GUIRO: User-Guided Matrix Reordering

Michael Behrisch, Tobias Schreck, and Hanspeter Pfister

Which matrix reordering algorithm to choose? What is hidden in my data? How to design better reordering algorithms?

Fig. 1: With GUIRO we address three essential matrix analysis questions: (1) Which matrix reordering algorithm produces useful results? (2) Can we steer these algorithms to support analytical tasks? (3) How to compare reorderings quantitatively & qualitatively?

Abstract—Matrix representations are one of the main established and empirically proven to be effective visualization techniques for relational (or network) data. However, matrices—similar to node-link diagrams—are most effective if their layout reveals the underlying data topology. Given the many developed algorithms, a practical problem arises: “Which matrix reordering algorithm should I choose for my dataset at hand?” To make matters worse, different reordering algorithms applied to the same dataset may let significantly different visual matrix patterns emerge. This leads to the question of trustworthiness and explainability of these fully automated, often heuristic, black-box processes. We present GUIRO, a Visual Analytics system that helps novices, network analysts, and algorithm designers to open the black-box. Users can investigate the usefulness and expressiveness of 70 accessible matrix reordering algorithms. For network analysts, we introduce a novel model space representation and two interaction techniques for a user-guided reordering of rows or columns, and especially groups thereof (submatrix reordering). These novel techniques contribute to the understanding of the global and local dataset topology. We support algorithm designers by giving them access to 16 reordering quality metrics and visual exploration means for comparing reordering implementations on a row/column permutation level. We evaluated GUIRO in a guided explorative user study with 12 subjects, a case study demonstrating its usefulness in a real-world scenario, and through an expert study gathering feedback on our design decisions. We found that our proposed methods help even inexperienced users to understand matrix patterns and allow a user-guided steering of reordering algorithms. GUIRO helps to increase the transparency of matrix reordering algorithms, thus helping a broad range of users to get a better insight into the complex reordering process, in turn supporting data and reordering algorithm insights.

Index Terms—Visual Analytics, matrix, black-box algorithms, seriation, ordering, sorting, steerable algorithm, interaction, 2D projection

1 Introduction

Relational datasets, commonly referred to as graph datasets, are getting more and more important in the research and commercial sectors. Social networks, modeling individuals and their interactions, are typically used for researching sociological and socio-dynamic phenomena, but also find their commercial application in the planning and evaluation of effective advertisement campaigns [64, 85]. Biological networks help us understand the intertwined relationships between the degree of efficiency and environmental influences for drugs or depicts complex phylogenetic trees [11, 41]. Telecommunication networks are combining relational data and geographic considerations to drive the development of our ubiquitous wired and wireless networks. A wide range of application fields for network analysis, in general, are known in the area. The NetworkRepository.com [66] alone groups its more than 4,500 datasets into 27 categories, representing also less commonly associated fields, such as ChemInformatics, brain-, power network analysis, or ecology and economy networks research.

Generally, two visualization techniques for large-scale graph data have been proven effective: Node-link diagrams and matrix visualiza-
Does the network represent a small-world phenomenon?" (globally visualization literacy to work with matrix plots. Users can experiment reordering process, in turn leading to novel data and algorithm insights. a broad range of users to get a better insight into the complex matrix the freedom to intervene in the row
column reordering process, helps a broad range of users to get a better insight into the complex matrix reordering process, in turn leading to novel data and algorithm insights. For row

A network analyst has to distill this information, by retrieving, e.g., who is the central actor connecting two subgroups in a social network. Having found this influencing person the subsequent question would be who is the primary contact point within the groups to distribute the information further.
T7 Adapt visual results to emphasize analytic importance. Presenting the above complex relationships is quintessential for a successful network analyst. However, based on our experience, these analysis interestingly do not lean necessarily on one automatic matrix reordering result but sacrifice the global reordering on subparts to emphasize a point on other parts. In GIRO, we address the usage scenarios of structure “pre-screening” with a projection-based index view (Sect. 4.2) and two interactions for a local pattern exploration, organization, and presentation (Sect. 5).

2.3 Algorithm Designer
Algorithm designers have a different set of tasks, compared to the network analyst and novice users. Their daily work is centered around quantitative evaluation schemes, justifiable algorithm design decisions, and stress tests.

T8 Inspect Quality Metrics. Algorithm designers are concerned, not only with the visual performance of their algorithms, but also quantitative quality metrics. Comparing a newly designed algorithm against the state-of-the-art is necessary for an objective comparative evaluation.

T9 Comparing/Weighing up of Design Decisions. Matrix reordering algorithms can be tweaked on two fronts: (a) the process of exploring the permutations space and (b) the notion of similarity between rows/columns. Giving the designer (visual) means to reflect on the impact of algorithm design decisions will lead to better, more stable algorithms for matrix reordering.

In GIRO, we give the algorithm designer various means to address their typical usage scenarios (see: Sect. 6). A performance metric view (T8) and the index-to-index comparison component (T10) enables a (visual) juxtaposition of matrix reordering results. Futhermore, we can overlay multiple reordering results in our projection-based index visualization, as we will describe in Sect. 4.2.

3 Matrices and Visual Patterns in Matrices
An adjacency matrix $M = (n \times n)$ is a two-dimensional vector with $n = |V|$ with $N$ being the vertices in an undirected network. A matrix cell $c_{ij}; 0 \leq i, j < n$ denotes the existence of an edge between vertices $i$ and $j$. In simplest case, $c_{ij} = 1$ if $i$ and $j$ are connected, otherwise $c_{ij} = 0$. In many real-world datasets, $c_{ij}$ models data attributes, such as strength (e.g., amount of messages exchanged between friends in a social network), or relation type (e.g., highway vs. interstate road in a transportation network) resulting in a weighted adjacency matrix. Matrix visualizations reflect this layout scheme by applying color, texture, or glyphs to each to cell. Rows and columns can be freely rearranged, also called a row/column permutation $n$, without changing the underlying data. A “good” reordering allows us to perceive Overview and Detail in one plot. Statistically, however, $(n! \times n!)/2$ non-redundant permutations exists of which only few expose the topological structure.

3.1 Visual Patterns in Matrices
A visual matrix pattern is a perceivable structure in the matrix visualization revealing information about the underlying graph topology. Some work has been conducted [7, 53, 80] to collect the central visual patterns in matrix plots, along with their graph-theoretic interpretations. For the sake of simplicity, let us analyze the simple graph shown in Fig. 2 while generalizing our findings to a taxonomy of interpretable visual patterns for matrices.

The graph in Fig. 2 contains the complete/fully connected subgraph (clique) $(v1 \bowtie v2 \bowtie v3)$. In our matrix representation, the red coherent rectangular block on the main diagonal represents this clique. Generally, a Block-diagonal pattern consists of one or multiple blocks (at least $2 \times 2$ cells) on the main diagonal of the matrix. Note, that the main diagonal is the center matrix line, where $i = j, 0 \leq i, j < n$, and is quintessential for a correct interpretation of matrix patterns. The density of the blocks lets us distinguish cliques (fully colored), strongly connected components or clusters (some uncolored matrix cells) and loosely connected components (less color density). Connected components are often the central visual pattern. It finds applications, e.g., in social network analysis representing groups of mutual friends or in bioinformatics groups of genes with their related expression characteristics.

The next visual pattern in our graph connects $v5 \bowtie v1, v5 \bowtie v2, v5 \bowtie v3, v5 \bowtie v4,$ and analogous $v6 \bowtie v1, v6 \bowtie v2, v6 \bowtie v3, v6 \bowtie v4$. The visually dense off-diagonal gray-blue rectangles show this connectivity pattern in matrices.

Fig. 2: An adjacency matrix (left) and a Node-Link diagram (right) allow interpreting a range of visual patterns. A network analysts’ tasks is to (a) find the row/column ordering that represents most topological structures best and (b) to represent the results effectively to the audience. Figure adapted from [54].

Albeit being similar in its visual appearance to block-diagonal forms, Off-Diagonal Block patterns do not overlap with the main diagonal; sometimes they are even at the corners of the matrix. Off-diagonal blocks refer to bi-cliques consisting of two sets of vertices where each vertex from the one set is connected to each vertex of the second set. Bi-cliques model for example, website visits and their user groups, or persons having visited a range of cities.

Next, our graph shows one four node path connecting $v5 \bowtie v6, v6 \bowtie v7, v7 \bowtie v8,$ and $v8 \bowtie v9$, with its matrix representation being the two bands or lines parallel to the main diagonal.

In undirected networks, this Band pattern consists of lines parallel to the matrix diagonal. These lines are mirrored and do not touch or cross the main diagonal. Bands indicate paths through the network, i.e., sequence of links from one to another vertex. Paths can represent a possible way of information flow through a computer network or a sequence of reactions in a biological network.

Lastly, we find a fan-like connection of the vertex $v10 \bowtie v8, v10 \bowtie v9, v10 \bowtie v11$, and $v10 \bowtie v11$ in green. In the matrix, this motif will be strongly visible by a so-called Star/Line Pattern

Generally, star or also called line patterns in matrices consist of one horizontal and one connected vertical line with or without discontinuities. This pattern represents a star graph motif, i.e., a highly connected vertex in the network, such as a famous person in a friendship network, a transportation hub, or a catalyst along reactions in a biological network.

Our walk-through should not be considered as exhaustive, but rather as a general methodology for retrieving matrix patterns. The large variety of visual patterns results from the combination of the aforementioned base patterns and so-called anti-patterns: Two established anti-patterns exist in the literature [7, 80]:

The most obvious Anti-pattern is Noise. We can describe it as a distribution of black cells without inherent visual structure. The more noise a matrix contains, the less information it shows. Interestingly, noise has two interpretations: (1) There is no apparent structure in the underlying graph topology and (2) the pattern extractor, i.e., the matrix reordering algorithm, was not able to grasp this underlying topology [10, 53].
The second Anti-pattern is the Bandwidth Anti-pattern. Bandwidth patterns are solely an artifact of algorithms, which are examining the permutation search space in a breadth-first search, such as the graph-based matrix reordering algorithm Cuthill-McKee and its predecessors [15, 24]. This breadth-first search finds its visual manifestation in a formed envelope, which does not add interpretable information unless the ordering within the formed envelope contains other patterns.

In this work, we want to establish a third general Anti-pattern: The Noise-Cluster Anti-pattern. Many matrix reordering algorithms (see also: Sect. 3.2) are designed to support partitioning tasks, i.e., their implementation tries (desperately) to reflect groupings/cluster in the data. However, two artifacts frequently clutter matrix visualizations: (1) the algorithm groups rows/columns next to each that are contradicting our humans’ gestalt-law expectations and (2) the algorithm separates rows/columns that our humans’ perception system would rather group together. Both visual artifacts can be traced back to the algorithm’s implementation, e.g., thresholding decisions. Unfortunately, these perceptual and related cognitive biases aspect find little attention in the literature, yet they contribute significantly to our interpretability problems for matrices.

### 3.2 Matrix Reordering (or Seriation)

Matrix visualizations scale well with the size of the input data, but their usefulness heavily depends on a suitable reordering of rows/columns [7, 53, 81, 83]. We can formalize the process reordering of an undirected network \( G \) in accordance to our state-of-the-art survey on matrix reordering approaches [7], as computing one permutation \( \pi \in S \) that maximizes or minimizes a quality criteria \( q(\pi, G) \), such that:

\[
\arg\min_{\pi \in S} q(\pi, G) \quad \text{for Example, } q \text{ can compute the sum of distances } d(x, y) \text{ between vertices according to the order } \pi \text{. The equation would find } \pi \in S \text{ that minimizes this sum.}
\]

While a variety of automated reordering methods have been developed, we want to focus our attention in this work on the underlying mechanisms and different quality criteria that guide the process as a whole.

**Automated Matrix Reordering:** Quality metrics for matrix reordering regularly include (a) the optimization of quality criteria, e.g., the row/ column dissimilarity, and (b) the emergence of visible structures from the matrix. Regarding (a), several quality criteria have been suggested that are partially congruent with the graph drawing domain; c.f., Diaz et al. [17]. The prominent quality criteria for matrix reordering are bandwidth and profile optimization, column/row gradient measures, such presented by Hubert et al. [36], the Hamilton path length [13], or Stress measures [55]. The linear arrangement (LA) [43] is the de facto standard for comparing matrix reordering algorithms quantitatively. It relates all node-to-node distances (or edge weights) to their index-to-index distances in a given reordering. Minimizing the LA causes similar vertices to be placed close to each other. Finding a global optimum for this problem is computationally hard, but efficient approximations, such as the Multi-Scale approach [43] exist. Regarding (b), other reordering strategies may aim at exposing interpretable visual structures (c.f., Sect. 3.1). These effect ordering techniques [22] make use of perception-driven quality criteria, such as Anti-Robinson Events/Deviations [14, 65, 71], Anti-Robinson arrangements [61], the barycenter heuristics [50], or McCormick et al.’s “Measure of Effectiveness” [51, 52], which can be used to find locally dense areas off the diagonal (c.f., Off-Diagonal Block pattern). Note that GUIRO allows analysts to compare 16 quality scores from all mentioned categories.

**Interactive Matrix Reordering:** Automatic approaches are making (non-)deterministic, sometimes greedy, and most importantly hard-coded decisions at many points of their process, i.e., how to start the reordering sequence or how to traverse and prune the permutation search space. To address the shortcomings of certain algorithms, interactive reordering techniques have been suggested. Bertifier [60], TableLens [63], InfoZoom [70], or Siirtola and Mäkinen’s “Reorderable Matrix” [69] on the one end of the spectrum, let the user drag-and-drop rows and columns manually, providing comparable functionality to Bertin’s mechanical reordering device. On the other end of the spectrum, semi-assisted (steering) approaches, such as MatrixExplorer [32], COMBat [73], or PermutMatrix [13] let the user apply distinct matrix reorderings while helping between context switches. The first approach to a semi-automatic matrix reordering was presented by Henry and Fekete in [32], where a traveling salesman-based reordering algorithm can be iteratively (re-)applied on a selected subset of rows/columns.

GUIRO unifies many of the mentioned interactive and semi-automatic approaches in one user interface and lets the user experience, steer, and evaluate matrix reordering systematically. Please note that this section is supposed to give the reader a tutorial into the topic. For didactical reasons, we discuss GUIRO in the context of its applied techniques in the dedicated Related Work Section 8.

### 4 User-Guided Matrix Reordering

![Fig. 3: Processing pipeline for our user-steerable matrix reordering approach: the rows, resp. columns, of a matrix are interpreted as high-dimensional (HD) vectors (1st image) and projected into a 2D space (2nd image) forming a set of vertices. Similar HD vectors are projected to similar 2D positions. A matrix reordering result is depicted by a path acyclically connecting all vertices. Selecting vertex groups allows a local application of reordering algorithms on submatrices (3rd image). The path can be manually modified, such that locally optimized submatrices or single vertices can be placed next to another (4th image).](image)

In this section, we provide an overview of GUIRO. We illustrate how the steerable reordering process can increase user understanding of the matrix plot, as well as the measurable quality of the reordering.

**Implementation:** We built GUIRO on top of a microservice infrastructure that encapsulates processing algorithms (matrix reordering, projections, and statistics) independent of their implementation language and makes them accessible through a REST API. The matrix reordering server, for example, bridges between Java (for the API) and its underlying algorithm implementations in R, Java, and C++. For all ten projection algorithms described in Sect. 4.2, we use the Smile (Statistical Machine Intelligence and Learning Engine) Java library [2]. Inter-service communication is implemented, for now, in REST. A central database stores all computed projections and matrix reordering results. GUIRO, as the visible client, is implemented in TypeScript. The interactive visualizations are written in D3.js. The client accesses all microservices through an API gateway, which enables horizontal scalability, service discovery, load distribution, health checks, and manages the authentication centrally. Source code, a demo instance, and a descriptive webpage will be available at [http://bit.ly/matrixreordering](http://bit.ly/matrixreordering).
4.1 Comparing Matrix Reordering Results

A multitude of matrix reordering algorithms have been developed with distinct design goals in mind. Consequently, the question arises which algorithm to choose (T1)? We claim that the research developments in the quality metrics for information visualization field [8] will soon allow us to approach this question with Visual Analytics means. For now, however, we have to rely on the user to make an informed decision.

In GUIRO, we use a diagram to choose from 70 matrix reordering algorithms (+1 identity sort). All implementations are accessible through a central server, whose purpose is to proxy our own Java implementations, 42 implementations from the outstanding R package seriation by Hahsler, Buchta, and Hornik [28, 29], nine R packages implementing implementations [37], two R packages correlplot algorithms [79], and three algorithms from the C++ library Boost [1]. All references for the accessible implementations can be found in the appendix. Note that, nine reordering algorithms are reoccurring with district implementation facets, making them interesting study objects for algorithm designers (T9, T10).

We depict the results of all applied algorithms first in a preview manner, next to their quantitative quality metric scores; see Figure above. The list allows us (a) to experience the algorithms’ visual quality and usefulness, i.e., produced amount of clutter (T1), and (b) demonstrates the algorithm’s pattern variability given the same dataset (T2). For instance, in the screenshot above we can see that 0LO shows a Star-Line form, but ARSA tries to capture a Block pattern. After having found an algorithm result, we can apply it to the main interactive matrix view to begin our in-depth analysis process (Sect. 4.3). Interestingly, it is even for experts challenging to distinguish deterministic from non-deterministic algorithms (T3). To aid this use case, we allow to reapply the same algorithm several times and see the results in a comparative display.

4.2 Representing Matrix Reorderings in the Projection Space

Our index ordering visualization relies on a projection of the matrix M onto the 2D plane, where the user can interact with the matrix elements by moving or grouping them (T5, T6, T7). We realize the projection by regarding the matrix rows or columns as a set of n-dimensional vectors with n = |N| with N being the nodes in our network. Similar concepts have already been shown for 2D/3D graph layouts [30], dynamic network analysis [74], or topological graph comparison [12, 44]. In order to examine the typological structure of M (T4), we can apply all projection techniques that find a lower dimensional representation y = (y1, ..., yN)T for each vector x = (x1, ..., xN)T with k ≤ l, such that y preserves the HD row/column similarity. In other words, that put visually similar rows/columns close to each other in the projection space. Note, for symmetric matrices (undirected graphs) rows and columns can be used interchangeably.

We could extend GUIRO for non-symmetric matrices, where rows and columns would need to be considered distinctively. In the current version of the system, we decided against this functionality, because it adds significant complexity to the interpretation of the projection space patterns.

The many existing projection algorithms all have their own advantages and disadvantages. Linear projection techniques are typically fast, but only preserve linear structures. Non-linear projection techniques are typically slow, but can also take non-linear structures into account. In order to be projection-agnostic, GUIRO implements ten projection techniques: PCA, MDS, t-SNE, IsoMap, Isotonic MDS, LLE, PPCA, RPCA, SammonMapping, LaplacianEigenMap.

To depict one or multiple matrix reordering results, we connect the 2D projection points in this display through directed edges representing a given ordering of the rows or columns. The resulting path has n − 1 edges representing the matrix’s row/column linearization. Its start and end point map to the first and last row or column in the matrix view. The path represents a model visualization, whose appearance (see: Sect. 4.3) allows us to scrutinize the effectiveness of a matrix reordering result.

4.3 Interpreting Reordering Results in the Projection Space

The 2D index visualization allows the user to perceive the reordering in terms of the similarity of rows or columns and gives access to interactive matrix reordering operations. We structure the interaction around visual patterns emerging from this space. These patterns prove to be a beneficial starting point for improving the matrix reordering. They visually represent a mismatch between close projection points, i.e., similar column-row vectors of the matrix, and long connection edges, i.e., the sequential placement of dissimilar column-row vectors by the reordering algorithm.

- **Star Edges:** The star edge pattern is a special case of the vertex group pattern, which can be often in matrix reordering results. As our illustration depicts, a dense group of vertices is connected to multiple satellite vertices. In order to optimize this pattern, a user or reordering algorithm would need to connect all satellite vertices linearly with one edge to the dense central vertex group.

- **Multiple parallel edges** are another indication for optimization potential. Our illustration (left) represents the case when similar vertices are not connected appropriately to their neighborhood, i.e., are not arranged correctly into “local” subsequences. A fix would first consider the local groupings and then connect the two groups.

**Discussion:** The index ordering visualization projects similar rows/columns in close proximity. Consequently, an intermediate result will appear visually cluttered (e.g., Fig. 4), due to the many, long, and unstructured index ordering edges. The users’ task is to retrieve the aforementioned projection space patterns and invoke the local (group) reordering solutions described in the following subsections. Users will be confronted with a combination of patterns (see also the accompanying videos), thus making this view to a typical Visual Analytics component, in which user understanding, prioritization and planning are of paramount importance for the success.

5 Interacting with Matrix Reorderings in the Projection Space

Once we retrieved the aforementioned structures in the 2D projection space we can start optimizing the reordering with two key interactions: (1) Invoke an automated local reordering on a selected group of vertices, and (T4, T5) (2) Rearrange vertices or groups by replacing one or more existing edges (T6, T7).

In the case of an automated local reordering, the user selects a group of vertices with a lasso selection tool and invokes thereby the matrix reordering algorithm of choice. The algorithm operates only on the selected matrix elements, effectively reordering a subregion of the selected matrix. This reordering approach can work recursively (T5), as user-selected groups can be organized into subgroups and sorted independently (see Sect. 5.1 for details). The second key interaction
Adaptations to submatrices 6, 8 and 9 must be applied in both the forward and backward edge modification algorithms to establish this valid selection. We propose a so-called PartSort algorithm to prepare the matrix for a local reordering. PartSort takes as an input a user selection, the current index permutation, and a reordering algorithm to be applied on the submatrix. It then linearizes the selected projection vertices sequentially without a notion of reordering quality. It ensures that the remaining 2D projection vertices, i.e., before and after the user selection, are logically connected by edges. After this initial step, all available matrix reordering algorithms can be applied on the selected submatrix as if the matrix stands on its own. For our example case, depicted in our matrix reassembling illustration (left), the user selection corresponds to the submatrix 5, which gets reordered independently. This reordering, i.e., the new ordering of the selection vertices, needs to be propagated into the full matrix. While the submatrix 1 remains untouched by the local change, all other submatrices have to be adapted. Depending on a submatrix’s position relative to submatrix 5, either a horizontal (column-wise) or vertical (row-wise) or horizontal and vertical adaptations have to be applied. Submatrices 2 and 3, for example, will only be affected in the horizontal direction. The same applies to submatrices 4 and 7 in a vertical direction. Therefore, a column-row swapping helps to reassemble the matrices correctly. Adaptations to submatrices 6, 8 and 9 must be applied in both the horizontal and vertical directions.

5.1 Local Group Reordering
We enable a submatrix reordering either with the help of an automated algorithm or a user-steered manual reordering decision (T4, T5). Obviously, this operation can only be applied if all row-/column projection vectors are in subsequent order thus forming a coherent submatrix. To establish this valid selection, we propose a so-called PartSort algorithm to prepare the matrix for a local reordering.

5.2 Edge Replacement Reordering Strategies
The second user intervention is the so-called edge replacement, depicted in Fig. 5. It rearranges a single row/column, respectively group of rows/columns relative to each other. This can be useful for detecting item-to-item, item-to-group, or group-to-group relationships (T6). Another use case is that an analyst may wish to emphasize a finding (T7), e.g., by “pulling” two influencing entities right next to each other.

A formally correct matrix reordering is present in the projection space if there exists precisely one acyclic path that connects all vertices. Whenever a user draws a new edge, its insertion will introduce cycles and ambiguities resulting from the temporarily two potential paths. Accordingly, every user intervention inevitably requires reestablishing our formal requirement. We need to deal with two edge manipulation cases depending on whether the user wants to put a vertex (or group) left/before or right/after another vertex (group):

1. Forward Edge Modifications: Forward edges are those edges whose linear reordering’s start index (index of the inserted edge’s outgoing vertex) is smaller than the end index (index of the inserted edge’s incoming vertex). To make an example: if the user wants to place the 1st row-/column vector next to the 5th row-/column vector.

2. Backward Edge Modifications: Backward edges are those edges whose start index is larger than the end index. For example, if the user wants to place the 5th row-/column vector next to the 1st row-/column vector.

As a proof of concept and a point of further research, we implemented two different reordering strategies. We found that both edge modification types can ultimately have a significant impact on the matrix’s overall reordering quality with respect to the linear arrangement.

Independent of the two cases we are reestablishing one acyclic path connecting all vertices by inserting so-called “reconstruction edges”. In our forward edge modification algorithm we split the vertex set into three disjoint subsets, as depicted in the upper part of Fig. 5. The first subset contains all vertices with an index before the insertion edge’s starting point (v2) in the Set1 = [v0, v1]. The second subset contains all edges with an index larger than the starting point and smaller than the endpoint (v8) in the Set2 = [v3, v4, v5, v6, v7]. Finally, the third subset contains all edges with an index after the endpoint in the Set3 = [v9, v10]. The new vertex sequence is calculated as follows: First, we start the permutation vector by retaining Set1 = [v0, v1] and then add the insertion edge’s start point and end point [v0, v1, v2, v8]. We append subsequently Set2 = [v0, v1, v2, v8, v9, v10] to the list. Lastly, a new reconstruction edge is added to connect all items in Set3, leading to the final linear reordering of [v0, v1, v2, v8, v9, v10, v3, v4, v5, v6, v7].

The backward edge modification algorithm is slightly more complicated and illustrated in the lower part of Fig. 5. We also split the vertex set into three subsets. In this case, Set1 contains all vertices after the insertion edge’s endpoint Set1 = [v3, v4, v5, v6, v7]. In the second set Set2, we collect the ordering from the initial linearization start until the insertion edge’s endpoint Set2 = [v0, v1]. Lastly, we store all vertices after the insertion edge’s s-t point in Set3 = [v9, v10]. The new permutation vector begins with the insertion edge sequence [v8, v2], followed by Set1 = [v8, v2, v3, v4, v5, v6, v7] and a reconstruction edge. Subsequently, we append Set2 = [v8, v2, v3, v4, v5, v6, v7, v8, v1] and another reconstruction edge to connect the missing vertices in Set3. The final result is [v8, v2, v3, v4, v5, v6, v7, v9, v1, v3, v5, v8, v10]. A wide range of other implementations for the edge modifications are possible. While the algorithms presented above are just one heuristic, they perform well in establishing a better linear arrangement in our experiments. However, more sophisticated algorithms that minimize the reconstruction edges’ length while retaining most of the input linear ordering are imaginable. These algorithms need to judge whether the user should be overruled for the sake of a better linear arrangement.

6 Comparing Performance Metrics for Reordering Algorithms
Another use case for GUIRO is the quantitative quality assessment. Algorithm designers constantly need to compare their designs to the state-of-the-art, similar approaches in the subfield, or algorithm modifications from the same algorithm family (T8).

We allow comparing 16 distinct quality metrics for matrix reordering algorithms, such as (Anti-)Robinson measures, graph-theoretic, stress or correlation quality metrics. Interestingly, we found that Wilkinson’s idea to compute the similarity of matrix reordering results with the help of a Spearman correlation-based metric [80, p. 532] can be intuitively depicted with a parallel-coordinate plot inspired visualization. For example, in Fig. 6 we directly see that the hierarchical clustering algorithms HC_Ward and HC_AVG produce in the identical row/column list independent of their linkage method (T9, T10).
We conducted an expert study with one of the leading researchers in the field of matrix reordering algorithm design and gathered valuable comments and thoughts on the design decisions, computational- and work flow aspects (all tasks, but particularly T4-7). (3) We conducted an expert study with one of the leading researchers in the field of matrix reordering algorithm design and gathered valuable comments and thoughts on the design decisions, computational- and work flow aspects (all tasks, but particularly T4-7 and T8-10).

7 Evaluation

We designed GUIRO to support users with a wide range of visualization and algorithm literacy to explore matrix reordering as a topic. As discussed in the Background Section 3.2 and the Related Work Sect. 8, either automatic or interactive approaches were present to tackle subparts of this problem. We claim that no single technique or software fulfills all our outlined requirements. Based on our experience though we also believe that it is possible to solve most tasks by switching back and forth between systems, such as presented in [13, 32, 60, 73].

Following the “Patterns for Visualization Evaluation” [20], we designed our evaluation procedure to reflect the three persona (novices, network analysts, algorithm designers) introduced in Sect. 2: (1) In a qualitative user study, we examine the usability and understandability of the system as a whole for novices and general users, incorporating all tasks, but put our focus on (T1-3). (2) We present a case study evaluation inspired by a typical network analysis scenario, demonstrating our proposed interaction techniques on a real-world dataset (T4-7). (3) We conducted an expert study with one of the leading researchers in the field of matrix reordering algorithm design and gathered valuable comments and thoughts on the design decisions, computational- and work flow aspects (all tasks, but particularly T4-7 and T8-10).

7.1 User Study

Experiment Design: After a careful consideration, we decided yet against a formal comparative study between different systems, since this would only allow us to compare that systems like PermutMatrix (7), MatrixExplorer (2), or GUIRO (71) have a distinct ranges of applicable matrix reordering algorithms. MatrixExplorer, however, allows us to interact with the matrix (T4 - T6) and retrieve consensus among algorithms (T9). While we value the latter in GUIRO only as a minor contribution, a comparative evaluation to our proposed interactive reordering would have been interesting, but practically challenging.

Fig. 6: GUIRO allows algorithm designers to compare 16 quality metrics (T8), as well as the row/column index (dis-)similarity. We see that the hierarchical clustering algorithms HC_Ward and HC_Avg produce in the identical index list independent of their linkage method (T9, T10).

We claim that this is a typical result. Although the novice user might be challenged by understanding high-dimensional spaces and lower-dimensional projection techniques, they can still use them as a conceptual data analysis tool. For most questions we used a 5-point Likert-scale rating ranging from strongly disagree (1) to strongly agree (5). We analyzed the survey results using a graphic qualitative method (Box-Plot) and report here the median (x) and interquartile range (IQR). In order to compare the resulting ordinal distributions according to their population mean ranks, we applied the non-parametric Wilcoxon signed rank test. Our survey questions and answers can be found along with the analysis journal in the Appendix.
are not comparable, i.e., the subjects did not apply the same lasso selections. In UT3, which we estimated to be the most challenging task, all participants found a 100% match between OLO and OLO_Complete, but interestingly also agreed on the next two most similar rankings; a non-obvious choice. It will be interesting to correlate these rankings with quantitative ranking comparison metrics.

We asked the participants before ($\bar{x}=3$, IQR=1) and after the study ($\bar{x}=4$, IQR=1.25) to evaluate their confidence in assessing matrix patterns. Notably, their confidence increased on average by 1.27 points; the Wilcoxon signed rank test confirms this difference with a strong statistical significance ($p < .002$). Overall, the System Usability Scale (SUS) score was, as expected, on average $\mu = 62.95$ with a $\sigma = 14.629$; a typical result for a sophisticated Visual Analytics system.

### 7.2 Case Study on Submatrix Reordering

Figure 7 demonstrates the applicability of our approach by interactively improving the expressiveness of subparts of the matrix ($T4,6$). The dataset depicts a $236 \times 236$ matrix capturing social interactions in a primary school [23]. We begin our analysis with the Optimal-Leaf-Ordering algorithm in the single linkage variant (OLO_Single) [5]. Note, we depict all other reorderings in the Appendix. We interpret Fig. 7(1) globally ($T4$) and locally ($T5,6$) as follows: OLO tends to favor Block-Diagonals while having problems to grasp dense Off-Diagonal areas [7]. Nested patterns are not apparent.

Overall, we see a satisfactory reordering result ($T1$), i.e., the school’s classes and class levels, although our algorithm seems to have problems in the upper left quadrant, respectively its associated rows and columns ($T6$). We want to focus our analysis, hence, on the hidden submatrix structures. We apply Optimal-Leaf-Ordering—a hierarchical clustering extension—to our dataset. The algorithm finds many distinct clusters on the diagonal, but fails to reveal structures in the regions on the top right and bottom left. The Kernel PCA projection depicts one large cluster and two subclusters. A lasso selection of the top left projection cluster reveals that many of the fragmented rows/columns on the top left are related to an existing, already coherent, block in the matrix. Reordering this submatrix with Multiscale has a major influence on the visual quality in the top right of the matrix, but reflects only mildly on the off-diagonal structures in this quadrant. Selecting and reordering the other projection cluster (for completeness) has only minor impact on the overall matrix appearance.

The third improvement idea relates to the guidance of users in the...
exploration of (local) patterns. The expert stated the vision, that users should be able to see descriptive, local highlights for automatically retrieved patterns. We support this vision; however, localized matrix pattern descriptors remain currently a research topic. One way to educate users in GUIRO, though, will be to provide a thorough selection of tutorial datasets each depicting one prominent visual pattern.

8 RELATED WORK

For didactic purposes, we discussed the gap in the research literature throughout the paper (e.g., in Sect. 7) and embedded GUIRO into matrix research field in Sect. 3.2. This Section should give the reader a broader context, focusing on advancements in algorithmic and interactive relational data analysis.

8.1 Automatic Relational Data Analysis

Relational data analysis and visualization is unarguably challenging. Not only the dataset sizes range between a few hundred to many million or even billion items, but rather the linking and complex relationships among items is in the focus of the analysis [76]. These relationships can even be multivariate in nature or can change over time [6]. The research community devotes a significant portion of its efforts into finding solutions for typical analysis tasks, such as graph partitioning, connectivity or similarity analysis, finding relationships in multi-variate or high-dimensional spaces [42, 58, 77]. These approaches require not only sophisticated algorithmic techniques but also elaborate solutions for understanding the results. While the algorithmic field shows promising advances with, e.g., deep convolutional neural networks [16, 38, 45], the graph visualization domain is primarily concerned with externalizing graph topology characteristics [4, 34, 68] or designing effective overviews for large graph datasets, i.e., solving the hairball rendering problem [32, 35, 78].

Our proposed methods rely on projection to depict and interact with matrix re reorderings. Many techniques exist to project high-dimensional data to 2D space [21, 59] and support the exploration process in data analysis tasks [84]. Most notably, our approach is similar to Harel and Koren’s work on Graph Drawing by High-Dimensional Embedding (HDE) [30]. In HDE, the positions of vertices are first computed in an HD space and then projected to 2D/3D for drawing. The projection technique is supposed to retain HD structures and reflect them in the low-dimensional space. However, HDE is well-known to be fast and frequently leads to unsatisfactory results. We still believe that the approach is valid and thus allow our users to examine different projection techniques. We share this claim with Turkay et al., who present a Representative Factor Generation, i.e., a pattern-finder for HD spaces [72]. Here, the global projection shows the overall HD space topology, while local artifacts can be resolved visually by user intervention.

It is also known that the humans’ performance on a visually presented Traveling Salesman Problem (TSP) decreases non-linearly with the number of displayed points [75]. While our projection-space interactions must also lead to a Hamiltonian path, we follow a VA approach. We act on the user’s intervention complete the path to obey our requirements. Consequently, our offered algorithmic support reduces the cognitive load to solve a TSP problem.

8.2 Interactive Matrix Visualizations

For many users, matrix representations are less intuitive than node-link diagrams, and should be augmented with interaction and other visualizations to increase user understanding [53]. In MatLink [33], matrices are enhanced by overlays to improve the performance of path finding tasks in large networks. In GeneaQuilts [11] and Compressed Adj. Matrices [18], the matrix’s symmetric structure is compressed and rearranged to guide the user to interesting aspects in layered networks. Semantic zoom interaction can help to navigate matrices which do not fit into the available screen space. In several papers, zooming and dynamic aggregation techniques support the user navigation process in large matrices [3, 9, 19]. Also recently, Wong et. al noted in [82] that the ease of understanding for node-link diagrams and the information density of matrix representations show a mutual benefit.

9 DISCUSSION AND FUTURE WORK

When compared to the related work on interactive matrix reordering our approach goes beyond the state-of-the-art by incorporating three main aspects that potentially lead to a reordering improvement in one visual analysis system. Firstly, our system guides the user to submatrices, which could potentially be improved by showing row/column similarity in a novel projection view. Secondly, the user may choose to apply an arbitrary automatic matrix reordering algorithm on selected submatrices. We help the user by showcasing thumbnails of the local reorderings to help to anticipate the operation’s outcome. Thirdly, the user is free to rearrange the rows and columns—or groups thereof—manually. In addition, our approach allows the hierarchical construction of new matrix reordering algorithms, by applying local optimizations on a global matrix reordering result.

Currently, our approach is restricted to symmetric matrices as they are common in adjacency matrices for undirected graphs or similarity matrices. In the future, we plan to generalize our approach to non-symmetric and even to non-square matrices. Furthermore, we are researching other algorithms, such as clustering, to improve the reordering interactions and guide the steering process. Heuristic graph-based algorithms can be applied to find (near-)optimal linear arrangement paths in either a group of selected vertices or after a manually reordering decision. The question which projection technique is able to reflect the HD structures is addressed in GUIRO pragmatically: We allow the user to choose between 10 implemented projection methods. We found, though, that linear projection techniques are easier to interpret, while non-linear techniques reveal our algorithm shortcomings patterns more easily. It would be interesting to screen the projection view appearances automatically for the patterns described in Sect. 4.2.

In terms of scalability, our system is primarily limited by two design choices: First, in order to allow a wide audience to use GUIRO, we decided for a web-based client/server architecture. This has significant performance implications since our bottleneck is the necessary serialization and marshaling for transmitting our data. Right now, we implemented consistently REST interfaces to access our servers. In the future, though, we will be able to process matrices larger than a few hundred rows and columns by changing the inter-service communication to binary and compressed transmission formats, such as gRPC, Protobuf, or Apache Thrift. Second, our matrix visualization implementation is SVG-based and consequently lets the HTML Document Object Model (DOM) grow exponentially in size with the number of rows/columns in the input data. An obvious solution will be to replace the renderer with an HTML canvas-based or WebGL-based alternative.

10 CONCLUSION

Matrix data are gathered in many application scenarios. However, only a suitable row and column reordering allows the perception of underlying matrix patterns. While previous work centered on fully-automated reordering algorithms, we propose with GUIRO a Visual Analytics approach. During the process, users can apply global matrix reorderings, while steering them in the local context. By applying several, potentially varying algorithms to distinct local submatrices, the users can construct their own new matrix reordering composition.

In our web- and microservice-based system, we introduce a novel technical solution for gathering user feedback in the process. The projection-based reordering representation allows us to understand and perceive the decisions made by the black-box matrix reordering algorithms and gives rise to foreseeable interactive intervention.

We evaluated our approach in a user study with 12 participants to test the usefulness of GUIRO, a case study-driven evaluation on a social network dataset to demonstrate local reorderings in a global context, and conducted an expert study to validate our approach from an external point of view. Our results reveal, that the presented interaction mechanisms prove to be effective and enhance user understanding, while the matrix reordering composition is powerful, but hard to understand.

In conclusion, the presented visual analytics system serves as a basis for a user-steerable matrix reordering process and allows to understand and steer the black-box matrix reordering algorithms.