Graph2DMatrix: Exploring Graph Neural Networks Ability to Visualize Multivariate Graphs with Reordered Matrices

Job Heuvelmans (5753066) *

Utrecht University, Department of Information and Computing Sciences, Princetonplein 5, Utrecht, The Netherlands.
Master Business Informatics.

Version of May 1st, 2022

Universiteit Utrecht

*E-mail: j.heuvelmans@students.uu.nl
First Supervisor: Dr. M. Behrisch
Second Supervisor: Dr. T. Mchedlidze

Abstract

Previous research in graph embedding concentrates mainly on using embeddings for downstream machine learning tasks such as node classification, edge prediction and, to a lesser extent, on visualizing these embeddings for analytical examination. This study aims to determine whether high dimensional graph embeddings can be used to uncover structures in graphs, and visualize these in two dimensional matrices. We propose a framework that embeds a graph in high dimensions; calculates the pairwise distance matrix; reorders rows and columns in this matrix; and visualizes the original graph in a new matrix exploration tool. The goal of this framework is to supply individuals with high level knowledge on relational data. We test the framework by analyzing visual quality by feeding in basic pre-generated graphs. The random walk algorithms (e.g. DeepWalk, Walkets and attentionWalk) are able to accurately visualize 4 out of 6 of the canonical data patterns for a high level understanding of the data. Nevertheless, these basic graphs do not reflect complex relational data used in many real world applications, and therefore, we introduce two novel algorithms for embedding numerical node-attributed graphs (i.e. featPMI and featWalk). These algorithms are tested on a subset of the attributed Slovakian social network Pokec, in which both the algorithms show increasing information retention over the naive embedding of DeepWalk. Furthermore, featWalk is found to be preferred over featPMI with a clearer separation of patterns, and better feature preservation. Our findings indicate the potentiality of embeddings to generate valuable high level matrix visualizations.

keywords: *graph embedding, graph visualization, random walk, seriation, matrix reordering, multivariate graphs*

Contents

1 Introduction

With the ever increasing need for larger visualizations of relational data, it becomes difficult for most existing methods to visualize node-link diagrams of over 140,000 vertices [1]. Elmqvist et al. [1] explain that a further increase in the size, density or attributes of graphs will lead to occlusion and a decreased readability of graphs [2, 3]. One of the suggested solutions for avoiding this large graph visualization problem is to encode graph structure in matrix form [2], which could additionally encode supplementary attributes. A recent comparison by Ghoniem, Fekete and Castagliola [4], and Keller, Eckert and Clarkson [5] found that the readability of graphs favored a matrix-based representations over node-link diagrams for large graphs. Nevertheless, these matrices are typically not very informative to the user if the arrangement of its rows and columns does not show any structure [6, 7]. When ordered appropriately however, they can show high level patterns more clearly in much larger graphs than node-link diagrams [1]. To the best of our knowledge no matrix reordering algorithm exists for ordering attributed graphs. Therefore, the purpose of this study is to investigate whether Graph Neural Networks can obtain useful embeddings for weighted (1), non-symmetric (2) and multivariate networks (3), to be used in constructing a visually 'useful' matrix. An example of increasingly useful matrix re-orderings is given in figure 1, with the first image consisting of no structure and the last image of highly visible structure. Ideally for comparison of algorithms, this row and column reordered matrix should be comparable on a quantitative and qualitative level to other matrix representations of the same graph.

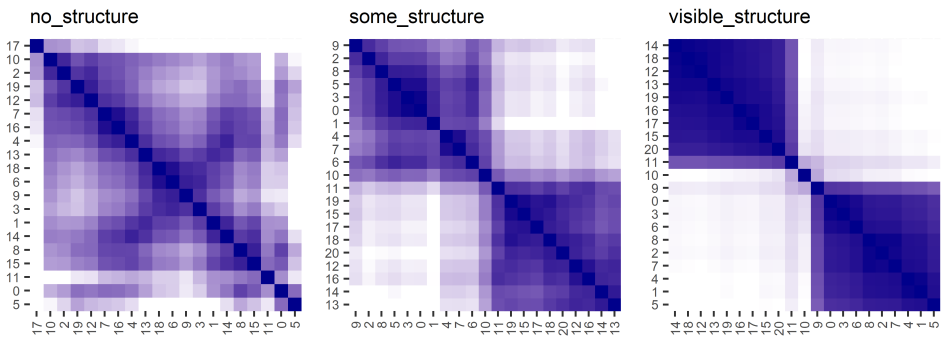


Figure 1: Matrix series with x-axis increasing interpretive quality

Goyal and Ferrara [8] argue that embedding graph data to the vector space is a necessity for increased usability and practicality of large graphs. For that reason, in our work embeddings generated by Graph Embedding

Algorithms will be used in conjunction with a seriation technique to derive a good matrix order. In this reordering high-level patterns should manifest and they should inform the user. But for all that, are graph neural networks even able to learn useful embeddings for a visually comprehensible matrix reordering? To answer this question, a resulting matrix reordering that is interactive (e.g. zoom, pan and move) and steerable would be of great value in this exploratory analysis.

The most important, evaluation metrics for comparing these matrices mainly involve the visual quality patterns in Behrisch, Shreck and Pfister [7]. Quantitatively however, the results will be tested by comparing the re-orderings on several quality metrics as in [7, 8], with the most important metric for our research being the Hamiltonian Path Length. An alternative to this approach is using the common metric minimum Linear Arrangement (LA) scores [9] to exploit block patterns as in [6, 10].

1.1 Problem Statement & Research Questions

The aim of this research is to explore whether graph embeddings can be used to visualize recognizable patterns in row and column reordered permutations. These visualizations should contain practical patterns such as the Wilkinsons [11] canonical data patterns and binary patterns of Behrisch et al. [6]. These patterns give interpretation of a graphs structure in the matrix visualization and are explained in more detail in section 4.6.1. Therefore, the main research question is aimed at fulfilling the visualization need for large graphs, and it intents to do so by exploring whether graph embeddings can be used to discover understandable patterns in the data. The main research question is therefore:

- **Research Question:** *How can we use vector point*[1] *Graph Neural Networks to derive a latent space for effective visualization(s) of complex graphs in a matrix?*

To answer this main research question five supporting sub questions have been defined:

- **Sub Question 1:** *What are the different types of vector point Graph Neural Networks?*
- **Sub Question 2:** *How to project a high dimensional embedding into a two dimensional matrix?*
- **Sub Question 3:** *What is a visually comprehensible two dimensional matrix?*

[1] vector point GNNs are neural networks that embed graph nodes in the latent space [12]

- **Sub Question 4:** *What nodal information is retained by embedding graphs with vector point Graph Neural Networks, and what is lost?*
- **Sub Question 5:** *How could a vector point GNN generate useful projection(s) of a multivariate graph?*

To answer sub questions 1,2 and 3 a semi-systematic literature review will be performed in section 3. In addition, this section will contextualize recent work on graph embeddings, seriation and quality metrics. Sub question 4 will be answered by generating artificial graphs with specific properties, of which the visualizations will be analyzed. And finally, for sub question 5 experiments will be performed to explore how multivariate graph embeddings and their matrix visualizations can be interpreted.

1.2 Research Approach

This research uses the methodology by Munzner [13] for visualization design. The nested approach is used to design and evaluate effective visualizations.

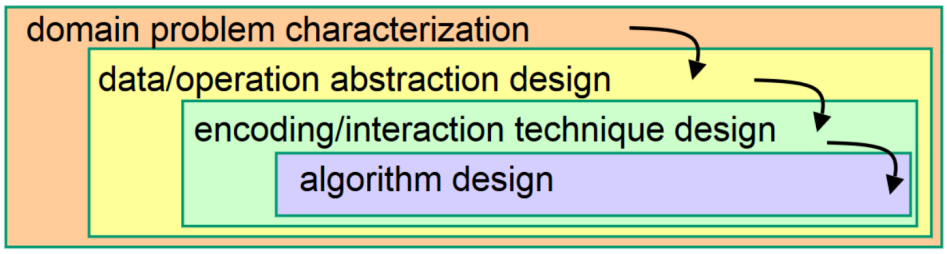


Figure 2: Nested model by Munzner [13]

The first step in this model is defining the domain problem and data characterization. For this research, the output would be to investigate the graph domain, the work flow from embedding to visualization, data types and matrix reordering described in sections 2 and 3. The next step is operation and data type abstraction, where the output is a rendition of the used data type(s) and operations to be performed (described in sections 4 and 7). Afterwards, the visual encoding and interaction design step defines how to present the data visually in section 4, and what type of user interactions can be applied in section 5. Finally, the algorithmic design step is used to create an algorithm for automatic graph embedding and visualization as described in sections 4 and 6. The aforementioned steps are summarized in figure 2.

1.2.1 Research Method

To obtain a good matrix representation of a complex graph it is often not enough to encode this graph in two or three dimensional space. Therefore,

Harel and Koren [14] suggest to embed a graph in high dimensions (e.g. 50) and, thereafter, reduce the dimensions to make visualization of the results possible. Unfortunately, a high dimensional object cannot be easily represented in low dimensions due to the simple fact that there is inherent confinement in the allotted space [14]. As a result, a non-trivial technique called dimensionality reduction is applied to the data which, if done incorrectly, can make the entire embedding useless. The practicality of any embedding rests on the assumption that the divergence between observations is meaningful for downstream tasks, or that the results are readable, if the goal is visualization.

In this thesis, the research approach is similar to that of the method introduced by Harel and Koren [14]. In their work node embeddings were obtained by creating n pivot nodes in the original graph. By calculating distances between these pivot nodes and all other nodes in the graph they essentially represent multiple dimensions for each distinct node. In contrast, in this thesis graph embeddings are generated by multiple graph embedding algorithms instead. Unfortunately, embedding in high dimensions as in [14] can be problematic since data becomes sparse in high dimensional space. In this high dimensional space concepts of distance and clusters can quickly become insignificant [15], and unfortunately, problems become worse whenever the dimensions increase. Nevertheless, the assumption that dimensions should be as low as possible (i.e. $d \ll |V|$), and choosing a good dimensionality reduction technique (i.e. DRT) should help mitigate this problem. Aggarwal, Hinneburg and Keim [15] show that distance metrics performance differs significantly in higher dimensions without applying a DRT beforehand. Note that the Manhattan distance (i.e. L1 norm) consistently outperforms the Euclidean distance (i.e. L2 norm) in higher dimensions [15]. The choice of including a DRT before calculating pairwise distances seems trivial in higher dimensions based on this study. Therefore, DRTs and its variants are discussed further in section 3.4. One notable DRT that was used in Harel and Koren [14] is PCA, but this DRT would likely lead to sub-optimal results in our study since linear PCA would nullify the intrinsic non-linearity of the embeddings.

1.2.2 Literature Protocol

The protocol of the literature review will be semi-structured since the main goal of this study is exploration. First, a few key domains and insights will be researched. Hereafter, a more unstructured study will follow to identify key themes, research gaps and model solutions in other fields.

For the structured part, a manual search will be performed by searching for papers on Google Scholar; searching on Github for implementations; and using R and Python documentations. The research protocol is given by figure 3 for the structured part. To get an understanding of the preliminaries

of the domain several articles will be read, whereafter the Research Question(s) and Scope of the project will be (re-)defined. To construct a search vocabulary a pre-search is done, after which the actual search for articles begins (with inclusion & exclusion conditions). After gathering and reading the literature, a written report of the natural divided parts will be made. And finally, key insights from the structured part in combination with thesis progression will be the guideline for adding unstructured literature.

The time period for the search will be 2000-2021 (with the exception of references to original papers). Nevertheless, TSP algorithms and heuristics have been developed far before this period and there are highly influential and essential papers from the 1950s onward thus these will be included. The keywords used in this search are enumerated below:

Keywords: *graph, embedding, graph embedding, matrix reordering, seriation, TSP, TSP algorithms, TSP heuristics, Deep Learning Graph Embedding, Random Walk Graph Embedding, Matrix Factorization Graph Embedding, Graph Neural Networks, GNN, dimensionality reduction, Dimensionality Reduction Techniques, High Dimensional Data, weighted graph embeddings, attributed graph embeddings, multivariate graph embeddings*

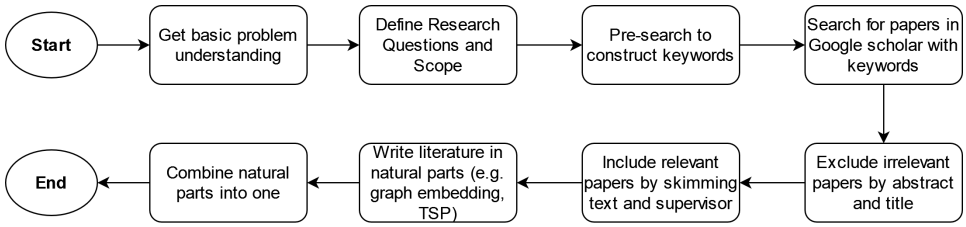


Figure 3: Research protocol

1.3 Relevance for Business and Society

Visualizing large scale and complex graphs are highly beneficial for understanding complex systems in scientific, business, healthcare and chemistry domains [16, 17]. By visualizing graphs it is possible to gain insights of the data that is hard to spot or previously unknown. In a business setting you enable an analyst to quickly identify outliers, trends, group structures and measure similarity. Some concrete applications could be the visualization of a social network and group similar societies, or visualize complex molecules and identify possible interactions, or visualize power grids to pinpoint weaknesses in a power grid. Visualizing graph data increases interpretations of the data in which complex ideas can be communicated in a single image [17].

Unfortunately, many of these real world graphs are very large and clever techniques need to be utilized to clearly visualize graph data. An option to visualize these graphs is to represent this data in matrix form and extract high level patterns. This option is ultimately scalable in graph size since it is possible to visualize just a single pixel per node!

Our vision is therefore to supply non-experts with value from relational data using high quality visualizations. Or in the acronym *G.R.A.P.H.*:

Great visualizations. value Realization. usable to Anyone.
everyday Problems. High-level insights.

1.4 Contributions

Combinations of the two literature streams named machine learning for graph embeddings, and matrix reordering led to new insights. These streams are combined in this research by encoding a simple, weighted or multivariate graph in the latent space, reducing the dimensions and applying an ordering to gain potential high level knowledge and discover topological patterns in graphs. Researchers could increase their understanding of the black box of Graph Embedding Algorithms by making changes in the algorithms and visualizing these differences. The contributions of this paper are summarized below:

- Insight in the black box of graph embeddings
- Insight into existing quality metrics and why they are not applicable for comparing visualizations
- Two 'new' metrics for comparing visualization
- Crude debugger to visualize and combine embeddings
- Multivariate algorithms: featureWalk and featurePMI
- High level matrix visualizations of basic, weighted and multivariate graphs

2 Background

In this section the imperative preliminaries for this study will be established. There will be an introduction into the different types of graphs used in this study; several graph definitions; graph data representations and an insight into the final framework.

2.1 Graph Definition

In this section the reader will be introduced to several graph theoretic definitions which are useful for understanding this study. The definition of a simple graph follows:

Definition 1 (graph definition): *A graph G where G = (V,E) contains a node set* $V = \{v_1, v_2, ..., v_n\}$ *and an edge set* $E = \{e_1, e_2, ..., e_m\}$. *Edge* e_{ij} *connects two nodes* v_i, v_j, $\forall e \in E$.

According to definition 1 each edge represents a relationship and connects two nodes [18]. Definition 2 shows that an edge in a graph can be represented by a node pair:

Definition 2 (edge definition): *An edge e can be defined as an ordered pair of vertices* $e_{ij} = (v_i, v_j)$. *The weight of an edge w(e) = 1 if the vertices are connected and 0 otherwise. A graph is symmetric iff* $e_{ij} = (v_i, v_j)$ *and edge* $e_{ji} = (v_j, v_i)$ *where* $w(e_{ij}) = w(e_{ji})$, $\forall e \in E$.

An historical application of graphs are cities (i.e. nodes) and whether you can travel between them (i.e. edges). In this example, an edge can also represent a one-way connection, or more formally it can be 'directed' if we do not assume symmetry in the node-node pairs. For non-symmetry, each edge $e \in$ E where $e_{ij} = (v_i, v_j)$ and $e_{ji} = (v_j, v_i)$ the edge weights are not equivalent or, $w(e_{ij}) \neq w(e_{ji})$.

Graphs are not restricted to displaying relationships on a binary level, they are able to model problems in which weight between nodes is important. The binary city problem can be extended by introducing proximity between cities as non-binary weights. For this example, a weight of 0 is infinitely far apart, and weights larger than zero increase closeness. The weighted graph WG extends G in definition 3 by assigning non-binary edge weights w for all edges in E [18].

Definition 3 (weighted graph definition): *A weighted graph WG is defined as WG = (V,E,W). WG is a special case of G where* $\forall w \in W$ *lie in the range* $[0,\infty)$. *The weight of an edge w(e) defines connection strength, with w(e) = 0 being disconnected nodes, w(e) > 0 connected nodes according to some weighting scheme.*

Furthermore, certain attributes can be modelled into graphs to introduce the multivariate graph in definition 4. While it is possible to attribute both edges and nodes, this study will only consider node attributes. A multivariate graph is defined as a node attributed graph with a feature vector F for each node.

Definition 4 (multivariate graph definition): *A multivariate graph M is an extension on G where M = (V,E,F), where* $\forall v \in V$ *there is a vector* $f \in F$ *which specifies one or more attributes of v [19].*

In the running example our cities will gain attributes, think of population figures, demographics and land area.

2.2 Types of Graphs

First the reader will be introduced to the many types of graphs and their differences. This master's thesis main focus is on basic graphs, weighted graphs and multivariate graphs.

2.2.1 Basic Graph

The *basic graph* is the simplest type of network in graph analytics. A basic graph consists of nodes and the edges wherein each edge represents a connection between nodes. In this simple form, there can only be one edge between each node pair and none of these edges can have the same source/target node (i.e. no loops). An example of a simple graph is the well-know social network Facebook. In this network a node is represented by an individual on the website, and an edge is represented by friendship between two people. Evidently, a requirement for this graph to be a simple network is that a friendship is mutual and you cannot be friends with yourself. In figure 4(a) there is an example of such a network.

2.2.2 Directed Graph

An extension on this basic form is made by allowing the edges to have a direction, thus introducing *directed and undirected graphs.* A *directed graph (or digraph)* is a graph where the edge between two nodes flows one way, but does not necessarily flow the other way. A *digraph's* edges are usually represented by arrows pointing from the source to the target vertex. An example of such a graph is a flow diagram, where processes are vertices and flows are edges. One can visualize such a graph as in figure 4(b). A simple graph is undirected.

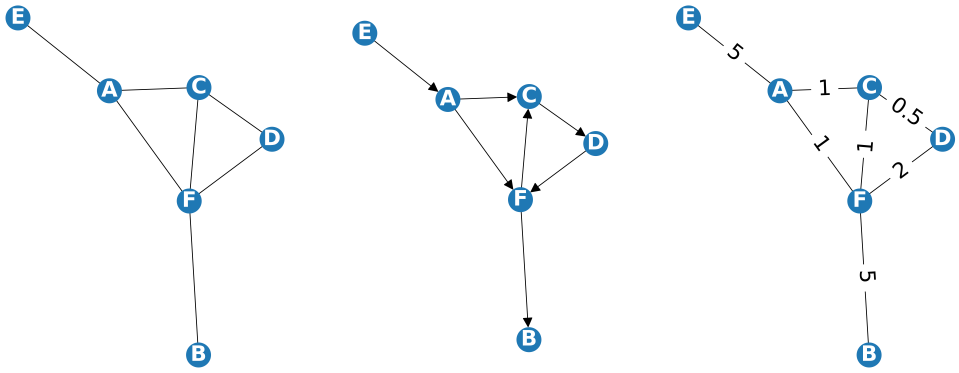


Figure 4: (a): Simple graph (b): Directed graph (c): Weighted graph

2.2.3 Weighted Graph

To increase complexity an extra variable could be introduced. By weighing the edges in a graph, the distinction between *weighted and unweighted graphs* is made. A *weighted graph* is a graph in which the connections between nodes are of different types of strength. The higher the weight on an edge the greater the 'cost' (or benefit, depends on definition of weights) to move from the source to target node. An example is a public transport network wherein edge weights imitate the time it takes to get from one city to the next, as represented in figure 4(c). A simple graph is unweighted.

2.2.4 Heterogeneous Graphs

Then, by allowing a graph to have different types of edges and/or nodes in a graph several types of *Heterogeneous Graphs* can be introduced. An example is given in figure 5(a) wherein nodes represent different life forms. Red nodes are omnivores, orange nodes are herbivores and green nodes are plants. Not only do the node types differ, also the relationships between them varies. In this example there are two types of edges, these are eating and fighting relationships. Note that many different types of *Heterogeneous Graphs* (or *Multilevel Graphs*) exist [20]. These graphs are out of the scope of this paper and will not be discussed further. A simple graph is homogeneous.

2.2.5 Multivariate Graph

And finally, the arguably most complex graphs are graphs with attributes called *multivariate graphs* or *attributed graphs*. These graphs can have any type of additional information next to nodes and edges, think of images, variables and/or (un)structured text. An example of a *multivariate graph* is given in figure 5(b) where nodes are cities and the attributes are represented by text. In this example, the cities that have more connections are more

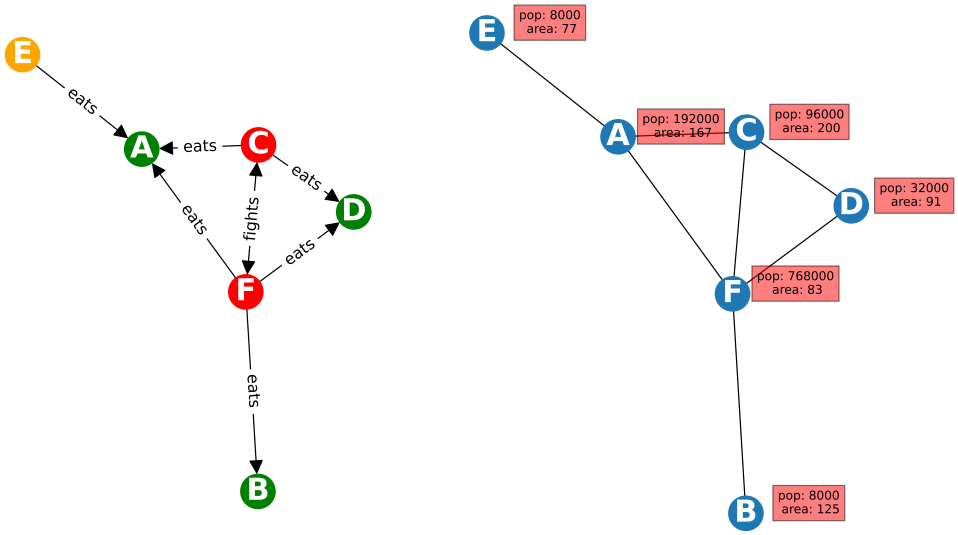


Figure 5: (a): Heterogeneous Graph (b): Multivariate graph

central and have a higher population. A simple graph does not have any variables.

2.3 Graph Data Representations

In this section an enumeration of possible different graph representations that are utilized in graph embedding algorithms are discussed. The current input of choice for this framework is the edge list.

2.3.1 Edge list

Simple graphs can be represented by an $|V| \times |2|$ *Edge List* L(G) where each entry l_i in L represents an edge as a node pair between vertex v_i and v_j. An example of an edge list is shown in figure 6, wherein among others, an edge exists between node 0 and 1. For directed graphs, v_i is the source of an edge and v_j is the target of the edge. And for weighted graphs, a third entry $w(e) > 0$ for all edges is introduced and indicates the weights between these two vertices. In an edge list it is easy to loop over all edges, but some operations such as finding the degree of a node is relatively hard.

2.3.2 Adjacency Matrix

An alternative to the edge list is the $|V| \times |V|$ *Adjacency Matrix* A(G) shown in figure 7. In this matrix A, each entry a_{ij} indicates whether node v_i, v_j are connected or not. For undirected graphs, the adjacency matrix is symmetric. For weighted graphs the entries can be any real number, or $a_{ij} > 0$.

source	target
0	1
0	2
0	3
1	2
2	3
...n	...n

Figure 6: Edge list

	v1	v2	v3	v4	v..
v1	0	1	1	0	...
v2	1	0	0	0	...
v3	1	0	0	1	...
v4	0	0	1	0	...
v..	...	...	...	...	...

Figure 7: Adjacency matrix

2.3.3 Incidence matrix

A relatively similar matrix can be fabricated with the $|V| \times |E|$ *Incidence Matrix* I(G). Each entry i_{ij} in matrix I indicates whether vertex v_i is adjacent to edge e_j. An incidence matrix usually exists of three values, -1 for incoming edges, 0 for no edges and 1 for outgoing edges, see figure 8 for an example.

	e1	e2	e3	e4	e..
v1	1	0	-1	1	...
v2	-1	1	0	0	...
v3	0	-1	1	0	...
v4	0	0	0	-1	...
v..	...	...	...	...	...

Figure 8: Incidence matrix

2.3.4 Adjacency list

A hybrid between an edge list and an adjacency matrix is the *Adjacency List* visualized in figure 9. An adjacency list is an array of $|V|$ lists with a space complexity $O(V + E)$. Each linked entry $d_{i...n}$ in adjacency list D(G) is a list of vertices v_j to $v_{j...n}$ connected to v_i with key v_i. For example, in this figure node 1 is connected to node 0 and 4. This representation benefits over the adjacency matrix due to the sparsity of most graphs since $O(V + E) \ll O(V \times V)$. Another advantage of this representation is that it allows for more efficient node operations (e.g. inserting and deleting nodes).

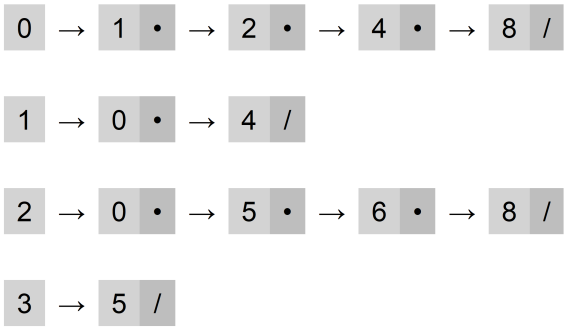


Figure 9: Adjacency list

2.4 Pairwise node preservation principles

In graph embedding algorithms it is paramount to measure nodal similarity in order to construct an embedding that preserves and optimizes proximity between nodes. Graph embedding algorithms differ in the way they calculate these pairwise similarities [21]. Nevertheless, a few of the most well-known and regularly used measures in graph embedding algorithms are enumerated below. The following definitions of first and second order proximities are based on the paper by Cai, Zheng and Chang [21].

The *first-order proximity* in definition 5 is a measure of closeness between pairs of nodes. This is essentially equal to the weight of an edge between two nodes, and values the local structure in a graph.

Definition 5 (first-order proximity): *The first-order proximity between a pair of vertices v_i, v_j is defined as the weight of the edge between two vertices $w(e_{ij})$ where $e_{ij} = (v_i, v_j)$. In a basic graph, the first-order proximity $\forall e \in E$ is equal to 1 if an edge exists and 0 otherwise. The first-order proximity for weighted graphs differs, $\forall e \in E$ is equal to $w(e_k)$ if an edge exists and 0 otherwise.*

The *second-order proximity* tries to capture node similarity by comparing the neighbourhoods of two nodes. The neighbourhood of a node is the set of adjacent nodes. Nodes are more distant from each other if they do not share a similar neighbourhood.

Definition 6 (second-order proximity): *The second-order proximity compares the neighbourhoods of two nodes and constructs a similarity measure between them. $\forall v \in V$ there is a neighbourhood n. The neighbourhood n_x for node x is equal to all edges connecting to x or, $\forall v \in V$ where $[w(e_{xv}) > 0] \in n_x$. The pairwise second-order proximity s_{ij} between two nodes is a similarity measure (e.g. cosine distance) between the neighbourhood of node i and j : n_i and n_j. The second-order proximity is defined as $s_{ij} = cosine(n_i, n_j)$.*

Most of the algorithms discussed in this paper will preserve either first, second order proximities, or both. There are even *higher-order proximity* preserving algorithms out there (e.g. Higher-Order Network Embeddings or HONE [22]). This paper will not go into detail about these algorithms. Nevertheless, it might be possible to preserve higher order proximities by increasing the size of the neighbourhood, according to Xu [12]. In addition, the authors of [8] mention the use of different metrics like Common neighbours, Katz Index, Academic Adar and Rooted PageRank to approximate higher-order proximities [23].

2.5 Mapping Graph Data

In this section a general outline for retrieving an embedding and mapping these embeddings to lower dimensions is discussed. The output of a nodal graph embedding is defined as $|V| \times d$ latent vector representation of all nodes v in a graph G, with dimensions d. Each node in the graph has a latent vector representation with the same dimensions and preserves one or more preservation principles. Definition 7 shows the definition of embedding a graph:

Definition 7 (embedding a graph): *All nodes in a graph are represented with dimensions d (i.e. features), and mapped with a non-linear function* $f_v : V \rightarrow \mathbb{R}^d$. *The result is an array EM of size* $|V| \times d$ *and represents each node in the same latent space.*

The latent representation from definition 7 is transformed to a 2-dimensional vector space by reducing the original embedding to a 2-dimensional embedding. In definition 8 this mapping to lower dimensions is defined.

Definition 8 (mapping to lower dimensions): *A 2-dimensional array* EM_{2D} *is derived from the EM in definition 7, where a (non-)linear transformation* $\mathbb{R}^d \rightarrow \mathbb{R}^2$ *transforms dimensions d to d=2.*

As a last step, the newly acquired embedding array is transformed to a distance matrix by calculating pairwise Euclidean distance for each node pair.

2.5.1 Visualization

The resulting graph embeddings in definition 7 are transformed to two dimensions as defined in definition 8. Each node can be represented in a two dimensional vector space by transformation with a distance measure or a dimensionality reduction technique, or both (more on this in section 3.3 and section 3.4).

Consequently, a distance matrix is derived and in this matrix the horizontal and vertical axis represent the nodes in a graph. Unfortunately, the derived matrix is usually of relatively little value for human exploratory analysis. This is due to the fact that it is extremely difficult spot patterns in a unordered matrix visualization. Therefore, this matrix is ordered by a seriation algorithm (more on this in section 3.5.1) to shuffle the rows and columns and obtain a more comprehensible result.

3 Related Work

The literature section builds upon the preliminaries specified in section 2. It introduces the reader to the relevant work in order to construct the framework and follows a similar order as the structure of this framework.

Section 3.1 and section 3.2 discuss a brief history of graph embeddings and why these embeddings are relevant for science and society. They introduce some of the challenges and solutions that graph embeddings and their corresponding algorithms face.

After this introduction, an understanding follows for why graph embeddings are hard to acquire, and their relative importance is explained. A discussion on different types of graph embedding algorithms follows in section 3.3. In this section, the different families of embedding algorithms are discussed along with some of their advantages and disadvantages. Furthermore, an enumeration of graph embedding algorithms used in this paper succeeds.

Additionally, in section 3.4 the reader is introduced to dimensionality reduction for displaying a high dimensional embedding in a low dimensional matrix.

Thereafter, to derive a good matrix order, section 3.5.1 will explain some history on matrix reordering algorithms. This section specifically zooms into TSP algorithms and their heuristics.

Afterwards, section 3.6 shows some of the quality metrics for determining the quality of a reordering. In this section the framework for comparing embedding algorithms to each other quantitatively is introduced.

And finally, a conclusion with the main findings will be given in section 3.7.

3.1 Graph Analytics

Graphs are common structures to represent complex real-world data and relationships. They model a set of objects and indicate whether these objects are related in one way or another. Hence, a graph can be used to model a wide variety of applications, think of social circles (e.g. friendships and people), web graphs (e.g. hyperlinks from website to website) and road networks (e.g. cities and roads). As a consequence, these structures can be exploited to gain insight in the irregular non-linear spaces to analyze higher level patterns in the data [8, 12]. By structurally employing graphs to represent real-world networks, it is possible for researchers to gain a lot of knowledge about the relationships between network entities [24].

Commonly, most problems on graphs are solved by directly applying a model on a graph, the adjacency matrix or a vector representation of a graph [8]. But, at first, researchers developed handcrafted rules to extract useful information from the adjacency matrix of a graph [25]. Unfortu-

nately, the structure of graphs are rather complex, and when they started to grow in size, most existing methods quickly became impractical. These earlier methods had relatively high time and space requirements, and thus performing computation on larger graphs or adjacency matrices directly is often undesirable. A solution for this problem was introduced in the early 2000s when graph embedding algorithms gained traction [21]. By assuming that the data could be mapped to a lower dimensional manifold these graph embedding methods tried to reduce the high dimensionality of graph data [21]. This time period saw the introduction of embedding algorithms like Locally Linear Embeddings [26] in 2000 and Laplacian Eigenmaps [27] in 2003. A decade later in the 2010s, graph embedding algorithms gained a lot of attention again due to wide applicability of graphs in many different fields. During this time period there were many advances in embedding algorithms. An example was the possibility of encoding additional information in an embedding [21]. Another influential example is the introduction of random walk graph embeddings in 2014 with DeepWalk [28], which marked a decisive event where embeddings could be approximated even cheaper than before. Therefore, many successive methods based their design on this popular approach.

Embedding algorithms aim to effectively preserve graph information for downstream tasks by mapping a high dimensional graph to a lower dimensional vector [8, 12, 21]. The most common tasks are node classification, link prediction, visualization, node clustering and graph classification [8, 21].

There exists many different types of graphs in graph analytics, with the most basic being the *basic graph*. In section 2.2 there is an enumeration and explanation of different graph types.

3.2 Representation learning

Traditionally the problem of graph embeddings falls in between two existing research streams [21]. With on the one hand feature learning, where a system tries to uncover a more useful representation of the data, by the means of examination, and uses this information in downstream tasks [29]. And on the other hand, there is graph analysis where the objective is to analyze different relationships in graph data to extract useful information [25].

The inherent challenges of graph embeddings can be attributed to the fact that graph embeddings try to represent a complex network in low dimensional vectors, while trying to preserve most information. This raises several challenges namely, what is the ideal dimensionality of the embedding, which features to preserve and is the method scalable [8]? The choice of vector dimensionality is directly related to the effectiveness of the algorithm in terms of time and space complexity. In real-world graphs there are often many millions of nodes and thus scalability is of utmost importance. Therefore, it is often proposed to encode a graph in a vector dimensionality

d that is much smaller than the number of nodes in a graph, or $d \ll |V|$.

Unfortunately, embedding algorithms will no longer be able to preserve all graph properties and relationships if a very low dimensionality is chosen. A reduction in precision is inevitable in this case and thus an embedding dimensionality has to be chosen with great care.

Furthermore, graphs can be embedded in four different dimensions of coarseness, which are defined as node, edge, graph and substructure embeddings [21]. The application and problem should determine which embedding granularity to use. Node embedding algorithms try to estimate similarities between nodes by placing nodes that are more similar closer to each other. This paper has node embeddings as a focal point.

The algorithm that was used to derive them these embeddings can be further refined into three different families discussed in section 3.3. Furthermore, algorithms differ by their preservation principles (as explained in section 2.5) and they have a trade off between precision and time/space complexity [21]. The next subsections and table 1 describe three families of graph embedding algorithms.

3.3 Graph Embedding Algorithms

Graph embeddings algorithms can be split into three distinct groups, these are Matrix Factorization (i.e. MF), Deep Learning (i.e. DL) and Random Walk (i.e. RW) approaches [8]. The methods in these groups can be either deterministic or stochastic in their node to vector mapping. Most MF methods are deterministic, whereas DL and RW algorithms are stochastic by design [12]. The algorithms in this paper will all have similar matrix output(s) as defined in definition 7, where only the dimensions between representations differ. This makes the choice of algorithms independent for downstream Machine Learning tasks, when not considering differences in time and space performance.

Embedding algorithms can be divided into vector point, Gaussian distribution and dynamic graph embeddings [12]. In vector point graph embeddings, embeddings are transformed to multiple d dimensional vectors, where the dimensions describe some features for all nodes. Common node similarities measures for (low dimensional) vector point embeddings are the dot product, cosine distance, Euclidean distance, Manhattan distance, and Mahalanobis distance.

Gaussian distribution based embeddings on the other hand, model every node as a distribution to capture inherent uncertainties in the data [30]. Node similarities in these models are usually measured by a divergence measure such as the Kullback–Leibler divergence between corresponding distributions.

And finally, there are dynamic graph embeddings which can learn representations of continuously adapting graphs and are able to learn temporal

patterns without retraining. Adaptations to the original graph can include adding new nodes or removing connections between existing nodes. The methods in this class are able to predict temporal behavior on graph data [31]. Examples of such algorithms are in the paper and python library of Goyal et al. [31].

To stay in the scope of this research all of the algorithms in this framework are vector point based. A list of the implemented algorithms is shown in table 1. Notice that most graphs have fewer nodes than edges (i.e. total degree is often larger than 1), and embedding dimensions are usually similar for the implemented algorithms. Table 1 shows time complexity of DeepWalk is $O(|V|d)$ versus Laplacian Eigenmaps time complexity of $O(|E|d^2)$. Therefore, scalability of random walk algorithms (except for struc2vec) is usually better than factorization and deep learning approaches.

Approach	Method	Preservation	Time complexity
Random Walk	Deepwalk	1st	$O(\|V\|d)^2$
	node2vec	1st and 2nd	$O(\|V\|d)^2$
	struc2vec	Roles	$O(\|V\|^3kd)^3$
	attentionWalk	1st and 2nd	$O(\|E\|d)^4$
	Walklets	Higher-order	$O(\|V\|d)^5$
Deep learning	SDNE	1st and 2nd	$O(\|V\|\|E\|)^2$
Factorization	HOPE	Higher-order	$O(\|V\|^3d)^4$
	Laplacian Eigenmaps	1st	$O(\|E\|d^2)^2$
	LLE	1st	$O(\|E\|d^2)^2$

Table 1: Implemented embedding algortihms

3.3.1 Random Walk Algorithms

One of the key assumptions of Random Walk graph embedding methods is that it is possible to preserve the structure of graphs by performing 'random walks' along the structure [28]. For different algorithms, these random walks vary in their sampling of walks, and differ in the a priori assumptions made. The aforementioned walks serve as input for the embedding algorithms to derive a latent representation of the nodes in the original graph. This brings about embeddings which preserve nodal proximities, and some can even preserve certain neighbourhood characteristics. Random Walk algorithms are very competitive and are able to scale better than most other methods in this paper.

[2]Complexity of the implementation by Goyal and Ferrara [8]
[3]Complexity of the implementation by Ribeiro, Saverese and Figueiredo [32]
[4]Complexity of the implementation by Abu-El-Haija et al. [33]
[5]Complexity in paper by Rozemberczki, Allen and Sarkar [34]

A description of the implemented RW algorithms in the framework will follow:

DeepWalk: DeepWalk [28] is the original random walk algorithm, where most if not all other algorithms in this family are based upon. The aforementioned Graph Neural Network focuses on optimizing first-order proximity (i.e. local similarity) between nodes, whereupon more locally similar nodes are embedded closer in the embedding space. The algorithm generates random walks, which are constructed randomly by moving from the origin node to one of its neighbours, and stacking these nodes to the existing path for a specific length. The algorithm 'walks' through a graph several times to get a local representation and reveal latent patterns in the data [28]. After constructing all of these random walks, the authors of Perozzi et al. [28] consider the graph and its nodes as words, and utilized the word2vec [35] algorithm within the graph domain. Consequently, random walks in DeepWalk are considered as a sequence of nodes in a path (or, a sequence of words in a sentence) and a fixed context window (e.g. CBOW or SkipGram [35]) is applied. Nodes that appear together in this context window are more likely to be similar [28].

node2vec: The node2vec [36] algorithm is similar to DeepWalk but it introduced the concept of biased walks. The objective function to optimise is directly reliant on the concept of a what constitutes a neighbourhood. In node2vec this neighbourhood definition is explicitly defined by two parameters. The return parameter p and in-out parameter q differentiate between local and global neighbourhood characterstics [36], which is also known as the exploration-exploitation tradeoff. A high (> 1) value for p indicates that the walks are more biased towards exploration, and on the other hand, a high value for q indicates a more locally biased search. A special case exists where node2vec is essentially the same as DeepWalk and that is when $p = 1, q = 1$ [36].

struc2vec: Struc2vec [32] is a graph embedding method that learns node representations from the global network structure, and from node to node relationships in a graph. This method utilizes the same framework as methods like node2vec and DeepWalk, but it differs fundamentally in its execution. In contrast to DeepWalk, struc2vec does not capture the proximity of nodes by distance, it rather expresses similarity by grouping nodes based on their role in a graph. This concept is called structural identity, where nodes can be equivalent to each other independent of their position in the graph. In order to generate embeddings, struc2vec establishes a progressive hierarchy of similarity in which the definition of being pairwise structurally equivalent differs. To create this hierarchy the algorithm uses the degree of nodes as one of the important features for calculating this similarity. As a final step, weighted random walks along a constructed hierarchical multi-layer graph are performed, where the contexts for all nodes

are constructed. If two nodes share a similar context they must be similar to each other [32].

attentionWalk: The RL method attentionWalk [33] is different from the other algorithms because of the usage of the attention mechanism. The attention mechanism steers the random walk in a certain direction by using distance to the original node as a metric [33]. Furthermore, attentionWalk also introduces trainable instead of fixed model parameters that can be automatically learned via backpropagation [33]. This method shows good preliminary results and the authors of Abu-El-Haija et al. [33] report a reduced link prediction error by 20-40% compared to other state of the art algorithms.

Walklets: In contrast to DeepWalk, which learns one representation of relationships between vertices, a new method was proposed by Perozzi et al. [37]. In the Walklets [37] algorithm a sequence of latent relationships between nodes is constructed where each successive representation expresses higher-order proximity. By skipping over steps in each sampled random walk (i.e. hop distance) a higher power adjacency matrix is formulated, which leads to higher-order walklets and thus higher-order relationships between nodes [37]. The authors of Walklets report a 10% increase in performance when compared to DeepWalk.

3.3.2 Deep Learning Algorithms

Deep Learning methods aim to capture the inherent non-linearity in graphs by creating a robust architecture that is able to find effective solutions in low-dimensional space. Most algorithms in this family use multiple layers in a deep neural network to embed an adjacency matrix directly [8]. Unfortunately, methods in this family usually have high computational cost compared to random walk methods.

Structural Deep Network Embedding: Wang, Cui and Zhu [38] argue that complex relationships in the structure of a graph cannot be effectively modelled with shallow models. They propose SDNE [38] a semi-supervised deep learning solution for embedding complex graph data in the non-linear latent space. The first part of SDNE consists of an unsupervised auto encoder which tries to embed nodes by many non-linear functions. The second part of SDNE is supervised and applies a penalty for nodes that are similar but embedded far from each other (similar to Laplacian Eigenmaps discussed in the next section). In this method there is explicit use of an objective function and the first order proximity (i.e. local network structure), and second order proximity (i.e. global network structure) are optimised in unison [38].

3.3.3 Matrix Factorization Algorithms

Matrix factorization was the first school of algorithms to pioneer in solving the graph representation problem. The idea behind these methods is to represent a graph in the form of a matrix, which can be of many forms (e.g. adjacency matrix, graph Laplacian). The next step is to derive an embedding by choosing an appropriate factorization method which will be applied to this generated matrix [8]. Unfortunately, due to the usage of large matrices, these methods are often unable to scale well to larger graphs, with time complexities that are often quadratic for the number of dimensions [21].

High-Order Proximity preserved Embedding: The HOPE [23] algorithm was developed to solve the asymmetric transitivity problem in directed graphs with the purpose of capturing structures of graphs more accurately. This method tries to approximate higher-order proximities by embedding the similarity matrix with Singular Value Decomposition. The similarity matrix is constructed by multiplying two polynomial matrices derived using different measurements. These matrices are fabricated by the usage of one of the several different similarity measures (e.g. Katz Index, Common Neighbours, Personalized Pagerank or Acadamic-Adar) [23].

Locally Linear Embedding: Locally Linear Embedding [26] is an unsupervised method which tries to preserve high dimensional non-linearities in a lower dimension. In contrast to other methods at the time, LLE modelled complex data to a lower dimension while preserving global geometry and neighbourhood similarity [26]. In the intermediate data, each node is a vector represented by a weighted linear combination of their k-nearest neighbours. Hereupon, a minimization of the loss function (i.e. optimizing the weights of the linear combination) maps all nodes to a lower dimension [26]. The input is different from Laplacian Eigenmaps method discussed next, where the Graph Laplacian is used instead of the adjacency matrix.

Laplacian Eigenmaps: Laplacian Eigenmaps [27] is a non-linear algorihm inspired by Locally Linear Embedding. The algorithm assumes that a set of high dimensional data points in Euclidean space lie among a lower dimensional manifold. It utilizes the graph Laplacian to estimate the low dimensional topological space of the data. First, the Laplacian matrix from the intial graph data is constructed, wherein edges represent geometrical similarity and the weights for this geometrical similarity can be represented by a simple binary scheme (e.g. 0 if there is no edge, 1 if there is) or any other scheme (e.g. Gaussian weights). To derive an optimal embedding, the Laplacian Eigenmaps algorithm computes the eigenvectors of the previously acquired Graph Laplacian matrix and assumes that the data, lies on a non-linear manifold. The Laplacian Eigenmaps algorithm only fo-

cuses on preserving first-order proximities by placing two similar nodes in the original data close together in lower dimensions, and is heavily penalised if two similar nodes are embedded far apart [27]. This algorithm is transductive and it is not able to map new points of data without retraining the model.

3.3.4 Multivariate Graph Embedding

All of the previously mentioned algorithms are basic graph embedding techniques. For weighted graphs embeddings the random walk algorithms can be easily adapted by adjusting their random walk sampling technique. For other techniques the implementations of Goyal and Ferrara [8] include a weighted version.

Unfortunately, embedding multivariate graphs is much harder since there exists many different types of multivariate data. Text-associated DeepWalk (i.e. TADW) [39] learns a vector representation for each node by jointly factorizing a combination of both the graph structure and the text feature matrix. Another method called Attributed Social Network Embedding (i.e. ASNE) [40], simply concatenates the aforementioned matrices and performs matrix factorization to retrieve graph embeddings. Then there are the Attributed Node Embedding (i.e. AE) and Multi-Scale Attributed Node Embedding algorithms (i.e. MUSAE) which use random walks and skip-gram to retrieve node embeddings from an attributed graph [34]. MUSAE differs from AE, wherein AE combines attribute embeddings into a single representation and MUSAE represents each node attribute embedding separately in a multi-scale approach similar to Walklets.

Observe that the discussed techniques above use either matrix factorization or random walk sampling strategies which is consistent with our basic graph embedding strategy.

3.4 Dimensionality Reduction

Unfortunately, most of the previously mentioned algorithms have an optimal dimensionality that is greater than 2, it is therefore impossible to visualize the embeddings directly. To be able to visualize these embeddings, the dimensionality needs to be reduced to 2 dimensions. In this section several approaches for dimensionality reduction are discussed.

One possibility for reducing the dimensionality of embeddings is simply using similarity measures as discussed in section 2.5. Nevertheless, these measures behave imperfectly in higher dimensions [15, 41, 42]. It is thus preferred to use a Dimensionality Reduction Technique (or DRT) before applying the L1 (i.e. Manhattan distance) or L2 (i.e. euclidean distance) norm. Applying a DRT constitutes a transformation of high dimensional vectors to low dimensional vectors in which classic norm distances between

nodes can be interpreted as a pairwise distance matrix. There exist different types of dimensionality reduction techniques that offer an effective method to reduce dimensionality.

In addition to the visualization difficulties discussed above, many downstream machine learning tasks suffer from high dimensionality in data [43]. DRTs can help solve these problems by making visualizations feasible, by compression of the data, by multicollinearity reduction and by the removal of outliers, all while preserving original data structures [44]. Methods in DRT can be broadly identified as methods that extract or transform features from the original data and methods that remove redundant features [45]. In this paper it is imperative that the multi-dimensional embeddings have to be transformed to a fixed two dimensional environment for visualization. This has to be achieved without a loss of observations, and that dictates we focus on the former feature transformation DRTs.

The chosen dimensionality of the embeddings is relevant for DRTs, in many of these techniques a small sample size in combination with high dimensions is highly detrimental for their performance. A general rule of thumb is that the dimensionality should be much smaller than number of observations [45], which is in accordance with the requirement in section 3.2.

The choice of a suitable DRT is no trivial task and depends heavily on the data and assumptions made [45]. The requirements and preferentials for DRT selection in our framework are as follows, the DRT has to unsupervised (labels are unknown), it should optimize a manifold (i.e. local and global optimization), it should preserve neighbourhood relations and a bonus would be optimized for visualization. Furthermore, the dimensionality reduction technique should be Non-Parametric since the distribution of the data is unknown a priori. Additionally, to achieve sub question 5, it is required to extract multivariate information from these higher dimensions, some DRTs are explicitly optimized to extract this information [45].

One of the primary candidates for a good DRT is the classic Principal Component Analysis (i.e. PCA) [46, 47]. PCA first constructs a covariance matrix of the original standardized data and computes eigenvectors and eigenvalues on this matrix. Afterwards, this technique uses Principal Components, which are linear combinations of the original variables, as new variables. A subset of the top-k PCs (new variables) is made, which account for as much of the variation as possible. When $k = 2$ the data is transformed to two dimensional data with minimal loss of quality. Kernel-PCA [48] on the other hand, is a kernel method and an extension to the original PCA where non-linearity in the data manifold is assumed. Another version of PCA is introduced by Raunak, Gupta and Metze [49], they implement an algorithm for PCA with several simple preprocessing steps to significantly improve the quality of the results. Even today, due to its relative simplicity and efficiency, PCA-based methods are a reasonable option for reducing dimensionality [49]. Nevertheless, since PCA tries to maximize variance

it usually has large pairwise distances between observations, which is not necessarily ideal for visualization.

Another candidate is the unsupervised DRT t-Distributed Stochastic Neighbor Embedding (i.e. t-SNE) [50, 51], which tries to measure similarities in high and low dimensions concurrently by measuring the Kullback-Leibler divergence between two probability distributions. In high and low dimensions, a Gaussian distribution, respectively a Cauchy distribution is plotted over each instance of the data to derive similarities between all pairs of nodes. Finally, these two probability distributions are mapped to be similar by optimizing the KL cost function. One of the applications of t-SNE is creating visualizations of high dimensional non-linear data structures [51].

Some other candidates that are worth mentioning are the Locality Preserving Projection (i.e. LPP) [52] and Multi Dimensional Scaling (i.e. MDS) [53] . LPP is an unsupervised method based on Laplacian Eigenmaps mentioned in section 3.3.3, whereas the non-linear MDS tries to map points in low dimensions by preserving distances globally [53].

Table 2 lists some of the properties of these techniques, this table has been constructed with information in Ayesha et al. [45]. According to the previously defined requirements, t-SNE and KPCA are candidates for our framework. In addition to these DRTs, PCA will be added since it is easily interpretable, calculation time is small $O(|N|)$ and usually serves as a benchmark [54].

DRT	V-Opt[1]	Linearity	Supervised	Parametric	Span
PCA	No	Linear	No	Parametric	Local
KPCA	No	Non-Linear	No	Parametric[2]	Manifold
t-SNE	Yes	Non-Linear	No	Not-Parametric	Manifold
LPP	No	Linear	No	Undef.	Local
MDS	Yes	Non-Linear	No	Not-Parametric	Global

Table 2: Dimensionality reduction techniques [1]Optimized for visualization [2]User defined

3.5 Matrix Reordering Approaches

The next step in this research is to find and display a 'good' matrix order. A 'good' permutation is found by applying a matrix reordering method to the dimensionality reduced embedding (i.e. distance matrix) and compare the resulting figures with various quality metrics. This section first explains what type of matrix reordering algorithms exist and goes into detail in a specific school of reordering algorithms next.

3.5.1 Seriation

Seriation algorithms are classified into three distinctive families: structure based, distance based and convolution based methods [55]. Structure based methods focus mainly on interchanging rows and columns to bring out a particular pre-determined pattern [6, 56]. In contrast, distance based methods rely on node to node distances in a dissimilarity matrix, and aim to maximize (or minimize) the merit (or loss) of the used function. And finally, there are convolution based methods which optimize an objective function defined by a kernel. These methods, blur the original image and compare this reordering to the original ordering [57]. The idea behind these methods (e.g. ConvoMap [57]) are that by minimizing variation in the resulting permutation, they remove noisy patterns.

An extensive comparison of different families of seriation techniques is tested in Hashler [58] and Liiv [59], while Hahsler, Hornik and Buchta [60] introduce a R package which implements many of these techniques for direct comparison. In this work, the focus is on one special case of distance-based seriation algorithms, which are Travelling Salesman Problem solvers. These directly optimize the Hamiltonian path length.

3.5.2 TSP & Heuristics

The Travelling Salesman Problem (i.e. TSP) is a prominent NP-hard problem in combinatorial optimization. This prominency stems from the many (indirect) applications for TSP, think of vehicle routing and machine sequencing [61]. The Classical TSP problem statement by Menger [62] suggest that there is at least one optimal solution, and it can be defined as follows: given a finite amount of places to visit and their corresponding distances, what is the shortest route visiting all places exactly once? Needless to say, the optimal solution can be identified by considering all possible routes. Unfortunately, modern machines are in practice almost unable to solve even medium city problems due to the brute-force approach's factorial time complexity, or $O(n!)$. Two additional options arise for retrieving optimal or near-optimal TSP solutions, either using faster exact algorithms, or using heuristics, with the latter trading accuracy for large temporal gains.

A breakthrough for optimally solving TSP came in 1954 when the cutting plane method using linear programming [63] solved an impressive 49 city problem. Thereafter, several other influential methods have been proposed to solve TSP, the branch and bound method [64] in 1960; dynamic programming [65] in 1962 and; branch and cut for TSP [66] in 1991. The symmetric TSP solver Concorde as described in Applegate et al. [67] of which development started in 1990, remains one of the best exact solvers for TSP today [68, 69, 70]. Concorde implements the branch and cut scheme, where unfortunately, running time is highly correlated with the ability to

find a good initial solution [67, 70, 71]. Therefore to speed up Concorde, a number of techniques exist that try to find a sub-optimal initial solution.

Whenever these optimal methods are no longer feasible, the implementations of approximate algorithms and heuristics have been proposed. Some examples include the Nearest Neighbour, Genetic and Greedy algorithms [72]. The Nearest Neighbor algorithm [73] is a simple algorithm that selects the city with the smallest distance to the current location and stops when all cities are added to the tour (i.e. path). In a different approach, Genetic Algorithms [74] simulate evolution by selecting the fittest individual tours, whereafter these individuals are able to reproduce and pass on genetic traits to their children. The idea behind this method is that after many generations, tours will iteratively become better. Furthermore, heuristics can speed up algorithms by making certain assumptions, some important insertion heuristics are described by Rosenkrantz, Stearns and Lewis [73]. Insertion heuristics construct smaller sub-tours, where nodes not in this initial sub-tour are added one after another. Insertion heuristics differ in the way they construct the intial sub-tour, which node to add next and where to add this next node. Likewise, with the k-Opt heuristic, which is an expansion on the 2-opt heuristic by Croes [75], there is a possibility to iteratively improve an existing tour by deleting k edges. The tour is split in k sub-tours and these sub-tours are reconnected by generating new connections between each sub-tour. The algorithm stops when no additional improvements can be found [68]. The quality of the k-Opt heuristic increases with an increase in k but so does the complexity, thus it would make sense to make k dynamic during execution instead of static. This expansion of the k-Opt heuristic is exactly what the the Lin-Kernighan heuristic [76] does, and it is considered one of the most effective heuristics for TSP [70], solving an 85.9k city problem optimally [71].

The Lin Kernighan heuristic (i.e. LKH) is continuously updated with new features and optimizations [70] which caused the heuristic to stay highly competitive and it "currently holds the record for all instances with unknown optima" [77]. A recent comparison between different configurations of the LKH heuristic and plain Concorde was made by Sanches, Whitley and Tinós [70]. One other notable expansion to Concorde is the Chained LK algorithm [78] which tries to find a good initial solution to speed up Concorde. This algorithm finds sub optimal tours, which are subsequently optimized locally with iterative search. It achieves this by reapplying the Lin Kernighan heuristic after making slight modifications on the current tour (i.e. kick), in order to find a set of locally optimal starting points [78]. This offered great advantages over the original Lin Kernighan heuristic [79]. Recently, Sanches et al. [70] extended the CLK algorithm by proposing to use partition crossover. The main idea of partition crossover is that by using recombination of two locally optimal parent tours, a high quality offspring will be produced with high probability [70]. The most recent version that

has been implemented for TSP is the GPX2 partition crossover by Sanches et al. [70]. Combining GPX2 with the Lin Kernighan 2.0 heuristic [80] and generating initial solutions led to an average run time decrease of Concorde for all of the 13 tested problems, with 7 of them being significantly different [70].

3.6 Matrix Visualization Quality metrics

To be able to compare different visualizations there is a need to establish a good comparison metric. This metric should be distinguishable for comparison between different algorithms, and the metric should distinguish between interpretable and non-interpretable visualizations. A good visualization should be able to show distinct nodes and edges; information on clusters and connected components in the data; how the data is distributed and high-level topology [81]. Consequently, Behrisch et al. [6] argue that matrix visualizations are especially well suited to fulfill these requirements. In the next sections visual patterns in the data and usage of visual quality metrics are discussed.

3.6.1 Visual Patterns

Visual patterns in matrix re-orderings are highly dependent on the seriation algortihm used [6]. Some algorithms, such as TSP, are more inclined to show block patterns, while others show off-diagonal patterns more often. Nevertheless, in this work TSP will be used to discover patterns in the data, since TSP is a highly researched problem and can produce (semi-)deterministic results (when using an exact solver). In the delineation section of this paper there will be more information on this choice.

The visual patterns to check for in the final visualizations are the binary data patterns defined in Behrisch et al. [6], which are based upon the canonical data patterns by Wilkonson [11]. These patterns describe connection within and between cliques; hub nodes; cycles and paths; potential bipartite relationships in the data; randomness and failure(s) within algorithms [6]. The visual patterns are discussed into more detail in section 4.6.1

3.6.2 Visual Quality Metrics

To compare matrix re-orderings Hashler [58] suggest the use of seriation criteria. Seriation criteria try to optimize visualizations by grouping similar objects close and pushing larger dissimilar values further from the diagonal. These types of measures fall into three classes namely gradient conditions, rank/dissimilarity agreement and path length [58]. Note that the visual quality metrics used in this paper are from the R package seriation [60] and, therefore the upcoming definitions are taken from the comprehensive paper by Hashler [58].

Gradient conditions compare closeness of the visualization in a dissimilarity matrix D with values d_{ij}, to an Anti-Robinson Matrix [58]. An Anti-Robinson Matrix is a matrix that has the property of only sustaining increased or decreased values when moving away from the diagonal [82]. An example of such a method is the Anti-Robinson events [83], which count the number of times this property is violated. Dissimilarity values in the visualization are compared within subsequent rows and within subsequent columns according to the gradient conditions formulated by Hubert, Arabie and Meulman [84]. Anti-Robinson events score with the following definition: $\sum_{i<k<j} f(d_{ik}, d_{ij}) + f(d_{kj}, d_{ij})$ and is bounded by $1 \leq i < k < j \leq n$, and f which represents a function with value 0 when $d_{ik} < d_{ij}$ and 1 otherwise [58]. Several measures edit the previously mentioned function by including the weight of the violation (i.e. AR deviations [83]), by calculating the divergence amidst violations and agreements (i.e. gradient measure [84]), and by weighting these divergences (i.e. weighted gradient measure [84]) [58]. In addition there are rank/dissimilarity agreement measures, which compare matrix visualizations by calculating pairwise dissimilarities and check whether their ranking in the ordering matches [58]. The Inertia criteria for instance, calculates $-1 \times \sum_{i,j=1}^{n} d_{ij}(i-j)^2$ [58] as a score and imposes large penalties for objects that are placed far from the diagonal and have a high similar value [85]. The linear seriation criterion (i.e. LS) [86] does not square the final terms as in the Inertia criterion and thus penalizes less for distances to the diagonal. Furthermore, the least squares criterion [85] penalizes similar to Inertia by taking the complete squared difference between ranks and dissimilarities, it also excludes the negative transformation [58]. And finally, there is the path length measure in which the objective is to minimize the total Hamiltonian path length [85, 87]. This measure visits each node in the graph once and only once and adds the dissimilarity values to derive a final score, or by definition $\sum_{i=1}^{n-1} d_{i,i+1}$ [58]. For a full distinction of the seriation criteria the reader is suggested to read the paper by Hashler [58].

In this paper, the distinction between measures that are more attentive towards local structure (e.g. Least squares criterion, Relative generalized Anti-Robinson events and Hamiltonian path length) and global structure (e.g. Anti-Robinson events, 2-SUM and Gradient measure) is made and could influence the choice for a subset of seriation criteria. In addition to that, there exists reordering algorithms which optimize certain criteria such as TSP, which will always score better or equal to other seriation algorithms on the Hamiltonian path length measure.

To compare the matrix reorderings of different seriation visualizations a selection of quality metrics could be made. The relevance of this selection is directly reliant on the implementation and what should be optimized [58].

3.7 Main Findings

This paper aims to address several questions which remain unanswered in the literature review. Most studies visualize graphs as node-link diagrams, and while these visualizations are useful, there is a general lack of studies on matrix visualizations. The question of what visual patterns resemble in embeddings generated by graph embedding algorithms remains unanswered.

Furthermore, literature on attributed graph embeddings focus mainly on generating embeddings for node classification, edge prediction, community detecting and graph clustering. There appears to be a gap in visualizing multivariate data in a single representation. In addition to that, most of these algorithms embedd only textual, binary and categorical data, while in this paper numerical data will be included.

And finally, there seem to be many different scores and visual patterns to assess the quality of a good matrix visualization. To the best of our knowledge, a single comparable score amongst different latent space visualizations does not yet exist.

4 Deep Learning Based Matrix Reordering

This chapter introduces the framework for visualising graphs in two dimension matrices. A barbell graph is used as an example throughout this section. An example of this graph is shown in figure 10 and consists of two fully connected groups (i.e. cliques), two hub nodes (node 11 and 9) and a single bridge node (node 10). The aim of this section is to provide an extensive step by step overview of the framework.

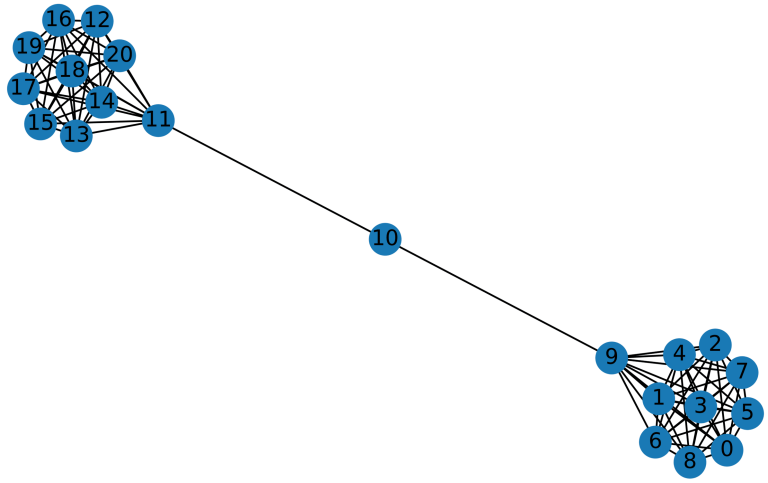


Figure 10: Barbell Graph

4.1 Overview of Pipeline

As a first step, a graph is represented in a data structure, thereafter this graph is embedded into N-dimensions for representing nodes as vectors. To visualize a large N-dimensional embedding we apply dimensionality reduction techniques. In the second step we calculate a distance matrix after projecting the graph in 2D. This distance matrix is uninformative since there is no structure [6]. To allow for structure in the data we apply a seriation technique and expect that patterns arise in the data. We qualitatively judge the visualization by identifying patterns and quantitatively judge the visualization by quality metrics. In appendix A a overview of the embedding and visualization phase is given in image form. In this image different colored nodes represent different roles of nodes in the embedded graph. The images do not show the quality assessment phase. An implementation of the architecture described in the next sections is available on gitlab[6].

[6]https://git.science.uu.nl/vig/mscprojects/gnn-framework-for-multivariate-matrix-reordering

4.2 Graph Representation

The first step in the framework is converting graph data to a NetworkX graph. The main reason for this conversion is that a NetworkX graph can be fed into most graph embedding algorithms implementations directly and the package makes visualizations for small graphs is rather convenient (see figure 10). To store a graph on the disk it can be represented with different data structures, the four most common structures are discussed in more detail in section 2.3. The choice of a representative structure is not critical for our framework and new structures can easily be added with little change of code. Nevertheless, since the loaded graph is directly transformed into a NetworkX graph, only space complexity is of interest. An edge list stores all edges and is often most economical with a space complexity of just $O(E)$, this compared to the hefty $O(V^2)$ for an adjacency matrix where a node by node matrix is stored. An additional small advantage for the framework would be the possibility for a single file representation for multivariate graphs, while for other structures a two file representation is more appropriate.

4.3 Embedding a Graph

The second step in the framework is acquiring embeddings for all nodes in the graph. To obtain node embeddings we can apply several different strategies, which include random walk, matrix factorization, and deep learning algorithms, which are explained in further detail in section 3.3.

The focus of this study is mainly on random walk algorithms and, in the following section a detailed application of the algorithm DeepWalk will be given.

4.3.1 Why generate embeddings

A computer is not able to understand the concept of a graph without transforming this graph to a computer understandable syntax. These could be edge lists, or any of the other representations explained in section 2.3. Nevertheless, these representations often suffer from the curse of high dimensionality and thus one of the techniques for representing nodes in lower dimensions is constructing simple summary features such as the degree and centrality of nodes [88]. Unfortunately, this solution proved rather time consuming with expert analysis and less effective than the automatic methods of today.

By constructing new vector spaces and embedding graph data in this space, much of the existing frameworks in machine learning can be applied faster and more accurately [89]. And a final reason, Harel and Koren [14] show a great increase in graph visualization quality by embedding graphs in a high dimensional space.

4.3.2 Obtaining Embeddings with Word2Vec

The random walk strategy is based upon a well known model used in natural language processing (NLP), namely word2vec. The goal of word2vec is to map all unique words in the corpus to a numerical N-dimensional vector space [35]. Each point in this vector space is a contextual aware vector representation of a word. Word2vec extracts certain features from the input text segments and captures syntactic and semantic similarity between words with proximity in the vector space. This allows for simple similarity extraction of words, and even more complex operations. For instance, researchers found that a combination of vector representations could lead to correct analogies, the arguably most famous example is: *king* is to *man*, as *woman* is to *queen* [35]. Note that a recent study by Nissim, van Noord and van Der Groot [90] show that in most of these examples a trick (e.g. reporting second answer as first answer) was used to get satisfactory results [90]. The idea of DeepWalk is based upon this embedding of words. DeepWalk tries to embed nodes in a vector space while preserving proximity of nodes.

To obtain embeddings, word2vec uses an architecture called Skip-Gram (or CBOW). The objective of Skip-Gram is to predict the context of the target word in a predefined range or context window [35]. In Skip-Gram we feed training pairs of words into the model and retrieve contextual aware embeddings. Figure 11 shows the Skip-Gram architecture where word $w(t)$ is fed as an input to a shallow feed-forward neural network. Generation of these training pairs occurs by sliding a window of size n along the training documents or text segments. For DeepWalk the idea is again rather similar, where instead of documents and word pairs, a random walk and node pairs are fed as an input.

Word2vec applies back-propagation and gradient descent on the intermediary weights to optimize these weights. Finally, a softmax function is applied and, with a cost function, the most probable words around the target word are maximized. The intermediary (hidden) weight layer can actually be considered the vector representations of the words trained in the network. For further details I suggest the reader to read the paper by Mikolov et al. [35].

4.3.3 Obtain Embeddings with DeepWalk

The embedding generation in DeepWalk does only differ with word2vec by acquiring training samples. When comparing word2vec to DeepWalk, Deepwalk considers nodes as words and a sequence of nodes as possible sentences. To generate actual sequences of words (i.e. nodes) the algorithm performs multiple random walks along the to be embedded graph. For every node in the graph a certain number of random walks will be performed whereafter these walks are considered as sentences and fed as an input to the

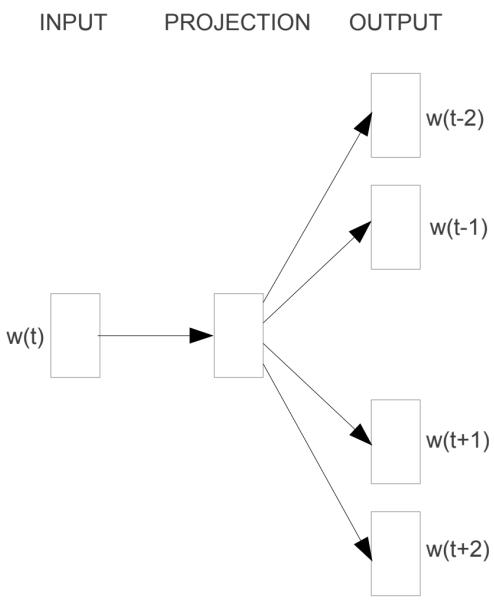


Figure 11: Skip-Gram architecture taken from Mikolov et al. [35]

Skip-Gram architecture (e.g. the target node is used to predict the context nodes). Figure 12 shows an example of such a random walk on a barbell graph.

In the barbell graph in figure 12 the walk starts at node 0, moves to node 2 and ends in node 8. A single training sequence in this case would be:

$$[0\ 2\ 3\ 4\ 5\ 8]$$

and is analogous to a list of words in word2vec:

$$['I',\ 'like','to','have','apples','and','pears']$$

Similar to word2vec a sliding window can be applied of any size on this training sequence. As an example in figure 13 a window of size two is applied to the random walk in figure 12. In figure 13 the highlighted yellow node is the target node and the nodes in the square box are the context nodes. In this example node 0 and node 2 are considered more similar than node 0 and node 5 since they appear in more training samples. Finally, Skip-Gram generates the real valued embeddings that maps graph distances in N-dimensional space [91]. All training samples are used and proximity preserving node embeddings will be the result.

In table 3 a typical result for an embedding is given. There is a clear separation in the embedding between nodes 0,1,2,3 and 5,6,7,8 where they differ in all of the dimensions. Furthermore, we see a clear separation of node

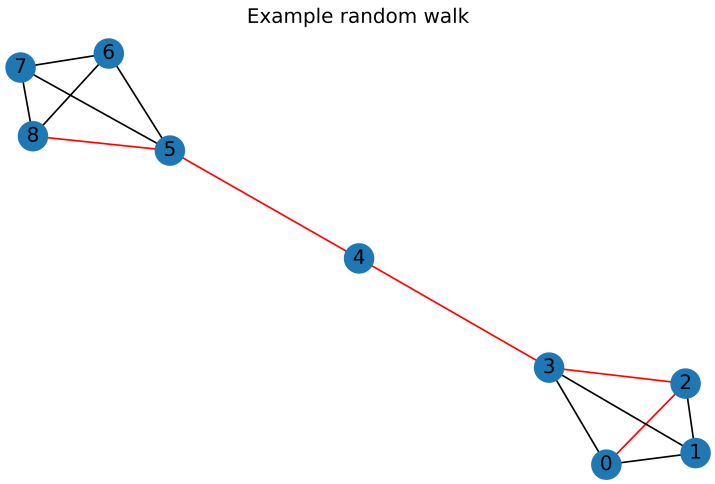


Figure 12: Random Walk

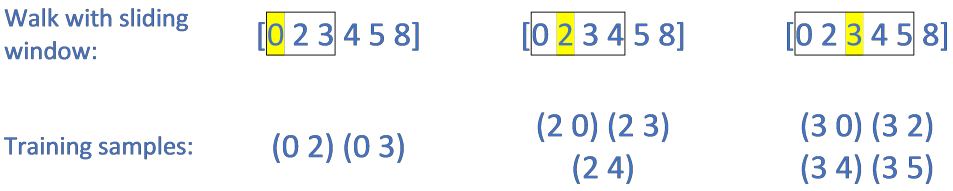


Figure 13: Sliding Window and training pair generation

3 and 5 from their respective cliques and this corresponds to the barbell graph structure, i.e. these nodes behave like hubs. Node 4 seems to be completely different from all the other nodes. In this simple example it is apparent that there are probably too many dimensions since this structure can easily be represented with fewer dimensions as shown in section 4.4.

4.4 Reducing Dimensions

According to research, calculating a distance matrix directly from high dimensional embeddings specified in section 4.3 will result in inaccurate distances between points in high dimensional space [15, 41]. Therefore, and in accordance with the method of Harel and Koren [14] a dimensionality reduc-

$node$	dim_1	dim_2	dim_3	dim_n
0	0.86	0.9	0.87	..
1	0.89	0.91	0.88	..
2	0.89	0.89	0.88	..
3	0.65	0.84	0.81	..
4	-0.98	0.32	0.001	..
5	-0.87	-0.75	-0.79	..
6	-0.87	-0.89	-0.87	..
7	-0.83	-0.88	-0.89	..
8	-0.82	-0.88	-0.88	..

Table 3: Fictional embedding example

tion technique will be applied to reduce the dimensionality of the embedding to two.

A choice was made to include three different dimensionality reduction techniques. The first being the linear Principal Component Analysis (i.e. PCA) which is also used in the paper by Harel and Koren [14]. The more modern and influential t-distributed Stochastic Neighbor Embedding (i.e t-SNE) which is optimized for visualizing high dimensional data [51] is also implemented. And finally, Uniform Manifold Approximation and Projection (i.e. UMAP) which claims to be faster, be able to better preserve the structure of the data for larger datasets [92].

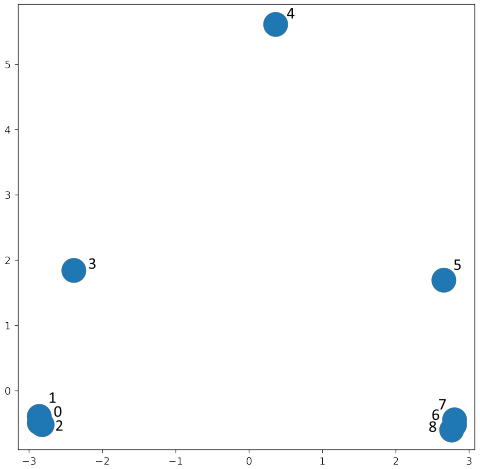


Figure 14: Projection of embeddings in 2D vector space

An example of the embedding obtained in table 3 reduced to 2 dimensions with PCA is given in table 4. These embeddings can be visualized in a 2D

vector space as in figure 14. For this easy example there is ample visual separation in the previously determined different nodes.

node	dim_1	dim_2
0	-2.86	-0.46
1	-2.86	-0.42
2	-2.87	-0.47
3	-2.39	1.83
4	0.37	5.61
5	2.65	1.69
6	2.79	-0.57
7	2.79	-0.51
8	2.79	-0.59

Table 4: 2D embeddings

node	0	1	2	3	4	..
0	0.0					
1	0.04	0.0				
2	0.0	0.06	0.0			
3	0.35	0.34	0.35	0.0		
4	1.0	1.0	1.0	0.7	0.0	
..	..	..	..	..	..	..

Table 5: Distance matrix (diagonal in blue)

Evidently this 2D embedding could be easily transformed into a distance matrix by calculating pairwise euclidean distances. It is apparent that the upper triangle of this matrix is exactly the lower triangle of the matrix because pairwise distances are not relative in our example. Therefore, we could save almost half on memory by just representing either triangle of the matrix as in table 5, a feat not possible for directed graphs. This table could be interpreted as a reversed heatmap, where larger values are worse than smaller values (i.e. distance).

4.5 Permuting a 2D matrix

In this section the reader will be introduced to the permutation process in the pipeline. In this section an example and the interpretive value of ordering this example will be shown.

4.5.1 Random Permutation

The output of table 5 can directly be represented by a data matrix where distances are visualized by color intensity. In the upcoming figures more similar nodes (i.e. lower distances) are represented by darker colors and completely distant nodes are represented by the color white. The more saturated a cell is, the more similar two nodes are.

Figure 15 shows a randomly permuted data matrix, which is based on the previous example. This matrix has been randomly permuted to highlight what a typical result for a more complex graph would look like. In this matrix node 6 is highly similar to node 7 and 8 (indicated by dark values), while being completely dissimilar from node 4. This figure seems to show the salt and pepper anti-pattern defined in Behrisch et al. [6] and is hard

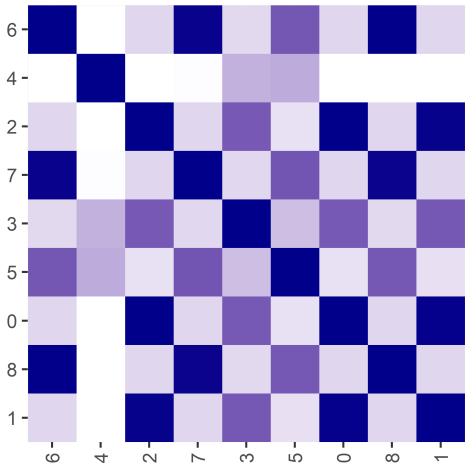


Figure 15: Unordered matrix visualization of example graph

to interpret even for this example with a small number of nodes. From now on such a matrix will be called an 'unordered' matrix.

4.5.2 Obtaining a useful permutation

To obtain an interpretable reordering a seriation algorithm for matrix reordering could be used. Hashler et al. [60] introduced an R seriation package[7] which includes many different types of seriation algorithms and a framework for showing matrices. The different types of seriation algorithms are further discussed in section 3.5. A choice was made to use a TSP algorithm for ordering a matrix because this is one of the most well-researched problems in optimization. The algorithm of choice is Concorde[8] for obtaining the optimal solution and a Lin-Kernighan heuristic for increased ordering speed. In section 8 there will be a discussion on whether there a better alternatives.

The exact TSP algorithm finds one of the smallest Hamiltonian paths in a graph, and for this use case we define a Hamiltonian path as a path that visits all nodes one and only one time. These Hamiltonian paths help proliferate the structure in the final visualization by placing points similar in high dimensional space close in the visualization. Furthermore, TSP clusters similar nodes close to the diagonal in the matrix for more distinct and informative clusters. By applying TSP to the example embedding in figure 14 the resulting path shown in figure 16 is retrieved. This path starts in node 2 and ends in node 8, and thus the following ordering is retrieved:

[7]An implementation of the R seriation package can be found here https://cran.r-project.org/web/packages/seriation/index.html

[8]Concorde is available here: https://www.math.uwaterloo.ca/tsp/concorde.html

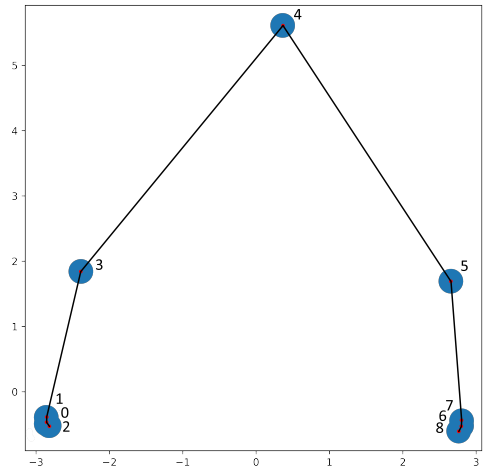


Figure 16: Hamiltonian Path on embedding (center of node in red)

[2 0 1 3 4 5 7 6 8]

The ordering is applied to figure 15 and the results are shown in figure 17. The ordered matrix is more interpretable than the random permutation and shows some clear structure in the data. In the next section there is a more formal justification of why this ordering is better than random.

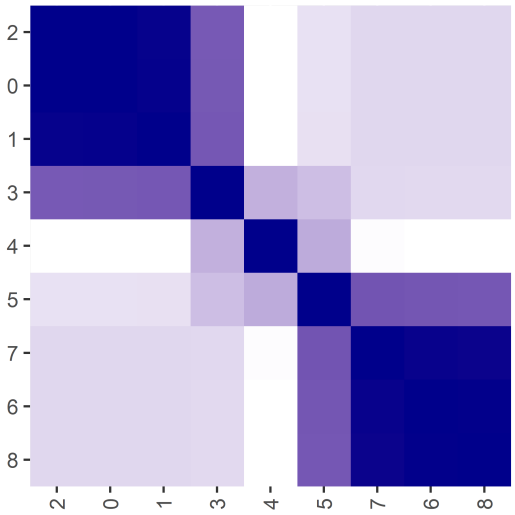


Figure 17: Ordered matrix visualization of example graph

4.6 Assessing Quality

To assess whether a reordering is qualitatively better than another reordering it is possible to compare two matrices visually with visual patterns highlighted in the next section. For a more quantitative approach several metrics could be used. Research suggest using Visual Quality metrics (i.e. VQM) introduced in section 3.6.2, unfortunately these metrics are doubtfully applicable for our study. Thus, an introduction to the self-defined scores will be given in section 4.6.3.

4.6.1 Visual patterns

To quantify the goodness of an embedding we can use visual patterns in the resulting matrix reordering. These patterns are able to show underlying structure in the graph data and can give the user high level knowledge on his or her data. Figure 18 shows the six patterns to abide to and anti-patterns to avoid [6].

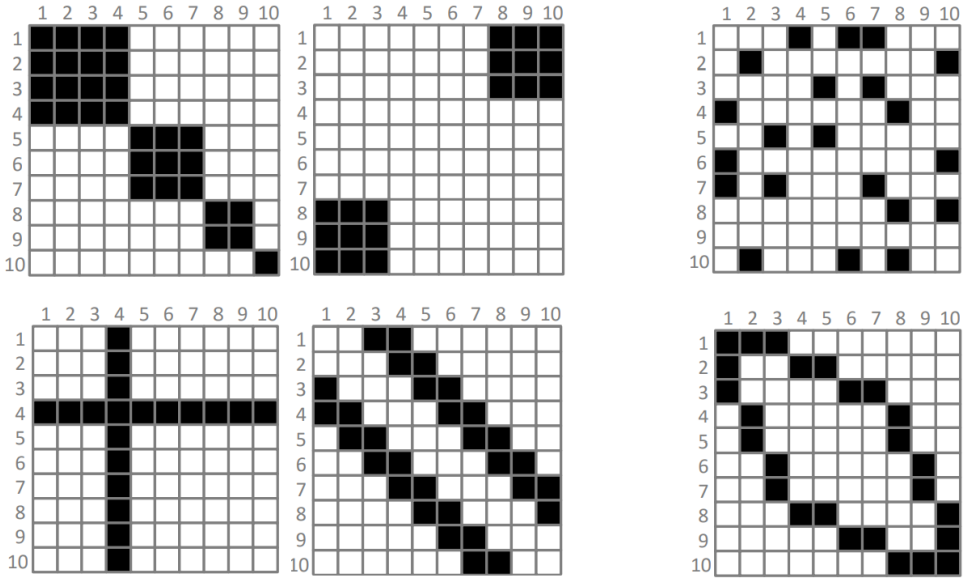


Figure 18: Visual Patterns taken from Behrisch et al. [6]. From left to right: (1) Block Pattern, (2) Off-diagonal Pattern, (3) Noise Anti-Pattern, (4) Line/Star Pattern, (5) Bands Pattern, (6) Bandwidth Anti-Pattern

In the figure above the top left block contains four patterns (i.e. Block, Off-diagonal, Line, Bands) that describe cliques, bi-graph structure, hub nodes and cycles respectively. In the top right column there are two anti-patterns which describe randomness in the data (top), and algorithmic peculiarities (bottom). To continue with our running example, figure 19 shows

a clear identification of the Block Pattern in the yellow and red boxes. These identified cliques do indeed align with the original graph structure. Furthermore, the visualization correctly identifies partial clique membership where node 3 belongs to the larger clique (i.e. 2,0,1) only partly. The green box in this figure shows the Off-Diagonal pattern which represents closeness of the two larger cliques in the graph data.

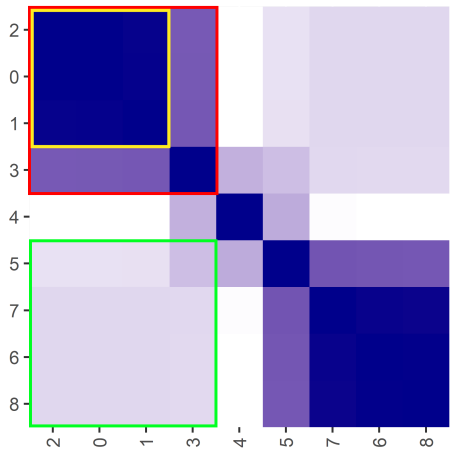


Figure 19: Block pattern

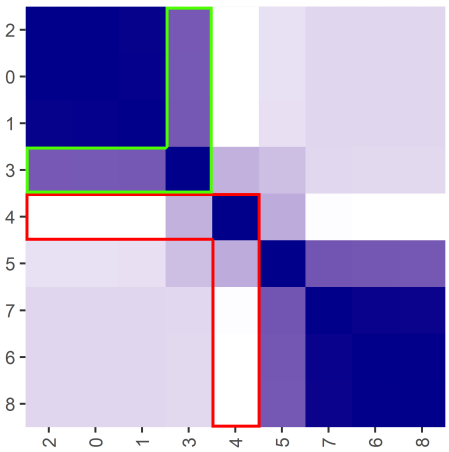


Figure 20: Star/Line pattern

Figure 20 shows the Line/Star Pattern in the red and green boxes. In the original graph node 3, node 5 are hub nodes for their corresponding cliques and node 4 is a hub node for node 4 and 5. These findings correspond with the original graph data.

4.6.2 Visual Quality Metrics

It is also possible to check the goodness of a visualization with the visual quality metrics described in section 4.6.2. In figure 21 a table with these metrics is shown on the running example. Unexpectedly, in this example there appear to be four Anti-Robinson Deviations, which after visual inspection of figure 17, should not exists. This is easily explained with data inspection though, there appear to be minor differences in distances between points and thus color in the matrix representation. One of the strengths of these quality metrics are therefore that they can discover inconsistencies in the data that are hard to spot on a visual level.

Unfortunately, visual quality metrics have been found to be applicable for intra comparison per algorithm, but being overly biased on the underlying data. This makes these metrics almost useless for inter-algorithmic comparison. The findings for these statements and a proposed solution are further explained in section 8.

Visual quality metrics															
2-Sum	Anti-Robinson Deviations	Anti-Robinson Events	Banded Anti-Robinson	Cor_R	Gradient measure	Gradient measure weighted	Inertia	Lazy path length	Least Squares	Linear Seriation	Measure of effectivene	Stress Moore	Stress Neumann	Hamiltonian path length	RGAR
600	4	51	2	0	63	45	891	9	751	232	71	13	6	2	0

Figure 21: Example problem and seriation scores

4.6.3 Reconstruction Metrics

In this section there will be a brief introduction to the reconstruction metrics used in section 7.2. These metrics were introduced to check for their applicability in selecting good visualizations. These metrics are fundamentally different from the other metrics used in this paper and do not directly calculate whether the visualization shows any patterns. Instead these methods give an idea of how good the original graph is represented in the visualization. Some of the original graph node similarity measures can be defined as measuring the first and second order proximity; k-hop distance between neighbours; probability in a random walk and adjacency matrix factorization [12]. In this work we define two separate reconstruction metrics inspired by the work of Xu [12]. The metrics are explained by introducing score calculation on basis of divergence between original graph and visualization vectors:

Khop Similarity score, which is a metric that calculates the pairwise distance for all nodes within three hops. First in the original graph, a triplet is generated with source nodes, target nodes and their distance. Thereafter, for all node pairs in the triplets the distance is calculated in the visualization (indicated by the darkness of the hue) and saved as a vector. The original triplets are transformed to a distance vector, and the cosine similarity between both vectors is calculated.

The *Degree Adjusted ARI* on the other hand clusters nodes with the exact same degree in the original graph as a ground truth, and this leads to n clusters (i.e. each distinct degree number is a cluster). For the visualization, each row is summed and represents the degree of the visualization. These are clustered with kmeans into the known a priori n clusters and compared to the original graph clusters by calculating the Jensen-Shannon Divergence (i.e. JSD) between the distributions. Some nice properties of the JSD in comparison to the Kullback-Leibler divergence is that it is symmetric and a metric [93].

5 Visual Debugger for Quality Assessment

Analysing high-level complex relations in 2D matrices works well for the problem highlighted in section 4. But, when introducing graphs with multiple variables, even more dimensions are introduced to the original problem, and it becomes increasingly difficult to truthfully visualize all of this information in 2D. Therefore, we propose a multi-scale approach where the user can compare, combine, adapt and subset multiple visualizations on the same graph.

Features for nodes 0–10

node	feat_1	feat_2
0	15	3
1	20	5
2	21	5
3	11	10
4	24	5
5	16	5
6	16	1
7	18	1
8	21	2
9	24	1
10	50	5

Features for nodes 11–20

node	feat_1	feat_2
11	41	5
12	40	5
13	34	2
14	15	4
15	20	3
16	21	1
17	1	5
18	21	10
19	1	1
20	2	3

Figure 22: Features of Barbell Graph

5.1 Descriptive Analytics

The developed tool for analysis of graph visualizations aims to describe graph data, and takes a data analytics approach. The tool named *Matrix Reordering Explorer* can be used for amateur and expert analysis of basic, weighted and multivariate graphs. It has been explicitly designed to cater to the needs of scholars and network analysts as defined in Behrisch et al. [7]. Requirements for scholars are: (*a*) compare visualizations of different algorithms; (*b*) possibility to compare different embeddings; and (*c*) define practicality of the embeddings [7]. Furthermore, for network analysts they

are: detect local & global patterns (d); possibility for adaptation of visualization(s) (e); and visualization of nested patterns (f)[7].

To give an example, the barbell graph was loaded into the tool in conjunction with the feature vectors as shown in figure 22. The graph was embedded with the featureWalk algorithm discussed in the next section and this resulted in figure 23. In the figure the barbell structure embedding is shown on the left, while the combined representation of the features is shown on the right. This figure shows the compare function of the tool, in which two plots can be laid side to side, from different algorithms or embeddings, or both. This satisfies requirement a and b of being able to compare.

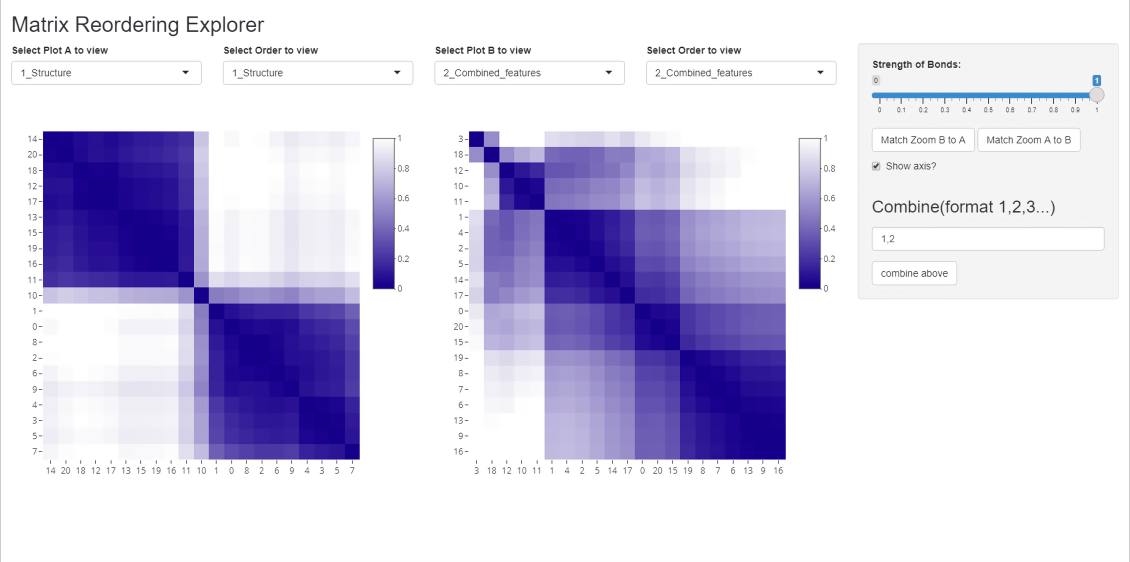


Figure 23: Matrix Reordering Explorer: Compare function

Thereafter, figure 24 shows the adaptive nature of this visualization tool. It is possible to directly view bond strength by hovering the mouse over each cell; the bond strength threshold for visualization can be adjusted; and it is possible to zoom; pan; and match zoom levels of distinct visualizations. With these interactions the tool aims to satisfy requirement e.

In figure 25 the focus is on visualizing two reorderings, based solely on the feature embeddings. These features have been embedded without including any graph structure. Feature 1 for instance, shows a dense clique in the bottom right with nodes 17,20 and 19. If we compare the feature values (in figure 22) of these nodes this clique is justified, with feature values of 1, 2 and 1 respectively. The embedding seems to correctly classify nodes into similar groups of values, thus for this image satisfying requirement c.

Furthermore, in this figure feature 2 is displayed, with the reordering of feature 1, highlighting patterns that are present in feature 1 and feature

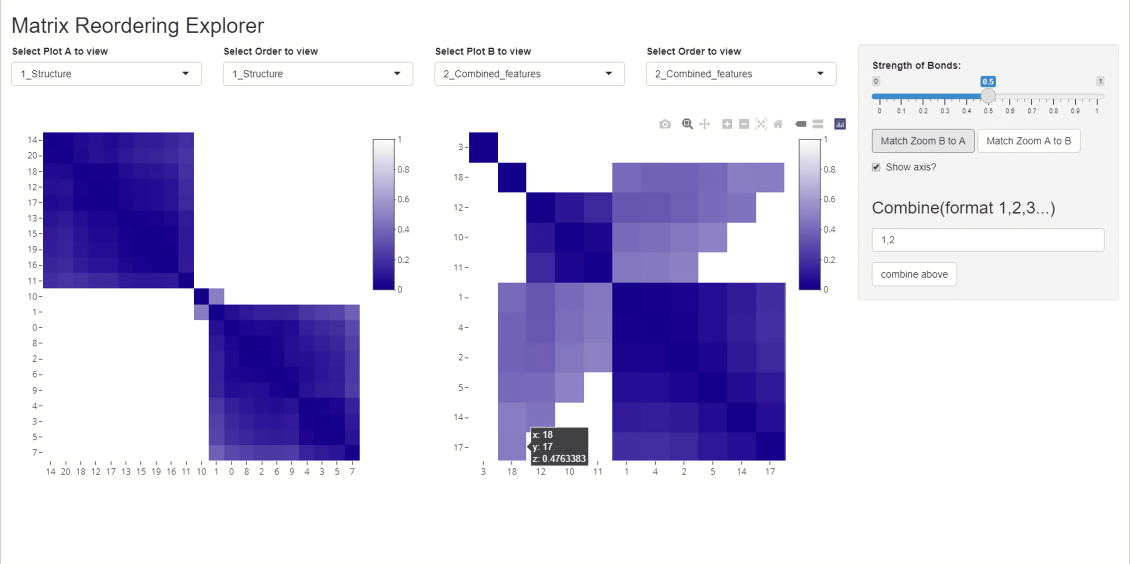


Figure 24: Matrix Reordering Explorer: Adaptive nature (e.g. zoom)

2. A prime example is the dense block pattern in the upper right corner consisting of nodes 10,12 and 11. Again, these match with the similarity between feature tuples namely: (50,5),(40,5) and (41,5). Note that even more information can be gained from this figure, node 4,2,1,3 and 17 belongs to the same block as nodes 10, 12 and 11 in feature 2. This is indicated by the dark stripes in the right figure. Nevertheless, they do not belong to this clique in feature 1 (indicated by the ordering in the right figure). With this ability we can combine, detect and subset patterns and we hope requirement d is satisfied.

And finally, figure 26 shows the ability of combining different embeddings with mid fusion (i.e. fusion after embedding, but before seriation). In this figure the early fusion combined embedding (i.e. embed both features at the same time) and the structure embedding are merged, indicated by the numbers they represent in the drop-down list (i.e. 1 and 2). In this figure, there is a clear high level separation from the structure, and the figure also shows multi-scale smaller block patterns within these larger blocks. The high level separation is indicated by hub node 10 in the middle, and groups of nodes 11-20 and nodes 0-9. Furthermore, there are two smaller cliques (10,1,2,4 and 8,6,9,7) in the lower right box, which correspond to the values in figure 22. With these examples the hope is that the nested requirement f is satisfied.

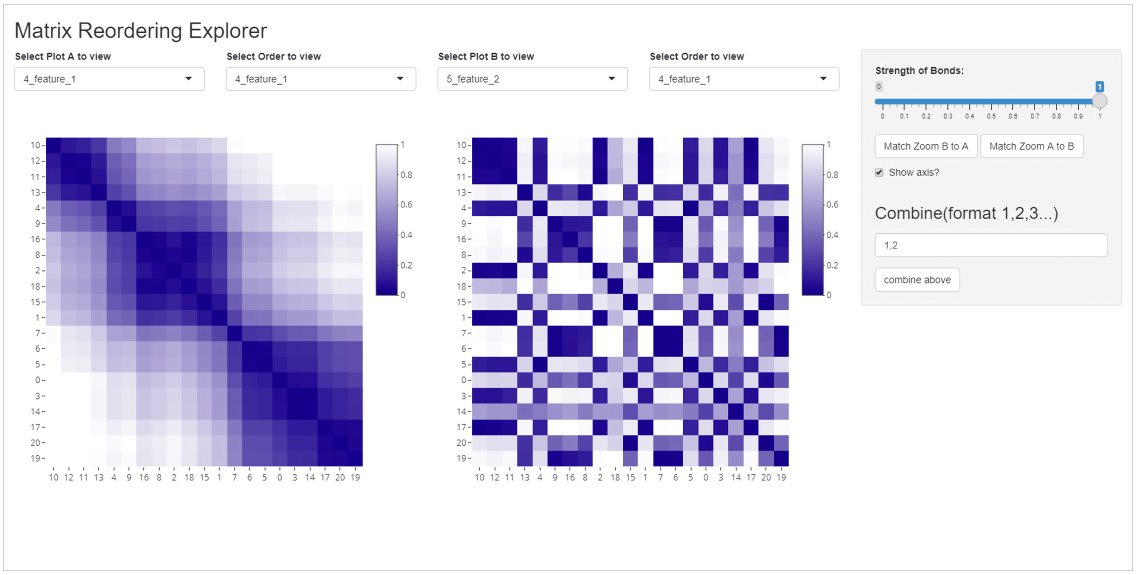


Figure 25: Matrix Reordering Explorer: Feature comparison

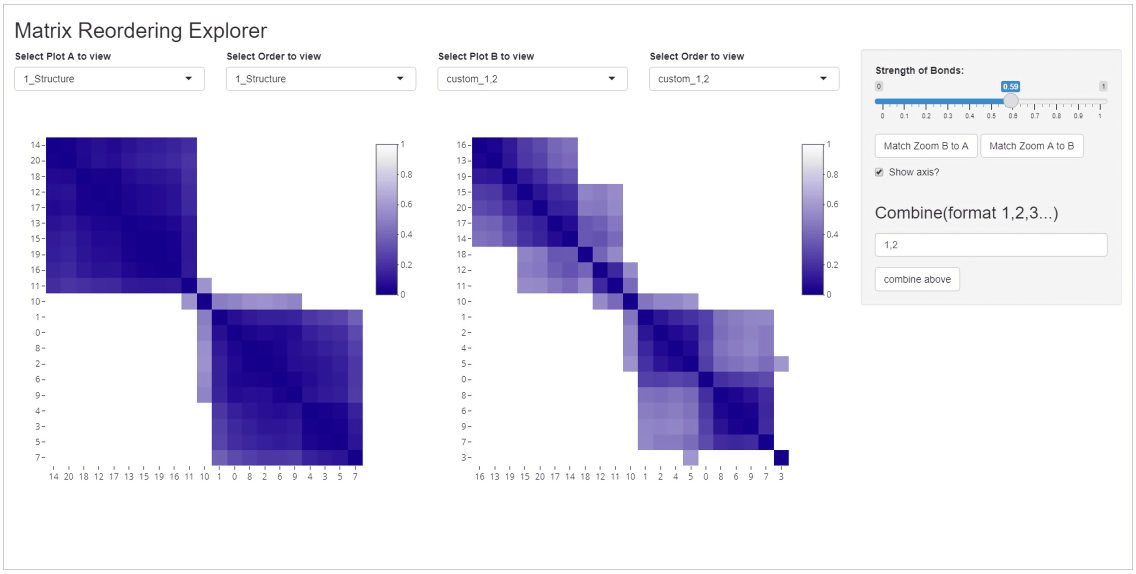


Figure 26: Matrix Reordering Explorer: Combine embeddings

6 Novel Algorithms for Visualizing Multi-Graphs

In this section, the problem of visualizing multivariate graphs will be solved by introducing two novel algorithms. The reader will be introduced to the inspiration for these algorithms, their pseudo-code and time complexities.

6.1 Need for Multivariate Visualizations

Unfortunately, most multivariate graph embedding algorithms either embed structure and features combined or calculate a separate structure embedding and a complete attributes embedding. For visualization purposes, and to satisfy the goals specified in section 5, there is a need for:

1. a structure embedding for the graph structure,
2. combined embeddings for a high level overview,
3. separate attribute embeddings for a local view, and
4. possibility to combine distinct embeddings for analysis.

To the best of our knowledge there exists no graph embedding algorithm that is able to embed a multivariate graph in which each feature is embedded separately. Thus, we introduce featurePMI and featureWalk, leveraging existing techniques in the graph embedding domain to generate combined and separate visualization(s) of multivariate graphs.

6.2 featurePMI

FeaturePMI is a graph embedding algorithm designed on the notion of matrix factorization. The strength of this algorithm is that it incorporates the structure of the graph into each separate attribute and combined attribute embedding.

This algorithm takes as input the random walks from an embedding algorithm (e.g. Walklets). These random walks are split into origin nodes, and a pre-specified window of target nodes (e.g. $\sqrt{n}$). Thereafter, these node vectors serve as an input into a count matrix P, and this matrix counts how many times each origin-target pair is present in the pre-calculated random walks. When dividing this matrix by the maximum value in P a transition probability matrix is returned. Note that this algorithm is highly efficient when matrix P is sparse, due to the sampling phase. We can assume that for any graph, apart from fully connected graphs, that the number of edges is smaller than the number of nodes squared, and for these graphs this algorithm offers temporal gains.

Algorithm 1 FeaturePMI: Get weighted feature matrix

input Origin node vector O, Target node vector T and,
Attribute dictionary A, window size w

Initialize empty PMI matrix P.
Initialize empty feature matrix F
Initialize empty weighted PMI matrix W
for each $o \in O$ **do** // Find transition probability with node occurrence
  $t \leftarrow T[i]$
  $P[o,t] \leftarrow P[o,t] + 1$ // Count matrix
  $i \leftarrow i + 1$
$P \leftarrow P/(max(P) + 1)$ // Transition probability matrix
for each $o \in O$ **do** // Use origin nodes to sample features
  $t \leftarrow T[i]$
  for all $f \in features$ **do**
    $distance \leftarrow (A[f,o] - A[f,t])^2$ // Distance between features
    $F[f,o,t] \leftarrow distance$
$F \leftarrow F/max(F)$
$W \leftarrow P * F$ // Derive feature weighted transition probability
return W

In the next step, calculation of an entire matrix with pairwise differences of each feature is possible, but rather time consuming. Therefore, the random walks are used to sample a very sparse feature matrix F. F is constructed with only the available samples in the random walks and it represents all feature distances between nodes. Finally, we multiply matrix P and F to get a weighted representation of the distances in matrix W. By combining matrix P with matrix F we implicitly model the graph structure in each feature embedding.

		Transition Probability	
		High	**Low**
Distance	**High**	High	Medium
	Low	Medium	Low

Table 6: Resulting scores from multiplication show that similar values for transition probability and distance show polarized values, and are arguably the most interesting observations.

Another interesting aspect we found for the algorithm featPMI is that the distance and transition probability multiplied, implicitly model highly divergent observations (instead of similar ones). For this part of the algorithm we assume that, the most interesting nodes are the nodes that have a high distance (i.e. low similarity) and a high transition probability, or the other way around. A question that could be asked is, why does node

a have a high transition probability to node b when their similarity is low? This is also true for the opposite case: why does a node with low distance (i.e. high similarity) have a high transition probability? Table 6 shows the interactions between both metrics and the ones we focus on in blue. This table, also shows an interaction in orange, which is of lesser importance due to the intrinsic proximity preserving property (i.e. high transition probability to close nodes) of using random walks. The cell in white contains high distance (i.e low similarity) and low transition probability and is likely to be of less importance due to the inherent random occurrence of this node-pair in the sampling. The resulting scores are fed into an matrix factorization algorithm, in which higher divergences appear to have a greater predictive value.

Complexity analysis: The featurePMI time complexity differs greatly based on the sampling strategy. In the brute force approach, it is necessary to calculate all pairwise node distances making this a complex operation of the number of features times n squared. It is possible to estimate a more fair complexity analysis by making an assumption about how n is influenced by this strategy. With random walks the complexity can be reduced to: window size $\times$ the number of walks. We set window size to $\sqrt{n}$ and, thus can approximate that $\sqrt{n} \times$ walk_number calculations have to be made, for simplicity we set the walk number to 20 for each experiment. Since this value is a constant, the walk_number can be ignored in the rest of the analysis.

The complexity of the loop in algorithm 1 in lines 4:7 is $T(n) \rightarrow O(o)$. This can be transformed to to $O(w*n)$ when considering that o is essentially a larger vector of n with multiple of size w. Each origin node is represented by the window size, or $o = w * n$. As mentioned in the section above, w is set to $\sqrt{n}$, and thus o is reduced to $n^{1.5}$.

Next, for the loops in lines 9:13 the complexity is $T(n) \rightarrow O(o * f)$, in which o can be substituted with $n^{1.5}$. This finally results into the following time complexity: $O(n^{1.5} * f)$.

The combined complexity of both steps are $O(n^{1.5} * f + n^{1.5})$. Note that algorithm 1 has separated both loops for clarification, but it is actually possible to combine both loops by assigning the values needed more efficiently. Removing one of the required loops directly removes the least time consuming loop from the complexity analysis, and results in a time complexity of: $O(n^{1.5} * f)$. In this specification, graphs with over 100,000 nodes can be embedded within 24 hours, and graphs with over 1 million nodes within 23 days (see section 7.4.3 for more details on runtime). Note that these example calculations are made with a single core, no parallelization and no usage of GPU, therefore optimization of these elements could decrease runtime significantly.

In step 8 the resulting complexity of dividing a matrix $n * w$ is equal to $O(n^{1.5})$. In step 14 however, the complexity is equal to $O(n^{1.5} * f)$ due

to the 3 dimensional matrix of $n * w * f$. Furthermore, In the last step the complexity is again equal to $O(n^{1.5} * f)$ due to elementwise multiplication of matrix P and F.

Combining all complexities of this algorithm results in a total of $O(3 * f * n^{1.5} + n^{1.5})$, which simplifies to $O(f * n^{1.5} + n^{1.5})$ when not considering constant variables.

Algorithm 2 FeaturePMI: Get embeddings

input Embedding dimension d, Number of features n, α
Weighted PMI matrix W

1: Initialize empty embeddings array E.
2: Initialize empty matrix M
3: **for** all f in $range(0, n)$ **do**
4: $M \leftarrow W[i]$ // Current feature matrix
5: $x_{emb}, y_{emb} \leftarrow ALS(M * \alpha, d)$ // Alternating Least Squares
6: $E[i] \leftarrow x_{emb}$ // Current feature embedding

return E

The next step is to derive embeddings from matrix W. For each feature, the Alternating Least Squares algorithm [94] is used to derive the final node embeddings. This is a method from recommender systems and can be used on incomplete matrices to obtain missing node embeddings.

The complexity of ALS is $O(2n * d^2 + (n + w) * d)$ per iteration [94], where d is the number of dimensions. Iterations are kept constant in this thesis and, therefore, the loop in 3:6 has a total time complexity of $T(n) \rightarrow O(2n*d^2*f+(n+w)*d*f)$. As specified above, $n+w$ is equal to $n^{1.5}$ reducing this complexity to $O(2n*d^2*f+n^{1.5}*d*f)$. The total complexity of the entire algorithm is an addition of the complexity of algorithm 1 and algorithm 2. This results into $T(n) \rightarrow O(f*n^{1.5}+n^{1.5}+2n*d^2*f+n^{1.5}*d*f)$. If f and d are set to be equal in experiments, the runtime compared to the number of nodes is expected to increase with the following complexity $O(3n^{1.5} + 2n)$, or without constants $O(n^{1.5} + n)$.

One final note for this algorithm is that there is a clear tradeoff between speed and accuracy. By changing the sampling strategy from $n^{0.5}$ to $n^{0.25}$ should reduce time complexities massively, and it makes this algorithm more scalable albeit less precise.

6.3 FeatureWalk

The next algorithm to discuss is the featureWalk algorithm which is compared to featurePMI in order to test their different strategies and discover different patterns in embedding multivariate graphs. In contrast to featPMI, FeatureWalk focuses on the concept of random walks on graphs and their

feature graphs. It differs by leveraging the random walk paradigm to sample features in the same manner as the graph structure. Furthermore, this algorithm does not explicitly model the graph structure in the combined embeddings, rather it introduces this structure after embedding and before seriation. Algorithm 3 shows the pseudocode of featureWalk in combination with two existing algorithms.

Algorithm 3 FeatureWalk: Generate Embeddings

input Feature matrix F, Dimension d, number of nodes n, d walk length wl and walk number wn

1: Initialize empty embedding matrix E.
2: **for** each $f \in F$ **do**
3: $k \leftarrow round(\sqrt{n})$ // k = number of neighbours
4: $k\_graph \leftarrow k\_neighbours(F[f], k)$
5: $k\_graph \leftarrow k\_graph - 1$ // reverse weights and distances
6: $G \leftarrow (k\_graph)^2$ // increase negative effect distance
7: $model \leftarrow Walklets(G, d, wl, wn)$ // Random walks per feature
8: $emb \leftarrow model.get\_embedding()$
9: $E[f] \leftarrow emb$

return E

In this algorithm, the first step is to construct a k-neighbours graph for each feature in the feature set. This creates several new graphs upon which the embedding algorithm Walklets will run and derive embeddings. In step 5 and 6 the weights and distances are switched for usage further in the pipeline, furthermore, the negative effect of larger distances is increased. Afterwards, this graph is fed into Walklets with a prespecified dimension, walk length and walk number. Note, that the algorithm is flexible and graph embeddings can be derived by any other embedding technique. Also note, different from featurePMI, that this algorithm does not implicitly model the structure in the feature embeddings, and thus a structure embedding should always be included in the final embeddings set.

Complexity Analysis In the featureWalk algorithm a KD tree [95] is used to construct the k-nearest neighbour graph. This KD tree has a time complexity of $O(n * log(n))$ [96], and since our algorithm creates such a graph for each feature, the total time complexity of tree generation is $O(n * log(n) * f)$.

For creating the embeddings with Walklets, the total time complexity is $O(n * d * f * p)$ when combining information from table 1 with the fact that Walklets has to be run f times. The total complexity is therefore, $O(n * log(n) * f + d * n * f)$ when p (i.e. number of walklets) is taken as a constant. According to our experiments and calculations, for over 100,000 nodes this algorithm runs in under 24 hours, for over 1 million

nodes featWalk is even more scalable than featPMI, running under 10 days (see section 7.4.3 for more details on runtime).

6.4 Algorithm Variants

As specified in section 6.1 we need structure embeddings, combined embeddings, separate embeddings, and subsets of different combinations of embeddings.

For structure embeddings, the algorithms in section 3.3.1 can be run in conjunction with featurePMI and featureWalk. Additionally, the separate embeddings can be obtained as explained in the sections above. Furthermore, in section 5 the possibility of combining embeddings has been explained and therefore, the only question that remains is the possibility of generating combined embeddings. For both algorithms there exists an extended version that includes a full early fusion of every feature. FeaturePMI for instance, sums all distance matrices F, squares the result and introduces a combined feature matrix to be embedded. For featureWalk, the k-neighbours graph estimates the nearest neighbours in higher dimensional space (i.e. all feature values represent a point in space) and this graph is fed into an embedding algorithm. In addition, it is also possible to generate medium fusion visualizations by combining several of the independent embeddings in the visualizer stage. With these additions the combined embeddings requirement is satisfied.

Some shortcomings of these algorithms are the inability to deal with textual, image and audio data. Additionally, the algorithms are not able to handle missing attributes, disconnected nodes and directed graphs.

Since these algorithms depend on the number of features, they can be further sped up by pre-processing/removing features to account for multicollinearity or non-significance. And secondly, a variant on featureWalk separates embeddings on binary attributes versus real valued attributes. The binary attributes will no longer be embedded by Walklets, and this has the advantage of more speed and accuracy. And finally, the construction of a Ball Tree in featureWalk before running the KD tree algorithm increases accuracy in higher dimensions (i.e. more features) [97], but is more expensive [96].

7 Experiments

In this section, the previously defined framework, and the novel algorithms are tested. First in section 7.1 the datasets used in the experiments are introduced with summary statistics. In section 7.2 the ability of graph algorithms to show canonical data patterns is observed with the assistance of small generated test graphs, and a single real graph. Thereafter, in section 7.3 a real world dataset which is weighted and directed is explored, and differences with respect to the unweighted version are highlighted. Subsequently in section 7.4 multivariate graphs are introduced and embedded with the novel algorithms in section 6. And finally, in section 8, inferences will be made on the readability of these visualizations.

All experiments without explicit hyperparameters are performed with the algorithmic settings in appendix B, or if not explicitly specified, with standard settings by the author of the algorithm. For temporal comparison, all experiments have been run on a single core adaptive 3.8-4.4 GHz processor, with 16 GB of 3200 MHz of RAM.

7.1 Dataset Collection

The generated graphs included in this section are well known graphs existing of a predefined structure and properties. These Python Networkx library [98] generated graphs are small, and specifically designed to uncover patterns in the data. These graphs essentially served as a proof of concept for pattern recognition on the patterns introduced in Behrisch et al. [6] and include the Block Pattern, Off-diagonal Block pattern, Line/Star Pattern, Bands Pattern and the Noise Anti-Pattern. To achieve this goal, the following graphs have been generated:

- (1) random, graph where two nodes are randomly connected with an edge
- (2) star, graph where each node is connected to a single hub node
- (3) barbell, graph with two cliques, two hub nodes and a connecting node
- (4) bipartite, graph with two disjoint sets of nodes
- (5) line graph, graph where each consecutive node has an edge

The graphs listed above are all basic graphs without directed edges, edge weights or nodal properties. In the first experiment the random graph inquires for the Noise Anti-Pattern and tests the inherent randomness (or non structure) in the graph. Second, the star graph tries to exploit the Line/Star pattern by having a single highly influential node. Thereafter, the

barbell graph will be used to highlight block patterns with the occurrence of cliques in the graph. Afterwards, the bipartite graph showed the ability of graph embedding algorithms to visualize the Off-diagonal Block pattern, with the bipartite graphs inherent disjoint sets. Consequently, the line graph tries to utilize its path property to accommodate for the visualization of a bands pattern. And finally, the football graph has tested the manifestation of patterns in a real world setting.

To test the ability of the weighted, and directed version of these embedding algorithms, a visualization of the US airports dataset is shown. In appendix C we included an extra visualization of another graph, a graph constructed with the API of the Nederlandse Spoorwegen (or NS) for the interest of the reader.

name	*kind*	*directed*	$\|V\|$	$\|E\|$	NP^*	$\Delta(G)$	$\delta(G)$	$d_{avg}(G)$
random	basic	no	20	89	0	13	5	8.9
star	basic	no	21	20	0	20	1	1.9
barbell	basic	no	21	92	0	10	2	8.8
bipartite	basic	no	20	100	0	10	10	10
line	basic	no	21	20	0	2	1	1.9
football	basic	no	115	613	0	12	7	10.7
NS	weighted	yes	247	354	0	17	1	2.9
USair	weighted	yes	286	4587	0	166	1	16.1
pokec-100	multi	no	100	313	6	86	1	6.3
pokec-1000	multi	no	995**	4411	6	230	1	8.9
pokec-5000	multi	no	4983**	41697	6	1030	1	16.7

*node properties and,
**divergence of original size due to node removal of nodes without profile data

Table 7: Summary statistics datasets

In the final experiment, the multivariate graph embedding algorithms introduced in section 6 will be compared against visualizations by naive DeepWalk. For this experiment, the Slovakian Pokec dataset was chosen since it is a very large online social network with 1.6 million users, 30 million friendships and most importantly at least six binary/numerical node attributes [99]. The novel algorithms will be run on a subset of the Pokec dataset, sampled with BFS. This allowed for easy visualization and explaining, reduced runtime and still gives a demonstration of the feasibility of visualizing larger graphs. Table 7 shows the descriptive statistics of the datasets used in this section.

7.2 Canonical data patterns in Basic Graphs

The experiments in this section highlight patterns that arise for simple artificially generated graphs with the algorithmic settings for this experiment

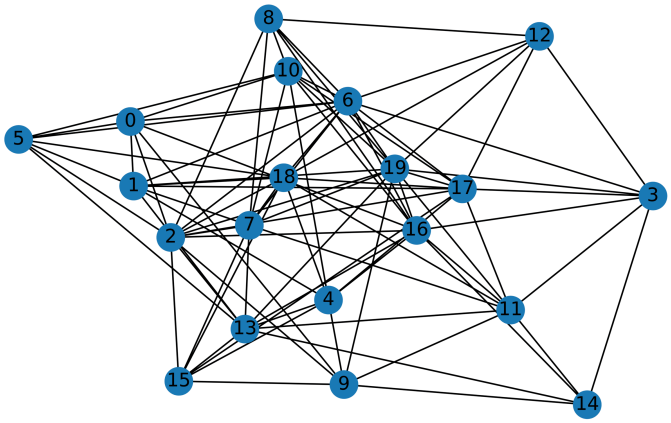
listed in appendix B. The section is structured by first: displaying the to-be embedded graph, and second showing the resulting reordered matrices of all graph embedding algorithms. Each experiment has been run five times and the best result (according to the minimal reconstruction scores) is visualised in a 3x3 grid.

The first graph to be tested was a random graph shown in figure 27a, and it was created with a Networkx Erdős-Rényi graph with the number of nodes equal to 20. Furthermore, the probability of edges between these nodes was set to 0.5 and there should be little to no patterns visible in the resulting embedding visualization.

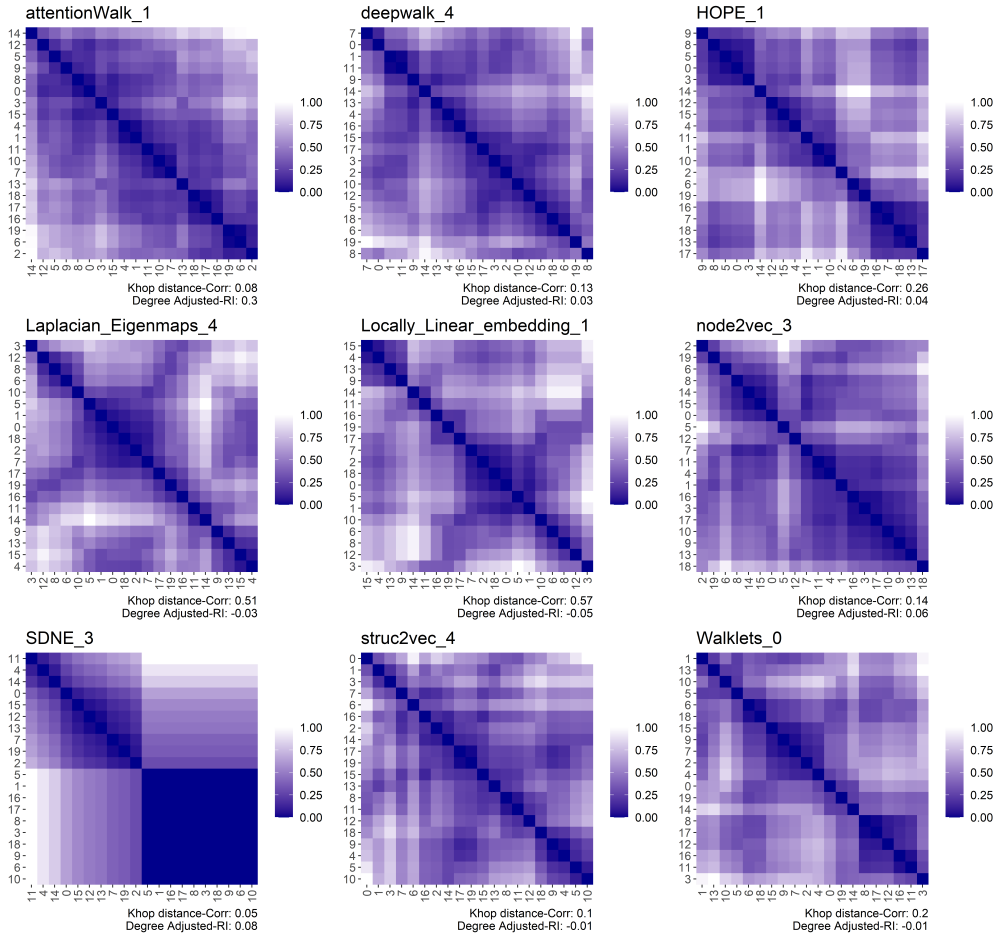
In figure 27b it was found that indeed most algorithms do not have clear patterns visible in their embeddings, with only some rare ill-defined block patterns visible. The exception of this was SDNE which visualizes two large block patterns (in which the upper contains multiple smaller block patterns), and many halves of a line/star pattern. It was found that SDNE overfits quickly on these simple problems and achieves subpar results. An inspection on the quality metrics however (low Khop-score, ARI close to zero) support our claim that most embeddings have indeed not discovered any pattern in the data.

In figure 28a a Networkx star graph was generated with a single central node connected to 19 peripheral nodes. This extreme network was generated to test for the line/star pattern. The embeddings in figure 28b illustrate that 6 out of the 9 embeddings are able identify node 0 as the central node. Furthermore, the algorithms: attentionWalk, HOPE, SDNE and Walklets have shown clear line patterns for the strongly connected central node and, note they have categorized all of the other nodes as similar nodes. These algorithms obtained perfect or near perfect reconstruction scores (i.e. 0.99 to 1). Furthermore, deepwalk and node2vec were less distinctive in recognizing the star/line pattern, obtained a lower Khop distance and a lower degree adjusted ARI score. On the other hand, the algortihms: Laplacian Eigenmaps, Locally Linear Embedding and struc2vec have shown to score the lowest on these metrics and completely fail to show any relevant higher level pattern.

In figure 29a a Networkx barbell graph has been generated with two identical complete graphs of 10 nodes (e.g. cliques) and a single connecting node (node 10). Node 9 and 11 are a special case which have an extra degree (compared to the clique nodes) and connect to the central node. This graph was generated to test for the block pattern. The resulting matrices in 29b illustrate that all algorithms display block patterns. Subsequently, the complete graph structure, the central node and hub node have been correctly identified for 8 out of 9 graph embedding algorithms. While most algorithms score high on the ARI score, only SDNE is observed to score low with an ARI of 0.16. In the visualization it was recognized that only SDNE distinctly groups node 9 and 11 in the same block as all other nodes. Note that for

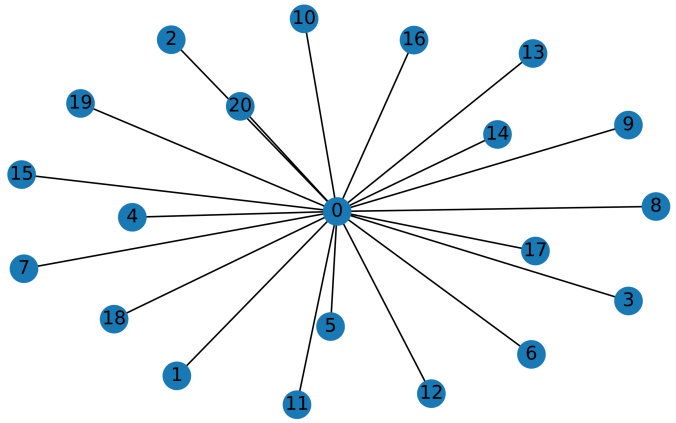


(a) Node Link diagram of a Random Graph

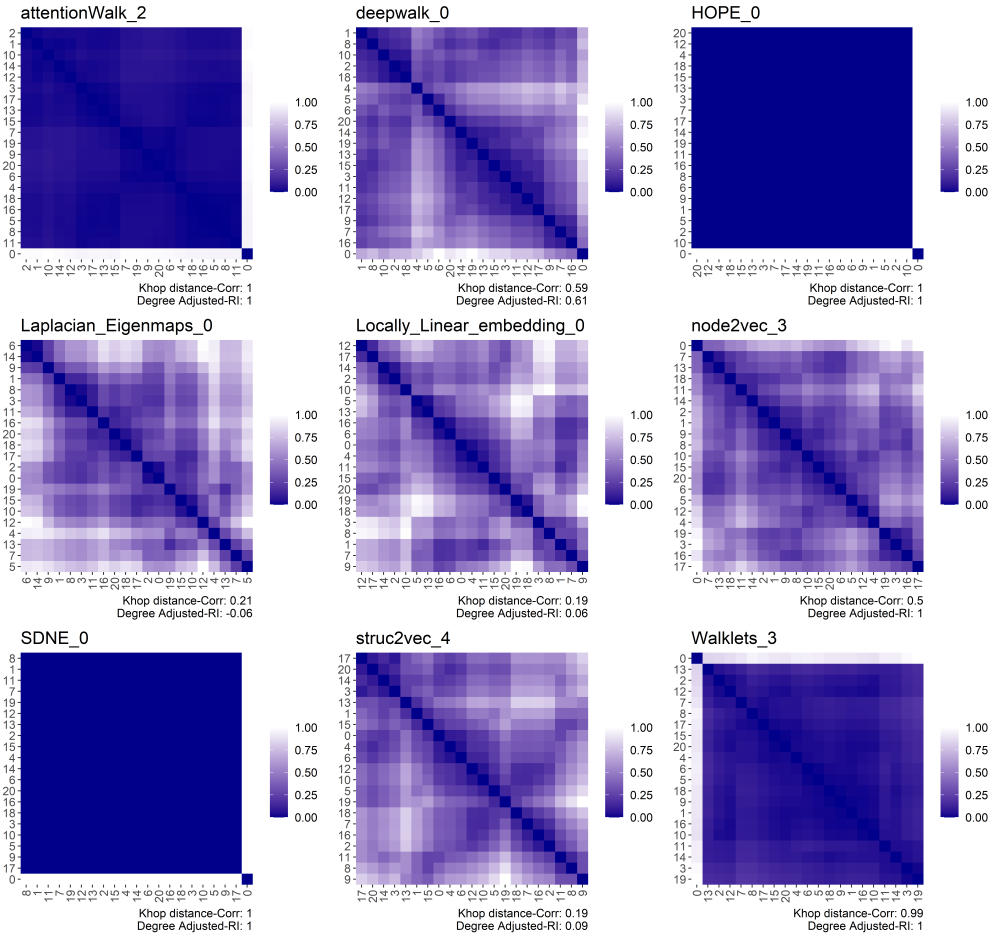


(b) 2D matrix visualization

Figure 27: Random Graph



(a) Node Link diagram of a Star Graph



(b) 2D matrix visualization

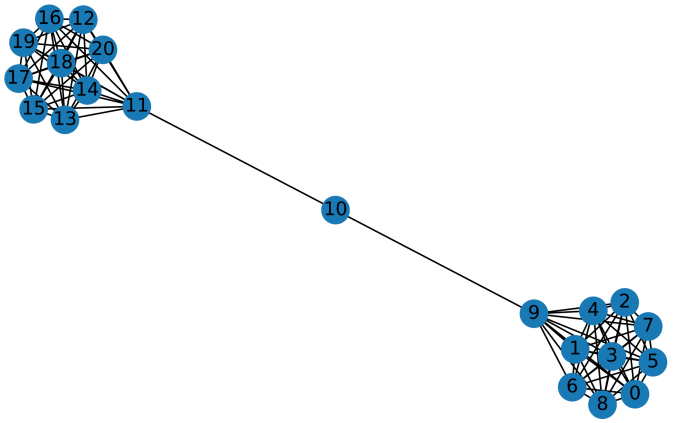
Figure 28: Star Graph

algorithms HOPE, deepwalk and node2vec this distinction is hard to spot due to very minor changes in hue. The only highly divergent result is in the visualization of struc2vec. Evidently, this was known a priori, since struc2vec explicitly preserves structure and not neighbourhood similarity. Therefore, the algorithm has placed all nodes in the same category with the exception of the nodes 9, 10 and 11. This is due to the fact that the peripheral nodes are structurally equivalent in the barbell graph. Additionally, the algorithm has recognized that node 9 and 11 are also structurally equivalent and only node 10 is highly dissimilar to any of the other nodes.

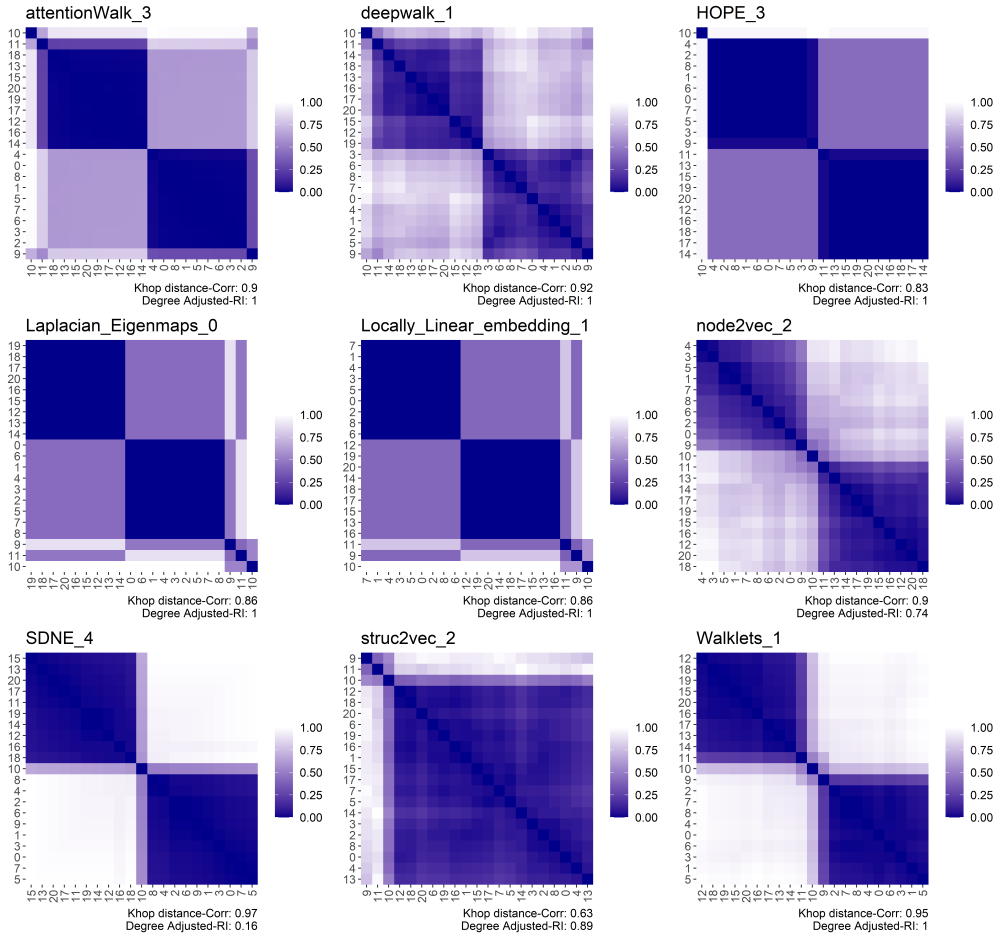
In figure 30a a Networkx complete biparite network with 10 nodes on either side was generated to check for the off-diagonal block pattern. In figure 30b the results have shown that 6 out of 9 algorithms are unable to clearly show any patterns in the data. For the algorithms of HOPE, SDNE and Walklets however, the results have been found to be similar to the barbell graph without the central nodes. Additionally, for the aforementioned algorithms, nodes 0-9 are correctly displayed on one side of the bipartite graph and node 10-19 on the other. Note that the Khop score clearly identifies algorithms that have better visualizations. As an example, all algorithms excluding HOPE, SDNE and Walklets have a low Khop score and show no patterns in the data. An ARI score of 1 however, indicates a good score on this metric. Unfortunately the ARI score is of no use in this example due to each algorithm scoring the maximum value. This is due to the definition of this score, the score will always be 1 for graphs where all nodes have an equal degree (see section 4.6.3 for more details).

In figure 31a a Networkx line graph (i.e. random lobster graph with 20 nodes and 0 probability of adding edges to the backbone) was visualized to illustrate the occurrence of the off-diagonal bands pattern. In this experiment we observed that 4 out of the 9 algorithms (HOPE, Laplacian Eigenmaps, LLE and Walklets) clearly show a band pattern on the diagonal instead of the expected off-diagonal pattern. Furthermore, it was found that 2 out of these 4 algorithms (i.e. HOPE and LLE) show consistent ascending/descending ordering of nodes. Moreover, note that Walklets has a consistent ordering, which, unexpectedly groups even and uneven nodes. Unfortunately, SDNE, struc2vec and attentionWalk seem to fail on this example and do not show a clear picture of the line graph. For the reconstruction scores it was found that HOPE, LLE and Laplacian Eigenmaps scored highest on Khop, while ARI has shown to be less consistent in this example.

To conclude, this experiment shows that the embedding algorithms are at least able to learn some of the patterns in Behrisch et al. [6]. Most notable are the block and line/star pattern which are clearly visible in the figures listed above. Nevertheless, in this experiment we observed mixed results for some of the algorithms when considering frequency of good visualizations in which some algorithms clearly performed better than others for different tests. On these experiments the most accurate algorithms are multiscale

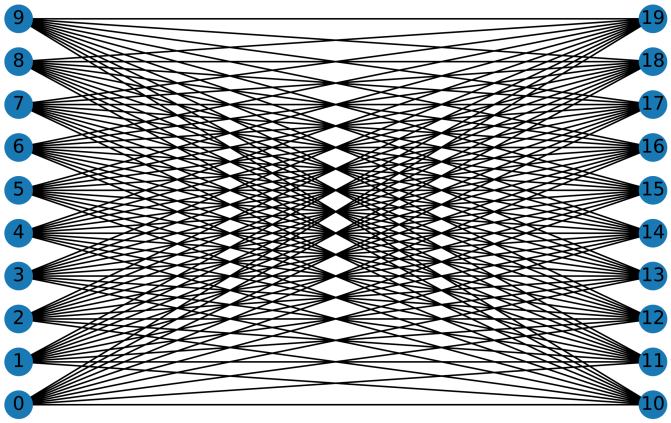


(a) Node Link diagram of a Barbell graph

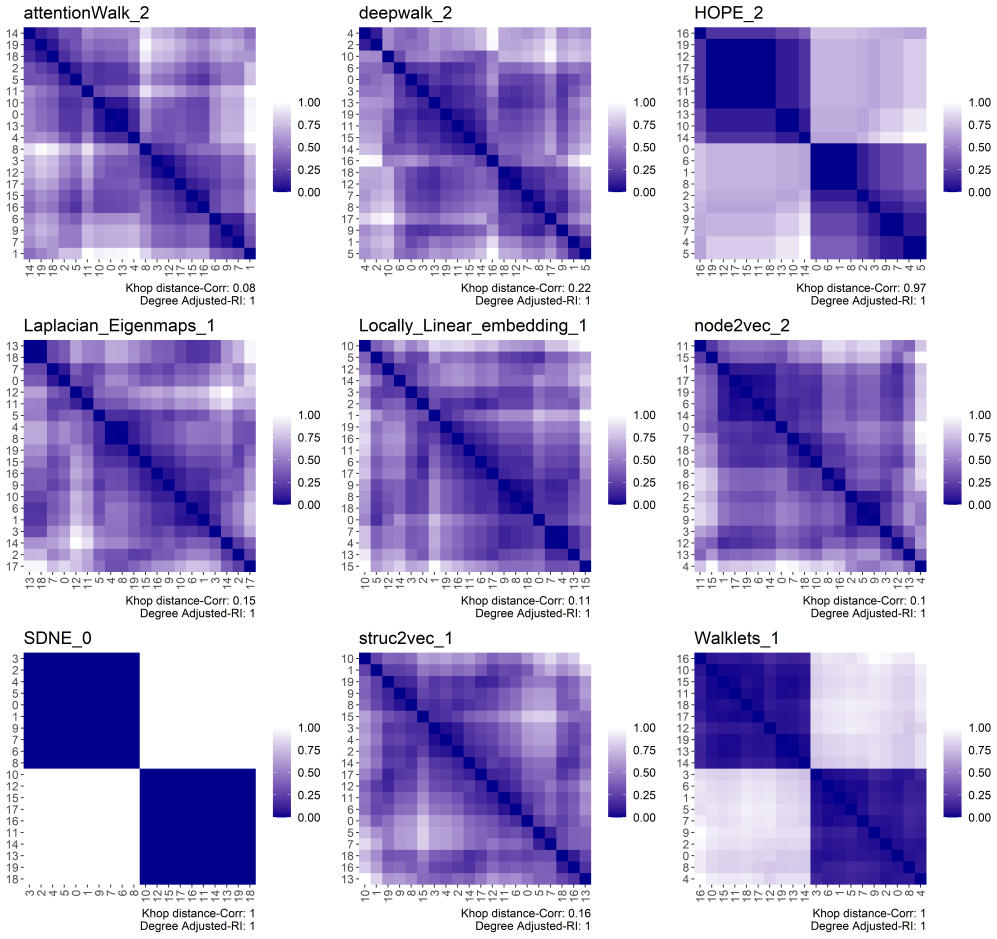


(b) 2D matrix visualization

Figure 29: Barbell Graph

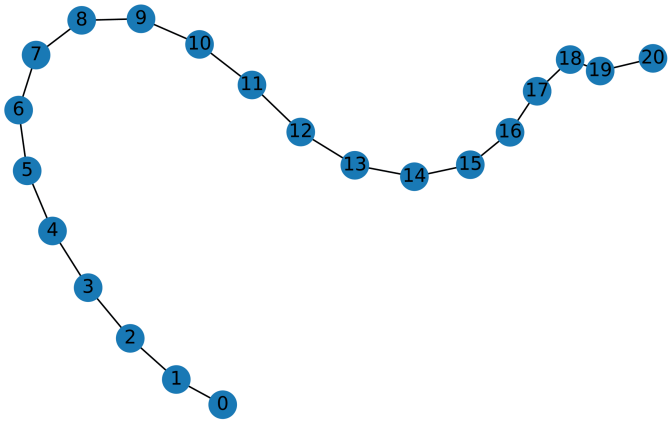


(a) Node Link diagram of a Bipartite graph

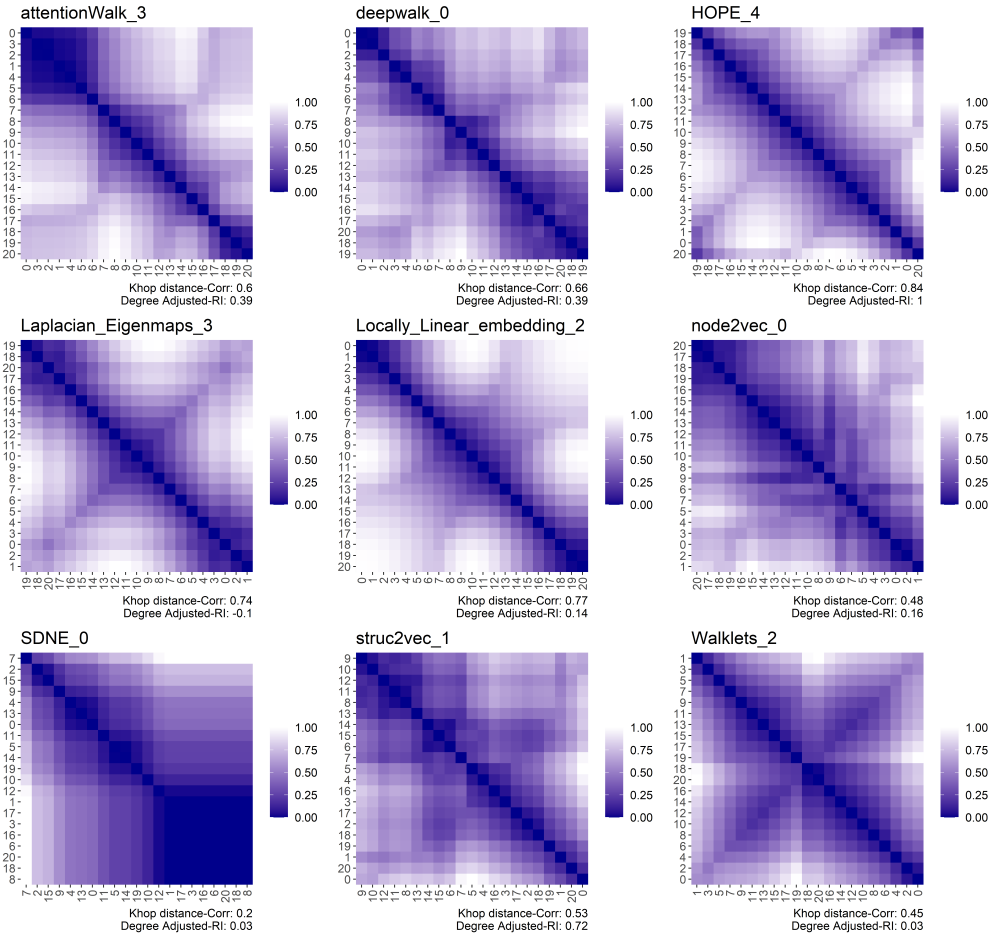


(b) 2D matrix visualization

Figure 30: Bipartite



(a) Node Link diagram of a Line graph



(b) 2D matrix visualization

Figure 31: Line

Walklets and HOPE, which showed reasonable results for all of the examples highlighted above. With these examinations we found that the patterns shown in these figures have given insights into answering research question 4.

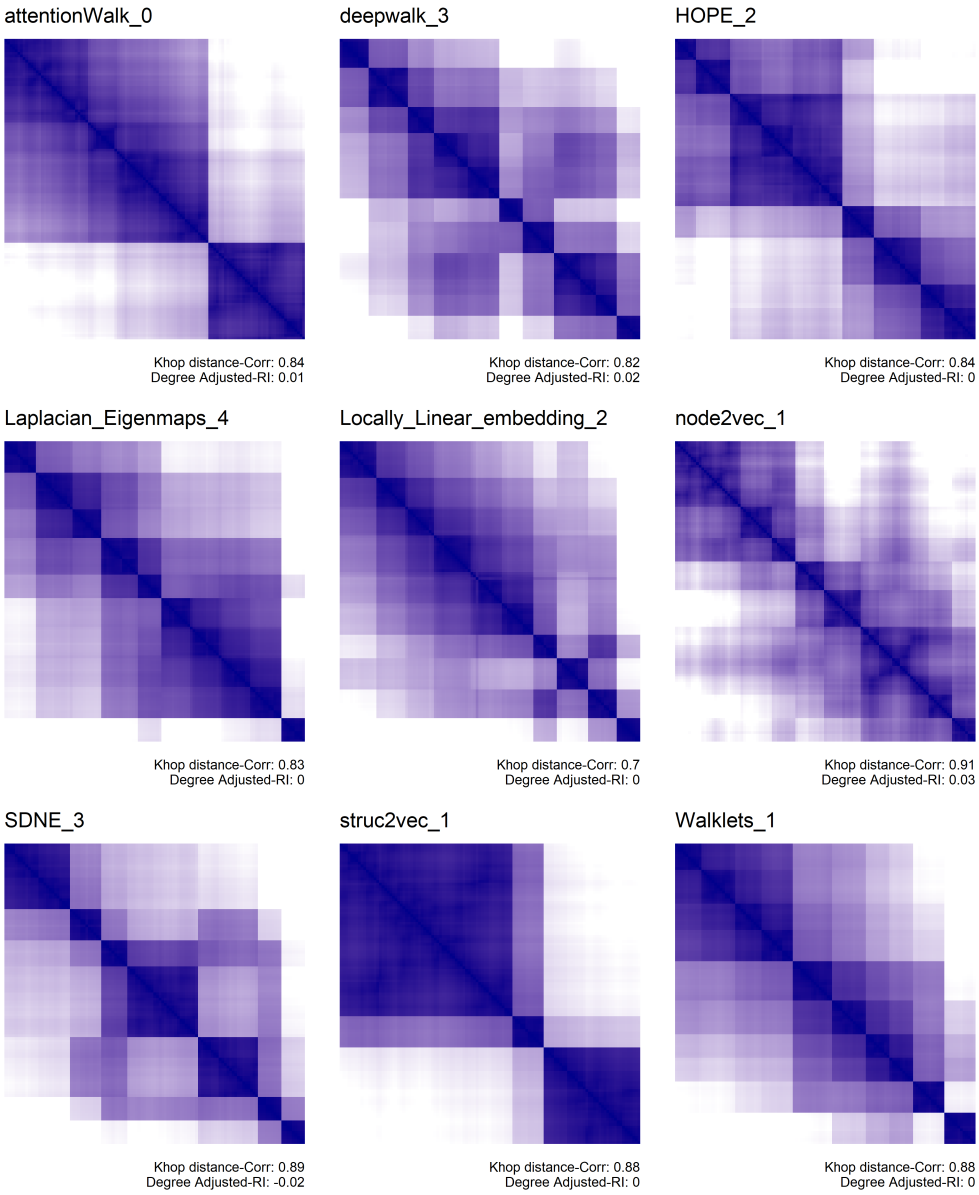


Figure 32: 2D matrix visualizations of dataset football

Finally, this section concludes with a simple graph visualization of the dataset football [100]. The football dataset consists of American football

Average Visual Quality Metric Scores of 5 Runs

Dataset: *football*

VQM	*atWalk*	*deepwalk*	*HOPE*	*Lap_Eig*	*Loc_Lin*	*node2vec*	*SDNE*	*struc2vec*	*Walklets*
2-Sum	1.77×10^{7}	1.67×10^{7}	1.66×10^{7}	1.74×10^{7}	1.67×10^{7}	1.66×10^{7}	1.68×10^{7}	1.69×10^{7}	**1.64×10^{7}**
Anti-Rob Deviations	2.63×10^{3}	1.21×10^{4}	5.88×10^{3}	5.41×10^{3}	7.82×10^{3}	8.32×10^{3}	1.05×10^{4}	**1.32×10^{3}**	5.95×10^{3}
Anti-Rob Events	**5.22×10^{4}**	9.87×10^{4}	7.88×10^{4}	6.68×10^{4}	6.74×10^{4}	7.88×10^{4}	8.16×10^{4}	6.15×10^{4}	6.25×10^{4}
Banded Anti-Rob	4.66×10^{3}	6.51×10^{3}	5.42×10^{3}	5.38×10^{3}	6.34×10^{3}	5.95×10^{3}	6.38×10^{3}	**3.84×10^{3}**	5.27×10^{3}
Cor_R	1.13×10^{-1}	1.13×10^{-1}	1.47×10^{-1}	**9.80×10^{-2}**	1.13×10^{-1}	1.33×10^{-1}	1.14×10^{-1}	1.68×10^{-1}	1.46×10^{-1}
Gradient measure	**3.71×10^{5}**	2.81×10^{5}	3.21×10^{5}	3.39×10^{5}	3.42×10^{5}	3.21×10^{5}	3.15×10^{5}	3.54×10^{5}	3.52×10^{5}
Weighted G measure	1.23×10^{5}	1.11×10^{5}	1.30×10^{5}	1.16×10^{5}	1.21×10^{5}	1.24×10^{5}	1.15×10^{5}	**1.51×10^{5}**	1.32×10^{5}
Inertia	2.06×10^{7}	2.23×10^{7}	2.29×10^{7}	2.08×10^{7}	2.27×10^{7}	2.30×10^{7}	2.23×10^{7}	2.31×10^{7}	**2.34×10^{7}**
Lazy path length	1.98×10^{2}	2.15×10^{2}	1.90×10^{2}	1.77×10^{2}	2.06×10^{2}	2.44×10^{2}	2.11×10^{2}	**1.35×10^{2}**	1.67×10^{2}
Least Squares	2.85×10^{7}	2.84×10^{7}	2.84×10^{7}	2.85×10^{7}	2.84×10^{7}	2.84×10^{7}	2.84×10^{7}	2.85×10^{7}	**2.84×10^{7}**
Linear Seriation	**3.73×10^{5}**	4.91×10^{5}	4.48×10^{5}	4.03×10^{5}	4.63×10^{5}	4.66×10^{5}	4.67×10^{5}	3.89×10^{5}	4.67×10^{5}
Measure of effect	1.45×10^{4}	1.22×10^{4}	1.30×10^{4}	1.37×10^{4}	1.26×10^{4}	1.25×10^{4}	1.26×10^{4}	**1.49×10^{4}**	1.26×10^{4}
Stress Moore	1.68×10^{2}	1.86×10^{2}	1.60×10^{2}	1.77×10^{2}	1.81×10^{2}	**1.08×10^{2}**	2.12×10^{2}	2.24×10^{2}	1.49×10^{2}
Stress Neumann	5.72×10^{1}	6.40×10^{1}	5.46×10^{1}	6.05×10^{1}	6.25×10^{1}	**3.74×10^{1}**	7.28×10^{1}	7.59×10^{1}	5.09×10^{1}
Ham Path length	3.22	3.78	3.31	3.16	3.78	4.27	3.63	**2.39**	3.02
RGAR	**1.06×10^{-1}**	2.00×10^{-1}	1.60×10^{-1}	1.35×10^{-1}	1.37×10^{-1}	1.60×10^{-1}	1.65×10^{-1}	1.24×10^{-1}	1.27×10^{-1}

*__Cyan__ (lower is better), indicates best low score. __Yellow__ (higher is better), indicates best high score.

Figure 33: Visual quality metrics of algorithms for dataset football

games in the highest college division. The vertices are teams from different colleges and the edges between these represent matches played. This dataset consists of 12 distinct labels placing each of the teams into conferences in which teams are more likely to play against each other. In figure 32 the visualization results are given. We observed that DeepWalk, Laplacian Eigenmaps, Locally Linear Embedding and Walklets recognize 10 distinct groups in the data, while the other algorithms had less block patterns in their visualization. Scores among visualizations differ slightly with Laplacian Eigenmaps scoring worst, and node2vec scoring highest on Khop similarity. Additionally, node2vec scored highest and SDNE lowest on degree adjusted ARI. Thereafter, figure 33 shows the visual quality metrics of the visualizations in which attentionWalk scores best on 4, Laplacian Eigenmaps scores best on 1, node2vec scores best on 2, struc2vec scores best on 6 and Walklets scores best on 3.

7.3 Patterns in Weighted Graphs

In order to explore the changes in matrix visualizations of unweighted graphs compared to weighted graphs, we analyzed the dataset US air. US air is a directed dataset with 286 airports in the United States, and includes 4587 directed edges between these airports. In this dataset each edge represents a unique flight path between two airports, while the weights describe the

number of flights on the flight paths. The data were embedded with the embedding algorithms DeepWalk and Walklets and hyperparameters of: 80 dimensions; walk number of 10; walk length of 80; window size of 5; and 10 epochs, for both algorithms.

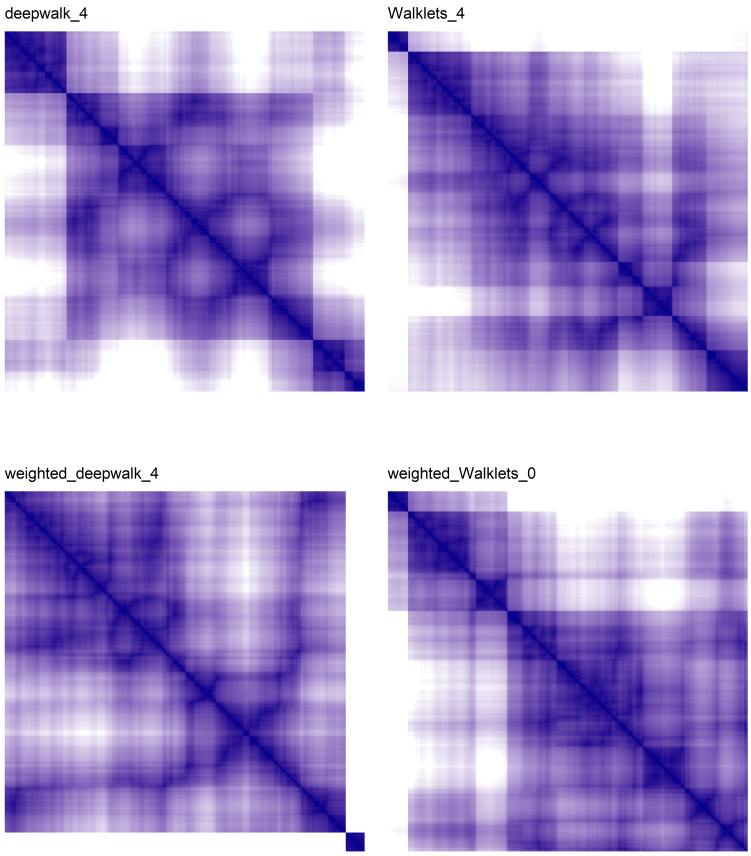
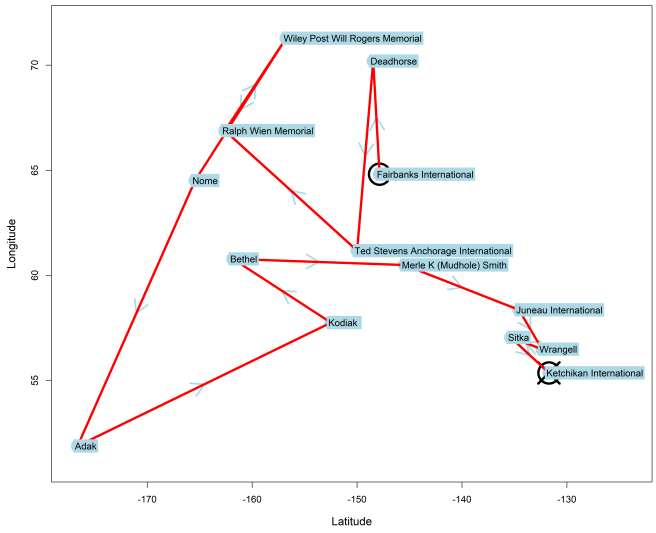


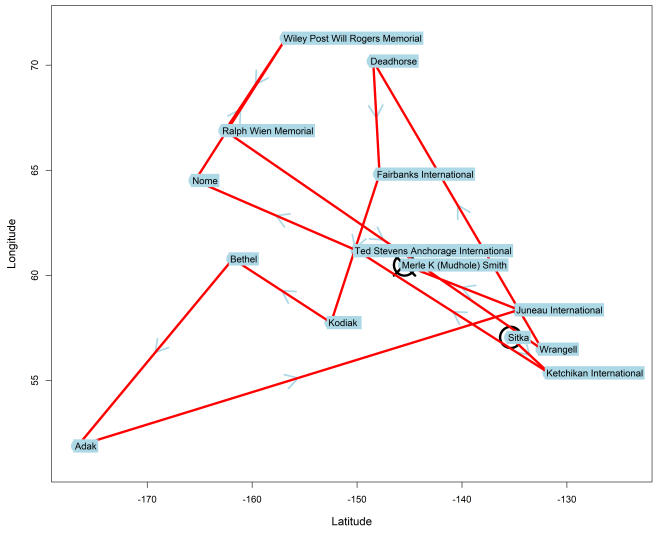
Figure 34: Distance matrices of unweighted and weighted dataset US air

In figure 34 the embeddings are visualized. When unweighted DeepWalk and weighted DeepWalk were compared, the most prominent observation was that the former shows two smaller block patterns on the edges of the matrix, and a single large block pattern in the center. The weighted version however, showed less distinction between blocks, but clearly separated a small block of outliers in the bottom right. On the other hand, intra-Walklets comparison showed less variation in these matrix visualizations, with only small changes in hue.

To explore this phenomenon further, in Walklets we plotted the TSP paths of both the weighted and unweighted version in figure 35. We opted



(a) Unweighted path



(b) Weighted path

Figure 35: TSP path of visiting airports for Alaska region

to plot only the Alaska region for explanation purposes. Note that in this image a circle indicates the start of the path, and a crossed circle represented the end of the path. It was observed that the unweighted Walklets (i.e. top image) favoured a proximity preserving approach, visiting mostly close (connected) airports, while the weighted version (i.e. bottom image) was

more inclined to adhere to the proximity including number of flights between airports.

Average Visual Quality Metric Scores of 5 Runs

Dataset: *directed_weighted_USair*

VQM	*deepwalk*	*Walklets*	*weighted_dee*	*weighted_Wal*
2-Sum	**6.44×10^{8}**	6.58×10^{8}	6.76×10^{8}	6.77×10^{8}
Anti-Rob Deviations	1.92×10^{5}	2.65×10^{5}	**1.59×10^{5}**	1.72×10^{5}
Anti-Rob Events	1.57×10^{6}	1.58×10^{6}	**1.45×10^{6}**	1.53×10^{6}
Banded Anti-Rob	9.79×10^{4}	1.04×10^{5}	8.99×10^{4}	**8.60×10^{4}**
Cor_R	1.07×10^{-1}	**8.97×10^{-2}**	9.90×10^{-2}	9.49×10^{-2}
Gradient measure	**4.34×10^{6}**	4.29×10^{6}	4.32×10^{6}	4.27×10^{6}
Weighted G measure	**1.74×10^{6}**	1.57×10^{6}	1.63×10^{6}	1.62×10^{6}
Inertia	**8.51×10^{8}**	8.15×10^{8}	7.74×10^{8}	7.71×10^{8}
Lazy path length	1.02×10^{3}	1.09×10^{3}	9.12×10^{2}	**9.05×10^{2}**
Least Squares	**1.10×10^{9}**	1.10×10^{9}	1.11×10^{9}	1.11×10^{9}
Linear Seriation	7.42×10^{6}	7.32×10^{6}	**6.45×10^{6}**	6.47×10^{6}
Measure of effect	7.64×10^{4}	7.71×10^{4}	8.35×10^{4}	**8.37×10^{4}**
Stress Moore	2.35×10^{2}	**2.25×10^{2}**	3.09×10^{2}	3.27×10^{2}
Stress Neumann	7.98×10^{1}	**7.65×10^{1}**	1.05×10^{2}	1.11×10^{2}
Ham Path length	7.17	7.49	6.71	**6.21**
RGAR	2.04×10^{-1}	2.05×10^{-1}	**1.87×10^{-1}**	1.98×10^{-1}

*__Cyan__ (lower is better), indicates best low score. __Yellow__ (higher is better), indicates best high score.

Figure 36: Visual quality metrics of US air

And finally, we compared visual quality metrics between all of the embeddings shown in figure 36. It was found that all algorithms preserve some VQM better than another, with unweighted DeepWalk scoring best in 5 out of 16 VQMs. In this example, DeepWalk is followed by weighted DeepWalk and weighted Walklets with both 4, and finally, Walklets with scoring best on 3.

7.4 Patterns in Multivariate Graphs

In this section a comparison was made between three different graph embedding algorithms. Naive DeepWalk served as a baseline, and is only able to discover structure in the graph, not taking into account node properties. Thereafter, featPMI shows the matrix visualizations consisting of: a structure embedding; a combined feature embedding; and six separated feature embeddings, respectively. Note that featPMI embedds the structure directly into each embedding, and thus we assume that this combined feature embedding is actually a combined feature and structure embedding by design. On the other hand, featWalk shows visualizations of: the structure; com-

bined features (without structure); six feature embeddings and; a combined feature and structure embedding. A final comparison is made by comparing VQM scores and pattern visibility in the visualization(s).

In this experiment the Pokec-1000 dataset is described by short codes in the following summarization. These codes are also present in the upcoming visualizations. There are six numerical and boolean attributes available in this dataset, and unscaled summary statistics are given in table 8. For the results of the Pokec-100 and Pokec-5000 dataset the reader is referred to appendix C & D.

Short codes:

- *Struct*: the structure embedding
- *Cfeat*: the combined features embedding
- *feature_1* (boolean): gender, with male and female
- *feature_2* (boolean): public, account is public or not
- *feature_3* (numerical): age of the user
- *feature_4* (numerical): account completion percentage
- *feature_5* (numerical): days since registration
- *feature_6* (numerical): days since login
- *CStruct*: the combined features and structure embedding

bool	min	max	n=0	n=1	σ	short code
gender	0	1	565	430	0.5	feature_1
public	0	1	350	645	0.48	feature_2
numeric	min	max	mean	median	σ	short code
age	0	52	20	16.3	9.97	feature_3
completion	12	97	59	49.4	23.55	feature_4
days reg	102	4384	1733	1763	597.34	feature_5
days log	0	844	5	31	65.96	feature_6

Table 8: Summary statistics features Pokec-1000

For this experiment, we need to adjoin that in literature most of the algorithms are run with variable x in 2^x dimensions. The inherent property of multiscale walks of Walklets caused an increase in dimension of $2^x * 5$. Since, we opted for a fair comparison between all algorithms, the dimensions of DeepWalk follow the same $2^x * 5$ dimensionality pattern. The Pokec-1000 dataset for DeepWalk has thus been run with $2^6 * 5 = 320$ dimensions

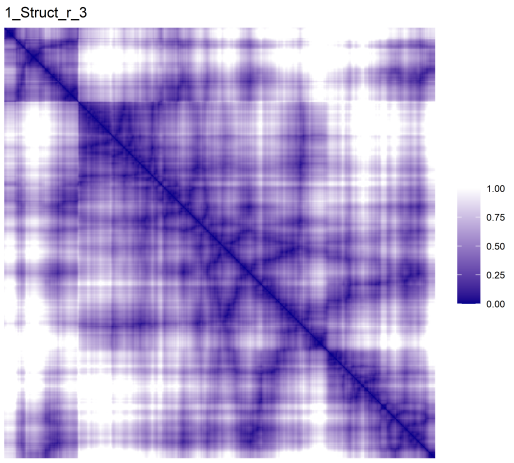


Figure 37: Structure Pokec-1000 DeepWalk

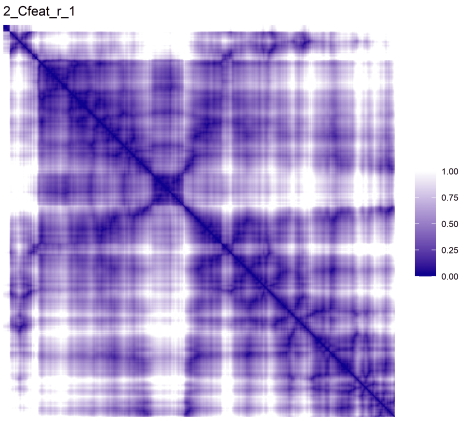
and still satisfies the $|V| << d$ requirement set before. The other influential hyperparameters of DeepWalk are usage of a walk number of 10, walk length of 80, window size of 5 and epochs of 10.

The structure visualization of DeepWalk is shown in figure 37. This figure has shown that structure in the DeepWalk embedding is either highly complicated or is not preserved well by DeepWalk. The figure shows multi-dimensional block patterns, most notably in the upper right corner. These patterns are occasionally part of a larger stair pattern in which there is only increasing or decreasing hue saturation over an axis. And finally, the figure shows curved line patterns (which resembles a fishbone) with highly similar nodes along this pattern. Moreover, the dividing line of the upper right block pattern and the rest of the figure, shows the star/line pattern, indicating a connection node. Thereafter, DeepWalk has shown white patches further away from the diagonal, indicating very dissimilar nodes.

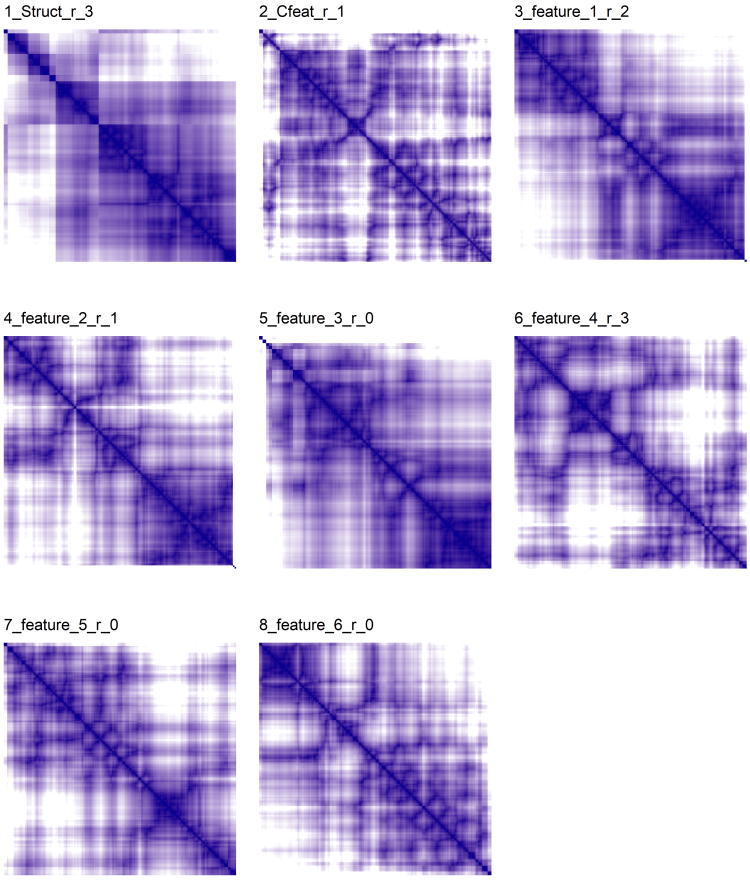
7.4.1 featPMI

In this section featPMI will show the visualizations of the final combined embedding and all individual embeddings consecutively. featPMI has used similar hyperparameters for the random walk part of the algorithm, with a walk number of 10, walk length of 80, window size of 5 and epochs of 10. Furthermore, featPMI was run with 200 iterations and a regularization of 0.1 for the matrix factorization phase.

Figure 38a shows the combined feature and structure embedding of featPMI. We found that this visualization is similar to DeepWalk, with similar uncertainty in the observed patterns. This image has shown fluctuating



(a) Distance matrix of entire graph in one single image



(b) Visualizations of all features and structure

Figure 38: featPMI Pokec-1000

hue coloration along both axis, indicating this uncertainty. Nevertheless, we noted that the second large block pattern is more distinctly defined in the embedding of featPMI. It contains a large block which seems to be fused with a line/star pattern near the middle of the image. And finally, there appears to be a highly distinctive group in the upper left corner, with high similarities within the group and very low similarities outside this group.

Figure 39b lists all visualized embeddings in a 3x3 grid. First, the structure visualization retained more distinctive block patterns along the entire image compared to DeepWalk. Secondly, this structure has a more gradient like hue change along the axes, which indicates more certainty of the reordering. Further, we have noticed that there are overlapping block patterns and multiple small and large line/star patterns in the matrix. On a different note, one should expect that embedding the boolean variables (i.e. feature 1 and 2) is rather easy in even a single dimension. Nevertheless, the algorithm appears unable to clearly visualize these variables, and only by extensive examination shows two larger block patterns. For the numerical variables, the same observations apply. One unexplained artifact was the one or two divergent block patterns appearing in either visualization corner.

node	$f1$	$f2$	$f3$	$f4$	$f5$	$f6$	*connections*
164	0	0	25	57	1361	159	[**4**,165]
157	0	0	22	21	2005	176	[**4**]
159	0	0	22	60	1422	1	[**4**,165,162,161]
166	1	0	25	12	2752	16	[**4**]
168	0	0	21	12	2593	276	[**4**,165,162,161,167]
σ	0.45 ↓	0 ↓	1.87 ↓	24.13 ↑	643.44 ↑	115.99 ↑	
Node features for Observations 1 to 5							

node	$f1$	$f2$	$f3$	$f4$	$f5$	$f6$	*connections*
917	0	0	18	24	1838	4	[**1**,132,217,740,755]
914	0	0	21	16	1650	290	[**1**,22,41,53,78,85,112, 131,691,692,711,713]
915	0	0	18	71	1880	1	[**1**,805,814]
911	1	1	27	12	1659	1	[**1**,35,146]
913	1	1	21	47	1702	2	[**1**,16,23,25,34,3,48,53,63,75 76,103,118,124,129,140,148 372,390,391,393,402,464] 584,612,617,639,720,721]
σ	0.55 ↑	0.55 ↑	3.67 ↓	24.73 ↑	106.23 ↓	128.8 ↑	
Node features for observations 991 to 995							

Table 9: featPMI: feature values and nodal links when maximizing TSP

To analyse the effectiveness of the feature grouping and proximity preservation of graph structure, we took an excerpt of the first 5 observations in

the upper left corner, and the last 5 observations in the downward right corner of the combined feature matrix. The nodes labeled 164 to 168 in table 9 show the input features for the first 5 observations. In this table, a green arrow indicates that the standard deviation among the samples is smaller than average, as shown in table 8, and a red arrow indicates the opposite. We found that the boolean features although imperfect, are similar for all nodes. Further we have noticed that feature 3 shows reasonable clustering compared to the data distribution. Feature 4,5 and 6 are less preserved in this visualization with none of the standard deviations scoring lower than random. And finally, the connections have shown that most nodes are not connected to each-other directly, instead they have similar neighbours. The nodes on the other end of the visualization have shown less consistency in clustering, with divergent binary features, and divergent values for features 1,2, 4 and 6. Another observation that was made is that these nodes have a higher degree, with less similar neighbours.

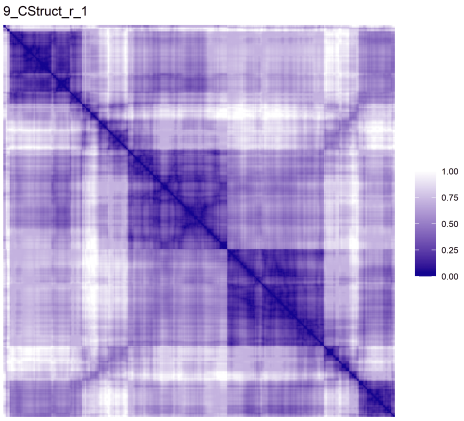
7.4.2 featWalk

In this section featWalk has been applied on the same graph for comparison with featPMI and DeepWalk. It has therefore been run with exactly the same hyperparameters.

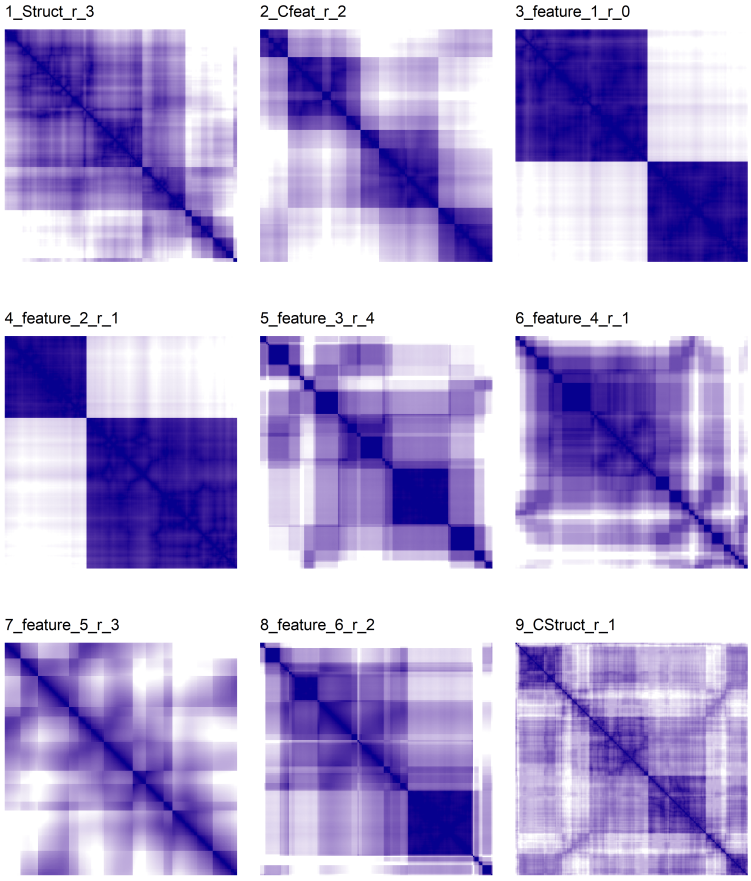
In figure 39a we observed that block patterns in this visualization are defined with more clear separation of color between distinct groups (i.e. the gradient differs more between blocks). In this figure, we noticed the arisal of shaded blocks, which indicated multi-scale similarities between groups. Nevertheless, while more clear separation exists, areas with highly similar (i.e. dark colors) nodes are more uncommon than in the visualization of featPMI.

In figure 39b we introduced embedding visualizations for structure, combined features, features, and finally the combined features and structure embedding. The observant reader could have noticed that that the structure in featWalk does not differ from featPMI, and this is due to the fact that both algorithms utilize identical embeddings for structure visualization since this step in their architecture does not differ. The boolean features (i.e. feature 1 and 2) displayed more defined borders and indicated the overall distribution of the data accurately. Furthermore, we recognized clearly defined borders in feature 3 and 6 in which almost all of the borders are characterized by an occurrence of the line/star pattern. Additionally, there are clear off-diagonal shaded block patterns in feature 3, which indicates that some block patterns are more similar to block patterns further along the diagonal. Feature 4 and 5 were found to contain more smaller block patterns, with less clear color differentiation between them. As a final note, feature 5 shows a more smooth transition between each element in the visualization.

When glanced over table 10, for all observations, featWalk is accurate



(a) Distance matrix of entire graph in one single image



(b) Visualizations of all features and structure

Figure 39: featWalk Pokec-1000

node	$f1$	$f2$	$f3$	$f4$	$f5$	$f6$	*connections*
167	0	1	20	28	2562	1	[**4**,165,168]
160	0	1	23	33	1832	1	[**4**]
919	0	1	22	41	507	10	[**4**]
158	0	1	23	45	2531	1	[**4**]
162	0	1	22	66	1791	139	[**4**,165,161,168,159]
σ	0 ↓	0 ↓	1.22 ↓	14.67 ↓	833.37 ↑	60.83 ↓	
Node features for Observations 1 to 5							

node	$f1$	$f2$	$f3$	$f4$	$f5$	$f6$	*connections*
598	1	0	23	45	1602	1	[**17**,452,496,635]
558	1	0	21	34	1705	1	[**17**,602,511]
577	1	0	21	12	1393	4	[**17**,591,602]
505	1	0	21	17	1597	2	[**17**,583,602]
526	1	0	21	12	2183	3	[**17**,135,476,482,529,532]
σ	0 ↓	0 ↓	0.89 ↓	14.82 ↓	294.83 ↓	1.3 ↓	
Node features for observations 991 to 995							

Table 10: featWalk: feature values and nodal links when maximizing TSP

in embedding boolean and numeric attributes into the visualization(s). It is observed that all respective boolean attributes are exactly the same, and features 3,4 and 6 have even smaller variances when compared to featPMI. Furthermore, we noted that the spread in feature 5 was similar to the spread in featPMI, and was least preserved by the embedding in this example. In the table it is again shown that the connections all have a common origin node, and that the nodes are not directly connected.

7.4.3 Additional experiments

The visual quality metrics in figure 40 suggested that DeepWalk shows the best visualization of the embedded multivariate graph. DeepWalk scored best on 11 out of 16 visual quality metrics, while featPMI scored best on 3, and featWalk on the remaining 2. For details on each separate embedding in featPMI and featWalk see appendix D.

In order to find divergences between the pokec-1000 dataset and the other pokec splits, we compared their resulting visualizations. For the pokec-100 dataset, the structure was well-defined in the algorithm DeepWalk, with two distinct blocks visible. This observation was not shared for the other algorithms. For the pokec-5000 dataset and featWalk, we observed even more clear separations between groups with line/star patterns. Nevertheless, the complete embedding was less defined for featWalk. FeatPMI on the other hand, showed consistent results among all pokec dataset sizes, with the exception of the feature visualizations which were more well defined in

Average Visual Quality Metric Scores of 5 Runs

Dataset: *Pokec-rel 1000*

VQM	*DeepWalk*	*featPMI*	*featWalk*
2-Sum	**9.78×10^{10}**	1.01×10^{11}	9.80×10^{10}
Anti-Rob Deviations	2.03×10^{7}	2.33×10^{7}	**1.46×10^{7}**
Anti-Rob Events	**9.94×10^{7}**	1.10×10^{8}	1.11×10^{8}
Banded Anti-Rob	6.21×10^{6}	**6.15×10^{6}**	6.73×10^{6}
Cor_R	6.19×10^{-2}	4.68×10^{1}	**5.95×10^{-2}**
Gradient measure	**1.20×10^{8}**	9.52×10^{7}	1.05×10^{8}
Weighted G measure	**4.62×10^{7}**	3.75×10^{7}	3.66×10^{7}
Inertia	**1.15×10^{11}**	1.07×10^{11}	1.11×10^{11}
Lazy path length	**1.20×10^{4}**	1.42×10^{4}	1.84×10^{4}
Least Squares	**1.63×10^{11}**	1.63×10^{11}	1.63×10^{11}
Linear Seriation	3.50×10^{8}	**3.45×10^{8}**	3.64×10^{8}
Measure of effect	8.69×10^{5}	**8.88×10^{5}**	8.34×10^{5}
Stress Moore	**1.06×10^{3}**	1.49×10^{3}	1.69×10^{3}
Stress Neumann	**3.58×10^{2}**	5.01×10^{2}	5.74×10^{2}
Ham Path length	**2.39×10^{1}**	2.87×10^{1}	3.81×10^{1}
RGAR	**3.03×10^{-1}**	3.36×10^{-1}	3.39×10^{-1}

*__Cyan__ (lower is better), indicates best low score. __Yellow__ (higher is better), indicates best high score.

Figure 40: Visual quality metrics of final visualization Pokec-1000

the other splits.

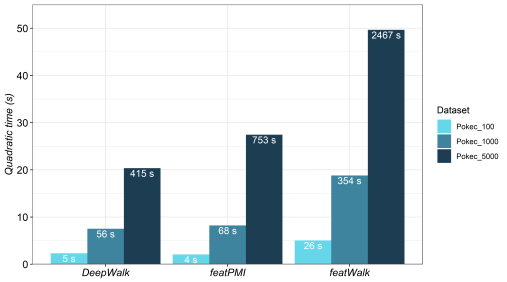


Figure 41: Quadratic time in (s) for embedding*

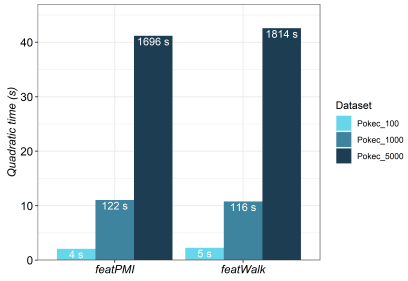


Figure 42: Quadratic time in (s) for seriation

Furthermore, the feasibility of the new algorithms was also tested by analyzing the temporal differences between Pokec-sizes and algorithms. In figure 41 the algorithms were plotted along the x-axis while time in quadratic seconds (e.g. 10 quadratic seconds is 100 seconds) is plotted along the y-axis.

For convenience, the time in seconds is also plotted in the bar itself. Observe for instance that the time complexity of featWalk for the Pokec-1000 and Pokec-5000 increases with factor 7 from 354 seconds to 2467 seconds, while size increases with factor 5. This indicates a more than linear increase in time with size (e.g. $7>5$), but less than quadratic (e.g. $7< 5^2$, for more details on time complexities see section 6). Note that this figure gives a slightly distorted image of time complexities, DeepWalk only embeds one fifth of the total size (i.e. only structure) and is therefore not a fair comparison. FeatPMI and featWalk on the other hand, embed both structure and features. We observed from comparing figure 41 and figure 42, that seriation in this example is less scalable than embedding.

8 Discussion

In this section we explore whether the visualizations introduced in section 7 can enhance our understanding of the underlying data. We interpret the results and investigate if graph embedding visualizations are able to distinguish between noise and signal for effective re-orderings. First we start with analyzing the results in section 8.1, thereafter we will discuss the limitations of our study in section 8.2 and finally we will conclude with directions for future work in section 8.3.

8.1 Findings

The results for basic graphs indicate that graph embedding algorithms are able to show canonical data patterns in small graphs. We found that the block, star/line, bands and noise anti pattern are clearly visible in the examples. Unfortunately, diagonal block patterns are absent in the examples, and we expect this is due to an artefact of the seriation algorithm TSP. This algorithm tries to reduce the Hamiltonian path length, and therefore, always favors placing observations close to the diagonal. Furthermore, in these examples the reconstruction scores Khop similarity and Degree Adjusted ARI appear to have potential to be a good indicator of visual quality. We proposed that if these scores reflect the data in a accurate manner, the visualization should be good as well. Nevertheless, Khop Similarity is expensive to calculate, and Degree Adjusted ARI is less accurate for larger graphs, so a lot of work remains to be done. As an alternative to existing metrics Beusekom, Meulemans and Speckmann [101] suggest the use of Moran's I which captures all patterns in the data for visual quality, instead of only block patterns. Unfortunately, as of now this spatial auto-correlation metric is only applicable for undirected graph. Subsequently, in the football dataset we observe that the Quality Metrics top 2 algorithms are attentionWalk and struc2vec, even though their visualizations are arguably worse than all other algorithms. We expect that this is due to the fact that these metrics favor large block patterns, which shows similarities with existing research [6, 101, 9]. Note that the graph embedding algorithms all generate different embeddings and therefore, we argue that these metrics are incomparable between visualizations. Consider for instance, in a worst case scenario, that an embedding completely fails and embeds each node exactly on the same point in the latent space. In this example the visual quality metrics would favor this embedding over all others, while it does not deliver any information to the user. We therefore argue that these metrics cannot be used for comparison on visualizations which have different data a priori. And finally, these basic graphs showed us that Walklets and HOPE generally performed best in creating accurate visualizations. The classic matrix factorization methods Locally Linear Embedding and Laplacian Eigenmaps

appeared competitive but fail to discover more complex patterns in the data (e.g. line/star and off diagonal). In addition to that, Node2vec and DeepWalk show good results for all problems (except bipartite), albeit with less clarity in their visualization. It appears that multiscale Walklets creates more information and certainty in the visualization. And finally, we are introduced to the shaded block pattern which indicates multiscale relationships within communities.

The figures in the weighted section appear less clear than the unweighted counterparts. A possibility could be that the dataset does not define groups in the data very well, or the structure and weighting scheme work against eachother. It appears that the US airport example does show a clear group of outliers suggesting that regularization or polarization of the data will create a more interpretable figure. After the TSP paths of both weighted and unweighted graphs were plotted, we found differences in their ordering and definition of proximity. Unweighted graphs favor proximity to nodes with whom they have a connection, while weighted graphs appear to take into account the weight of this connection for traversal. Unfortunately, again the Visual Quality Metrics do not highlight a clear top algorithm.

The first thing we noted when plotting large graphs in distance matrices is that their interpretation is much harder than in binary matrices such as those in Behrisch et al. [6]. This is probably due to more variation and uncertainty in the data, but it does allow for a richer representation of the underlying data. By allowing each cell in the matrix to be a numeric value, we were able to create multiscale visualizations for multivariate graphs. DeepWalk showed us the new fishbone-like pattern which we think represents paths along the edges of unconnected inner communities. We expect that the nodes inside the fish-bone are related feature wise, but do not share an edge and in this example could be an artefact of the breadth first search Pokec sampling. Furthermore, we also have some criticism on the Pokec dataset since the attributes in this dataset are probably not the most useful in predicting edges, and thus combining features and structure could allow for a less interpretable figure. Nevertheless, when inspecting featPMI and especially featWalk, we observed the viability of splitting embeddings to explore the underlying data. The visual quality divergences between these two algorithms is probably explained by the fact that featPMI requires a more careful selection of hyperparameters. FeatWalk on the other hand shows much clearer patterns in the data, in which, for instance, we can immediately identify the two boolean variables. In this visualization we noted what is probably another artefact of BFS Pokec, the heavy occurrence of the line/star pattern. Furthermore, featWalk combined structure and feature figure appears less clear cut than the rest, and this is probably due to the fact that we used medium fusion to construct this image. A good solution could be to integrate the feature embedding and structure embedding into a single early fusion operation, which as shown by the combined

feature visualizations (i.e. features without structure), yields a more clear image. Nonetheless, research shows that higher level fusion has its own advantages and disadvantages [102, 103]. Next, we argue that interpretation of these matrices is still hard without domain knowledge, but basic information about community structure or important nodes can be often be easily extracted in these high level images. For the experiments on feature preservations we found that the features appear to be much better preserved in featWalk and less so in featPMI. A peculiarity we found was that featPMI scored, on average, lower than random for feature preservation. This could be due to non optimal parameters, difficulties of preserving all features in a single image, or the sample we took was a particularly bad one. Unfortunately, contrary to our results, DeepWalk shows best on the quality metrics, but did not result in a more comprehensible figure.

In general, we found that larger graphs appear less interpretable, but that this could also be due to using a dataset in which it is hard to discover patterns. Nevertheless, an image that is less clear can indicate something useful, a highly divergent group of nodes is in the data (e.g. outliers that reduce hue interpretability), or it indicates that all nodes are relatively similar (i.e. less well defined). In the experimental phase we noted that the choice of a DRT appeared to be highly influential for the quality of the resulting images, in which PCA appears more stable (in line with Gisbrecht et al. [54]) and preferred for the simple problems in section 7.2. In contrast and in line with McInnes et al. [92], UMAP preserves more of the structure of the data in the other experiments. Note that the non-linearity in UMAP could explain the less clear cut borders in easier to solve basic graph visualizations. Unfortunately, the biggest downside in all experiments is the focus of the TSP algorithm to show block patterns while many other interpretable patterns exist. However, we believe this could easily be solved by applying different seriation algorithms.

To summarize, the research in this paper strengthens the claim of exploring large (multivariate) relational datasets in 2D matrices. This work highlights the importance of understanding relational data, and introduces an highly adaptable framework to create high level images of graphs. First, we found that graph embedding algorithms are able to learn embeddings for high quality matrix visualizations with canonical data patterns. Secondly, matrix factorization methods had more trouble creating a feasible visualization with more complex patterns (e.g. star, off diagonal) in the data. In this experiment, the algortihms HOPE and Walklets had most consistent results. Thirdly, the visual quality metrics in this paper appeared insufficient for optimal selection of a good visualization and, among other things, leads to the block pattern being preferred most among all visualizations. Fourthly, we noticed an increased difficulty in the readability of graphs when complexity was increased, which we partially solved by splitting embeddings and creating separate visualization. Furthermore, visual quality is higher in

multivariate featWalk, but is reduced in the larger complete visualizations possibly due to intermediate fusion in featWalk. And finally, node features appear to be highly preserved in featWalk, indicating the applicability of this algorithm for visualizing complex relational data.

8.2 Delineation

In this research we did not consider heterogeneous graphs or graphs with multiple node types, due to these demanding additional research, adaptation of existing embedding algorithms and increase of scope. For similar reasons, we opted to exclude Gaussian and dynamic graph embedding algorithms, and shift our attention to vector point graph embeddings. Note that visualizations with the excluded algorithms would probably show fundamentally different visualizations (e.g. for dynamic, a temporal aspect). In vector point graph embedding there exists many different algorithms, and thus it was necessary to create a subset for comparison. When we made this subset we took into account the importance of the algorithm in history, the number of citations and novelty of the algorithm. And finally, adaptive embeddings could be considered in the future, these could be used to retrieve embeddings with adaptive focal point attributes. The embeddings would empower the user to increase and decrease the importance of certain attributes.

Furthermore, for larger graphs we did not visualize node link diagrams for comparison due to temporal constraints, and possible high cluttering. In addition, visualization in 2D matrices could be enhanced by adding more information in a single cell as in the tool Zoomable Adjacency Matrix Explorer (i.e. ZAME) [1]. ZAME visualizes statistical glyphs on nodal relationships and increases the understanding of the underlying data. Nevertheless, the goal of our research is to give a high level overview, and thus this idea was discarded. In addition, note that including ZAME in our architecture would reduce the scalability property of our framework. Another way to add more information in a single image is to use 3D visualizations. Gaining embeddings in 3 dimensions can be obtained by a single value change in our framework, however visualizing a 3D matrix and retrieving a good matrix order is not trivial.

The existing framework can be used with a variety of different embeddings, sizes, seriation algorithms and dimensionality reduction techniques. Nevertheless, in this study we did not consider usage of very large graphs due to temporal and memory constraints. Likewise, although showing promising preliminary results, the naive combination of embeddings of different algorithms before seriation were not considered. Once more, due to temporal constraints and good preliminary results, we opted for the usage of the DRT UMAP for graphs larger than 40 nodes and DRT PCA for the rest. There exists plenty more dimensionality reduction techniques that have not been considered. In addition, it would even be possible to skip the reduction of di-

mension phase at all and calculate the distance in high dimensions. Research however, points out that this distance calculation is not optimal in higher dimensions [15]. Furthermore, we did not consider other seriation algorithms that could generate visualizations with different patterns in the data. Additionally, the framework could benefit from transfer learning, especially in the graph embedding phase. Thereafter, adaptations for the visualization phase exists with methods that reorder in combination with regressive blurring in an iterative manner. This reduces noise and highlights the global structure [55]. Hybrid sort on the other hand, is a pattern-based method and utilizes a matrix classifier to select the optimal method from an existing set of structure based methods, which is able to uncover most [104] if not all canonical data patterns [56]. Unfortunately, temporal constraints did not allow us to explore these options.

Finally, since attributed embedding algorithms mainly use text based or categorical attributes, we opted to explore the novel numerical attribute embeddings and visualizations. Again since, edge attributes are fundamentally different we did not consider embedding these.

8.3 Future Work

The work in this paper suggests many new research directions. These directions range from adaptation to a more unified platform for the framework (with usability research) to optimization and discovery of new patterns. However, we argue the most important new research direction is the development of a new metric for assessing the quality of the visualization. In this research, the largest issue we faced was the inability to quickly determine what visualization was preferred. Therefore, this metric should take into consideration all canonical data patterns, and, eventually could even be integrated into the framework as a determining score to select the best visualization. Furthermore, we suggest experimentation with larger graphs, different embedding algorithms and different seriation algorithms. Note that in the matrices in this paper the top triangle and bottom triangle are identical, and either the top or bottom triangle could be used to explain node properties or any other relationship between nodes. And finally, future research could implement embedding edge properties in the algorithms, analyze new graphs for discovery of new patterns, and suggests comparative studies against node link diagrams.

9 Conclusion

This thesis tested the feasibility of using graph neural networks for projecting a complex graph in a matrix visualization. The methodology of Harel & Koren [14] effectively inspired the usage of graph embeddings for graph visualization. Nevertheless, to the best of our knowledge, this methodology has not been applied for matrix visualization while they offer benefits in size, and high level understanding when compared to node-link diagrams. This thesis introduced an adaptive framework for graph embeddings and visualization. The graph embedding stage uses existing graph embedding algorithms, and is fed as an input to the visualization stage. By analyzing the occurrence of canonical data patterns in generated graphs, this thesis has shown the viability of pattern-based understanding of graph matrix visualizations. By allowing the cells of 2D matrix numerical values instead of binary only, several new multi-scale patterns arose in the experiments. Included are the shaded block pattern, the fishbone pattern and the stair pattern. Numeric multivariate graphs can be embedded with two new algorithms named featPMI and featWalk. These algorithms, in combination with separate visualizations of structure, features and a selective combination of the two, showed great promise in accurately visualizing a graph in a 2D matrix. To account for comparison of visualizations, we introduced a matrix visualization tool for users to edit, compare and combine different visualizations. Thereafter, experiments showed that the underlying data is the biggest determinant of retrieving good visualizations, with optimization as a close second. Additionally, we introduced two new metrics to quantify the goodness of a matrix visualization. These metrics are inoperable for weighted and multivariate datasets and they perform worse for larger datasets. Therefore, we stress the need for new metrics to select good visualizations. Future research could additionally focus on including edge properties in the embedding, and applying the framework to dynamic graphs. To visualize a complex graph in a matrix, we propose to split the graph into structure, feature and combined embeddings. On these splits an embedding algorithm computes embeddings of n dimensions to preserve proximity. Afterwards, these dimensions are reduced to 2 with a dimensionality reduction technique. Subsequently, we apply TSP to generate a good ordering, and visualize the result for analytical interpretation. As a result we found that Graph Neural Networks are a viable option for visualizing a multidimensional graph. The latent space preserves proximity and features in the data that we can visualize, and recognize in a TSP reordered 2D matrix.

10 References

[1] N. Elmqvist, T. Do, H. Goodell, N. Henry, and J. D. Fekete, "ZAME: Interactive large-scale graph visualization," in *2008 IEEE Pacific Visualization Symposium*. IEEE, Mar. 2008, doi: 10.1109/pacificvis.2008.4475479.

[2] T. Horak, P. Berger, H. Schumann, R. Dachselt, and C. Tominski, "Responsive matrix cells: A focus context approach for exploring and editing multivariate graphs," *IEEE Transactions on Visualization and Computer Graphics*, vol. 27, no. 2, pp. 1644–1654, Feb. 2021, doi: 10.1109/tvcg.2020.3030371.

[3] M. Okoe, R. Jianu, and S. Kobourov, "Node-link or adjacency matrices: Old question, new insights," *IEEE Transactions on Visualization and Computer Graphics*, vol. 25, no. 10, pp. 2940–2952, Oct. 2019, doi: 10.1109/tvcg.2018.2865940.

[4] M. Ghoniem, J. Fekete, and P. Castagliola, "On the readability of graphs using node-link and matrix-based representations: a controlled experiment and statistical analysis," *Information Visualization*, vol. 4, no. 2, pp. 114–135, May. 2005, doi: 10.1057/palgrave.ivs.9500092.

[5] R. Keller, C. M. Eckert, and P. J. Clarkson, "Matrices or node-link diagrams: which visual representation is better for visualising connectivity models?" *Information Visualization*, vol. 5, no. 1, pp. 62–76, April. 2006, doi: 10.1057/palgrave.ivs.9500116.

[6] M. Behrisch, B. Bach, N. H. Riche, T. Schreck, and J. D. Fekete, "Matrix reordering methods for table and network visualization," *Computer Graphics Forum*, vol. 35, no. 3, pp. 693–716, Jun. 2016, doi: 10.1111/cgf.12935.

[7] M. Behrisch, T. Schreck, and H. Pfister, "GUIRO: User-guided matrix reordering," *IEEE Transactions on Visualization and Computer Graphics*, vol. 26, no. 1, pp. 184–194, Aug. 2019, doi: 10.1109/tvcg.2019.2934300.

[8] P. Goyal and E. Ferrara, "Graph embedding techniques, applications, and performance: A survey," *Knowledge-Based Systems*, vol. 151, pp. 78–94, Jul. 2018, doi: 10.1016/j.knosys.2018.03.022.

[9] J. Petit, "Experiments on the minimum linear arrangement problem," *Journal of Experimental Algorithmics (JEA)*, vol. 8, dec 2003, doi: 10.1145/996546.996554.

[10] Y. Koren and D. Harel, "A multi-scale algorithm for the linear arrangement problem," *Revised Papers from the 28th International Workshop on Graph-Theoretic Concepts in Computer Science*, pp. 296–309, Jan. 2002, doi: 10.1007/3-540-36379-3_26.

[11] L. Wilkinson, *The Grammar of Graphics (Statistics and Computing)*. New York, Inc., Secaucus, NJ, USA: Springer-Verlag, 2005.

[12] M. Xu, "Understanding graph embedding methods and their applications," *SIAM Review*, vol. 63, no. 4, pp. 825–853, 2021, doi: 10.1137/20M1386062.

[13] T. Munzner, "A nested model for visualization design and validation," *IEEE Transactions on Visualization and Computer Graphics*, vol. 15, no. 6, pp. 921–928, Nov. 2009, doi: 10.1109/tvcg.2009.111.

[14] D. Harel and Y. Koren, "Graph drawing by high-dimensional embedding." *Journal of Graph Algorithms and Applications*, vol. 8, no. 2, pp. 195–214, 2004. [Online]. Available: http://eudml.org/doc/52327

[15] C. C. Aggarwal, A. Hinneburg, and D. A. Keim, "On the surprising behavior of distance metrics in high dimensional space," in *Database Theory — ICDT 2001*. Springer Berlin Heidelberg, Oct. 2001, pp. 420–434, doi: 10.1007/3-540-44503-x_27.

[16] N. Bikakis, J. Liagouris, M. Krommyda, G. Papastefanatos, and T. Sellis, "Graphvizdb: A scalable platform for interactive large graph visualization," in *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*. IEEE Computer Society, May. 2016, pp. 1342–1345, doi: 10.1109/ICDE.2016.7498340.

[17] A. Komarek, J. Pavlik, and V. Sobeslav, "Network visualization survey," in *Computational Collective Intelligence*. Springer International Publishing, 2015, pp. 275–284, doi: 10.1007/978-3-319-24306-1_27.

[18] M. S. Rahman, *Basic Graph Theory*, 1st ed. Springer International Publishing, 2017, doi: 10.1007/978-3-319-49475-3.

[19] C. Bothorel, J. D. Cruz, M. Magani, and B. Micenková, "Clustering attributed graphs: Models, measures and methods," *Network Science*, vol. 3, no. 3, pp. 408–444, Mar. 2015, doi: 10.1017/nws.2015.9.

[20] M. Kivelä, A. Arenas, M. Barthelemy, J. P. Gleeson, Y. Moreno, and M. A. Porter, "Multilayer networks," *Journal of complex networks*, vol. 2, no. 3, pp. 203–271, 2014, doi: 10.1093/comnet/cnu016.

[21] H. Cai, V. W. Zheng, and K. C.-C. Chang, "A comprehensive survey of graph embedding: Problems, techniques, and applications," *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, no. 9, pp. 1616–1637, Sep. 2018, doi: 10.1109/tkde.2018.2807452.

[22] R. A. Rossi, N. K. Ahmed, E. Koh, S. Kim, A. Rao, and Y. A. Yadkori, "HONE: Higher-order network embeddings," Jan 2018, doi: 10.48550/arXiv.1801.09303.

[23] M. Ou, P. Cui, J. Pei, Z. Zhang, and W. Zhu, "Asymmetric transitivity preserving graph embedding," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, Aug. 2016, pp. 1105—-1114, doi: 10.1145/2939672.2939751.

[24] J. Leskovec, J. Kleinberg, and C. Faloutsos, "Graph evolution: Densification and shrinking diameters," *ACM transactions on Knowledge Discovery from Data (TKDD)*, vol. 1, no. 1, pp. 2–43, Mar. 2007, doi: 10.1145/1217299.1217301.

[25] N. Satish, N. Sundaram, M. M. A. Patwary, J. Seo, J. Park, M. A. Hassaan, S. Sengupta, Z. Yin, and P. Dubey, "Navigating the maze of graph analytics frameworks using massive graph datasets," in *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. Association for Computing Machinery, Jun. 2014, p. 979–990, doi: 10.1145/2588555.2610518.

[26] S. T. Roweis, "Nonlinear dimensionality reduction by locally linear embedding," *Science*, vol. 290, no. 5500, pp. 2323–2326, Dec. 2000, doi: 10.1126/science.290.5500.2323.

[27] M. Belkin and P. Niyogi, "Laplacian eigenmaps for dimensionality reduction and data representation," *Neural Computation*, vol. 15, no. 6, pp. 1373–1396, Jun. 2003, doi: 10.1162/089976603321780317.

[28] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. Association for Computing Machinery, Aug. 2014, doi: 10.1145/2623330.2623732.

[29] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 8, pp. 1798–1828, Aug. 2013, doi: 10.1109/tpami.2013.50.

[30] A. Bojchevski and S. Günnemann, "Deep gaussian embedding of graphs: Unsupervised inductive learning via ranking," in *International Conference on Learning Representations*, Apr. 2018, doi: 10.48550/arxiv.1707.03815.

[31] P. Goyal, S. R. Chhetri, N. Mehrabi, E. Ferrara, and A. Canedo, "Dynamicgem: A library for dynamic graph embedding methods," Nov. 2018b, doi: 10.48550/arxiv.1811.10734.

[32] L. F. Ribeiro, P. H. Saverese, and D. R. Figueiredo, "struc2vec: Learning node representations from structural identity," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Association for Computing Machinery, Aug. 2017, pp. 385—-394, doi: 10.1145/3097983.3098061.

[33] S. Abu-El-Haija, B. Perozzi, R. Al-Rfou, and A. Alemi, "Watch your step: Learning node embeddings via graph attention," *NeurIPS*, pp. 9180–9190, Oct. 2018, doi: 10.48550/arxiv.1710.09599.

[34] B. Rozemberczki, C. Allen, and R. Sarkar, "Multi-scale attributed node embedding," *Journal of Complex Networks*, vol. 9, no. 2, Apr. 2021, doi: 10.1093/comnet/cnab014.

[35] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," in *Proceedings of Workshop at ICLR*, Jan. 2013, pp. 1–12, doi: 10.48550/arxiv.1301.3781.

[36] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. Association for Computing Machinery, Aug. 2016, pp. 855–864, doi: 10.1145/2939672.2939754.

[37] B. Perozzi, V. Kulkarni, H. Chen, and S. Skiena, "Don't walk, skip! online learning of multi-scale network embeddings," in *Proceedings of the 2017 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2017*. Association for Computing Machinery, Jul. 2017, pp. 258—265, doi: 10.1145/3110025.311008.

[38] D. Wang, P. Cui, and W. Zhu, "Structural deep network embedding," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Association for Computing Machinery, Aug. 2016, pp. 1225—-1234, doi: 10.1145/2939672.2939753.

[39] C. Yang, Z. Liu, D. Zhao, M. Sun, and E. Chang, "Network representation learning with rich text information," in *Proceedings of the 24th*

International Conference on Artificial Intelligence. AAAI Press, Jul. 2015, p. 2111–2117, doi: 10.5555/2832415.2832542.

[40] L. Liao, X. He, H. Zhang, and T.-S. Chua, "Attributed social network embedding," *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, no. 12, pp. 2257–2270, Dec. 2018, doi: 10.1109/tkde.2018.2819980.

[41] P. Domingos, "A few useful things to know about machine learning," *Communications of the ACM*, vol. 55, no. 10, pp. 78–87, Oct. 2012, doi: 10.1145/2347736.2347755.

[42] E. M. Mirkes, J. Allohibi, and A. Gorban, "Fractional norms and quasinorms do not help to overcome the curse of dimensionality," *Entropy*, vol. 22, no. 10, p. 1105, Sep. 2020, doi: 10.3390/e22101105.

[43] P. Naik, M. Wedel, L. Bacon, A. Bodapati, E. Bradlow, W. Kamakura, J. Kreulen, P. Lenk, D. M. Madigan, and A. Montgomery, "Challenges and opportunities in high-dimensional choice data analyses," *Marketing Letters*, vol. 19, no. 3, pp. 201–213, May 2008, doi: 10.1007/s11002-008-9036-3.

[44] A. Juvonen, T. Sipola, and T. Hämäläinen, "Online anomaly detection using dimensionality reduction techniques for HTTP log analysis," *Computer Networks*, vol. 91, pp. 46–56, Nov. 2015, doi: 10.1016/j.comnet.2015.07.019.

[45] S. Ayesha, M. K. Hanif, and R. Talib, "Overview and comparative study of dimensionality reduction techniques for high dimensional data," *Information Fusion*, vol. 59, pp. 44–58, Jul. 2020, doi: 10.1016/j.inffus.2020.01.005.

[46] H. Hotelling, "Analysis of a complex of statistical variables into principal components." *Journal of educational psychology*, vol. 24, no. 6, pp. 417–441, 1933, doi: 10.1037/h0071325.

[47] K. Pearson, "LIII. on lines and planes of closest fit to systems of points in space," *The London, Edinburgh, and Dublin philosophical magazine and journal of science*, vol. 2, no. 11, pp. 559–572, Nov. 1901, doi: 10.1080/14786440109462720.

[48] B. Schölkopf, A. Smola, and K. R. Müller, "Kernel principal component analysis," in *International conference on artificial neural networks*. Springer Berlin Heidelberg, 1997, pp. 583–588, doi: 10.1007/bfb0020217.

[49] V. Raunak, V. Gupta, and F. Metze, "Effective dimensionality reduction for word embeddings," in *Proceedings of the 4th Workshop on Representation Learning for NLP*. Association for Computational Linguistics, Aug. 2019, pp. 235–243, doi: 10.18653/v1/w19-4328.

[50] G. Hinton and S. Roweis, "Stochastic neighbor embedding," in *Proceedings of the 15th International Conference on Neural Information Processing Systems*, ser. NIPS'02. MIT Press, Jan. 2002, pp. 857—-864, doi: 10.5555/2968618.2968725.

[51] L. Maaten and G. E. Hinton, "Visualizing data using t-sne," *Journal of Machine Learning Research*, vol. 9, no. 86, pp. 2579–2605, Nov. 2008. [Online]. Available: http://www.jmlr.org/papers/v9/vandermaaten08a.html

[52] X. He and P. Niyogi, "Locality preserving projections," in *Proceedings of the 16th International Conference on Neural Information Processing Systems*. MIT Press, Dec. 2003, pp. 153—-160, doi: 10.5555/2981345.2981365.

[53] J. Kruskal and M. Wish, *Multidimensional Scaling*. Newbury Park, CA: SAGE Publications, 1978, doi: 10.4135/9781412985130.

[54] A. Gisbrecht and B. Hammer, "Data visualization by nonlinear dimensionality reduction," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 5, no. 2, pp. 51–73, 2015, doi: 10.1002/widm.1147.

[55] G. V. Vracem and S. Nijssen, "Iterated matrix reordering," in *Machine Learning and Knowledge Discovery in Databases. Research Track*, vol. 12977. Springer International Publishing, Sep. 2021, pp. 745–761, doi: 10.1007/978-3-030-86523-8_45.

[56] C. G. da Silva, "Hybrid sort a pattern-focused matrix reordering approach based on classification," in *14th International Conference on Computer Graphics, Visualization, Computer Vision and Image Processing*. IADIS Press, Jul. 2020, doi: 10.33965/cgv2020202011l005.

[57] T. Bollen, G. Leurquin, and S. Nijssen, "Convomap: Using convolution to order boolean data," in *17th International Symposium on Intelligent Data Analysis*. Springer, Oct. 2018, pp. 62–74, doi: 10.1007/97830300176826.

[58] M. Hahsler, "An experimental comparison of seriation methods for one-mode two-way data," *European Journal of Operational Research*, vol. 257, no. 1, pp. 133–143, Feb. 2017, doi: 10.1016/j.ejor.2016.08.066.

[59] I. Liiv, "Seriation and matrix reordering methods: An historical overview," *Statistical Analysis and Data Mining*, vol. 3, no. 2, pp. 70—–91, Apr. 2010, doi: 10.5555/3160819.3160821.

[60] M. Hahsler, K. Hornik, and C. Buchta, "Getting things in order: An introduction to the R Package seriation," *Journal of Statistical Software*, vol. 25, no. 3, pp. 1–34, Mar. 2008, doi: 10.18637/jss.v025.i03.

[61] G. Gutin and A. P. Punnen, Eds., *The Traveling Salesman Problem and Its Variations*. Boston, MA: Springer US, 2007, doi: 10.1007/b101971.

[62] K. Menger, "Das botenproblem," in *Ergebnisse eines Mathematischen Kolloquiums*, vol. 2. Leipzig: Teubner, 1932, pp. 11–12.

[63] G. Dantzig, R. Fulkerson, and S. Johnson, "Solution of a large-scale traveling-salesman problem," *Journal of the Operations Research Society of America*, vol. 2, no. 4, pp. 393–410, Nov. 1954, doi: 10.1287/opre.2.4.393.

[64] A. H. Land and A. G. Doig, "An automatic method of solving discrete programming problems," *Econometrica*, vol. 28, no. 3, p. 497, Jul. 1960, doi: 10.2307/1910129.

[65] M. Held and R. M. Karp, "A dynamic programming approach to sequencing problems," *Journal of the Society for Industrial and Applied Mathematics*, vol. 10, no. 1, pp. 196–210, Mar. 1962, doi: 10.1137/0110015.

[66] M. Padberg and G. Rinaldi, "A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems," *SIAM Review*, vol. 33, no. 1, pp. 60–100, Mar. 1991, doi: 10.1137/1033004.

[67] D. Applegate, R. Bixby, V. Chvatal, and W. Cook, "TSP cuts which do not conform to the template paradigm," in *Computational Combinatorial Optimization*. Springer Berlin Heidelberg, Nov. 2001, vol. 2241, pp. 261–303, doi: 10.1007/3-540-45586-8_7.

[68] M. Hahsler and K. Hornik, "TSP infrastructure for the traveling salesperson problem," *Journal of Statistical Software*, vol. 23, no. 2, Dec. 2007, doi: 10.18637/jss.v023.i02.

[69] H. H. Hoos and T. Stützle, "On the empirical scaling of run-time for finding optimal solutions to the travelling salesman problem," *European Journal of Operational Research*, vol. 238, no. 1, pp. 87–94, Oct. 2014, doi: 10.1016/j.ejor.2014.03.042.

[70] D. Sanches, D. Whitley, and R. Tinós, "Improving an exact solver for the traveling salesman problem using partition crossover," in *Proceedings of the Genetic and Evolutionary Computation Conference*. Association for Computing Machinery, Jul. 2017, doi: 10.1145/3071178.3071304.

[71] D. L. Applegate, R. E. Bixby, V. Chvátal, W. Cook, D. G. Espinoza, M. Goycoolea, and K. Helsgaun, "Certification of an optimal TSP tour through 85.900 cities," *Operations Research Letters*, vol. 37, no. 1, pp. 11–15, Jan. 2009, doi: 10.1016/j.orl.2008.09.006.

[72] H. A. Abdulkarim and I. F. Alshammari, "Comparison of algorithms for solving traveling salesman problem," *International Journal of Engineering and Advanced Technology*, vol. 4, pp. 76–79, Aug. 2015. [Online]. Available: https://www.ijeat.org/wp-content/uploads/papers/v4i6/F4173084615.pdf

[73] D. J. Rosenkrantz, R. E. Stearns, and P. M. Lewis, "An analysis of several heuristics for the traveling salesman problem," *SIAM Journal on Computing*, vol. 6, no. 3, pp. 563–581, Sep. 1977, doi: 10.1137/0206041.

[74] J. H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. Cambridge, MA: MIT Press, 1992, doi: 10.5555/531075.

[75] G. A. Croes, "A method for solving traveling-salesman problems," *Operations Research*, vol. 6, no. 6, pp. 791–812, Dec. 1958, doi: 10.1287/opre.6.6.791.

[76] S. Lin and B. W. Kernighan, "An effective heuristic algorithm for the traveling-salesman problem," *Operations Research*, vol. 21, no. 2, pp. 498–516, Apr. 1973, doi: 10.1287/opre.21.2.498.

[77] K. Helsgaun, "LKH an effective implementation of the lin-kernighan traveling salesman problem heuristic," LKH, Accessed on: Sept. 10, 2021. [Online]. Available: http://webhotel4.ruc.dk/~keld/research/LKH/

[78] D. Applegate, W. Cook, and A. Rohe, "Chained lin-kernighan for large traveling salesman problems," *INFORMS Journal on Computing*, vol. 15, no. 1, pp. 82–92, Feb. 2003, doi: 10.1287/ijoc.15.1.82.15157.

[79] D. S. Johnson, "Local optimization and the traveling salesman problem," in *Automata, Languages and Programming*. Springer-Verlag, Jun. 1990, pp. 446–461, doi: 10.1007/bfb0032050.

[80] K. Helsgaun, "General k-opt submoves for the lin–kernighan TSP heuristic," *Mathematical Programming Computation*, vol. 1, no. 2-3, pp. 119–163, Jul. 2009, doi: 10.1007/s12532-009-0004-6.

[81] B. Lee, C. Plaisant, C. S. Parr, J. D. Fekete, and N. Henry, "Task taxonomy for graph visualization," in *Proceedings of the 2006 AVI workshop on Beyond time and errors: novel evaluation methods for information visualization*. Association for Computing Machinery, May 2006, pp. 1–5, doi: 10.1145/1168149.1168168.

[82] W. S. Robinson, "A method for chronologically ordering archaeological deposits," *American antiquity*, vol. 16, no. 4, pp. 293–301, Apr. 1951, doi: 10.2307/276978.

[83] C. H. Chen, "Generalized association plots: Information visualization via iteratively generated correlation matrices," *Statistica Sinica*, pp. 7–29, 2002. [Online]. Available: http://www3.stat.sinica.edu.tw/statistica/oldpdf/A12n11.pdf

[84] L. Hubert, P. Arabie, and J. Meulman, *Combinatorial Data Analysis: Optimization by Dynamic Programming*. Philadelphia, PA: Society for Industrial and Applied Mathematics, Mar 2001, doi: 10.5555/377800.

[85] G. Caraux and S. Pinloche, "Permutmatrix: a graphical environment to arrange gene expression profiles in optimal linear order," *Bioinformatics*, vol. 21, no. 7, pp. 1280–1281, Apr. 2005, doi: 10.1093/bioinformatics/bti141.

[86] L. Hubert and J. Schultz, "Quadratic assignment as a general data analysis strategy," *British Journal of Mathematical and Statistical Psychology*, vol. 29, no. 2, pp. 190–241, 1976, doi: 10.1111/j.2044-8317.1976.tb00714.x.

[87] L. Hubert, "Some applications of graph theory and related non-metric techniques to problems of approximate seriation: The case of symmetric proximity measures," *British Journal of Mathematical and Statistical Psychology*, vol. 27, no. 2, pp. 133–153, Nov. 1974, doi: 10.1111/j.2044-8317.1974.tb00534.x.

[88] S. Bhagat, G. Cormode, and S. Muthukrishnan, "Node classification in social networks," in *Social network data analytics*. Boston, MA: Springer, Mar. 2011, pp. 115–148, doi: 10.1007/978-1-4419-8462-3_5.

[89] I. Makarov, D. Kiselev, N. Nikitinsky, and L. Subelj, "Survey on graph embeddings and their applications to machine learning problems on graphs," *PeerJ Computer Science*, vol. 7, p. e357, Feb. 2021, doi: 10.7717/peerj-cs.357.

[90] M. Nissim, R. van Noord, and R. van der Goot, "Fair is better than sensational: Man is to doctor as woman is to doctor," in *Computational Linguistics*, no. 2, Jun. 2019, pp. 487—497, doi: 10.1162/coli_a_00379.

[91] D. Jurafsky and J. Martin, *Speech and Language Processing*, 3rd ed. Upper Saddle River, NJ: Prentice Hall, 2019.

[92] L. McInnes, J. Healy, and J. Melville, "UMAP: Uniform manifold approximation and projection for dimension reduction," 2018, doi: 10.48550/arxiv.1802.03426.

[93] J. Briët and P. Harremoës, "Properties of classical and quantum jensen-shannon divergence," *Physical review A*, vol. 79, no. 5, p. 052311, May 2009, doi: 10.1103/PhysRevA.79.052311.

[94] X. He, H. Zhang, M.-Y. Kan, and T. S. Chua, "Fast matrix factorization for online recommendation with implicit feedback," in *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*. Association for Computing Machinery, Jul. 2016, pp. 549–558, doi: 10.1145/2911451.2911489.

[95] A. Moore, "An intoductory tutorial on kd–trees," *Computer Laboratory, University of Cambridge*, 1991, doi: 10.1.1.28.6468.

[96] H. Munaga and V. Jarugumalli, "Performance evaluation: Ball-tree and kd-tree in the context of mst," in *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*. Springer Berlin Heidelberg, Oct. 2012, pp. 225–228, doi: 10.1007/978-3-642-32573-1_38.

[97] S. M. Omohundro, "Five balltree construction algorithms," International Computer Science Institute Berkeley, Berkely, CA, Tech. Rep., Dec. 1989. [Online]. Available: http://www.icsi.berkeley.edu/ftp/global/pub/techreports/1989/tr-89-063.pdf

[98] A. Hagberg, P. Swart, and D. S Chult, "Exploring network structure, dynamics, and function using networkx," in *Proceedings of the 7th Python in Science Conference (SciPy)*, Jan. 2008, pp. 11–15. [Online]. Available: http://conference.scipy.org/proceedings/SciPy2008/paper_2/

[99] L. Takac and M. Zábovský, "Data analysis in public social networks," *International Scientific Conference and International Workshop Present Day Trends of Innovations*, vol. 1, no. 6, pp. 1–6, Jan. 2012. [Online]. Available: https://snap.stanford.edu/data/soc-pokec.pdf

[100] M. Girvan and M. E. Newman, "Community structure in social and biological networks," *Proceedings of the national academy of sciences*, vol. 99, no. 12, pp. 7821–7826, Jun. 2002, doi: 10.1073/pnas.122653799.

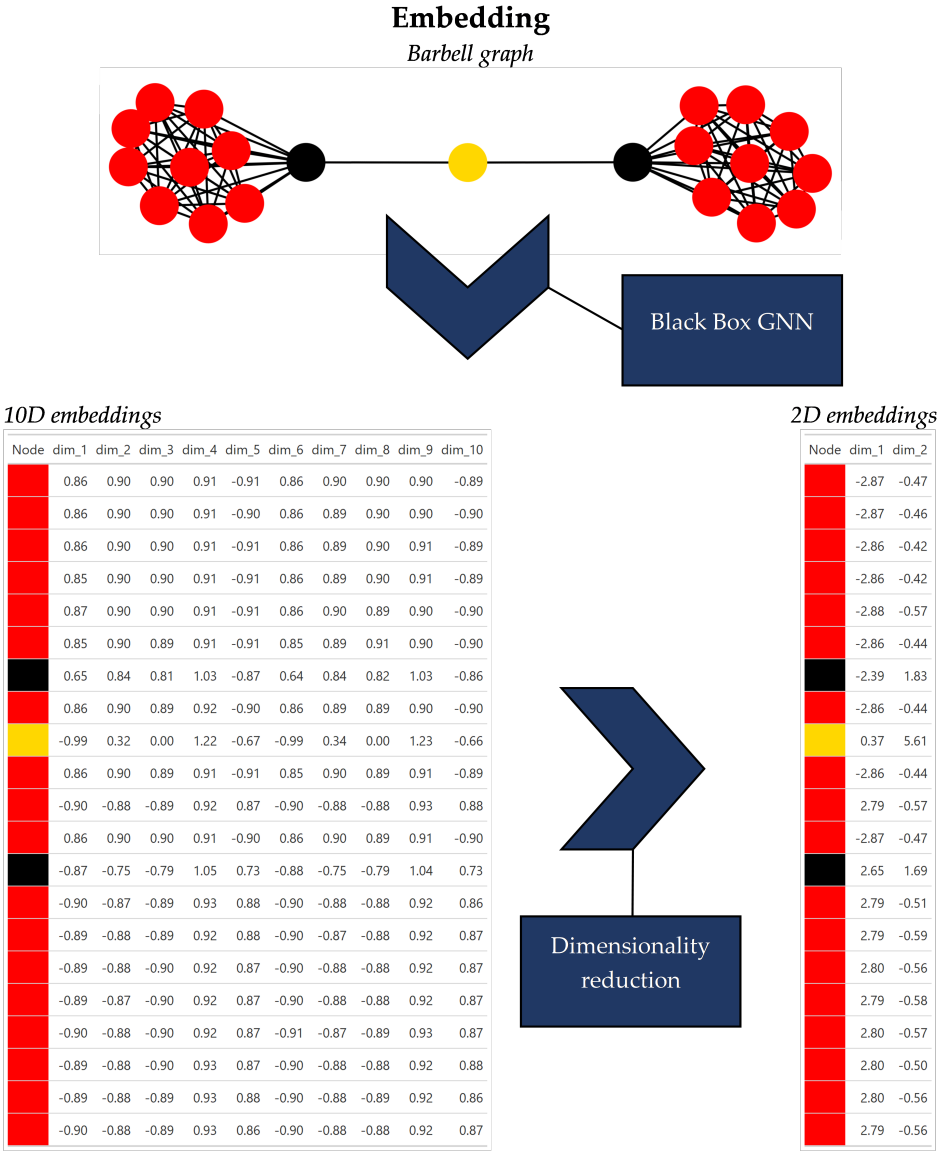
[101] N. van Beusekom, W. Meulemans, and B. Speckmann, "Simultaneous matrix orderings for graph collections," *IEEE Transactions on Visualization and Computer Graphics*, vol. 28, no. 1, pp. 1–10, Jan. 2022, doi: 10.1109/tvcg.2021.3114773.

[102] K. Gadzicki, R. Khamsehashari, and C. Zetzsche, "Early vs late fusion in multimodal convolutional neural networks," in *2020 IEEE 23rd International Conference on Information Fusion (FUSION)*. IEEE, Jul. 2020, pp. 1–6, doi: 10.23919/fusion45008.2020.9190246.

[103] D. Lahat, T. Adali, and C. Jutten, "Multimodal data fusion: an overview of methods, challenges, and prospects," *Proceedings of the IEEE*, vol. 103, no. 9, pp. 1449–1477, Sep. 2015, doi: 10.1109/jproc.2015.2460697.

[104] B. F. Medina, W. H. Kawakami, M. R. da Silva, and C. G. da Silva, "Smoothed multiple binarization – using PQR tree, smoothing, feature vectors and thresholding for matrix reordering," in *2016 20th International Conference Information Visualisation (IV)*. IEEE, Jul. 2016, pp. 88–93, doi: 10.1109/iv.2016.22.

A Overview of Framework



Node	dim_1	dim_2	dim_3	dim_4	dim_5	dim_6	dim_7	dim_8	dim_9	dim_10
	0.86	0.90	0.90	0.91	-0.91	0.86	0.90	0.90	0.90	-0.89
	0.86	0.90	0.90	0.91	-0.90	0.86	0.89	0.90	0.90	-0.90
	0.86	0.90	0.90	0.91	-0.91	0.86	0.89	0.90	0.91	-0.89
	0.85	0.90	0.90	0.91	-0.91	0.86	0.89	0.90	0.91	-0.89
	0.87	0.90	0.90	0.91	-0.91	0.86	0.90	0.89	0.90	-0.90
	0.85	0.90	0.89	0.91	-0.91	0.85	0.89	0.91	0.90	-0.90
	0.65	0.84	0.81	1.03	-0.87	0.64	0.84	0.82	1.03	-0.86
	0.86	0.90	0.89	0.92	-0.90	0.86	0.89	0.89	0.90	-0.90
	-0.99	0.32	0.00	1.22	-0.67	-0.99	0.34	0.00	1.23	-0.66
	0.86	0.90	0.89	0.91	-0.91	0.85	0.90	0.89	0.91	-0.89
	-0.90	-0.88	-0.89	0.92	0.87	-0.90	-0.88	-0.88	0.93	0.88
	0.86	0.90	0.90	0.91	-0.90	0.86	0.90	0.89	0.91	-0.90
	-0.87	-0.75	-0.79	1.05	0.73	-0.88	-0.75	-0.79	1.04	0.73
	-0.90	-0.87	-0.89	0.93	0.88	-0.90	-0.88	-0.88	0.92	0.86
	-0.89	-0.88	-0.89	0.92	0.88	-0.90	-0.87	-0.88	0.92	0.87
	-0.89	-0.88	-0.90	0.92	0.87	-0.90	-0.88	-0.88	0.92	0.87
	-0.89	-0.87	-0.90	0.92	0.87	-0.90	-0.88	-0.88	0.92	0.87
	-0.90	-0.88	-0.90	0.92	0.87	-0.91	-0.87	-0.89	0.93	0.87
	-0.89	-0.88	-0.90	0.93	0.87	-0.90	-0.88	-0.88	0.92	0.88
	-0.89	-0.88	-0.89	0.93	0.88	-0.90	-0.88	-0.89	0.92	0.86
	-0.90	-0.88	-0.89	0.93	0.86	-0.90	-0.88	-0.88	0.92	0.87

Node	dim_1	dim_2
	-2.87	-0.47
	-2.87	-0.46
	-2.86	-0.42
	-2.86	-0.42
	-2.88	-0.57
	-2.86	-0.44
	-2.39	1.83
	-2.86	-0.44
	0.37	5.61
	-2.86	-0.44
	2.79	-0.57
	-2.87	-0.47
	2.65	1.69
	2.79	-0.51
	2.79	-0.59
	2.80	-0.56
	2.79	-0.58
	2.80	-0.57
	2.80	-0.50
	2.80	-0.56
	2.79	-0.56

Figure 43: Embedding phase with a barbell graph as input and n dimensional embeddings from GNNs shown afterwards. Note that red, black and gold nodes represent different roles in the graph. The output is fed as input to figure 44

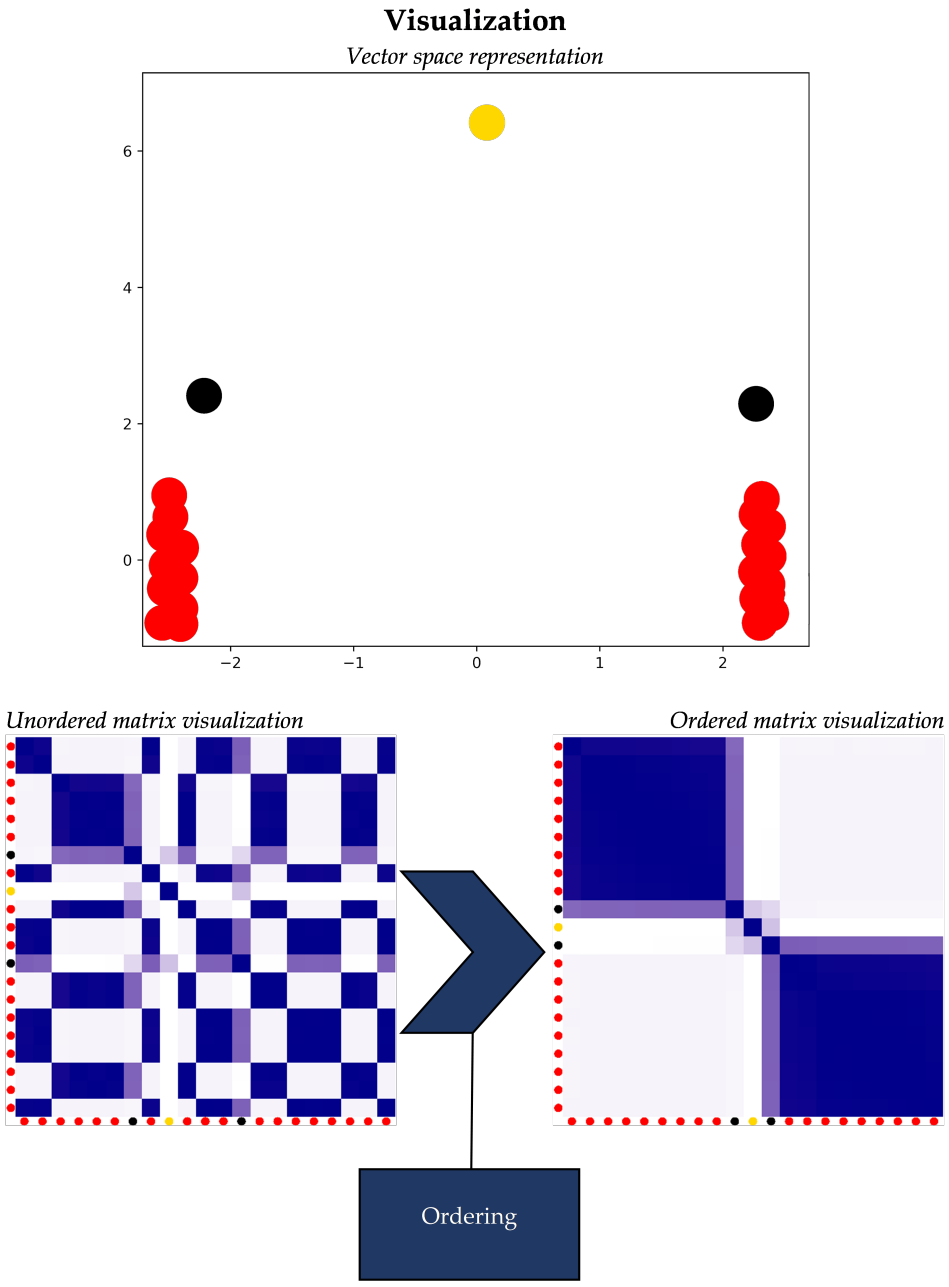


Figure 44: Visualization phase with a vector space representation on top, and the same data in matrix representation below. An unordered representation is shown on the left, and after ordering an ordered version on the right

B Algorithmic settings

The algorithmic dimensions for Laplacian Eigenmaps, Locally Linear embedding are 2. HOPE was used with a dimension of 4 and a β of 0.01. SDNE has he following parameters in this experiment: $\beta : 2, \alpha : 0.2, \mu_1 : 1e-3, \mu_2 : 1e-3, K : 2, n\_units : [20, 10], n\_iter : 20, \zeta : 1e-2, n\_batch : 15$. And finally, the random walk algorithms had the parameters listed in table 11.

	DeepWalk	node2vec	Walklets	struc2vec	attention Walk
dim	10	10	10	10	10
walk_number	20	20	20	20	80
walk_length	80	80	80	80	-
window_size	5	5	5	5	5
p	1	2	-	-	-
q	1	0.25	-	-	-
epochs	-	-	10	-	100
learning_rate	-	-	0.05	-	0.01
γ	-	-	-	-	0.5

Table 11: Random Walk Algorithmic parameters

C Extra Visualizations

DeepWalk & Walklets: (Un-)Weighted NS

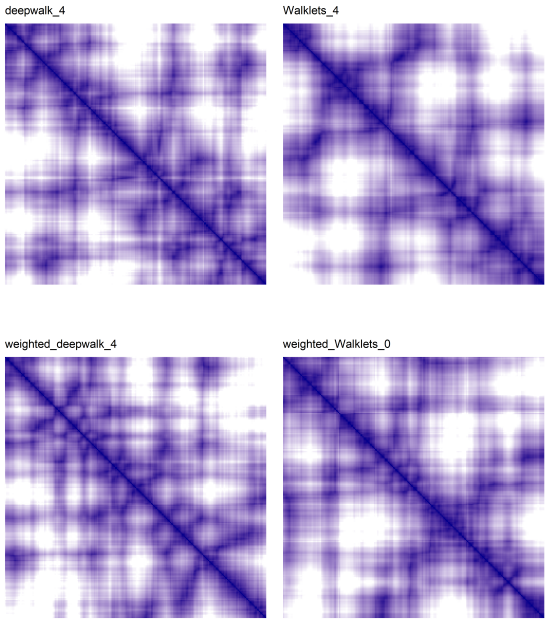


Figure 45: Structure visualization

DeepWalk: Pokec-100

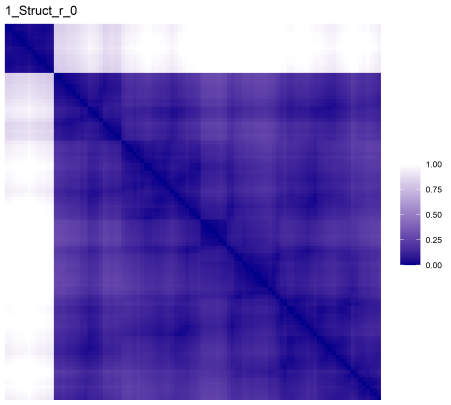


Figure 46: Structure visualization

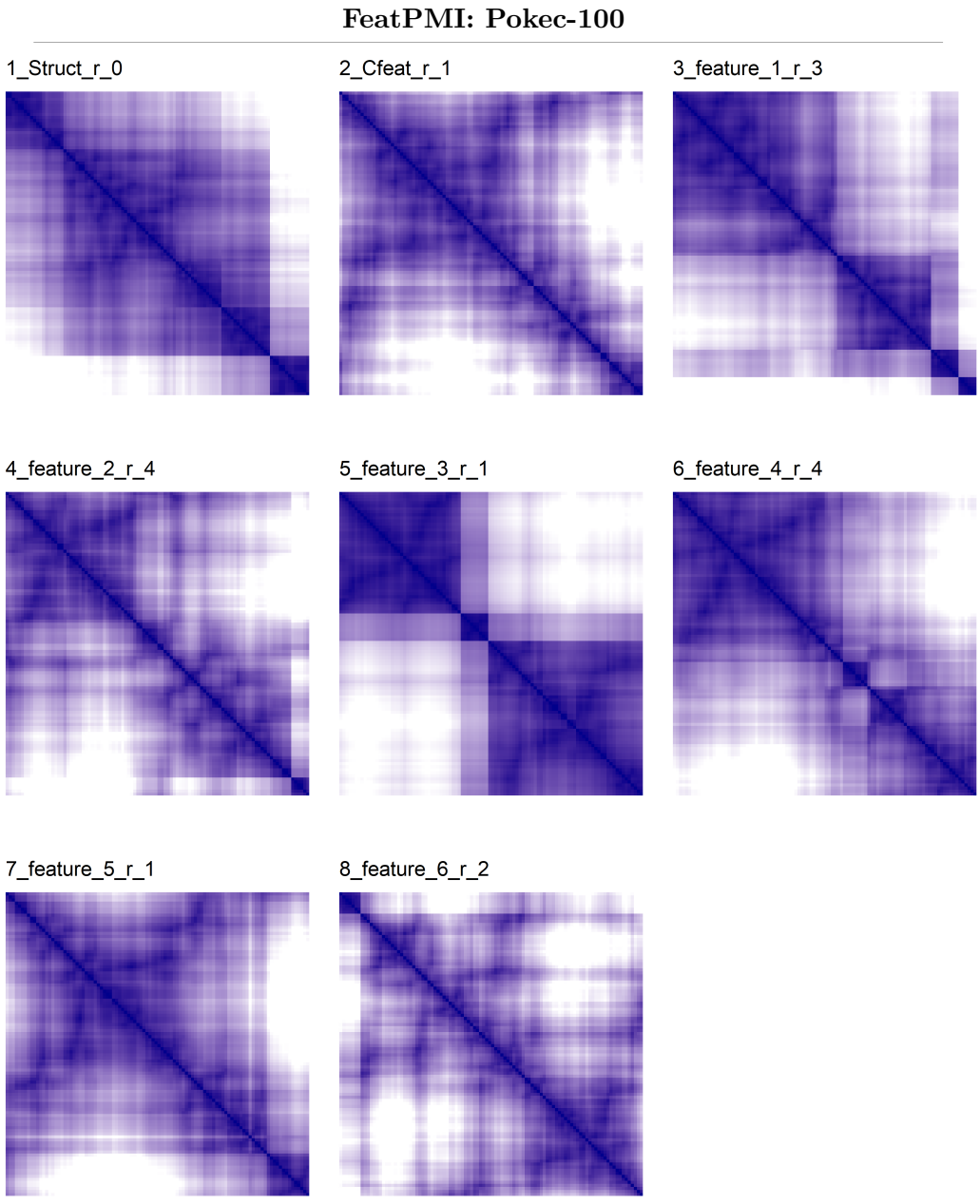


Figure 47: All visualizations

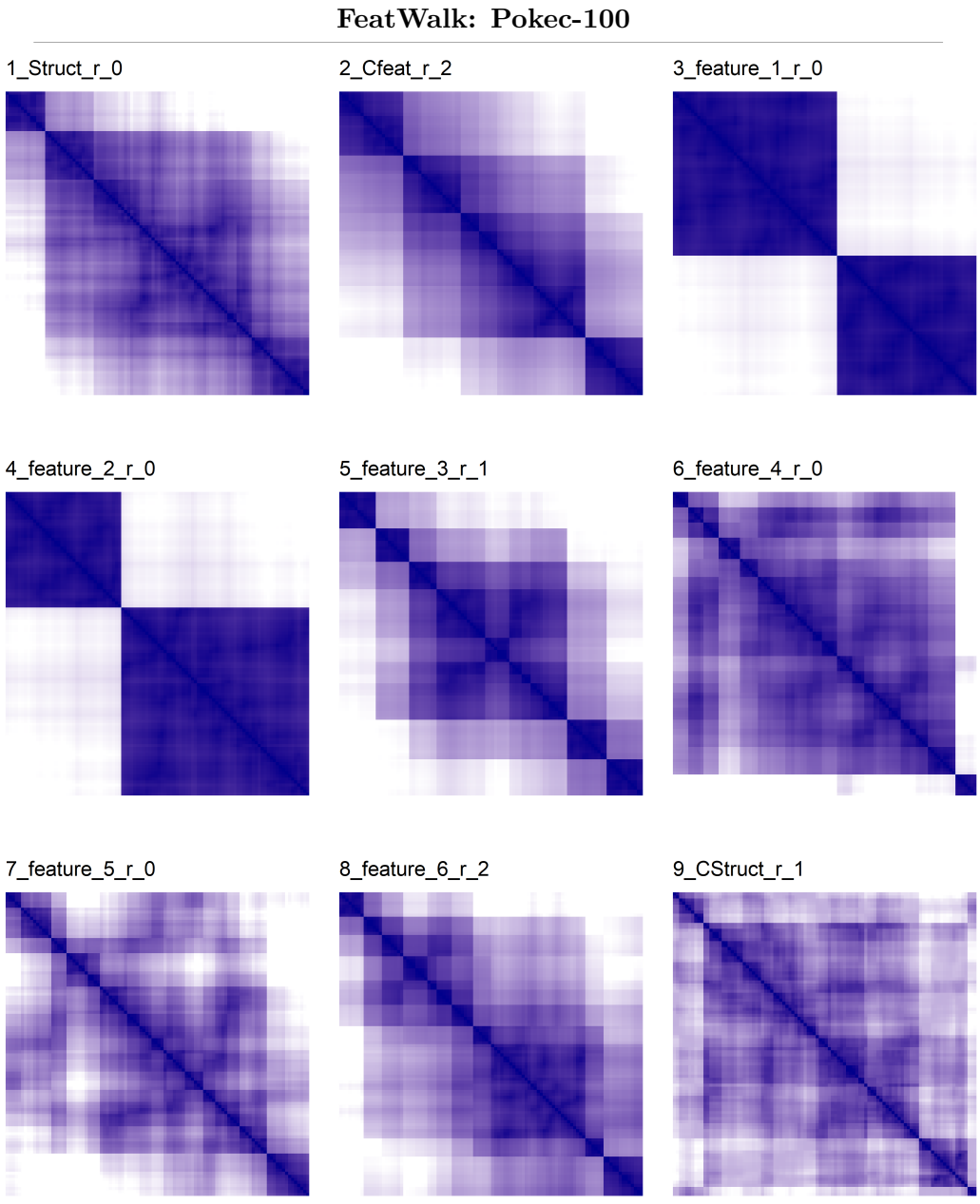


Figure 48: All visualizations

DeepWalk: Pokec-5000

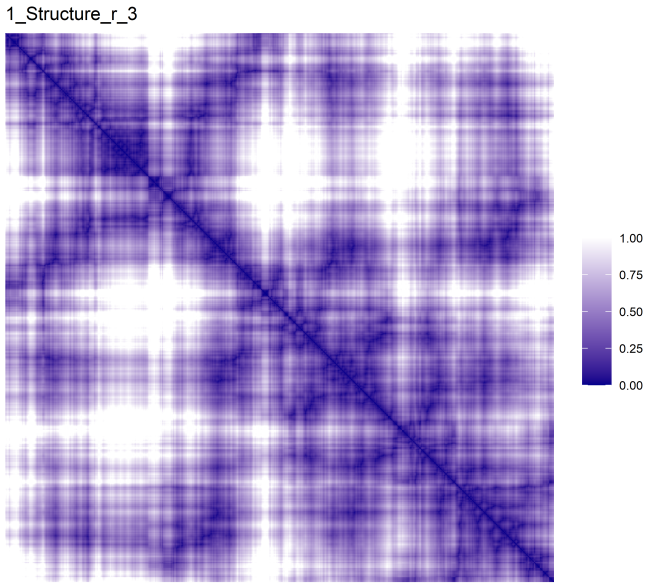


Figure 49: Structure visualization

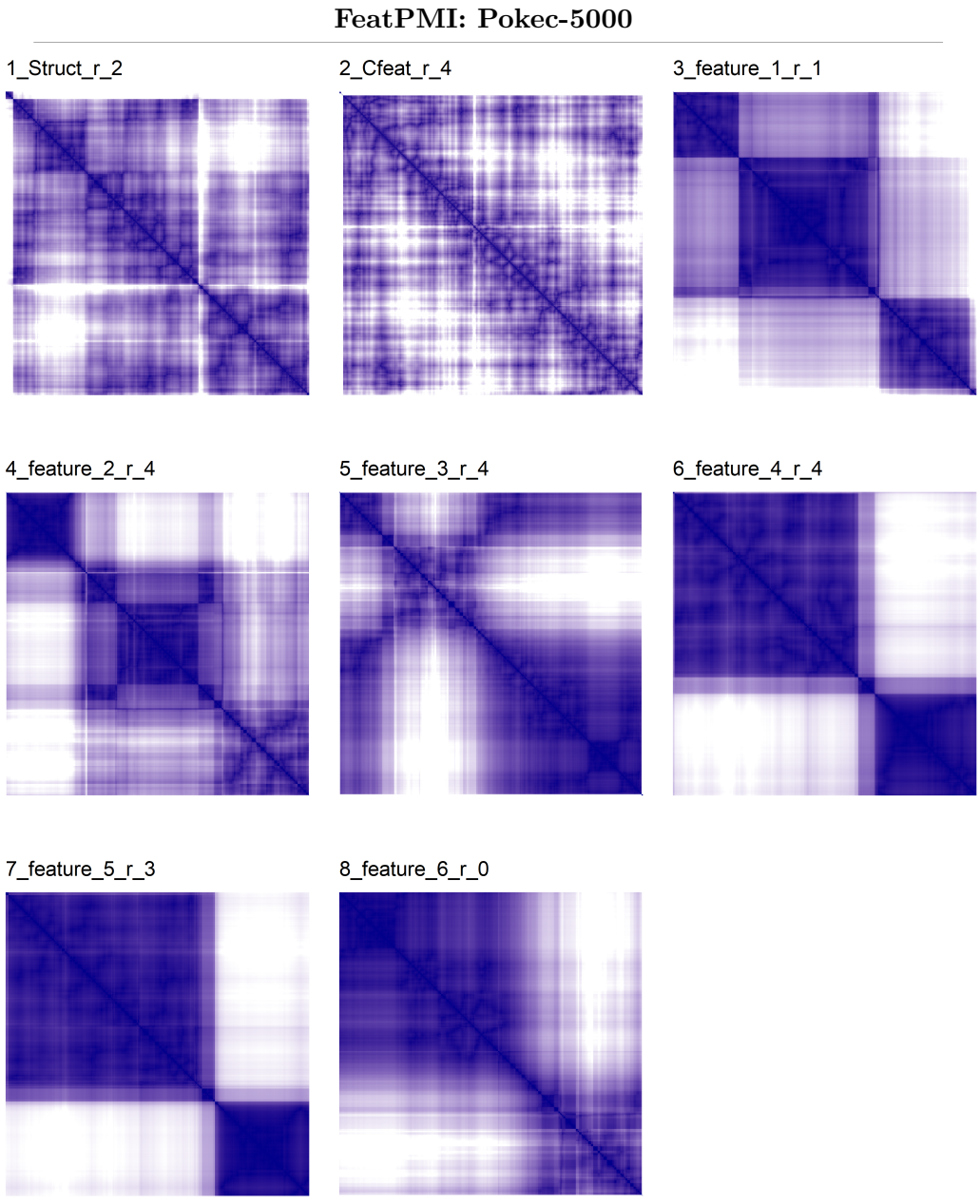


Figure 50: All visualizations

FeatWalk: Pokec-5000

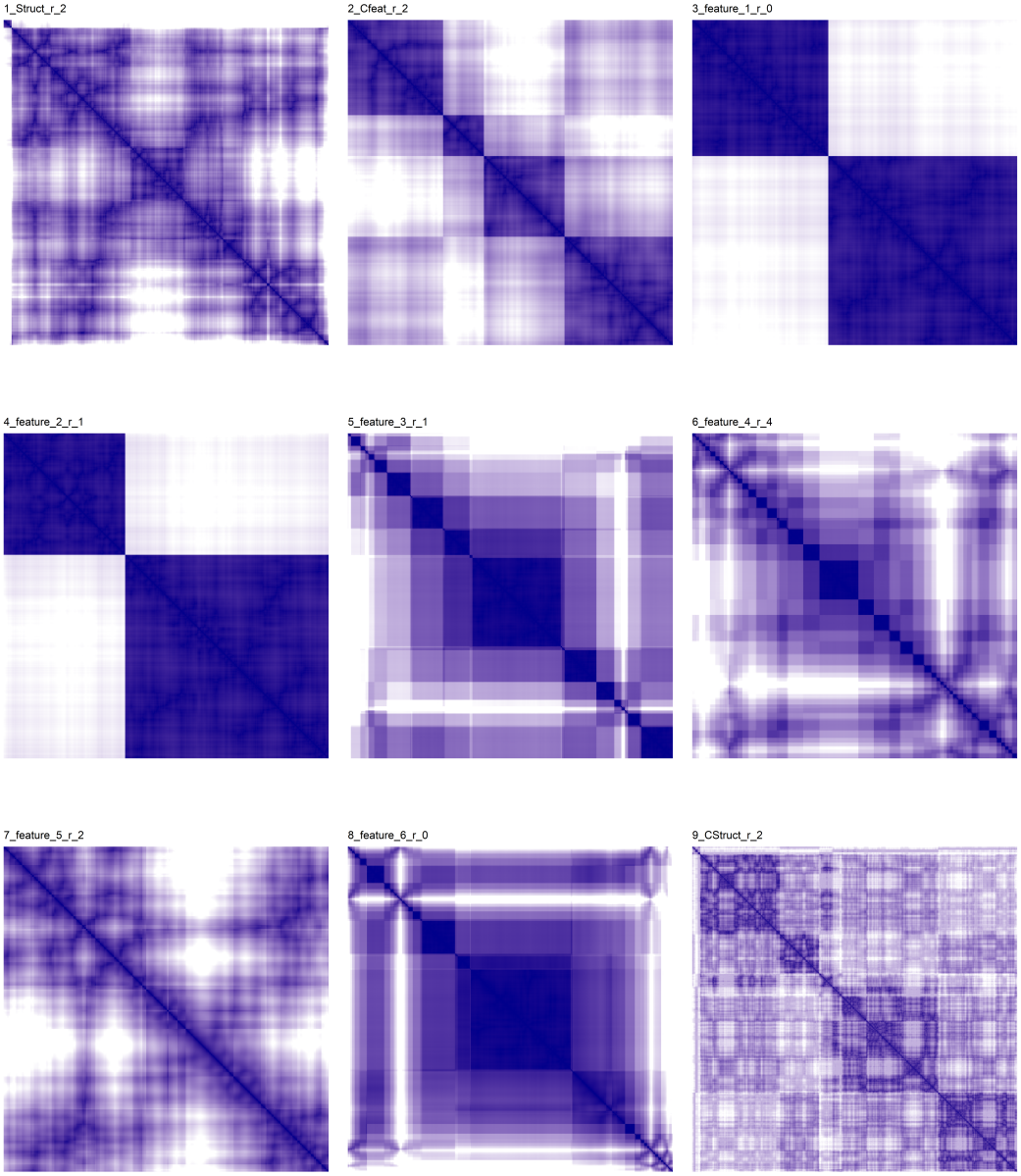


Figure 51: All visualizations

D VQM tables Pokec

DeepWalk & Walklets: (Un-)Weighted NS

Average Visual Quality Metric Scores of 5 Runs

Dataset: *directed_weighted_NS*

VQM	*deepwalk*	*Walklets*	*weighted_dee*	*weighted_Wal*
2-Sum	3.70×10^{8}	3.74×10^{8}	**3.69×10^{8}**	3.71×10^{8}
Anti-Rob Deviations	3.14×10^{5}	3.24×10^{5}	**3.08×10^{5}**	3.18×10^{5}
Anti-Rob Events	1.51×10^{6}	1.58×10^{6}	**1.46×10^{6}**	1.55×10^{6}
Banded Anti-Rob	8.93×10^{4}	**7.94×10^{4}**	9.12×10^{4}	8.35×10^{4}
Cor_R	**7.12×10^{-2}**	7.48×10^{-2}	7.21×10^{-2}	7.36×10^{-2}
Gradient measure	1.82×10^{6}	1.68×10^{6}	**1.91×10^{6}**	1.74×10^{6}
Weighted G measure	7.22×10^{5}	7.00×10^{5}	**7.33×10^{5}**	7.13×10^{5}
Inertia	4.41×10^{8}	4.31×10^{8}	**4.44×10^{8}**	4.37×10^{8}
Lazy path length	1.56×10^{3}	**1.15×10^{3}**	1.61×10^{3}	1.37×10^{3}
Least Squares	**6.13×10^{8}**	6.14×10^{8}	6.13×10^{8}	6.13×10^{8}
Linear Seriation	5.46×10^{6}	5.42×10^{6}	**5.39×10^{6}**	5.45×10^{6}
Measure of effect	5.26×10^{4}	**5.34×10^{4}**	5.29×10^{4}	5.29×10^{4}
Stress Moore	2.54×10^{2}	**1.65×10^{2}**	2.64×10^{2}	2.06×10^{2}
Stress Neumann	8.77×10^{1}	**5.67×10^{1}**	9.10×10^{1}	7.10×10^{1}
Ham Path length	1.26×10^{1}	**9.19**	1.30×10^{1}	1.08×10^{1}
RGAR	3.04×10^{-1}	3.18×10^{-1}	**2.95×10^{-1}**	3.13×10^{-1}

*__Cyan__ (lower is better), indicates best low score. __Yellow__ (higher is better), indicates best high score.

Figure 52: Visual quality metrics

FeatPMI: Pokec-100

Average Visual Quality Metric Scores of 5 Runs

Dataset: *pokec_rel_100_featPMI*

VQM	*1_Struct*	*2_Cfeat*	*3_feature_1*	*4_feature_2*	*5_feature_3*	*6_feature_4*	*7_feature_5*	*8_feature_6*
2-Sum	9.98×10^{6}	1.07×10^{7}	1.02×10^{7}	9.77×10^{6}	9.51×10^{6}	1.00×10^{7}	1.04×10^{7}	1.07×10^{7}
Anti-Rob Deviations	3.77×10^{3}	1.11×10^{4}	3.35×10^{3}	1.16×10^{4}	5.61×10^{3}	8.81×10^{3}	1.21×10^{4}	1.40×10^{4}
Anti-Rob Events	4.41×10^{4}	8.20×10^{4}	4.66×10^{4}	7.66×10^{4}	6.39×10^{4}	6.54×10^{4}	8.30×10^{4}	9.13×10^{4}
Banded Anti-Rob	3.50×10^{3}	3.69×10^{3}	3.16×10^{3}	4.47×10^{3}	3.06×10^{3}	4.00×10^{3}	4.24×10^{3}	4.57×10^{3}
Cor_R	1.03×10^{-1}	7.21×10^{-2}	1.11×10^{-1}	1.20×10^{-1}	1.79×10^{-1}	1.12×10^{-1}	7.30×10^{-2}	6.32×10^{-2}
Gradient measure	2.24×10^{5}	1.48×10^{5}	2.16×10^{5}	1.60×10^{5}	1.85×10^{5}	1.81×10^{5}	1.46×10^{5}	1.28×10^{5}
Weighted G measure	7.81×10^{4}	6.01×10^{4}	7.69×10^{4}	7.01×10^{4}	9.37×10^{4}	7.12×10^{4}	5.84×10^{4}	4.84×10^{4}
Inertia	1.20×10^{7}	1.04×10^{7}	1.14×10^{7}	1.23×10^{7}	1.32×10^{7}	1.18×10^{7}	1.09×10^{7}	1.03×10^{7}
Lazy path length	1.86×10^{2}	2.58×10^{2}	1.63×10^{2}	3.02×10^{2}	1.79×10^{2}	2.70×10^{2}	2.86×10^{2}	2.95×10^{2}
Least Squares	1.62×10^{7}	1.63×10^{7}	1.63×10^{7}	1.62×10^{7}	1.62×10^{7}	1.62×10^{7}	1.63×10^{7}	1.63×10^{7}
Linear Seriation	2.60×10^{5}	2.58×10^{5}	2.43×10^{5}	3.12×10^{5}	2.79×10^{5}	2.82×10^{5}	2.82×10^{5}	2.83×10^{5}
Measure of effect	1.04×10^{4}	1.08×10^{4}	1.10×10^{4}	9.47×10^{3}	1.02×10^{4}	9.98×10^{3}	1.01×10^{4}	1.03×10^{4}
Stress Moore	1.03×10^{2}	9.09×10^{1}	9.80×10^{1}	7.92×10^{1}	6.50×10^{1}	7.21×10^{1}	8.28×10^{1}	1.14×10^{2}
Stress Neumann	3.54×10^{1}	3.17×10^{1}	3.39×10^{1}	2.80×10^{1}	2.26×10^{1}	2.54×10^{1}	2.93×10^{1}	3.97×10^{1}
Ham Path length	4.05	4.86	3.70	5.94	3.74	5.39	5.86	6.01
RGAR	1.36×10^{-1}	2.54×10^{-1}	1.44×10^{-1}	2.37×10^{-1}	1.98×10^{-1}	2.02×10^{-1}	2.57×10^{-1}	2.82×10^{-1}

*__First__ number indicates order. __Second__ number (if available) indicates the feature number.

Figure 53: Visual quality metrics

FeatWalk: Pokec-100

Average Visual Quality Metric Scores of 5 Runs

Dataset: *pokec_rel_100_featWalk*

VQM	*1_Struct*	*2_Cfeat*	*3_feature_1*	*4_feature_2*	*5_feature_3*	*6_feature_4*	*7_feature_5*	*8_feature_6*	*9_CStruct*
2-Sum	9.99×10^{6}	9.50×10^{6}	9.38×10^{6}	9.63×10^{6}	9.56×10^{6}	1.02×10^{7}	9.72×10^{6}	9.52×10^{6}	1.00×10^{7}
Anti-Rob Deviations	4.14×10^{3}	2.93×10^{3}	7.43×10^{2}	1.18×10^{3}	4.60×10^{3}	1.25×10^{4}	1.16×10^{4}	3.31×10^{3}	8.92×10^{3}
Anti-Rob Events	4.38×10^{4}	3.69×10^{4}	4.82×10^{4}	5.16×10^{4}	4.32×10^{4}	7.63×10^{4}	7.18×10^{4}	4.17×10^{4}	8.21×10^{4}
Banded Anti-Rob	3.52×10^{3}	3.01×10^{3}	1.96×10^{3}	2.16×10^{3}	4.06×10^{3}	5.05×10^{3}	5.19×10^{3}	4.06×10^{3}	6.12×10^{3}
Cor_R	1.02×10^{-1}	1.48×10^{-1}	2.32×10^{-1}	2.03×10^{-1}	1.06×10^{-1}	6.51×10^{-2}	8.63×10^{-2}	1.25×10^{-1}	6.26×10^{-2}
Gradient measure	2.24×10^{5}	2.39×10^{5}	2.17×10^{5}	2.10×10^{5}	2.26×10^{5}	1.58×10^{5}	1.71×10^{5}	2.28×10^{5}	1.59×10^{5}
Weighted G measure	7.75×10^{4}	8.98×10^{4}	1.17×10^{5}	1.09×10^{5}	8.09×10^{4}	5.47×10^{4}	6.43×10^{4}	8.35×10^{4}	5.00×10^{4}
Inertia	1.20×10^{7}	1.31×10^{7}	1.43×10^{7}	1.37×10^{7}	1.30×10^{7}	1.12×10^{7}	1.25×10^{7}	1.31×10^{7}	1.14×10^{7}
Lazy path length	1.96×10^{2}	1.23×10^{2}	9.46×10^{1}	1.20×10^{2}	1.71×10^{2}	2.73×10^{2}	2.88×10^{2}	1.94×10^{2}	4.21×10^{2}
Least Squares	1.62×10^{7}	1.62×10^{7}	1.62×10^{7}	1.62×10^{7}	1.62×10^{7}	1.63×10^{7}	1.62×10^{7}	1.62×10^{7}	1.62×10^{7}
Linear Seriation	2.61×10^{5}	2.80×10^{5}	2.55×10^{5}	2.48×10^{5}	2.98×10^{5}	2.99×10^{5}	3.27×10^{5}	2.93×10^{5}	3.21×10^{5}
Measure of effect	1.04×10^{4}	1.00×10^{4}	1.17×10^{4}	1.18×10^{4}	9.87×10^{3}	9.79×10^{3}	9.14×10^{3}	9.70×10^{3}	9.12×10^{3}
Stress Moore	1.01×10^{2}	1.11×10^{2}	2.22×10^{2}	1.97×10^{2}	1.95×10^{2}	1.53×10^{2}	1.29×10^{2}	1.25×10^{2}	1.34×10^{2}
Stress Neumann	3.47×10^{1}	3.78×10^{1}	7.50×10^{1}	6.67×10^{1}	6.72×10^{1}	5.33×10^{1}	4.49×10^{1}	4.33×10^{1}	4.83×10^{1}
Ham Path length	4.05	2.56	1.91	2.27	3.41	5.41	5.59	4.02	8.55
RGAR	1.35×10^{-1}	1.14×10^{-1}	1.49×10^{-1}	1.59×10^{-1}	1.34×10^{-1}	2.36×10^{-1}	2.22×10^{-1}	1.29×10^{-1}	2.54×10^{-1}

*__First__ number indicates order. __Second__ number (if available) indicates the feature number.

Figure 54: Visual quality metrics

FeatPMI: Pokec-1000

Average Visual Quality Metric Scores of 5 Runs

Dataset: *pokec_rel_1000_featPMI*

VQM	*1_Struct*	*2_Cfeat*	*3_feature_1*	*4_feature_2*	*5_feature_3*	*6_feature_4*	*7_feature_5*	*8_feature_6*
2-Sum	9.82×10^{10}	1.01×10^{11}	9.45×10^{10}	9.74×10^{10}	1.01×10^{11}	9.97×10^{10}	9.77×10^{10}	9.73×10^{10}
Anti-Rob Deviations	7.74×10^{6}	2.33×10^{7}	9.77×10^{6}	1.44×10^{7}	1.34×10^{7}	2.14×10^{7}	1.65×10^{7}	1.67×10^{7}
Anti-Rob Events	7.10×10^{7}	1.10×10^{8}	6.71×10^{7}	8.34×10^{7}	8.12×10^{7}	9.90×10^{7}	8.78×10^{7}	8.74×10^{7}
Banded Anti-Rob	3.90×10^{6}	6.15×10^{6}	4.53×10^{6}	4.52×10^{6}	4.32×10^{6}	5.31×10^{6}	5.07×10^{6}	5.11×10^{6}
Cor_R	9.08×10^{-2}	4.88×10^{-2}	1.06×10^{-1}	9.82×10^{-2}	7.47×10^{-2}	6.05×10^{-2}	8.33×10^{-2}	8.82×10^{-2}
Gradient measure	1.75×10^{8}	9.52×10^{7}	1.80×10^{8}	1.47×10^{8}	1.47×10^{8}	1.18×10^{8}	1.41×10^{8}	1.42×10^{8}
Weighted G measure	6.97×10^{7}	3.75×10^{7}	7.21×10^{7}	6.27×10^{7}	5.68×10^{7}	4.68×10^{7}	5.65×10^{7}	5.71×10^{7}
Inertia	1.16×10^{11}	1.07×10^{11}	1.24×10^{11}	1.17×10^{11}	1.07×10^{11}	1.11×10^{11}	1.16×10^{11}	1.17×10^{11}
Lazy path length	4.85×10^{3}	1.42×10^{4}	8.45×10^{3}	8.66×10^{3}	7.67×10^{3}	1.02×10^{4}	9.93×10^{3}	9.27×10^{3}
Least Squares	1.63×10^{11}	1.63×10^{11}	1.63×10^{11}	1.63×10^{11}	1.63×10^{11}	1.63×10^{11}	1.63×10^{11}	1.63×10^{11}
Linear Seriation	2.82×10^{8}	3.45×10^{8}	3.11×10^{8}	3.15×10^{8}	2.87×10^{8}	3.36×10^{8}	3.28×10^{8}	3.30×10^{8}
Measure of effect	9.98×10^{5}	8.88×10^{5}	9.29×10^{5}	9.33×10^{5}	9.87×10^{5}	9.01×10^{5}	9.02×10^{5}	9.00×10^{5}
Stress Moore	9.47×10^{2}	1.49×10^{3}	1.12×10^{3}	1.22×10^{3}	1.03×10^{3}	1.06×10^{3}	1.06×10^{3}	8.74×10^{2}
Stress Neumann	3.17×10^{2}	5.01×10^{2}	3.76×10^{2}	4.09×10^{2}	3.46×10^{2}	3.56×10^{2}	3.56×10^{2}	2.94×10^{2}
Ham Path length	9.24	2.87×10^{1}	1.66×10^{1}	1.82×10^{1}	1.48×10^{1}	1.99×10^{1}	1.95×10^{1}	1.87×10^{1}
RGAR	2.17×10^{-1}	3.36×10^{-1}	2.05×10^{-1}	2.55×10^{-1}	2.48×10^{-1}	3.02×10^{-1}	2.68×10^{-1}	2.67×10^{-1}

*__First__ number indicates order. __Second__ number (if available) indicates the feature number.

Figure 55: Visual quality metrics

FeatWalk: Pokec-1000

Average Visual Quality Metric Scores of 5 Runs

Dataset: *pokec_rel_1000_featWalk*

VQM	*1_Struct*	*2_Cfeat*	*3_feature_1*	*4_feature_2*	*5_feature_3*	*6_feature_4*	*7_feature_5*	*8_feature_6*	*9_CStruct*
2-Sum	9.81×10^{10}	9.16×10^{10}	9.29×10^{10}	9.55×10^{10}	9.45×10^{10}	9.88×10^{10}	9.44×10^{10}	9.72×10^{10}	9.80×10^{10}
Anti-Rob Deviations	7.76×10^{6}	9.25×10^{6}	2.34×10^{6}	1.57×10^{6}	7.64×10^{6}	1.09×10^{7}	1.42×10^{7}	1.01×10^{7}	1.46×10^{7}
Anti-Rob Events	6.98×10^{7}	7.20×10^{7}	7.17×10^{7}	5.12×10^{7}	6.31×10^{7}	7.57×10^{7}	7.88×10^{7}	7.23×10^{7}	1.11×10^{8}
Banded Anti-Rob	3.92×10^{6}	3.63×10^{6}	2.02×10^{6}	1.95×10^{6}	4.16×10^{6}	4.45×10^{6}	4.79×10^{6}	4.35×10^{6}	6.73×10^{6}
Cor_R	8.95×10^{-2}	1.35×10^{-1}	2.16×10^{-1}	1.88×10^{-1}	9.16×10^{-2}	7.21×10^{-2}	9.06×10^{-2}	9.70×10^{-2}	5.95×10^{-2}
Gradient measure	1.77×10^{8}	1.75×10^{8}	1.73×10^{8}	2.15×10^{8}	1.88×10^{8}	1.62×10^{8}	1.61×10^{8}	1.69×10^{8}	1.05×10^{8}
Weighted G measure	7.00×10^{7}	7.70×10^{7}	1.10×10^{8}	1.05×10^{8}	7.24×10^{7}	6.19×10^{7}	6.21×10^{7}	6.58×10^{7}	3.66×10^{7}
Inertia	1.17×10^{11}	1.31×10^{11}	1.36×10^{11}	1.31×10^{11}	1.24×10^{11}	1.14×10^{11}	1.24×10^{11}	1.17×10^{11}	1.11×10^{11}
Lazy path length	4.55×10^{3}	2.64×10^{3}	2.79×10^{3}	2.49×10^{3}	3.05×10^{3}	3.24×10^{3}	2.84×10^{3}	4.20×10^{3}	1.84×10^{4}
Least Squares	1.63×10^{11}	1.63×10^{11}	1.63×10^{11}	1.63×10^{11}	1.63×10^{11}	1.63×10^{11}	1.63×10^{11}	1.63×10^{11}	1.63×10^{11}
Linear Seriation	2.81×10^{8}	3.42×10^{8}	2.54×10^{8}	2.38×10^{8}	3.10×10^{8}	2.96×10^{8}	3.47×10^{8}	2.99×10^{8}	3.64×10^{8}
Measure of effect	9.98×10^{5}	8.95×10^{5}	1.15×10^{6}	1.17×10^{6}	9.37×10^{5}	9.76×10^{5}	8.64×10^{5}	9.72×10^{5}	8.34×10^{5}
Stress Moore	9.59×10^{2}	1.21×10^{3}	1.88×10^{3}	1.72×10^{3}	1.56×10^{3}	1.33×10^{3}	7.01×10^{2}	1.75×10^{3}	1.69×10^{3}
Stress Neumann	3.21×10^{2}	4.06×10^{2}	6.29×10^{2}	5.73×10^{2}	5.22×10^{2}	4.45×10^{2}	2.35×10^{2}	5.88×10^{2}	5.74×10^{2}
Ham Path length	9.23	5.26	5.42	5.04	6.19	7.05	5.48	8.10	3.81×10^{1}
RGAR	2.13×10^{-1}	2.20×10^{-1}	2.19×10^{-1}	1.56×10^{-1}	1.93×10^{-1}	2.31×10^{-1}	2.41×10^{-1}	2.21×10^{-1}	3.39×10^{-1}

*__First__ number indicates order. __Second__ number (if available) indicates the feature number.

Figure 56: Visual quality metrics

FeatPMI: Pokec-5000

Average Visual Quality Metric Scores of 5 Runs

Dataset: *pokec_rel_5000_featPMI*

VQM	*1_Struct*	*2_Cfeat*	*3_feature_1*	*4_feature_2*	*5_feature_3*	*6_feature_4*	*7_feature_5*	*8_feature_6*
Banded Anti-Rob	6.82×10^{8}	7.95×10^{8}	5.19×10^{8}	4.85×10^{8}	4.23×10^{8}	2.59×10^{8}	2.69×10^{8}	3.01×10^{8}
Cor_R	6.44×10^{-2}	5.70×10^{-2}	8.46×10^{-2}	9.11×10^{-2}	7.73×10^{-2}	1.88×10^{-1}	1.68×10^{-1}	1.58×10^{-1}
Inertia	6.96×10^{13}	7.11×10^{13}	7.13×10^{13}	7.33×10^{13}	6.43×10^{13}	8.21×10^{13}	7.90×10^{13}	7.82×10^{13}
Lazy path length	1.03×10^{5}	2.56×10^{5}	6.28×10^{4}	5.98×10^{4}	6.25×10^{4}	1.70×10^{4}	1.68×10^{4}	3.45×10^{4}
Least Squares	1.03×10^{14}	1.03×10^{14}	1.03×10^{14}	1.03×10^{14}	1.03×10^{14}	1.03×10^{14}	1.03×10^{14}	1.03×10^{14}
Measure of effect	2.20×10^{7}	2.04×10^{7}	2.40×10^{7}	2.47×10^{7}	2.61×10^{7}	2.87×10^{7}	2.94×10^{7}	2.73×10^{7}
Stress Moore	6.38×10^{3}	8.74×10^{3}	6.57×10^{3}	8.55×10^{3}	8.58×10^{3}	4.24×10^{3}	5.05×10^{3}	7.48×10^{3}
Stress Neumann	2.13×10^{3}	2.95×10^{3}	2.19×10^{3}	2.86×10^{3}	2.87×10^{3}	1.42×10^{3}	1.68×10^{3}	2.51×10^{3}
Ham Path length	4.11×10^{1}	9.77×10^{1}	2.43×10^{1}	2.68×10^{1}	2.45×10^{1}	7.68	7.32	2.25×10^{1}

*__First__ number indicates order. __Second__ number (if available) indicates the feature number.

Figure 57: Visual quality metrics

FeatWalk: Pokec-5000

Average Visual Quality Metric Scores of 5 Runs

Dataset: *pokec_rel_5000_featWalk*

VQM	*1_Struct*	*2_Cfeat*	*3_feature_1*	*4_feature_2*	*5_feature_3*	*6_feature_4*	*7_feature_5*	*8_feature_6*	*9_CStruct*
Banded Anti-Rob	6.82×10^{8}	5.02×10^{8}	2.33×10^{8}	2.32×10^{8}	5.20×10^{8}	6.64×10^{8}	6.18×10^{8}	4.75×10^{8}	1.01×10^{9}
Cor_R	6.14×10^{-2}	1.31×10^{-1}	2.17×10^{-1}	2.00×10^{-1}	6.70×10^{-2}	6.44×10^{-2}	9.35×10^{-2}	3.93×10^{-2}	5.60×10^{-2}
Inertia	7.05×10^{13}	7.95×10^{13}	8.63×10^{13}	8.33×10^{13}	6.75×10^{13}	7.06×10^{13}	7.28×10^{13}	6.01×10^{13}	7.26×10^{13}
Lazy path length	1.02×10^{5}	3.37×10^{4}	2.18×10^{4}	2.19×10^{4}	3.04×10^{4}	2.39×10^{4}	2.87×10^{4}	2.14×10^{4}	4.63×10^{5}
Least Squares	1.03×10^{14}	1.03×10^{14}	1.03×10^{14}	1.03×10^{14}	1.03×10^{14}	1.03×10^{14}	1.03×10^{14}	1.03×10^{14}	1.03×10^{14}
Measure of effect	2.11×10^{7}	2.10×10^{7}	2.89×10^{7}	2.99×10^{7}	2.39×10^{7}	2.16×10^{7}	2.19×10^{7}	2.76×10^{7}	1.86×10^{7}
Stress Moore	5.73×10^{3}	5.36×10^{3}	1.03×10^{4}	1.03×10^{4}	8.92×10^{3}	6.95×10^{3}	3.95×10^{3}	6.79×10^{3}	2.31×10^{4}
Stress Neumann	1.91×10^{3}	1.79×10^{3}	3.43×10^{3}	3.43×10^{3}	2.98×10^{3}	2.32×10^{3}	1.32×10^{3}	2.27×10^{3}	7.74×10^{3}
Ham Path length	4.10×10^{1}	1.35×10^{1}	8.71	8.71	1.06×10^{1}	9.66	1.17×10^{1}	1.10×10^{1}	1.87×10^{2}

*__First__ number indicates order. __Second__ number (if available) indicates the feature number.

Figure 58: Visual quality metrics