hill M. O.: Decorana-a fortran program for detrended correspondence analysis and reciprocal averaging. 5

[HK02] Harel D., Koren Y.: Graph drawing by high-dimensional embedding. In *Revised Papers from the 10th International Symposium on Graph Drawing* (London, UK, UK, 2002), GD '02, Springer-Verlag, pp. 207–219. 5, 8, 9, 10

[Hub74] Hubert L.: Some applications of graph theory and related nonmetric techniques to problems of approximate seriation the case of symmetric proximity measures. *British Journal of Mathematical and Statistical Psychology 27*, 2 (1974), 133–153. 5, 7, 17

[JXFD08] Jin R., Xiang Y., Fuhry D., Dragan F. F.: Overlapping matrix pattern visualization: A hypergraph approach. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on* (2008), IEEE, pp. 313–322. 5, 16

[Kai11] Kaiser S.: *Biclustering: Methods, Software and Application*. PhD thesis, lmu, 2011. 5, 16

[KCH02] Koren Y., Carmel L., Harel D.: Ace: a fast multiscale eigenvectors computation for drawing huge graphs. In *Information Visualization, 2002. INFOVIS 2002. IEEE Symposium on* (2002), pp. 137–144. 14, 18

[Ken63] Kendall D. G.: A statistical approach to flinders petries sequence-dating. *Bulletin of the International Statistical Institute 40*, 2 (1963), 657–681. 5

[KH02] Koren Y., Harel D.: A multi-scale algorithm for the linear arrangement problem. In *Revised Papers from the 28th International Workshop on Graph-Theoretic Concepts in Computer Science* (London, UK, UK, 2002), WG '02, Springer-Verlag, pp. 296–309. 14, 17

[Kin70] King I. P.: An automatic reordering scheme for simultaneous equations derived from network systems. *International Journal for Numerical Methods in Engineering 2*, 4 (1970), 523–533. 5, 13

[KL08] Kaiser S., Leisch F.: A toolbox for bicluster analysis in r, 2008. 5, 17, 18

[Kny01] Knyazev A. V.: Toward the optimal preconditioned eigensolver: Locally optimal block preconditioned conjugate gradient method. *SIAM Journal on Scientific Computing 23*, 2 (2001), 517–541. 8

[Kor05] Koren Y.: Drawing graphs by eigenvectors: Theory and practice. *Comput. Math. Appl. 49*, 11-12 (June 2005), 1867–1888. 5

[KY05] Kim D., Yum B.-J.: Collaborative filtering based on iterative principal component analysis. *Expert Systems with Applications 28*, 4 (2005), 823–830. 10

[LDGM12] Lozano M., Duarte A., Gortázar F., Martí R.: Variable neighborhood search with ejection chains for the antibandwidth problem. *Journal of Heuristics 18*, 6 (2012), 919–938. 5, 14

[LDGM13] Lozano M., Duarte A., Gortázar F., Martí R.: A hybrid metaheuristic for the cyclic antibandwidth problem. *Knowledge-Based Systems 54* (2013), 103–113. 5, 14

[Len74] Lenstra J.: Technical note—clustering a data array and the traveling-salesman problem. *Operations Research 22*, 2 (1974), 413–414. 5, 14

[LHGY03] LIU L., HAWKINS D. M., GHOSH S., YOUNG S. S.: Robust singular value decomposition analysis of microarray data. *Proceedings of the National Academy of Sciences 100*, 23 (2003), 13167–13172. 5, 10

[Lii10] LIIV I.: Seriation and matrix reordering methods: An historical overview. *Statistical analysis and data mining 3*, 2 (2010), 70–91. 1, 2

[LK75] LENSTRA J. K., KAN A. R.: Some simple applications of the travelling salesman problem. *Operational Research Quarterly* (1975), 717–733. 5, 14

[LO*02] LAZZERONI L., OWEN A., ET AL.: Plaid models for gene expression data. *Statistica sinica 12*, 1 (2002), 61–86. 5, 15, 16

[LPP*06] LEE B., PLAISANT C., PARR C. S., FEKETE J.-D., HENRY N.: Task taxonomy for graph visualization. In *Proceedings of the 2006 AVI workshop on BEyond time and errors: novel evaluation methods for information visualization* (2006), ACM, pp. 1–5. 4, 5, 21

[LS76] LIU W.-H., SHERMAN A. H.: Comparative analysis of the cuthill-mckee and the reverse cuthill-mckee ordering algorithms for sparse matrices. *SIAM Journal on Numerical Analysis 13*, 2 (1976), 198–213. 5, 13

[LVW84] LEUNG J. Y., VORNBERGER O., WITTHOFF J. D.: On some variants of the bandwidth minimization problem. *SIAM Journal on Computing 13*, 3 (1984), 650–667. 5, 13, 14

[MBK97] MCGRATH C., BLYTHE J., KRACKHARDT D.: The effect of spatial arrangement on judgments and errors in interpreting graphs. *Social Networks 19*, 3 (1997), 223 – 242. 2

[McQ68] MCQUITTY L. L.: Multiple clusters, types, and dimensions from iterative intercolumnar correlational analysis. *Multivariate Behavioral Research 3*, 4 (1968), 465–477. 5, 9

[MDMS69] MCCORMICK W. T., DEUTSCH S. B., MARTIN J. J., SCHWEITZER P. J.: *Identification of Data Structures and Relationships by Matrix Reordering Techniques*. Tech. rep., DTIC Document, 1969. 5, 12

[MK03] MURALI T., KASIF S.: Extracting conserved gene expression motifs from gene expression data. In *Pacific Symposium on Biocomputing* (2003), vol. 8, World Scientific, pp. 77–88. 5, 16

[MML07a] MUELLER C., MARTIN B., LUMSDAINE A.: A comparison of vertex ordering algorithms for large graph visualization. In *Visualization, 2007. APVIS'07. 2007 6th International Asia-Pacific Symposium on* (2007), IEEE, pp. 141–148. 1, 2

[MML07b] MUELLER C., MARTIN B., LUMSDAINE A.: Interpreting large visual similarity matrices. *2007 6th International Asia-Pacific Symposium on Visualization* (Feb. 2007), 149–152. 2, 4

[MO04] MADEIRA S. C., OLIVEIRA A. L.: Biclustering algorithms for biological data analysis: a survey. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB) 1*, 1 (2004), 24–45. 15, 16

[MS00] MÄKINEN E., SIIRTOLA H.: Reordering the reorderable matrix as an algorithmic problem. In *Theory and Application of Diagrams*. Springer, 2000, pp. 453–468. 5, 12

[MS05] MÄKINEN E., SIIRTOLA H.: The barycenter heuristic and the reorderable matrix. *Informatica (Slovenia) 29*, 3 (2005), 357–364. 12

[MS14] MAFTEIU-SCAI L. O.: The bandwidths of a matrix. a survey of algorithms. *Annals of West University of Timisoara-Mathematics 52*, 2 (2014), 183–223. 12, 14

[MSW72] MCCORMICK W. T., SCHWEITZER P. J., WHITE T. W.: Problem decomposition and data reorganization by a clustering technique. *Operations Research 20*, 5 (1972), 993–1009. 5, 12

[Mue04] MUELLER C.: Sparse matrix reordering algorithms for cluster identification. *For I532, Machine Learning in Bioinformatics* (2004). 2

[Net] NETWORKX: NetworkX. https://networkx.github.io/. online, last visited November 19, 2015. 17

[Nie05] NIERMANN S.: Optimizing the ordering of tables with evolutionary computation. *The American Statistician 59*, 1 (2005). 5, 12

[OKLS15] OSEI-KUFFUOR D., LI R., SAAD Y.: Matrix reordering using multilevel graph coarsening for ilu preconditioning. *SIAM Journal on Scientific Computing 37*, 1 (2015), A391–A419. 14

[PBZ*06] PRELIĆ A., BLEULER S., ZIMMERMANN P., WILLE A., BÜHLMANN P., GRUISSEM W., HENNIG L., THIELE L., ZITZLER E.: A systematic comparison and evaluation of biclustering methods for gene expression data. *Bioinformatics 22*, 9 (2006), 1122–1129. 5, 16

[PDF14] PERIN C., DRAGICEVIC P., FEKETE J.-D.: Revisiting Bertin matrices: New Interactions for Crafting Tabular Visualizations. *IEEE Transactions on Visualization and Computer Graphics* (Nov. 2014). 1, 5, 8, 20

[Pet03] PETIT J.: Experiments on the minimum linear arrangement problem. *J. Exp. Algorithmics 8* (Dec. 2003). 2, 4, 17

[Pic09] PICH C.: *Applications of Multidimensional Scaling to Graph Drawing*. PhD thesis, Universität Konstanz, Konstanz, 2009. 11

[PM14] POP P., MATEI O.: An efficient metaheuristic approach for solving a class of matrix optimization problems. *METAHEURISTICS AND ENGINEERING* (2014), 17. 5, 14

[RC94] RAO R., CARD S. K.: The table lens: merging graphical and symbolic representations in an interactive focus+ context visualization for tabular information. In *Proceedings of the SIGCHI conference on Human factors in computing systems* (1994), ACM, pp. 318–322. 5, 20

[Rob51] ROBINSON W. S.: A method for chronologically ordering archaeological deposits. *American antiquity 16*, 4 (1951), 293–301. 5

[Ros68] ROSEN R.: Matrix bandwidth minimization. In *Proceedings of the 1968 23rd ACM national conference* (1968), ACM, pp. 585–595. 5, 13

[RSS*09] RASPAUD A., SCHRÖDER H., SÝKORA O., TOROK L., VRT'O I.: Antibandwidth and cyclic antibandwidth of meshes and hypercubes. *Discrete Mathematics 309*, 11 (2009), 3541–3552. 14

[RT92] RODGERS J. L., THOMPSON T. D.: Seriation and multidimensional scaling: A data analysis approach to scaling asymmetric proximity matrices. *Applied psychological measurement 16*, 2 (1992), 105–117. 5, 11

[SBB96] SPENKE M., BEILKEN C., BERLAGE T.: Focus: the interactive table for product comparison and selection. In *Proceedings of the 9th annual ACM symposium on User interface software and technology* (1996), ACM, pp. 41–50. 20

[SG74] SPENCE I., GRAEF J.: The determination of the underlying dimensionality of an empirically obtained matrix of proximities. *Multivariate Behavioral Research 9*, 3 (1974), 331–341. 5, 11

[Sii99] SIIRTOLA H.: Interaction with the reorderable matrix. In *Proceedings of the 1999 International Conference on Information Visualisation* (Washington, DC, USA, 1999), IV '99, IEEE Computer Society, pp. 272–. 5

[SLL01] SIEK J. G., LEE L.-Q., LUMSDAINE A.: *Boost Graph Library: User Guide and Reference Manual, The*. Pearson Education, 2001. 5, 18

[Slo86] SLOAN S. W.: An algorithm for profile and wavefront reduction of sparse matrices. *International Journal for Numerical Methods in Engineering 23*, 2 (1986), 239–251. 5, 14, 15

[Slo89] SLOAN S.: A fortran program for profile and wavefront reduction. *International Journal for Numerical Methods in Engineering 28*, 11 (1989), 2651–2679. 5, 14

[Ste65] STERNIN H.: *Statistical Methods of Time Sequencing*. Tech. rep., DTIC Document, 1965. 5, 9

[SW68] STEIGLITZ K., WEINER P.: Some improved algorithms for computer solution of the traveling salesman problem. 15

[TBK05] TURNER H., BAILEY T., KRZANOWSKI W.: Improved biclustering of microarray data demonstrated through systematic performance tests. *Computational statistics & data analysis 48*, 2 (2005), 235–254. 5, 16

[TSS05] TANAY A., SHARAN R., SHAMIR R.: Biclustering algorithms: A survey. *Handbook of computational molecular biology 9*, 1-20 (2005), 122–124. 8, 16

[vRHB*15] VON RÜDEN L., HERMANNS M.-A., BEHRISCH M., KEIM D., MOHR B., WOLF F.: Separating the wheat from the chaff: Identifying relevant and similar performance data with visual analytics. In *Proceedings of the 2nd Workshop on Visual Performance Analysis* (New York, NY, USA, 2015), VPA '15, ACM, pp. 4:1–4:8. 14

[Wat91] WATKINS D. S.: *Fundamentals of Matrix Computations*. John Wiley & Sons, Inc., New York, NY, USA, 1991. 8

[Wei13] WEI T.: *Corrplot: Visualization of a correlation matrix*, r package version 0.73. ed., Oct 2013. 5, 17, 18

[WF09] WILKINSON L., FRIENDLY M.: The history of the cluster heat map. *The American Statistician* (2009). 2

[Wil05] WILKINSON L.: *The Grammar of Graphics (Statistics and Computing)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005. 2, 4, 5, 8, 10, 11, 13

[WMFM13] WONG P. C., MACKEY P., FOOTE H., MAY R.: Visual matrix clustering of social networks. *IEEE computer graphics and applications 33*, 4 (2013), 88–96. 15

[WTC08] WU H.-M., TZENG S., CHEN C.-H.: *Handbook of Data Visualization*. Springer, 2008, ch. Matrix Visualization, pp. 681–708. 2