UTRECHT UNIVERSITY

GRADUATE SCHOOL OF NATURAL SCIENCES
Department of Information and Computing Science

Human Computer Interaction master thesis

Graph Hashing for drawing Multivariate Networks (GH4D)

First examiner:

Dr. M. Behrisch

Candidate:

Naoufal A. Haddou

Second examiner:

Dr. I.R. Karnstedt-Hulpus

July 4, 2024

Abstract

Traditional graph drawing creates structures based on structural proximity. More recent works on Multivariate networks (MVNs), can visualize networks based on both structural as well as attribute proximity. Yet, in the case of such Multivariate network visualization techniques, prior knowledge on how the attributes are visualized in the graph drawing is necessary. Recent advances in the field of network embedding, dimensionality reduction graph drawing and locality sensitive hashing have inspired us to create a new multi-purpose graph drawing approach for multivariate networks. In this work we purpose a new approach named Graph Hashing for Drawing (GH4D) that makes graph drawings for multivariate networks, based on both structural as well as attribute proximity, using state of the art Graph hashing algorithm named #GNN, and dimensionality reduction technique very sparse random projection. To the best of our knowledge, this is the first work that uses graph hashing for graph drawing. In addition, we are not aware of any works that leverages network embedding, or dimensionality reduction graph drawing, to make graph drawings in which both topology and node-attributes are represented. We were able to build a working pipeline to generate graph drawings using graph hashing, and our graph drawing evaluation produced promising results in which node-attributes are better represented in the graph drawing compared to existing methodologies. In addition we identified several directions for future research.

Contents

1. Introduction

A network or graph is a data-structure that consists of nodes which can be seen as entities, and edges which indicate relationships between the entities. A multivariate network, more commonly known as a knowledge graph, can be seen as such a network data-structure where an extra layer of complexity, data is linked to each node and each edge. A *Social network*, for example, not only contains relations between actors, but also data present on each actor. For example, demographic information such as Names, Gender, Nationality. Multivariate networks are used extensively across many domains. One of which are using Multivariate networks for Biological applications [1]. In the work [2] Multivariate networks are used to visualize metabolome data in so called metabolic pathway map. Many existing strategies to visualize Omics datasets exist [3], which refers to biological structures such as genomes, proteomes, metabolomes. It states that visualization of such systems is key as human judgment and intervention are often needed. These networks are usually used to confirm existing knowledge and deliver novel correlations of potential biotechnological importance. Multivariate networks play an important role in software engineering as well [1]. Entities such as files, classes, functions, are visualized using multivariate networks to make sense of large and complex systems. Examples are UML diagrams [4], treemaps [5] and icicle plots [6]. Social network analysis aims at understanding structural properties and the associated attributes of the connected to the nodes and edges [REF]. Social media networks and friendship networks are studied to find such that user profiles can be connected to other individuals or groups. Within that domain, community detection aims to identify groups in large complex, often multivariate, network structures.

Multivariate network visualization aims to generate meaningful visualization that can be used across the many domains, like the ones previously mentioned. There have been many works on visualizing Multivariate networks networks, in particular node link layouts [1]. The main focus has been on with creating new layouts, or layouts specifically designed for specific domain.

In traditional graph drawing there are many algorithms that deal with creating graph layout based on structure, such as the family of force directed and related algorithms. In addition, there are several aesthetic metrics that tell us how good the graph drawing is. Yet, as for Multivariate visualization approaches, they are either heavily topology driven or they are attribute driven. Such that an approach that takes advantages of both, and thus does graph drawing based on both topology as well as attributes, is lacking. In addition, we are not aware of any metrics aimed at Multivariate networks that tell us how good the visualization reflects node-and edge attribute information.

In addition specific knowledge to visualize the data is needed. In most attribute driven approaches specific attributes must be selected to be used to separate or group nodes in the visualization. So the question arises, what methodology do you use if you want a good node-link visualization, in which the graph drawing rep- resents both nodal-relations as well as the node-attributes, but you do not know what part of the relations, or attributes are important?

1.1 Contributions

In this work we try to bridge the gap between MVN *topology driven layouts* and *attribute driven layouts*. While Nobres' typology makes a clear distinction between using graph structure to encode typology and using graph structure to encode attribute values, this work we consider an combination of the two. To the best of our knowledge there is no work that does graph drawing based on both attribute and topology simultaneously. The following research question naturally follows from our observations in this section:

Literature review RQ: What methodologies can we (re)use to make graph drawings for Multivariate Networks, in which the drawing represents both node-attributes as well as node-topology? This question is answered in section 2.

In this work we describe an approach to represent both topology as well as attributes in graph drawings for Multivariate networks:

- In our literature review, we classified 27 state of the art network embedding algorithms based on whether they encode topology, attributes or both.
- Inspired by recent works on Multivariate network embedding, we leverage performance wise efficient graph hashing algorithm, HashGNN, to incorporate both node attributes and structure into node embeddings. We try to maximize structural proximity as well as attribute proximity by incorporating node topology as well as node attributes into the node's high dimensional vector representation.
- Inspired by dimensionality reduction graph drawing, we make use of dimensionality reduction to make low dimensional projections from high dimensional node representations.
- And we evaluate the resulting graph drawing using state of the art graph drawing metrics, using datasets used within graph drawing community. However, from graph drawing pov, there are no metrics to measure attribute proximity. We evaluate attribute proximity using so called trustworthiness based on attribute vector encoding. We see this as a pioneering step towards evaluating graph drawing from an node-attribute point of view.

The following research questions naturally follow out of the above stated points:

- **RQ1**: How can we use graph hashing algorithm HashGNN to make Multivariate graph drawings which incorporate both node attribute information as well as node topology information?
- **RQ2**: How can we evaluate attribute proximity in a graph drawing?

Our hypothesis is that we can reuse graph Hashing algorithm HashGNN for multivariate graph drawing in which attributes are better encoded while achieving similar visual quality metrics (delta 10 percent) compared to a state of the art graph drawing algorithm. To test our hypothesis, we implemented a pipeline based on previously mentioned literature. The pipeline and implementation of our approach, Graph Hashing for Drawing (GH4D) is discussed in section 4.

1.2 Applications and Human computer interaction

We see several domains of application, where opportunities lie for our proposed approach. In Knowledge graph exploration (KGE) [7], for instance, which is a machine assisted process in which the analyst tries to understand certain aspects of the graph-data using exploratory strategies. Three goals have been specified for knowledge graph exploration. Understanding structure wise aspects of the dataset at hand, identify whether the dataset satisfies current information need or research question, retrieving the portion of the dataset in case of a vague or hard to define information need [8]. Three main tasks are defined as to reach those goals: summarization and profiling, exploratory search, and exploratory data analytics [8]. We see opportunities for our approach to be applied in KGE to get a better idea of an unknown dataset, especially in exploratory search tasks. Usually exploratory search starts with tentative queries that might lead to identifying specific parts of interest, our approach can aid the analyst by visualizing the tentative query results.

In addition, identifying highly connected nodes (or hubs) play an important role in the structure of a network, and has been extensively studied in network science [9] [10]. In human mobility network for example, hub locations play a crucial role in spread of disease [11] and economic activity [12]. In addition to topology information in such networks, other node-attribute information is usually present. For example, an analyst might be interested in identifying potential candidate (future) hubs, for example in the case of spread of disease. And might want get an overview of nodes that are attribute wise similar to the existing hub node. In that case, it might be beneficial to have drawing that incorporates both topological information, to identify the hub, together with attribute information such that attribute-wise similar nodes must be identified as well.

Data exploration in general, and KG exploration in particular is tightly coupled with Human computer interaction. Since data exploration is a process that cannot be disconnected from the specific user need [8]. User involvement is crucial in development of graph visualization techniques, for both the graph drawing itself, and the steps that the user goes trough to analyze specific graph data. The quality of graph drawings, for instance, is based on metrics that reflect Usability and readability. These metrics are validated through empirical studies of human understanding of graphs. An other important aspect within KGE are interactity and personality, where active search [13] [14] for instance, is aimed towards personalization of search results during a users exploratory search. Also, other works are able to learn interestingness from the user input and interaction with the system [15] [8]. In those cases, our multivariate network drawing approach might serve the users with a overview of network structure, which in turn aids to determine the next steps in their exploratory approach.

2. Background and Related work

This part gives an introduction to the domain of interest in this work, and the opportunities that we identified within the given domain and relevant definitions are stated and the scope of this work is further stressed. From a literature point of view, we looked at two main domains, *graph drawing*, which is further discussed in section 2.1. And the information visualization domain, and more specifically, *Multivariate network visualization* which is further discussed in section 2.2. In section 2.3 we explore the work done in the field of *Graph Representation learning* (network embedding). We observed that network embedding has been used before to make graph drawings, this is further discussed in section 2.4. An emerging technique named *Locality sensitive hashing* that has recently been used in a variety of methodologies is discussed in section 2.5. In section 2.6 we discuss an network embedding technique based on graph hashing, that uses Locality sensitive hashing to generate node embeddings based on both structural, as well as attribute proximity. And we will explain the opportunities we identified to make *graph drawings* on both structural and attribute proximity.

In this paper we consider a *simple graph*, where $G = (V, E)$ is a mathematical structure consisting of two sets, where $V = V(G)$ and $E = E(G)$. In addition, each edge has a set of two vertices associated to it, such that $y \epsilon E$ $f(y) = \{v_i, v_j\}$, where $v_{i,j} \epsilon V$.

A *multivariate network*, in short (MVN), or *knowledge graph*, $G = (V, E, A)$ consists of an underlying graph G, plus a set of n attributes $A = \{A_1, ..., A_n\}$ for the nodes and edges. For each node $v \epsilon V$, there is a column in the set of attributes associated with it, $Av = \{a_{v1}, ..., a_{vn}\}$. Similar for each edge $e \epsilon E$, there is $Ae = \{a_{e1}, \dots .a_{en}\}$.

This work is aimed towards MVN's that have somewhat a form similar to a traditional graph structure. From the layout taxonomy perspective given in work of Nobre [1], the MVN's considered in this work fall under the *Node link layout* category.

These type of networks have shown great value across several domains. From a layout perspective, the multivariate networks in this work have some sort of *on-node/on-edge encoding* (figure 2.1). In this work, we solely use colors to indicate differences between nodes and edges. There are several other ways to encode attribute values on nodes/edges. This is usually done using different colors, shapes for each attribute set, or by nodes/edge sizes. For more information we refer to the work of Nobre [1].

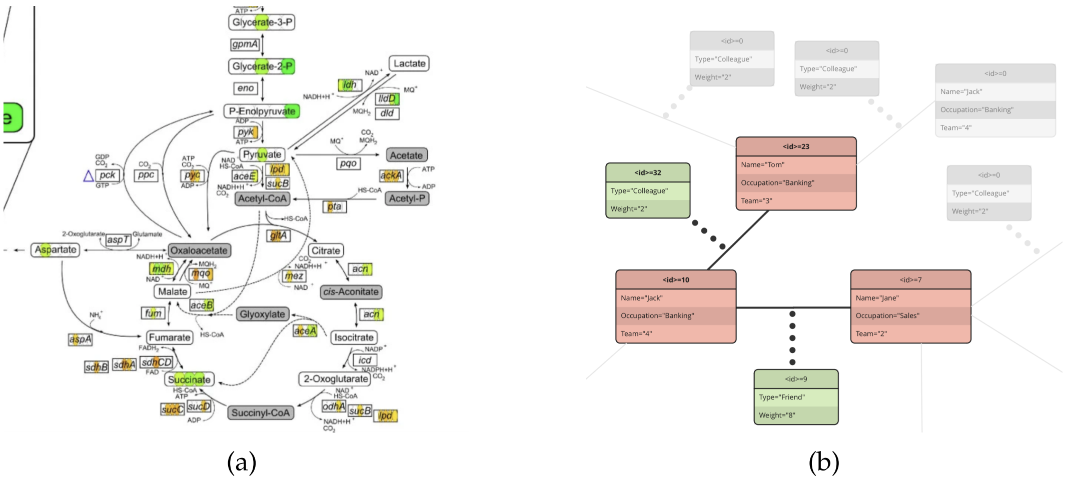


Figure 2.1: Example of on-node and on-edge encoding. (a) Multiple attributes (metabolite concentrations) are encoded directly on the nodes/edges using color and text (b) Fictional example of underlying data directly encoded on nodes (red) and edges (green).

In addition, we consider node and edge attributes that are of of *discrete* type. Discrete attributes are attributes that have values that are of a finite set such as binary. For example the attribute Gender, which can have value Male,Female, which in turn can be converted into 0,1 such that a layout algorithm can make use of it.

A *graph drawing* (or graph layout) is a spatial encoding of graph elements V and E, such that $x \epsilon V$ $f(x) = Pos(x)$. Such that each element x in set of Vertices V has a corresponding position calculated by function Pos.

We define structural proximity and attribute proximity as defined in [16]. *Structural Proximity* denotes the proximity of nodes that is evidenced by links. For nodes v_i and v_j, if there exists a link e_{ij} between them, it indicates the direct proximity; on the other hand, if v_j is within the context of u_i, it indicates the indirect proximity. So, if you make a drawing based on structural proximity, closely connected nodes should be drawn closer. *Attribute Proximity* denotes the proximity of nodes that is evidenced by attributes. The (attribute) intersection of A_i and A_j indicates the attribute proximity of v_i and v_j.

This work we are interested in making graph drawings for multivariate networks, that reflect both structural as well as attribute proximity. When we consider the domain of graph drawing, drawings are solely based on structural proximity, since that is the only information available. This is further explained in section "Graph drawing". The field of Multivariate visualization considers both structural proximity as well as attribute proximity, yet there are still opportunities. This is further explained in section "Multivariate network visualization"

2.1 Graph Drawing

Readability and usability of a given graph drawing of graph G is affected by quantitative measurements called *aesthetic criteria* (or Aesthetic Heuristics) [17]. *Graph drawing algorithms* generally deal with the ways of drawing graphs according to the set of predefined aesthetic criteria [18] and based on heuristics these algorithms try to meet each criterion.

One of the earlier works on these aesthetic critereia are stated in the work of Purchase [5], and for each a *quality metric* is defined. Metrics in this context are objective measurements which measure the extent to which a graph drawing conforms to a specific aesthetic. In this case, these metrics are continuous measurements (ranging between $0, 1$). The seven criteria given are Symmetry, edge crossings, angular resolution, edge length. Considering the seven aesthetic criteria, is it stated that it is assumed within the graph drawing community that the aesthetics improve readability.

In Bennett et al. [19] a conceptual model is provided, that gives an overview of the concepts that surround such graph aesthetic criteria (figure 2.2). The model is based on Norman's levels of processing. In short, Aesthetic Heuristics are defined based on Perceptual Principles, and are evaluated using empirical studies to understand how they affect Readability and/or Usability. Our work is mostly based inside the Aesthetic Heuristics domain. In addition, Aesthetic Heuristics are divided into four main categories: those for *Node placement*, *Edge placement*, *Overall layout*, and *Domain-specific* heuristics.

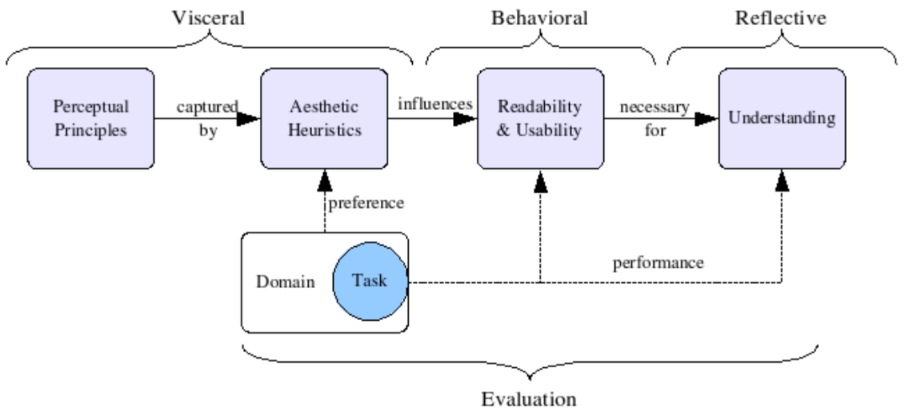


Figure 2.2: Bennett et als' [19] conceptual model of Aesthetic Heuristics based on Norman's levels of processing.

So, to create a good graph drawing, a selection is made of what aesthetic heuristics mostly reflect how well the graph is drawn, and the most appropriate algorithm is that which can maximize these heuristics. In addition, the issues of drawing good graph can be looked at from two point of views, *syntactic* and *semantic*. Semantic issues are domain specific, meaning they are influenced by the task that is to be performed on the data. While syntactic issues are not focused on tasks but on the structural aspects of a graph.

In survey Gibson et al. [20] a categorization is made of two-dimensional graph drawing algorithms. It states that most of the graph drawing techniques are from the family of force-directed and related graph drawing algorithms. This family of algorithms is based on the force directed paradigm, which applies principles of attraction and repulsion between graph elements to iterate towards an optimal graph layout. They are further categorized into three categories: force-directed, those based on dimension-reduction and computational improvements such as multi-level techniques.

Cheong et al's [21] survey provides a taxonomy of previously mentioned force directed algorithms for graph drawing. First it divides them into *classical* and *hybrid* approaches. The classical are further divided into: accumulated force models, energy function minimization models, and combinatorial optimization models. The hybrid models main focus is improving performance wise aspects of force directed algorithms. In this work we focus on the aesthetics of multivariate graph drawing methods, and we do not prioritize performance. For more details on these models, we refer to the survey. Furthermore, it is stated for each force directed algorithm what aesthetic criteria it considers.

Yet as for the domain of graph drawing, the metrics and drawing algorithms make graph drawings solely based on structural proximity, nodes that are closely connected are drawn together. To find proper techniques that draw on both structural as well as attribute proximity, we had to go to the information visualization world. There are works on so called "Multivariate network visualization", which is discussed on the next part.

2.2 Multivariate Network visualization

The work of Nobre is a state of the art report on Multivariate network visualization techniques. As for the type of networks that we are interested in, Multivariate networks that have a node link layout, the typology makes a clear distinction between so called attribute driven layouts and topology driven layouts. As the name suggests *attribute driven layouts* are layouts where node positioning is determined solely based on node attribute value. As for *topology driven layouts*, the node positioning is solely based on structural proximity.

Attribute driven layouts are further divided into attribute-driven faceting and attribute-driven positioning. Attribute-driven positioning, places nodes exactly based by node attribute value. Works where nodes are placed based on geographical coordinates [22] [23] use this type of encoding. In attribute driven faceting, the nodes are placed in regions corresponding to a categorical attribute, but the exact node position within that region is determined in another way. Work [24] defines Semantic Substrates in which a categorical attribute is selected which is used to layout nodes based on category, and the analyst can further specify to show links within or between categories.

Gibson et al's survey offers Multivariate network graph drawing methods. Gibson et al. categorizes *Multivariate graph drawing methods* into three groups: constraint based graph layouts, clustering based graph layouts, mapping attributes to 2D space. First, the so called *constraint-based graph layouts* try to incorporate node attributes into graph drawings using methods similar to previously mentioned. Most of them use some type of force directed algorithm, together with some Aesthetic Heuristics to draw the graph layout. In addition they use constraint-based techniques to impose specific predefined set of constraints on a selection of the nodes. These constraints can be user defined based on specific knowledge, or might be derived from the attributes that the MVN encodes. The work [25] uses Stress Majorization as a constraint.

Secondly there are the *clustering-based graph layouts*, where MVN graph drawings are determined based on the clustering of nodes. These methods mostly use a clustering-algorithm, which determines how to cluster based on node attributes, and some sort of *space filling* algorithms, which determines how to make use of the 2D space efficiently. For these clustering methods, it is said that users completely neglect edge crossing (which is an Aesthetic Heuristic) in favor of clustering the graph [26]. That why there is usually less emphasis on the so called edge crossings aesthetic in this category of MVN drawing algorithms. An example is shown in figure 2.3

Lastly there is the *mapping attributes to 2D space* group, which directly maps nodes into a 2D space using node attributes. These methods use some type of mapping, for example direct mapping or dimension reduction, to calculate coordinates for each node in the graph. There are also hybrid approaches that combine the above. EdgeMaps by Dork et al. [28] and Gibson and Faith's [27] both use a combination of force directed combined with

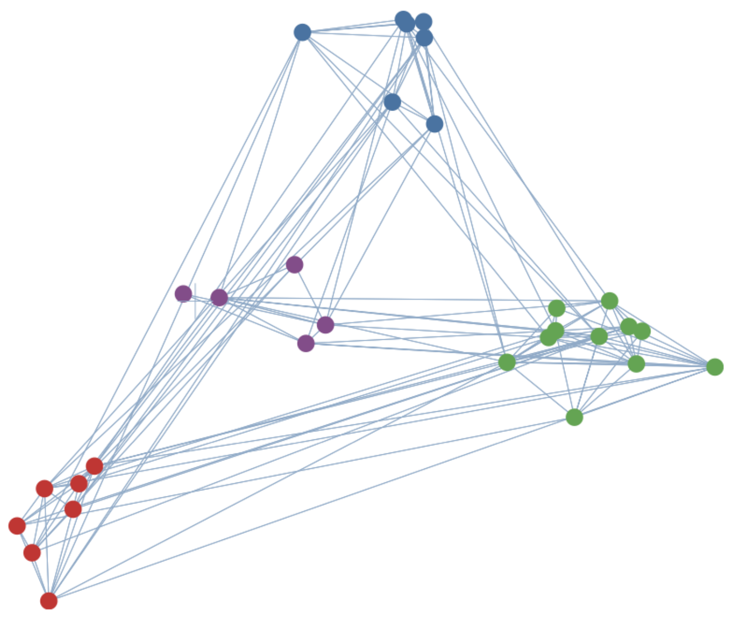


Figure 2.3: Clustering based layout created to visualize small world networks from the work of [27], that uses color on-node encoding.

mapping attributes to 2D space.

While each of these Multivariate graph drawing methods can be of great value for specific use cases, you need to have knowledge of the data in order to visualize the data. Specific attributes must be selected to separate or group nodes. So the question arises, what methodology do you use if you want a good node-link visualization, in which the graph drawing represents both nodal-relations as well as the node-attributes, but you do not know what part of the relations, or attributes are important?

This question is answered in the following section.

2.3 Graph Representation Learning

When we consider literature in the field of creating data-structure that represents both topology and attribute, there is a bit of a shift of domains from Graph Drawing towards Machine Learning. So called *Graph representation learning* or *Network Embedding* aims to map each node to a vector where the distance characteristics among nodes is preserved. Such that for a given (multivariate) graph $G = (V, E, A)$ we can define a function that maps each

Vertex v_i to an embedding x_i:

$f : v_i \rightarrow x_i \epsilon R_d$

Where each $x_i = \{x_1, x_2, ..., x_d\}$ is a vector of length d. To derive such a vector, graphs are usually represented by an adjacency matrix or a derived vector space representation [29]. In the case of a multivariate graph, the vector is derived from both node topology, as well as attributes.

While there are similarities between Multivariate network embedding and Multivariate network drawing the goals are different, in network embedding the focus on creating predictive models such as neural networks, or node classification. To the best of our knowledge, network embedding has never been used to make drawings based on both topology as well as attributes

We have considered several surveys on graph representation learning, Chen [30], Ju [31] and Cui [32] to get an overview of existing network embedding methods that might be suitable for MVN embedding.

The survey of Chen [30] gives an overview of several of such Network embedding methods, and it mentions several works that embed Multivariate network. They refer to MVN as "Graphs with auxiliary information" [33], meaning *"those that have labels, attributes, node features"*. It is said that nodes with different attribute information (such as labels) should be embedded further away in the embedding space than those with the same attribute information.

The survey of Chen makes a distinction between Classical network embedding methods and Emerging network embedding methods. As for the *classical network embedding methods* are mostly directed towards embedding simple graph, without node attributes. Examples are DeepWalk [34] and Node2Vec [35], which derive vectors from nodes based on random walks to reserve nodal proximity's and preserve neighborhood.

Neural network based methods

A trend can be seen in the *emerging network embedding methods* to incorporate both topology as well as attributes. In particular the *Neural network* based methods. *The Graph Neural Network* (GNN) framework [36] is a collection of such neural network based methods, that use message passing scheme paired with neural networks to learn node embeddings. There are several algorithms based on the GNN framework that can learn embeddings based on both topology as well as attribute information. SEAL [37], and P-GNN [38] are examples of such algorithms, SEAL extracts subgraphs based kn K-hops to do efficient link prediction. P-GNN is similar, it uses layers to map node positions with relation to all other nodes in the network, which is more effective performance wise. An other particularly interesting algorithm is the Weisfeiler-Lehman Kernel Neural Network (WLKNN) [39] which will be discussed in more detail in section "Locality sensitive hashing". Graph Convolutional Network (GCN) [40], uses convolution operator to iteratively aggregate nodes neighbor embeddings. And GraphSage [41], which uses aggregate function to classify nodes.

The survey of Cui [32] mentions more neural network based methods that can learn both topology as well as attributes (figure 2.4), in the paper so called "Network Embedding with Side Information". In [42] a so called Tri-party deep network representation model is presented, TriDNR, which uses a coupled neural network architecture to embed topology and attributes. It learns a low-dimensional vector for each nodes such that similar nodes are close in the representation space. The work of [43] similarly learns embeddings based on topology and attributes. It proposes a deep architecture by separately learning label embeddings using Doc2Vec and topology embedding using DeepWalk, and linearly combines them together. In [44] an embedding is learned that preserves first and second order proximity's while also incorporating node attributes.

Graph Hashing

One other emerging field besides neural network based methods we identified that was not mentioned by any of the surveys mentioned is *Graph Hashing*. Hashing techniques have been used to approximate similarities between high dimensional data. *Learning to hash functions* such as Spectral hashing [45] and semantic hashing [46] learn data specific functions. While *Randomized hashing* functions such as MinHash represent data as hash

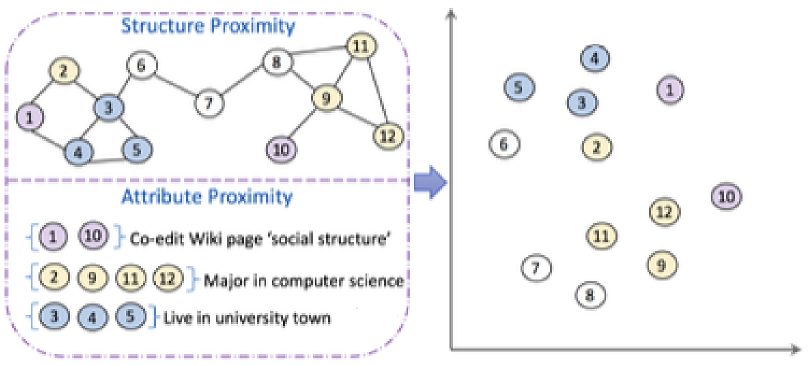
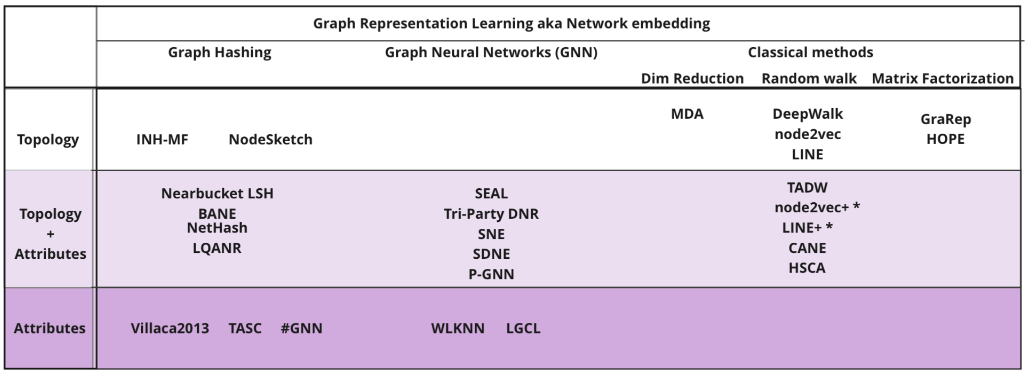

Structure Proximity
Attribute Proximity
Co-edit Wiki page 'social structure'
Major in computer science
Live in university town

Figure 2.4: This image from the work of Liao [32], shows how their proposed network embedding algorithm encodes both structural as well as attribute proximity

codes and uses randomized hash functions for classification. Besides capturing similarities, algorithms have been developed that can create node embeddings on both topology as well as attribute information. Binarized Attributed Network Embedding (BANE) [47] and w-Bit Quantization for Attributed Network Representation Learning (LQANR) [48] are examples of such algorithms that use such learning to hashing to acquire node embeddings. To learn the embeddings, they make use of hash codes based on both topology as well as node attributes. However, they are usually expensive in either time or space because they contain massive matrix factorization operations. NetHash [49] on the other hand uses randomized hashing to acquire embeddings, which is more efficient time and space performance wise. We made an overview of algorithms we considered, classified by whether the network embedding algorithm embeds topology, attributes or both. In figure 2.5 a total of 27 network embedding algorithms are categorized. They are further categorized based on the three domains mentioned, classical methods, neural network based methods, and graph hashing.

Current network embedding methods' focus lies on creating predictive models such as neural networks, or node classification, while we think that they might serve a purpose for graph drawing as well. However, because of the multi-dimensionality of the embeddings generated by these methods, it is not possible to map the data directly into 2d space for graph drawing purposes.



	Graph Representation Learning aka Network embedding				
	Graph Hashing	Graph Neural Networks (GNN)	Classical methods		
			Dim Reduction	Random walk	Matrix Factorization
Topology	INH-MF NodeSketch		MDA	DeepWalk node2vec LINE	GraRep HOPE
Topology + Attributes	Nearbucket LSH BANE NetHash LQANR	SEAL Tri-Party DNR SNE SDNE P-GNN		TADW node2vec+ * LINE+ * CANE HSCA	
Attributes	Villaca2013 TASC #GNN	WLKNN LGCL			

Figure 2.5: 27 network embedding algorithms are categorized based on whether they learn embeddings based on topology, attributes or both. They are further categorized into 3 groups: Graph Hashing, Neural Network based, and Classical methods.

Nonetheless, from a graph drawing pov Graph embedding might serve as an intermediate step towards generating a graph drawing based on learned embeddings that incorporate both attributes as well as topology. A technique called *dimensionality reduction* exists that maps high dimensional vectors into low (2d) dimensional representation that can be drawn. This idea, using network embeddings for graph drawing, is not new. Little work has been done on using network embedding for graph drawing. Yet, to the best our our knowledge, we are not aware of any works that use such embeddings to incorporate node attributes. In the next section we will discuss these works in more detail.

2.4 Network embedding for graph drawing

Network embedding has been used for graph drawing purposes. We did find some works where network embedding has been used, we did not find any works that encode both node topology and attributes into the embedding. While we have seen in the previous section that there are several algorithms that can generate such vector based node embeddings for Multivariate data.

Dimensionality reduction graph drawing

Before we mention works that use network embedding for graph drawing, we want to bring to the attention that the embeddings are usually multidimensional vectors that cannot be directly mapped into 2d space. The domain of *dimensionality reduction graph drawing* deals with reducing the dimensionality of such high-dimensional representations. Dimensionality reduction based methods, such as Principal component analysis (PCA) [50], turn a high dimensional representation of a graph into a low dimension while trying to maintain distances. t-SNE [51] measures similarities in high and low dimensions concurrently, and several functions are used to calculate low dimension representation of the high dimension data-structure.

Dimensionality reduction graph drawing take advantage of these techniques which provide increasingly more accurate, flexible, scalable. Tsnet [52] is such a DR graph drawing method developed based on a modification t-SNE method. It represents a graph with a shortest path adjacency matrix, which is used to compute the pair-wise distance between node pairs and node similarity with conditional probability. DRGRaph [53] is an other work that uses a sparse graph adjacency matrix to simplify the computation of distance matrix between the node pairs which reduces the computational complexity. Other works on Dimensionality reduction graph drawings are the works of Plant [54], which is more focused on performance, and is aimed towards machines equipped Multi-core CPU to improve graph drawing performance.

An other work that uses such dimensionality reduction for graph drawining is the work of Elzen [55]. Which describes an simple, yet effective, approach that leverages dimensionality reduction graph drawing for *dynamic network visualization*. It makes use of vectorization of complete networks and dimensionality reduction in order to visualize all dynamic network instances into 2d space. Such that snapshots where networks are similar are positioned closer to each other and clusters of points indicate stable or recurring network states. Then it uses a juxtaposed view where in one screen each of such network instance is represented as a point in a 2d visualization (node-link graph), and in the other screen the corresponding network is show the selected node in the first screen.

Network embedding graph drawing

Network embedding for graph drawing can be seen as adding an other layer of complexity to DR graph drawing. Instead of using the adjacency matrix representation to represent a graph, node embeddings are used. Deep Neural Network for DrawiNg Networks (DNN)2 [56] for example, uses Graph Convolution Networks to learn the embeddings, and leverages dimensionality reduction t-SNE and dimensionality reduction graph drawing tsNET to make graph drawings. It states that DNN^2 performs well compared to these algorithms despite some Deep Learning related limitations. The results highly suggest that Deep Learning is a promising approach for the future of graph drawing. An other work is GraphTSNE [57] which learns a shallow Neural Network made of Graph Convolutions to predict a graph layout. In Bohm [58] an graph embedding algorithms for graph drawing GEMPE is purposed, which focuses more on parallelization.

But as previously mentioned, none of the previous works has leveraged the potential to encodes attributes into the embedding for graph drawing. While we have seen that there are several algorithms that can learn such embeddings. In addition, we have seen that neural networks based network embedding techniques have been used for traditional graph drawing, we are not aware of any works that use graph hashing for graph drawing, while they might be more suitable from a performance wise perspective.

In the next section we want to deeper dive into graph hashing, and look at *Locality sensitive hashing*. It has been used in several graph related tasks such as k-nearest neighbor search. Interestingly for our use case, it has been used for node embedding, and we will point out which graph hashing methodology we will use in this work.

2.5 Locality sensitive hashing

Locality Sensitive Hashing (LSH) is a widespread randomized method that tackles the efficiency challenge for similarity search in high dimensional spaces [59] [60]. Hash values of similar nodes collide with high probability. It maps a vector in high dimensional space into representation in low dimensional space, such that probability that two vectors collide equals their similarity under the given measure. Thus, similar to neural network based embedding, yet it is able to do this in sub-linear time. Kraus NearBucketLSH [60] for example uses Locality sensitive hashing and is based on Gionis et als' work [61] which partitions data vectors into buckets. Yet, Kraus his version does efficient similarity search to the near buckets. They introduce a cached version that is more efficient performance wise. Simply put, nodes are converted into vectors, vectors are hashed using LSH into buckets, and buckets are used to retrieve results from queries, and do similarity search between buckets. Other similar works [62], [63], [64] we found in the survey of Jafari [65] on Locality sensitive hashing that use Vector space models to represent the nodes, and then use hashing to index all nodes.

Libraries such as FAISS [66] by Facebook and SCANN [23] by Google both exploit LSH for k-Nearest neighbor (similarity) search. SCANN and FAISS are mostly focused on a new quantization approach. Several techniques have been proposed in the literature based on hashing, graph search, or quantization to solve the approximate maximum inner product search problem efficiently, and the quantization based techniques have shown strong performance. Simply put, SCANN does maximum inner product search (MIPS), by reducing the number of items that are scored to identify the top result and improving the rate at which items are scored. User queries and documents are embedded into a dense vector space of the same dimensionality and MIPS is used to find the most relevant documents given a user query. LSH is used as a fast method of space partitioning. SCANN supports various vector representations: Bag of words, embeddings, feature vectors. SCANN indexes them using LSH, which groups similar vectors into the same bucket with high probability. It uses a so called LSH index, which is a fast way to identify promising buckets for nearest neighbours. In the refinement stage it uses optimal transport to refine results. Which is a mathematical framework for measuring the distance between probability distributions. Which allows to measure distance between query vector and bucket more efficiently.

FAISS on the other hand uses Product Quantization lookup tables based on IVFADC. At its core, the IVFADC requires computing the distance from a vector to a set of product quantization reproduction values. This means, do a search based on "words per vector". So a vector is decomposed into words, and when a query vector is given, its checks what words it consists of and can then lookup similar vectors (that are made up of the same words).

2.6 HashGNN

The recent advances in LSH, such as works as FAISS and SCANN that leveraged hashing techniques for efficient K-NN search, inspired us to look into LSH based Graph hashing techniques. Graph hashing algorithm HashGNN (#GNN) [67] stood out in particular. HashGNN is a node embedding algorithm that resembles Graph Neural Networks (GNN) but does not require training. HashGNN is based on Weisfeiler-Lehman Kernel Neural Network (WLKNN) [39], but in HashGNN the neural networks are replaced by random hash functions, in favor of the min-hash locality sensitive hashing. Thus, it can be said that HashGNN combines ideas of GNNs and fast randomized algorithms.

The *Weisfeiler-Lehman Kernel Neural Network* (WLKNN) is a message passing neural network. It is based around the principle that nodes pass their own embedding to all their neighbors, so in the end the embedding not only contains information on the node itself, but also information from neighboring nodes. This is done for several iterations such that, e.g., the second iteration messages are passed again to each nodes neighbors, which will cover nodes at 2 hops distance. And after k iterations we have information from nodes that are k hops away. Normal WLKNN uses 3 neural networks to learn network structure. HashGNN, which combines WLKNN with Minhash (which is a type of LSH), uses 3 hashing schemes instead of neural networks to learn network structure. Minhash [68] approximates the Jaccard similarity of two sets without comparing them directly. Instead, it uses random hash functions that use hyperplane rounding to classify nodes. For Minshash it is shown that the signature numbers of the random hash functions collide with with the probability of each occurred similarity of the input sets.

Technical explanation of HashGNN:
Given an Multivariate network $G = (V, E, A)$, the number of iterations T, the size of node representation K, and three arrays of randomized hash functions, suppose we are at the t-th iteration and the k-th hashing process. The following is from the paper to get an embedding for node v with binary vector representation x_v:

$$x_{v,1}^{(t,k)} \leftarrow argmin\big(\pi_3^{(t,k)}(x_v^{(t-1)})\big) \tag{2.1}$$

Line 4: [67]: Randomized Hash Function 3 $\pi_3^{(t,k)}$ is used to generate hashed vector $x_{v,1}$, using Minhash which randomly selects K features (1's) from initial vector x_v.

$$x_{v,neighbors}^{(t,k)} \leftarrow \bigcup_{u \epsilon N(v)} \{x_{u,1}^{(t,k)}\} \tag{2.2}$$

Line 7: Hashed vectors from all neighbors are combined into $x_{v,neighbors}$, from each hashed neighbor representation $x_{u,1}$ where u is in the set of neighbors of node v $N(v)$.

$$x_v^t[k] \leftarrow argmin\big(\pi_1^{(t,k)}(x_v^{(t-1)}) \bigcup \pi_2^{(t,k)}(x_{v,neighbors}^{(t,k)})\big) \tag{2.3}$$

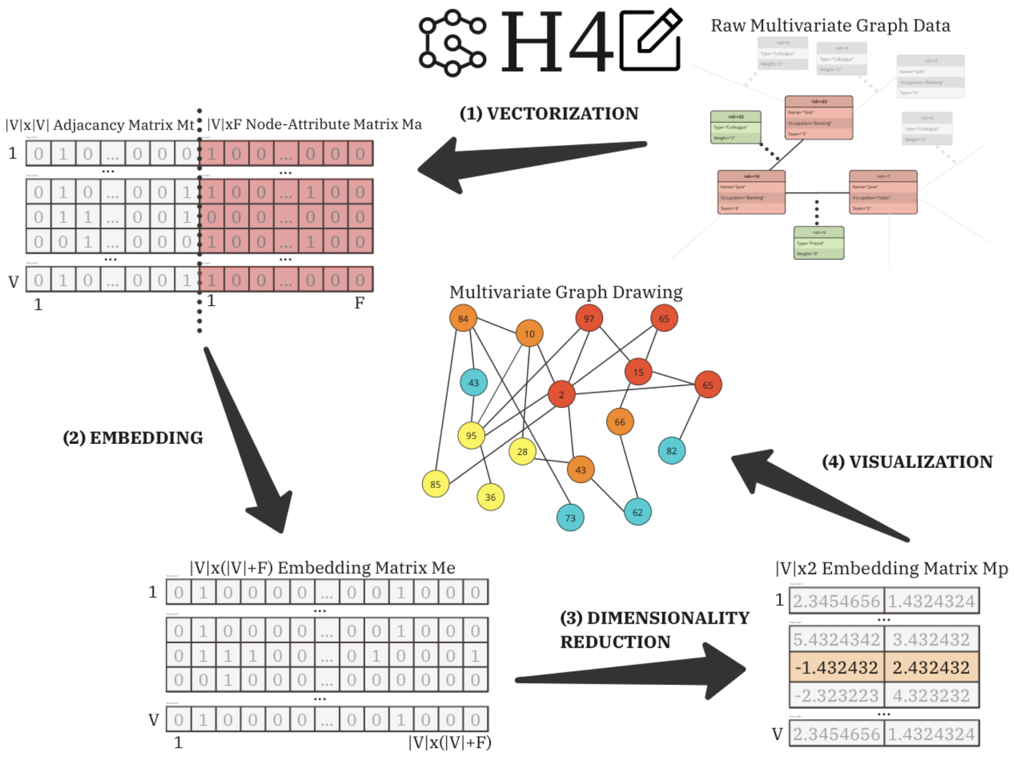
Line 8: Apply Randomized Hash Function 2 $\pi_2^{(t,k)}$, to get hashed neighbor vector (right part of the union $\bigcup$). Hash x_v with Randomized Hash Function 1 $\pi_1^{(t,k)}$, and combine both hashed vectors, to get final vector $x_v^t[k]$. Vector $x_v^t[k]$ is used as input for next iteration.

HashGNN has shown to be just as accurate as neural network based methods, yet much more efficient performance wise [67].

In this work we describe an approach to represent both topology as well as attributes in graph drawings for Multivariate networks. We implemented a pipeline based on previously mentioned literature. The pipeline and implementation of our approach, Graph Hashing for Drawing (GH4D) is discussed in chapter 4.

3. Graph Hashing for Drawing (GH4D)

We followed similar steps in our approach as the work of Elzen [55]. Our approach is also similar to the pipeline of other approaches of dimensionality reduction graph drawing [52]. We start with a high dimensional representation, which is altered by graph hashing, and in the end the dimensionality is reduced to 2 dimensions, and the resulting projection is visualized. Our approach consists of 4 steps: *Vectorization*, *Embedding*, *dimensionality reduction*, *visualization*, which are visualized in figure 3.1. The following section discussed each step in detail.



H4
Raw Multivariate Graph Data
(1) VECTORIZATION
|V|x|V| Adjacency Matrix Mt
|V|xF Node-Attribute Matrix Ma
(2) EMBEDDING
Multivariate Graph Drawing
(4) VISUALIZATION
|V|x(|V|+F) Embedding Matrix Me
(3) DIMENSIONALITY REDUCTION
|V|x2 Embedding Matrix Mp
2.3454656 1.4324324
5.4324342 3.432432
-1.432432 2.432432
-2.323223 4.323232
2.3454656 1.4324324

Figure 3.1: Graph Hashing for Drawing pipeline visualized: (1) Raw graph data is converted into a node-vector matrix (2) Which is embedding using HashGNN, (3) then the dimensionality is reduced to two dimensions (4) such that is can be visualized as a graph drawing.

3.1 Implementation

Implementation is made in Python 3 and uses standard library packages such as NetworkX [69], NumPy [70], and SciKit [71], SciPy[72]. In addition, the Graph Data Science (GDS) library published by Neo4j to use HashGNN.

3.1.1 Vectorization

Given an Multivariate network $G = (V, E, A)$, in the vectorization step, we define for each node in V a binary vector x_v, such that it can be used by the embedding algorithm in the next step. We assume that the data is valid and does not contain nonzero values. Each vector x_v consists of two parts, a *topology encoding* and a *attribute encoding*. To get the topology encoding, the graph is represented by $|V|x|V|$ adjacency matrix M_t. Such that, in the adjacency matrix, each row vector of length $1x|V|$ represents the topology encoding of a node. In addition, in the case of unweighted and unattributed edges, the above described topology encoding is enriched by edge-attributes. For integer values present in the adjacency matrix (edge weights), the values present in the adjacency matrix are binarized using boolean thresholding strategy.

To get the node-attribute encoding, for each node the attributes are first binarized and then concatenated. For the vectorization of node-attributes, we follow the same procedure used in these works [16] [42]. Categorical variables are converted to a set of binary features via one-hot encoding 3.2. For example, the gender attribute such as $Male, Female$, a node with has attribute values "female" is concerted into $v = \{0, 1\}$ where the second binary feature of value 1 denotes “female". Such that we end up with a $|V|xF$ matrix M_a which contains node-attribute vectors, where F denotes the length of the binary attribute encoding.

Following the same procedure as in [16] [42], the matrix M_t (topology) and M_a (attribute) are concatenated 3.3. Such that we end up with matrix M $|V|x(|V|xF))$, where row u of length $1x(|V| + F)$ represents node v_u as a binary vector encoding containing both attribute as well as topology infor-

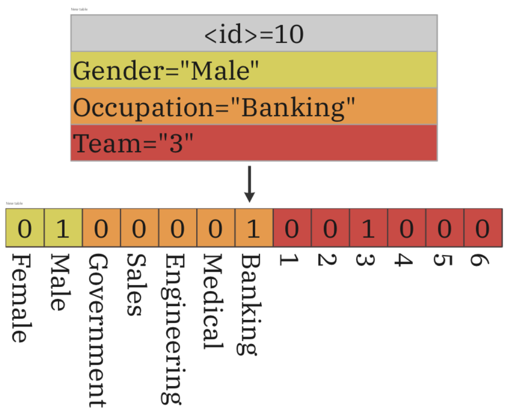


Figure 3.2: One-hot encoding process visualized: Given a node that is of gender "Male", has occupation "Banking", and is from Team "3" (top table), the node-attribute data is converted in a binary node-attribute vector (bottom).

mation.

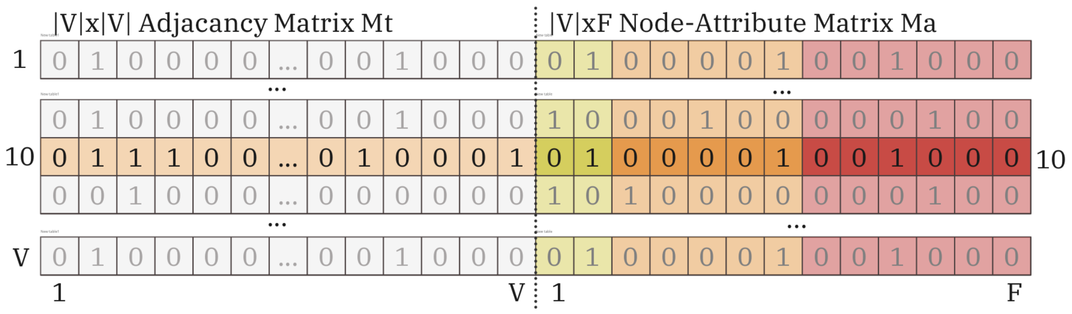


Figure 3.3: Concatentation of matrix M_t (topology) and M_a (attribute), to get Matrix M.

3.1.2 Embedding

Given an Multivariate network $G = (V, E, A)$, and a matrix M contains binary node vector encodings, the embedding is calculated using HashGNN. HashGNN is executed in Python using the Graph Data Science library published by Neo4j. To run HashGNN, first, an in memory graph of G is constructed in the Neo4j database, of which, for each node, its binary vector from M is attached to the in memory graph. Using the Graph Data Science python module, a Cypher query can be run in script which takes the in memory graph as an argument, runs the HashGNN algorithm, and returns the embedding matrix M_e. How individual node vectors are influenced by their neighbors is visualized in figure 3.4. The embedding matrix is of the same dimension as the input matrix M, and contains the embedding for each node as node vector encodings.

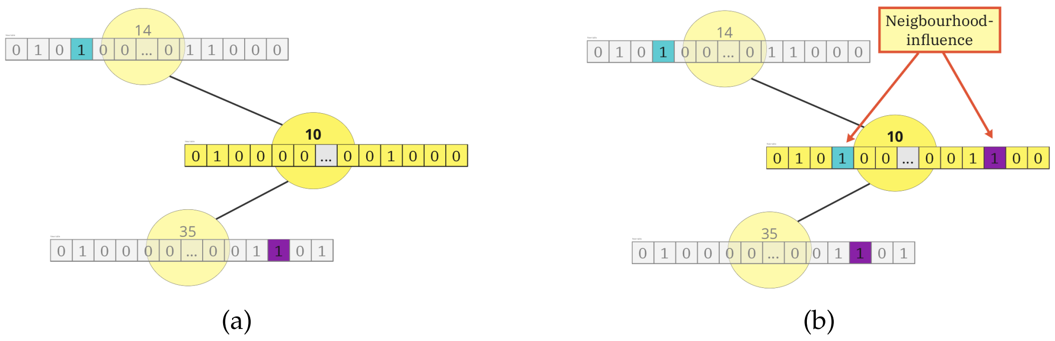


Figure 3.4: HashGNN embedding procedure. (a) Pre-embedding: node 10's vector representation, and its neighbors node 14 and 35, from matrix M (b) Post-embedding: we can see the neighbour-influence of 14 and 35 in cyan and purple. These embeddings make up embedding-matrix M_e

Parameters

There are four parameters that each influence the produced embedding by HashGNN. First, the *randomSeed* parameter influences the hash functions are used inside the algorithm. This affects which features are sampled each iteration. The parameter neighborInfluence determines how prone the algorithm is to select neighbors' attributes over features from the same node. The value of neighborInfluence is 1.0 by default, in which on average a attribute will be selected from the neighbors 50% of the time. And increasing the value leads to neighbors being selected more often. The parameter iterations sets the iterations of HashGNN, which is the maximum number of hops between a node and other nodes that affect its embedding. The embeddingDensity parameter denotes the size of the node representation as discussed in the previous section, where it was named k in the equation.

3.1.3 Dimensionality Reduction

Given the high dimensional embedding matrix M_e, we reduce the dimensionality to 2 dimensions using very sparse random projection (VSRP), using the implementation FastRP. Fast Random Projection, or FastRP for short, is a node embedding algorithm in the family of random projection algorithms. [73]. FastRP uses very sparse random projection, which reduces the dimensionality while preserving pairwise distances between the nodes using with strong theoretical guarantees [74]. Very parse random projection [75] is an improvement of Gaussian projection. In Gaussian projection the high dimensional matrix M is reduced by constructing a random projection matrix R of given lower dimension d. Then, both matrices M and R are multiplied: $N = MR$. If the entries of R are Independent and identically distributed with zero mean, N is said to preserve the pairwise distances. Very parse random projection speeds the process of finding R, by replacing the entries with probabilities. This gives a computational performance benefit over other methods such as Skip-gram. In the end, using VSRP on embedding matrix M_e, we end up matrix M_p of size $|V|x2$, which contains for each node in V a 2 dimensional vector 3.5. One thing we noticed is, the low dimensional embedding of VSRP has a grid like structure, in which similar nodes are given the exact same embedding. This is similar to the result of other Locality sensitive hashing methods, such as NearBucket-LSH, in which similar vectors are placed into similar buckets.

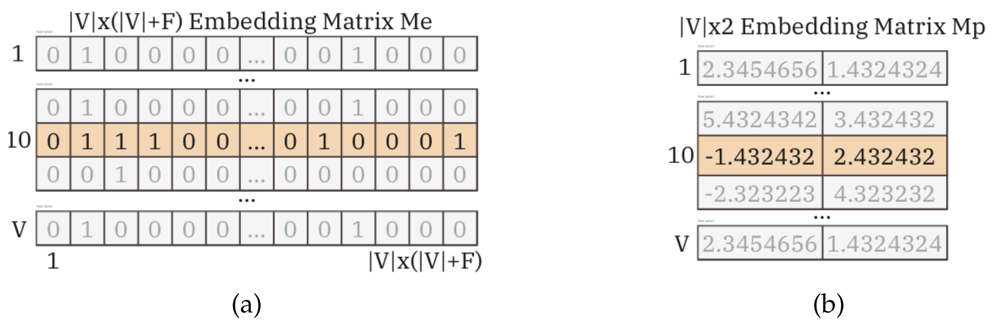


Figure 3.5: Dimensionality reduction procedure, embedding of fictional node 10 in orange. (a) The embedding matrix M_e is visualized, which contains vector embeddings produced by HashGNN, with the (b) After very sparse dimensionality reduction is applied, we end up with matrix M_p, that contains the two dimensional mapping of matrix M_e

3.1.4 Visualization

Given an Multivariate network $G = (V, E, A)$, and matrix M_p, we make graph drawing of G using the 2 dimensional node representations in M_p as 2 dimensional coordinates in the 2D space. The graph drawing is made using Networkx, we first construct a Networkx in memory graph of G, and for the drawing, M_p is fed to Networkx as positional arguments for the nodes in G. For the drawing, we use on-node and on-edge color encoding, such that one is able to distinguish nodes and edges with similar attributes visually, and the drawings can be evaluated visually As mentioned before, in VSRP similar nodes are given the exact same embedding. From a graph drawing point of view however, this creates node overlap, and such nodes are not visible in a 2d representation. To make the visualization more informative, nodes that belong to the same point in the grid are randomly scattered around the grid-point.

3.2 Evaluation

In this work, we do a *quantitative individual evaluation* supplemented by a *visual comparison* in order to validate the performance and applicability of our Multivariate network layout approach. Works that use a similar approach to their evaluation are [76] [77] [78]. We follow the procedure that is usually done in the graph drawing community, we evaluate our approach using datasets frequently used by the graph drawing community, using certain quantitative metrics against a benchmark graph drawing algorithm.

In this evaluation, we want to show how well our approach is able to map both structural as well as attribute proximity. We do this by increasing the complexity of the network step by step. Starting with a network that does not contain any node or edge attributes (Karate club dataset). After which, we evaluate our approach on a weighted network (les miserables dataset). Then we evaluate on a network that has categorical node attributes, but is unweighted (Football dataset). And finally, we evaluate our approach on a dataset that has both edge weights, as well as categorical node attributes (Hollywood Film Music dataset). In addition to the quantitative evaluation, we do a visual evaluation on the Multivariate network, and examine how node-attributes influence the drawing produced by GH4D.

3.2.1 Datasets

- **Zackary Karate club** dataset [79] consists of 34 nodes and 78 edges. The data was collected over two years by Wayne Zachary in the 1970s. The nodes in the data represents members of a karate club at an American university, and the edges represent their social interactions. The graph has no node or edge attributes and is un-directed.
- **Les Misérables** dataset [80] contains an weighted which consists 77 nodes and 254 edges. The nodes in the dataset represent characters in Victor Hugo's novel 'Les Misérables', and the edges represent that these two characters appeared in the same chapter of the the book. The data contains weighted edges, in which the weights indicate how often such a co-appearance occurred.
- **Football** dataset [81] contains an attributed network which consists of 115 nodes and 616 edges. The network represents the schedule of Division I games for the 2000 season. Nodes represent college football teams, and edges represent played games between the teams. All 115 teams were divided into conferences, which can be seen as communities, since games played happened more frequently with teams inside the same conference. The data contains labeled nodes, which indicates to which conference the team belongs.
- **Hollywood Film Music** dataset [82] contains an attributed network which consists of 102 nodes and 193 edges. The network represents collaboration of 40 composers of film scores and 62 producers, who produced at least 5 movies in Hollywood. The nodes represents these composers/producers, and the edges exists between composers and producers and represent that the composer produced a film score for the producer. Edges are weighted, the weight indicates the number of movies the composer created scores for the respective producer. There are two node attributes, one indicates whether the node is a composer or a producer, the other whether the composer belongs to the top composers, which have earned 1.5% or more of the total income of Hollywood movie score composers.

Name	|V|	|E|	type
Karate	34	78	binary
Les Misérables	77	254	weighted edges
Football	115	616	categorical nodes
Hollywood Film Music	102	193	weighted edges, categorical nodes

Table 3.1: 4 datasets, from graph drawing community

3.2.2 Algorithms

We compare our approach against two algorithms Fruchterman Reingold algorithm and Random node placement algorithm:

- **Fruchterman Reingold:** [83] (also known as Spring) is a graph drawing algorithm from the family of force-directed and related graph drawing algorithms. This family of algorithms is based on the force directed paradigm, which applies principles of attraction and repulsion between graph elements to iterate towards an optimal graph layout.
- **Random algorithm:** we compare our approach to a random algorithm, in which no node placement strategy is used, and nodes are placed randomly inside the 2D space.

We selected the FR algorithm because it is well known and established within the graph drawing community. And because of that, we were able to find an working implementation within the Networkx Python library, that we could use. And, as previously mentioned, there is currently no algorithm similar to our algorithm, in which both topology as well as attributes are considered in the graph drawing process.

Because of the performance wise efficiency our Graph Hashing algorithm, we also considered using performance wise improved versions of the Fruchterman Reingold algorithm. The *multilevel Fruchterman Reinold* (FR) algorithm [84] and the *parallel FR algorithm* based on openCL [85] for instance, improve the performance of force-directed algorithms and reduce execution time, enabling the algorithms to visualise large and complex networks in an efficient manner. However, we did not find an implementation that we could use, and it was not feasible to make such an implementation within the scope of this work.

3.2.3 Metrics

We selected edge-crossings, Angular Resolution, Edge length to measure Aesthetic wise aspects of the graph drawing. We use the same definition as in [86] [87].

- **Edge Crossings:** the *Edge Crossings* aesthetic metrics counts the number of pairwise edge crossings, where an edge crossing is defined as a point where two edges intersect. The Angular resolution metric
- **Angular Resolution:** The *Angular Resolution* aesthetic metrics calculates the angular deviation of each edge and the ideal angle (based on node degree), and then the mean is taken over all nodes.
- **Edge Length:** The *Edge length* aesthetic metric calculates the mean edge length of all edges (which is said to be the ideal edge length), and then the mean deviation of all edges from this ideal is calculated.
- **Neighbourhood preservation:** This metric measures how well structure is retained as in, nodes that are close in terms of graph-theoretic distance should be near each other in the layout. The Neighbourhood Preservation uses the Jaccard-similarity index to find both the theoretic neighbors of a node in the graph, and the nearest neighbors in the graph drawing, and calculates what fragment of neighbors in the drawing are graph-theoretic neighbours.

The metrics we use are all normalised, their values lie between 0 and 1, where 1 represents the best preferable outcome, and 0 the least, the same way as defined in [87]. For example, in the case of edge crossings, the amount of edge crossings is divided by the maximum amount of edge crossings, and the outcome is deducted from 1. Such that, with a lower amount of edge crossings, the metrics is closer to 1.

As for metrics that measure how well structure is retained in the graph drawing, we use the metric neighborhood preservation, as defined in [86]

[87]. While neighbourhood preservation works well to indicate how well structural proximity is maintained in the graph, it does not tell us anything of attribute proximity. We did not find any metrics within in the graph drawing community that measure how well the node-attribute similarity is retained in the graph drawing. In other words, nodes with similar attributes are closer in the embedding space. We did consider works on attribute driven layouting, but did not find such a metric.

The closest metric we found was a metric used in information visualization as a quality metric for projections named *trustworthiness* [88]. Trustworthiness measures to what extent the local structure is retained when projecting data of high dimension into low dimension. Square distance matrices are used for both the high dimensional as well as the low dimensional representation, and given k neighbors, and similar to the neighborhood preservation metrics, it measures to what extent neighbors in high dimension are neighbors in low dimension.

Driven by the fact that there no graph drawing metrics that measure how well node-attribute similarity is retained in the graph drawing, and inspired by the trustworthiness in projections, we reused trustworthiness to measure node-attribute similarity separately in a graph drawing. We define two metrics as the following:

- **Topology-Trustworthiness:** we define *topology-trustworthiness* as a measure that can be used in dimensionality reduction graph drawing, that tells us to what extent local structure is retained in the graph drawing, given the high dimensional *structural representation* of the graph. The trustworthiness measure is calculated over the binary (high dimensional) adjacency matrix, and the graph drawing (2 dimensional representation).
- **Attribute-Trustworthiness:** We define *attribute-trustworthiness* as a measure that can be used in dimensionality reduction graph drawing, that tells us to what extent local structure is retained in the graph drawing, given the high dimensional *attribute representation* of the graph. The trustworthiness measure is calculated over the binary (high di-

mensional) node-attribute matrix (as defined in section 4.1.1), and the graph drawing (2 dimensional representation).

3.2.4 Experiment setup

Quantitative evaluation:
For each given dataset (Karate,LesMiserables,Football,Hollywood), each algorithm (GH4D,Spring,Random) will generate a graph drawing 10 times. Then each of the 10 produced graph drawings is evaluated based on 6 metrics (EdgeCrossing, EdgeLength, AngularResolution, NeighborhoodPreservation, topologyTrustorthiness, AttributeTrustworthiness). Below the stepwise experimental setup given the above parameter settings for the GH4D approach.

Algorithm 1 Pseudocode of GH4D evaluation pipeline

```
  **Datsets**={Karate,LesMiserables,Football,Hollywood}
  **Algorithms**={GH4D,Spring,Random}
  **EvaluationMetrics**={EdgeCrossing,EdgeLength,AngularResolution,
  NeighborhoodPreservation,topologyTrustorthiness,
  AttributeTrustworthiness}
**for each:** $d \in \mathcal{D}atasets$
**for each:** $a \in \mathcal{A}lgorithms$
**for each:** $i = 0, i < 10, i++$
  drawing = createGraphDrawing(d,a)
  result = evaluateGraphDrawing(drawing,EvaluationMetrics)
  results.concat(result)
```

The results are afterwards visualized such that our approach (GH4D) can be compared to the benchmark strategies (Spring,Random). For GH4D, each iteration, the RandomSeed fed to the algorithm is different. We use fixed values for the other three parameters. For neighbourinfluence, we used the default value 1.0. For iterations we used 2 iterations, since it is said that often a value of 2 to 4 is sufficient. For embeddingdensity we use 50% the embedding dimension. It is said that this parameter should be roughly 25%-50% of the embedding dimension. We experimented with different values for neighborinfluence, embeddingdensity, and iterations. We did not see a positive effect on the visual quality metric results when tuning the parameter values. We did see an increase in execution time for higher values of embeddingdensity and neighbourinfluence. We also see a negative influence on edge length when increasing both iterations and embeddingdensity.

Multivariate network visual evaluation:

In addition to the quantitative evaluation, we will evaluate the Multivariate network based on how node-attributes influence the drawing. We will do this for the Multivariate dataset "Hollywood Film Music". We will generate three drawings: one that is purely based on an topological embedding, one that is purely based on a node-attribute embedding, and one that is drawn on both.

4. Results

4.1 Quantitative and visual evaluation

4.1.1 Karate Dataset

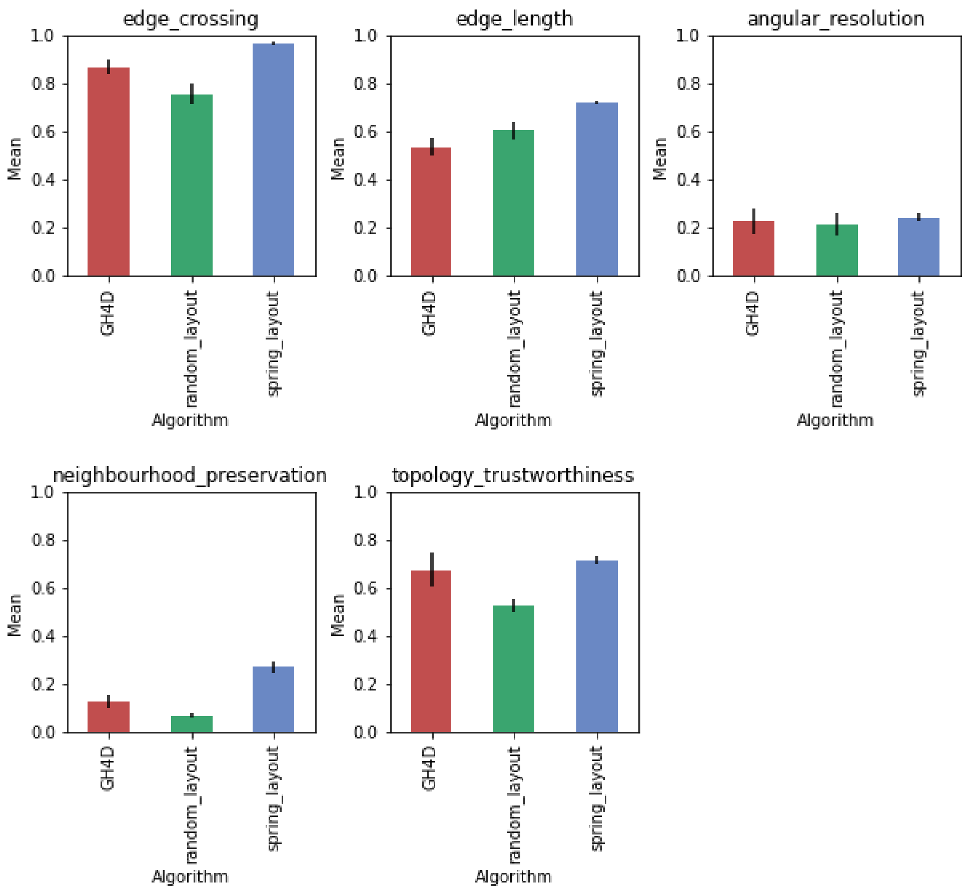


Figure 4.1: Karate Barplots with error-bars of each algorithms performance on Six visual quality metrics: a clear difference can be seen between the means of the best (spring), the second best (GH4D), and the worst (random)

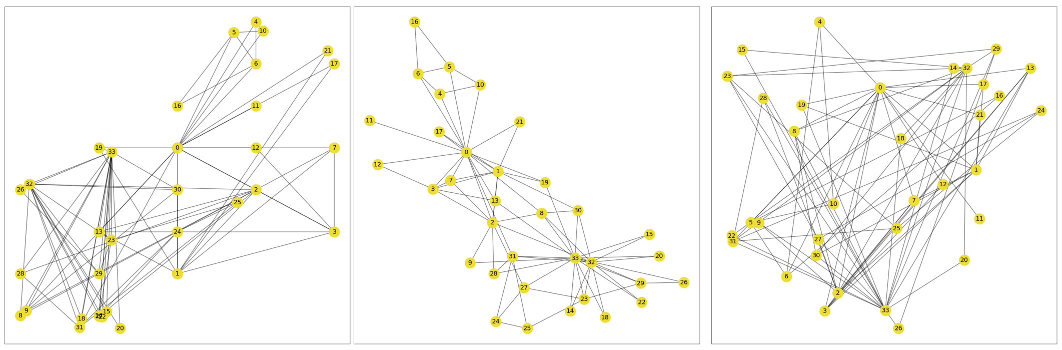

(a) (b) (c)

Figure 4.2: Karate graph drawings produced by each algorithm (a) Structure/communities is visible in the drawing (b) Clear structural proximity preservation is present (c) No clear structure or community is present

Algorithm	EC	EL	AR	NP	TT	FT
GH4D	0.869	0.537	0.227	0.125	0.675	NA
Spring	**0.967**	**0.721**	**0.242**	**0.273**	**0.717**	NA
Random	0.756	0.605	0.214	0.068	0.526	NA

Table 4.1: Mean VQM and Information distribution metric results for each algorithm: Spring has the best results in **bold**, followed by GH4D and lastly Random

Algorithm	Execution time
GH4D	29,11 ms
Spring	5,78 ms
Random	**0,08** ms

Table 4.2: Average execution time for each algorithm: The best performing algorithm is spring, which has for each metric the highest value in *bold*

When we consider the results of the karate dataset from a VQM (visual quality metric) perspective, (edge-crossings: EC, edge-length: EL, angular-resolution: AR) in table 4.1, spring has the best results for each VQM. Also from the Information distribution perspective (neighborhood preservation: NP, topology-trustworthiness: TT) spring has the best performance. This is also visible in the graph drawing in figure 4.2 (b), when we compare it to the other two (a) and (c). We can clearly see less edge crossings, the angles between edges seems also more balanced compared to the other drawings and we don't see many long edges, which reflects that closely connected nodes are also not far away in the graph drawing.

When we look at the results of GH4D from a VQM pov, it can be said that the results are in-between spring (best) and random (worst). The same can be said about the graph drawing in figure 4.2. From the same VQM pov, it looks better then the random drawing in figure 4.2 (c) but worse then spring. While there are visibly more edge crossings and edges-lengths more variable, there is still clearly a better structure visible compared to random. From a information distribution pov, while GH4D performs worse then Spring, the difference between both is reasonable (0.042 difference between TT means). Looking at both graph drawings of Spring and GH4D, we can see why these results are close. We can see that they recognize somewhat similar communities.

This is better visualized in figures 4.3 4.4, 4.5, where we highlighted the similar communities. For example, sub-graph of nodes (4,5,6,10,16) (at the top in both drawings) are closely connected, and all connected to node 0, which roughly placed in the center. This is not the case in random (sanity check). In addition, (3,7,12,17,21) and (14,15,18,20,22,23,29) are also close in both drawings.

In addition, from a performance perspective, random is the fastest, spring is second and GH4D is roughly 6 times slower then spring.

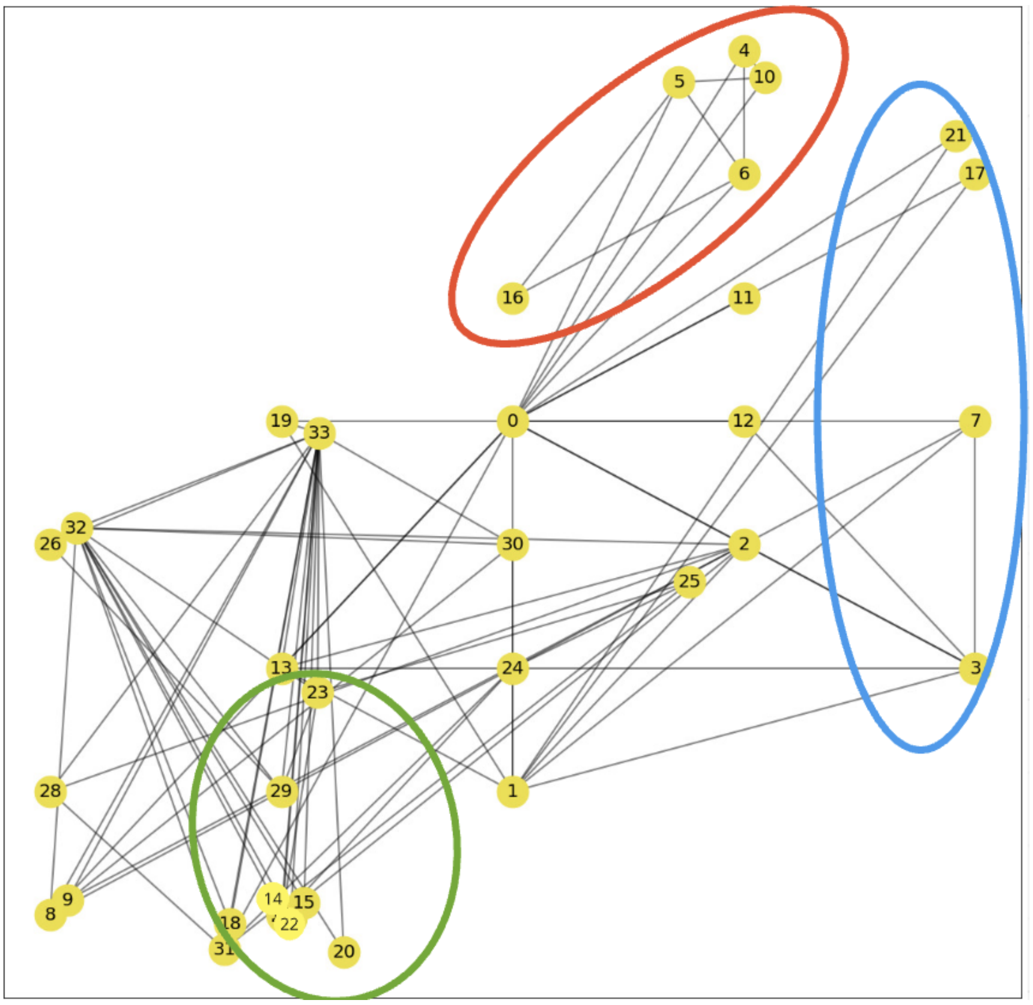


Figure 4.3: GH4D drawing, similar closely connected communities are highlighted in red blue and green: community (4,5,6,10,16) can be found in both GH4D and spring, and (3,7,12,17,21) and (14,15,18,20,22,23,29) are very similar

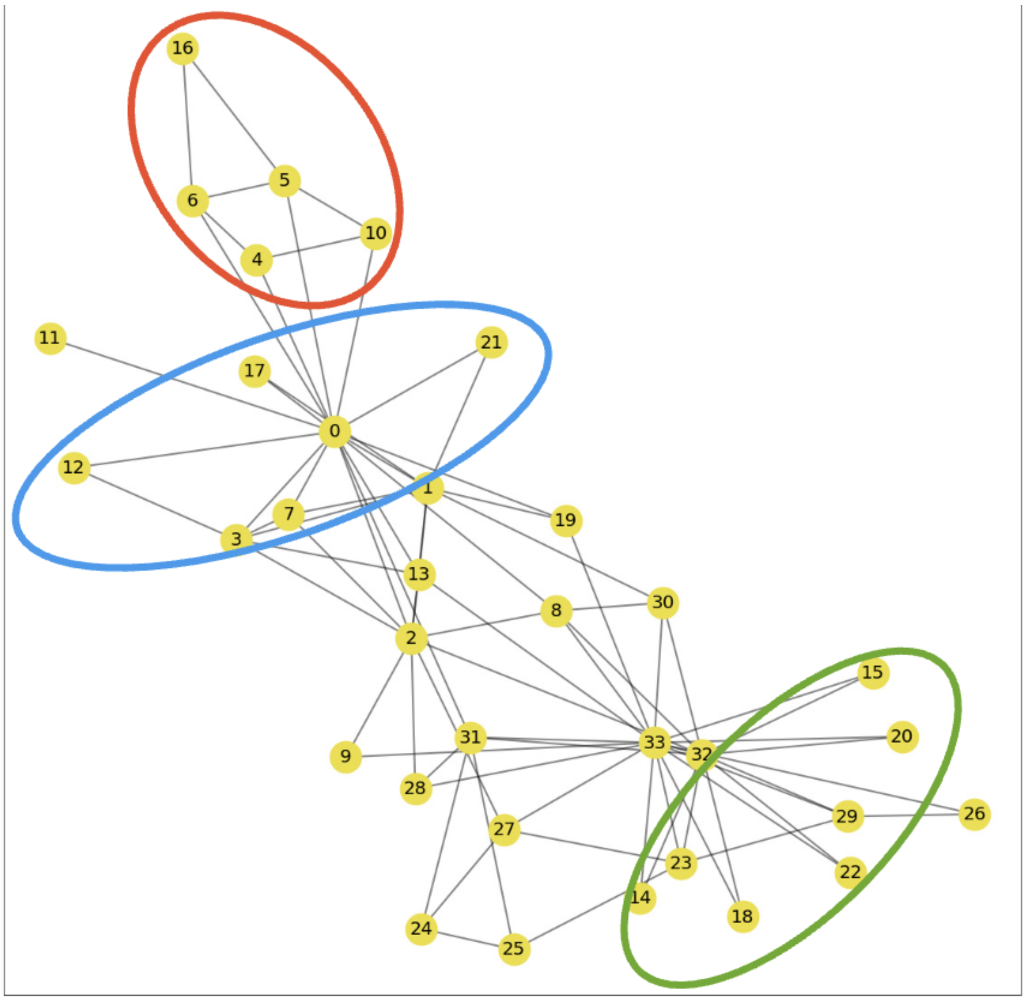


Figure 4.4: Spring drawing, similar closely connected communities are highlighted in red blue and green: community (4,5,6,10,16) can be found in both GH4D and spring, and (3,7,12,17,21) and (14,15,18,20,22,23,29) are very similar

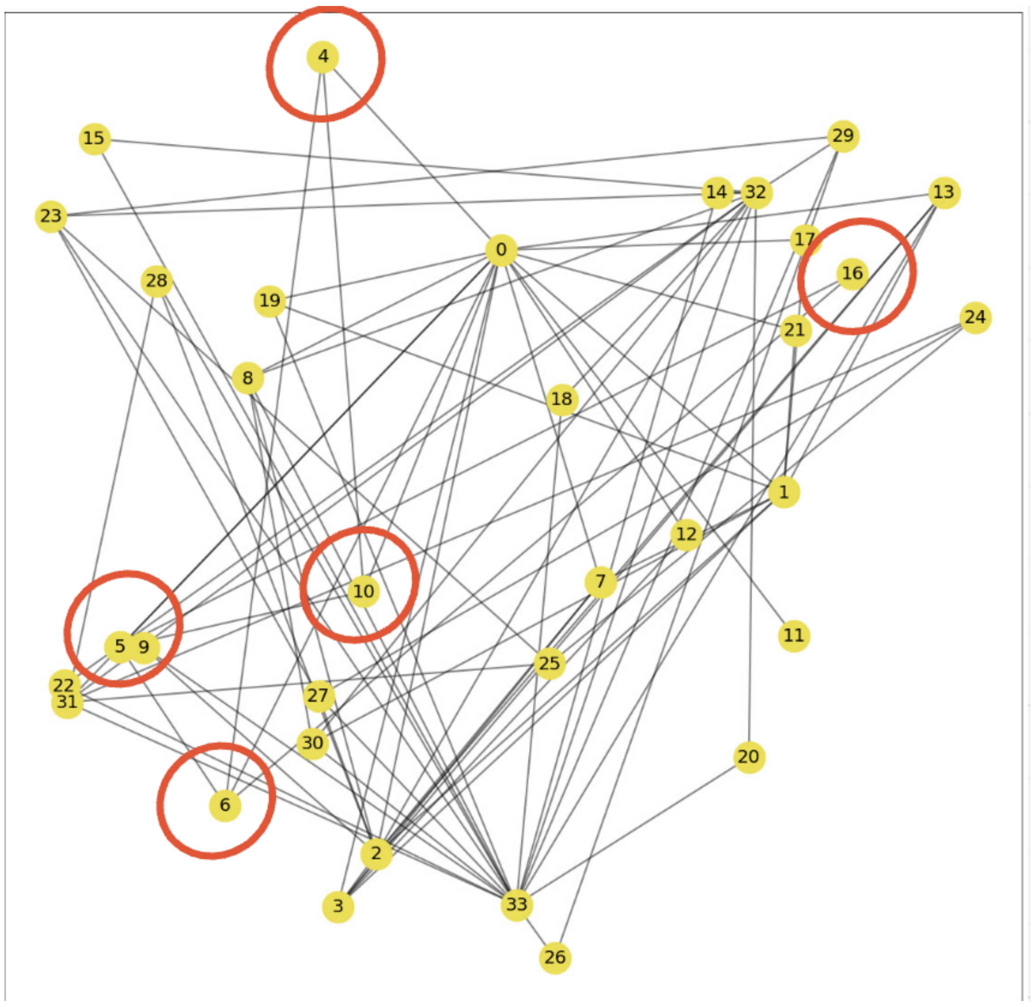


Figure 4.5: Random, similar closely connected communities are highlighted in red blue and green: We do not find the same communities in this drawing as is the case with Spring and GH4D

4.1.2 Les Miserables Dataset

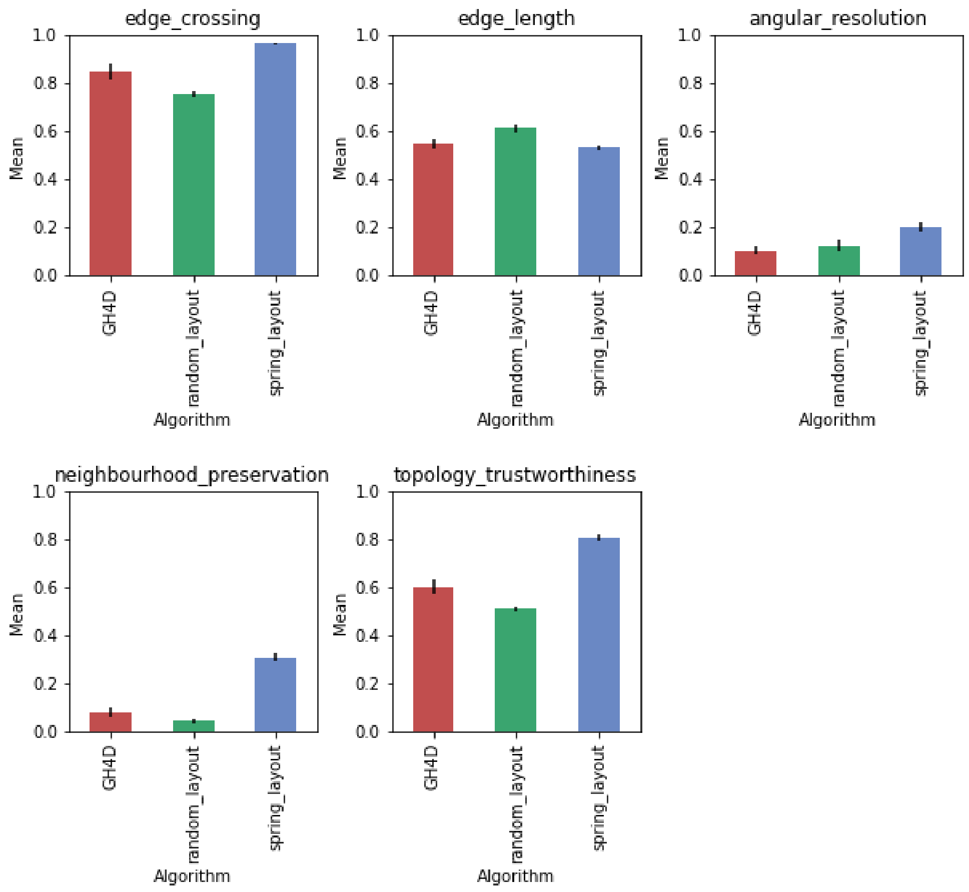
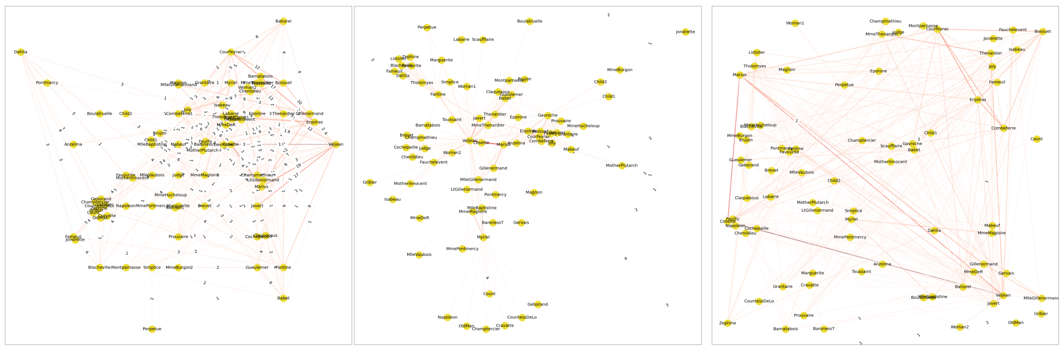


Figure 4.6: Les Miserables Barplots with error-bars of each algorithms performance on Six visual quality metrics: a difference can be seen between the means of the best (spring), the second best (GH4D), and the worst (random)



(a) (b) (c)

Figure 4.7: Les Miserables Bar-plots of each algorithm (a) GH4D: some structural proximity preservation presernt (b) Spring: Clear structural proximity preservation (c) No proximity preservation

Algorithm	EC	EL	AR	NP	TT	FT
GH4D	0.849	0.547	0.102	0.078	0.602	NA
Spring	**0.966**	0.532	**0.200**	**0.308**	**0.806**	NA
Random	0.755	**0.611**	0.122	0.045	0.511	NA

Table 4.3: Les Miserables mean VQM and Information distribution metric results for each algorithm: Spring has the best results in **bold**, followed by GH4D and Random are closer with the edge to GH4D

Algorithm	Execution time
GH4D	39,43 ms
Spring	13,84 ms
Random	**0,10 ms**

Table 4.4: Average execution time for each algorithm: The best performing algorithm is spring, which has for each metric the highest value in *bold*

When we consider the results of the Les Miserables dataset from a VQM perspective, in figure 4.6 and table 4.3, spring has the best results for each VQM (except EL), and for each information distribution metric (NP,TT). This is also reflected in the graph drawing produced by spring in figure 4.7 (b). We see few edge crossings, and edge-lengths and edge-angles are well distributed. We see clear communities that are closely connected. In the case of GH4D's VQM, the edge-crossings score better than random (difference 0.094), while EL and AR are closer (in the favor of random). However, when we look at the graph drawings of GH4D and random, it is more clear that GH4D produces the better drawing. In GH4D we see better structure, closely connected nodes (communities), while this is not present in random. From information distribution metrics pov, GH4D worse than Spring (TT of 0.602 vs 0.806) while better than random (random TT is 0.511), and NP GH4D, Spring and random (0.308,0.078,0.045) respectively. This is also reflected by the graph drawings 4.7. When we compare GH4D and Spring, we see that GH4D can detect some of the communities that spring detects. In figures 4.8 4.9, 4.10 we highlighted some of these similar communities. For example, the community of (Napoleon, Oldman, Count, Champtercier, Cravatte, countlessdelo, geborand) can be found in both drawings (in red). And there are communities that are very similar (Brujon, Montparnasse, Claquesous, Geulemer, Babet) and (Courefac, Combeferre, Grantaire, Enjolras, Feuilly, Bahorel, Bossuet). Yet, as a sanity check, the same communities in random are not present.

In addition, since the edges are weighted, GH4D should draw nodes with similar weighted edges closer. If we look at the drawing, is is not very clear, but we can somewhat see that higher weighted edges are clustered more on the top right corner of the drawing. This is also present in spring, but not in random.

In addition, from a performance perspective, random is the fastest, spring is second and GH4D is roughly 3 times slower then spring.

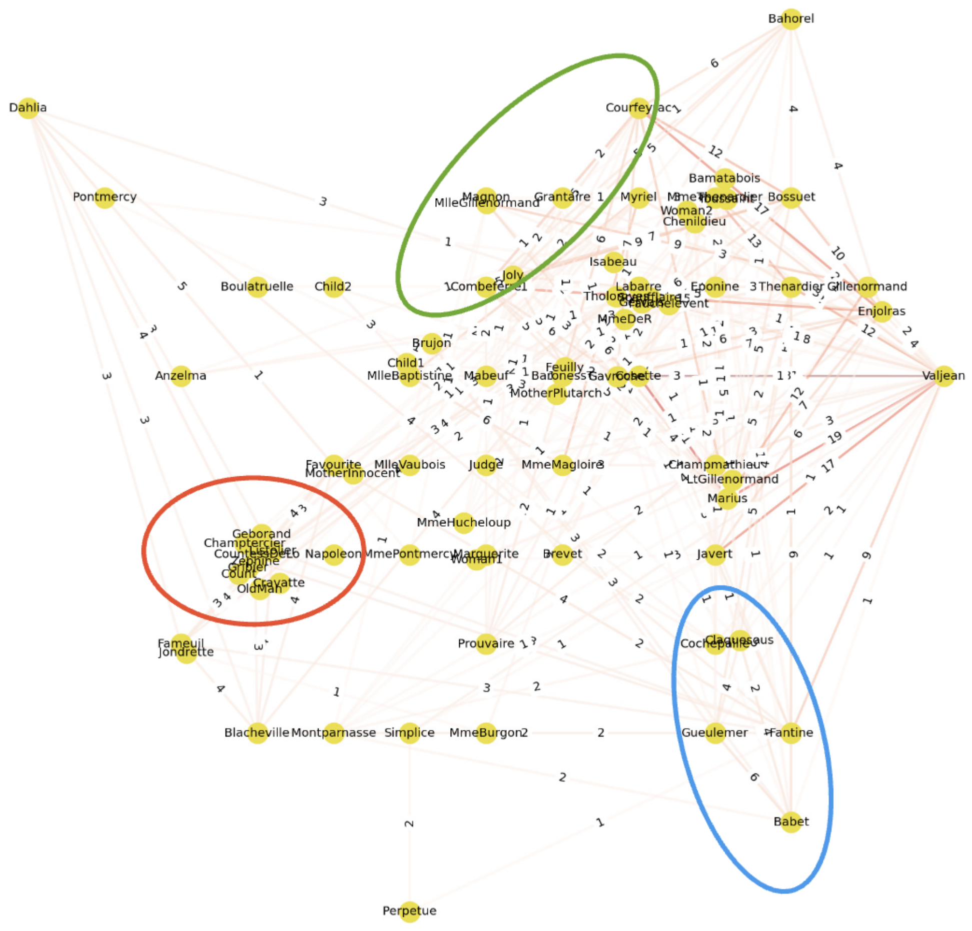


Figure 4.8: GH4D drawing, similar closely connected communities are highlighted in red blue and green: For example the community in red of (Napoleon, Oldman, Count, Champtercier, Cravatte, countlessdelo, geborand) is also present in Spring

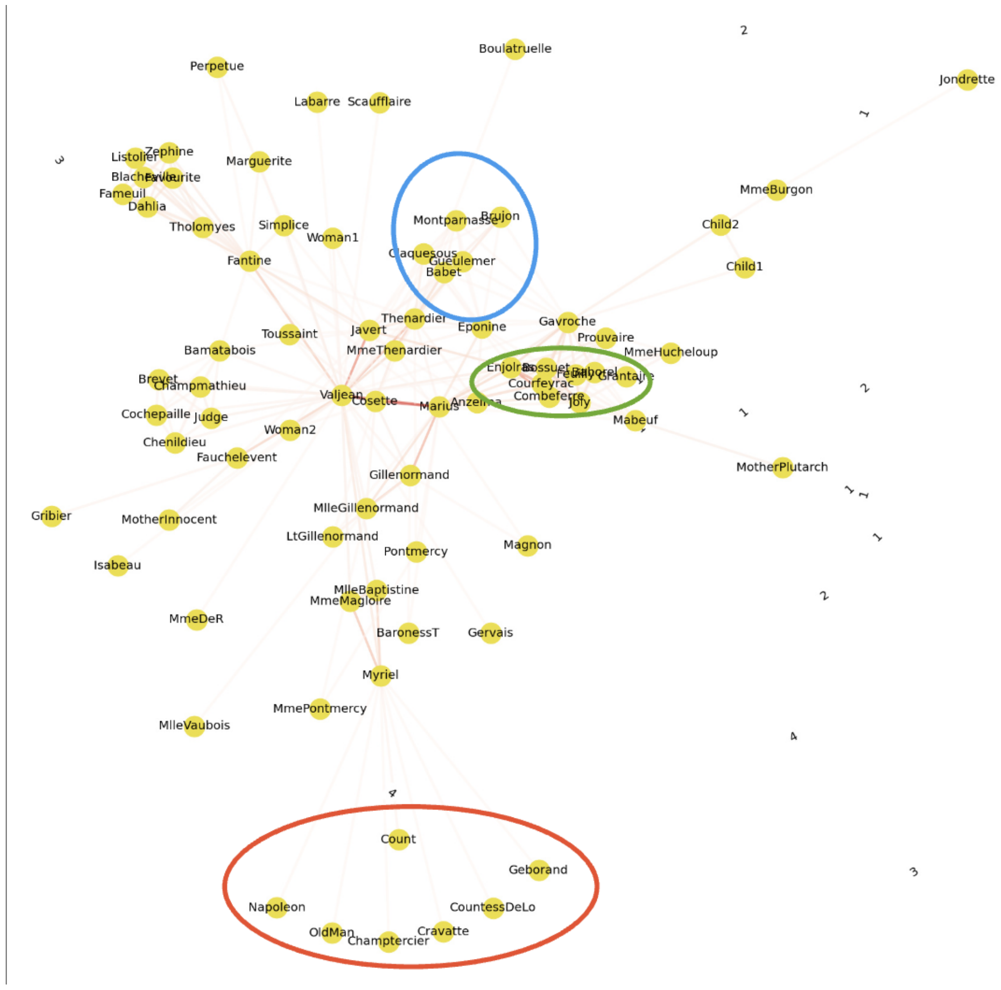


Figure 4.9: Spring drawing, similar closely connected communities are highlighted in red blue and green: For example the community in red of (Napoleon, Oldman, Count, Champtercier, Cravatte, countlessdelo, geborand) is also present in GH4D

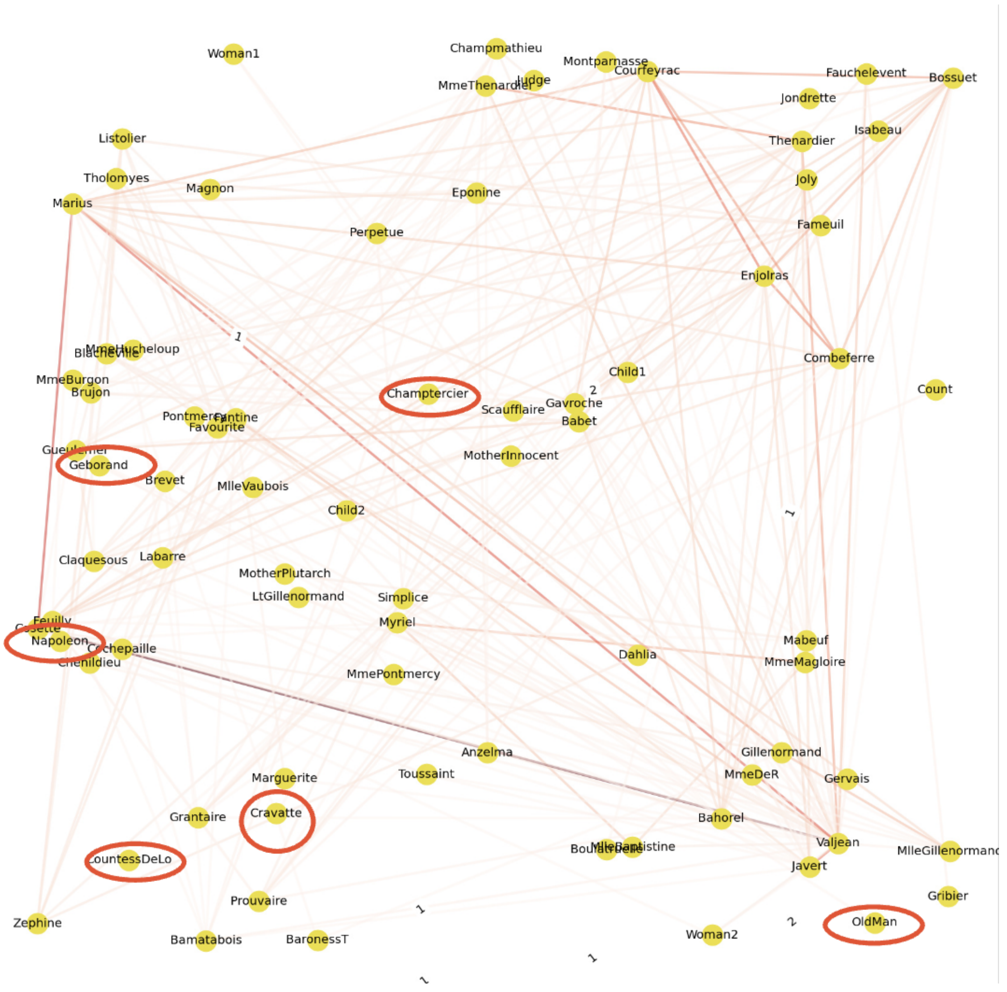


Figure 4.10: Random, similar closely connected communities are highlighted in red blue and green: We do not find the same communities in this drawing as is the case with Spring and GH4D

4.1.3 Football Dataset

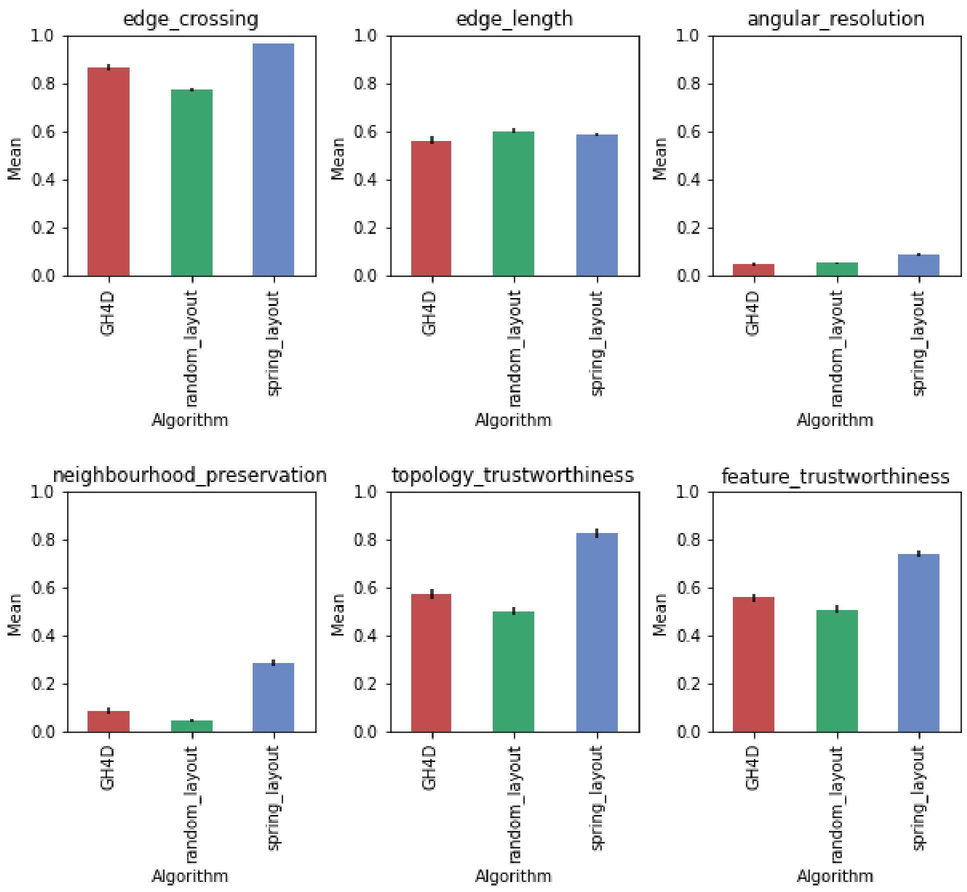
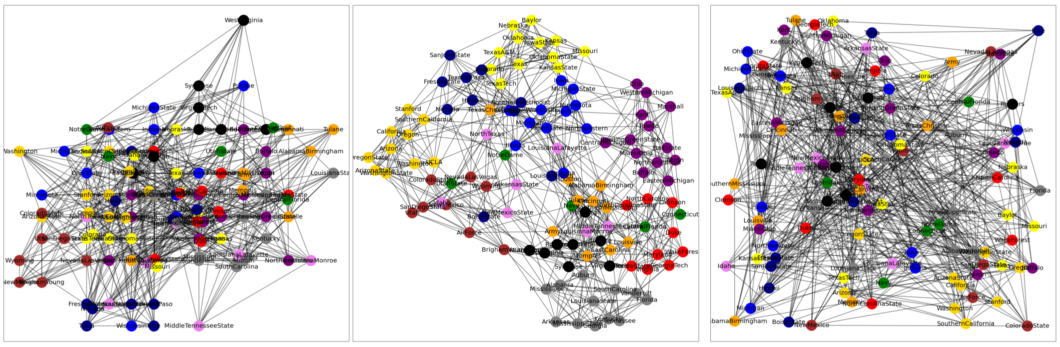


Figure 4.11: Football Bar-plots with error-bars of each algorithms performance on Six visual quality metrics: Except for edge crossing, the VQM metrics are much closer. From an information distribution pov best to worst are more clear (Spring, GH4D and Random)



(a) (b) (c)

Figure 4.12: Football graph drawings of each algorithm (a) GH4D: there is separation of groups present, yet not very clear (b) very clear separation of groups (c) No clear separation of groups

Algorithm	EC	EL	AR	NP	TT	FT
GH4D	0.866	0.563	0.048	0.089	0.574	0.558
Spring	**0.968**	0.587	0.088	**0.286**	**0.826**	**0.739**
Random	0.766	**0.609**	**0.208**	0.019	0.515	0.505

Table 4.5: Football Mean VQM and Information distribution metric results for each algorithm: The VQM results are much more distributed, while in the information distribution results spring is the best followed by GH4D and the worst is random

Algorithm	Execution time
GH4D	65,70 ms
Spring	26,13 ms
Random	**0,22 ms**

Table 4.6: Average execution time for each algorithm: The best performing algorithm is spring, which has for each metric the highest value in *bold*

When we consider the results of the Football dataset from a VQM perspective, in figure 4.11 and table 4.5, the results are a bit more divided. We can see that spring minimizes edge-crossings very well, and GH4D is second best, while random is worst. While for the other two, EL and AR, both spring and GH4D are worse than random. However, this is not reflected by the drawing. We can see the best structure in Spring, in which several highly connected sub-graphs are visualized, which is less visible, but somewhat there in GH4D, while from the visualization pov random is the worst drawing. From information distribution metrics pov, GH4D is worse than Spring (TT of 0.826 vs 0.574) while better than random (random TT is 0.515). The same holds for the FT of all three (0.739 vs 0.558 vs 0.505). And NP of GH4D, Spring and random (0.308,0.078,0.045) respectively. This is also reflected by the graph drawings 4.12. Since this dataset has categorical node attributes, we visualized each nodes attribute in figure 4.12 with a different color. In spring, the communities are best identified, nodes of the same colors are for the most part placed in the same group. While in GH4D's drawing this is less visible, they are still somewhat of a division color wise, yet they are more scattered then spring. While in random, we can say that this is not the case.

We expected that GH4D would do better from a node-attribute encoding perspective. Because, while spring determines the communities solely on connections, GH4D gets the attribute information (attribute matrix) fed as an input. Thats why we wanted to take a closer look closer into GH4D's performance in relation to the football dataset. We reran GH4D again, but this time we fed the algorithm a vector representation that is solely based attribute information, instead of both topology and attribute. However, the results were similar to the one where both topology and attribute information is fed to the algorithm.

In addition, from a performance perspective, random is the fastest, spring is second and GH4D is roughly less than 3 times slower then spring.

4.1.4 Hollywood Film Music Dataset

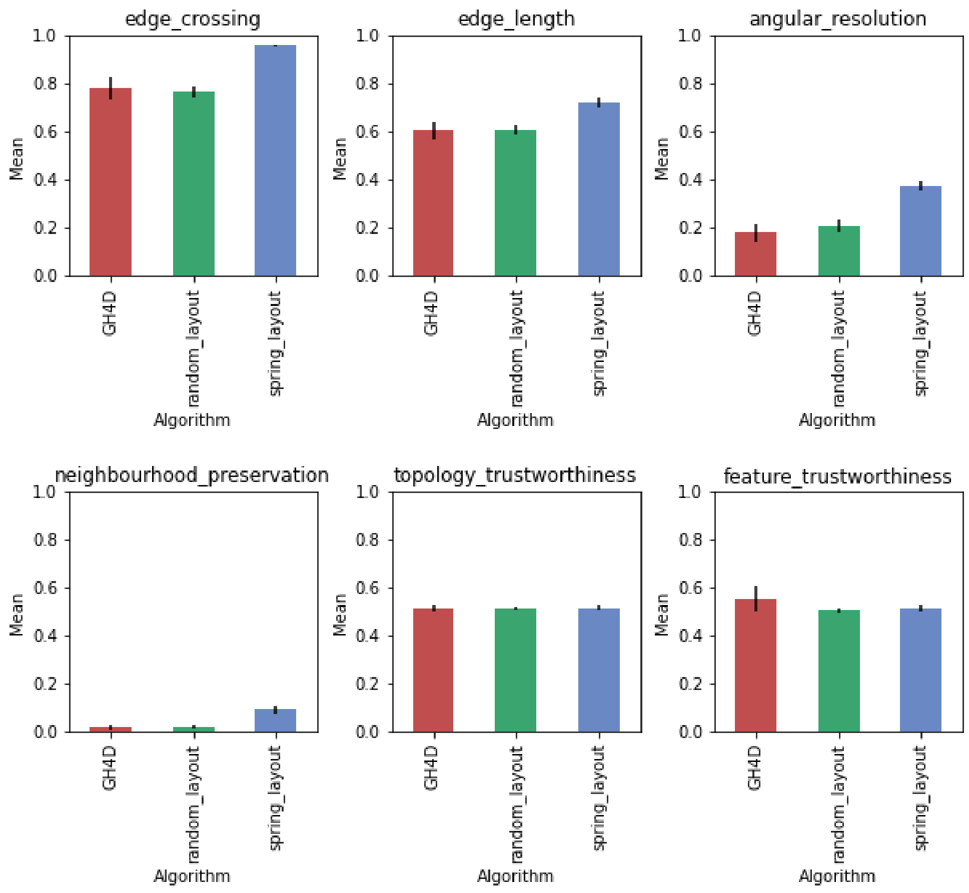


Figure 4.13: Hollywood Film music Bar-plots with error-bars of each algorithms performance on Six visual quality metrics: Spring is the best on each VQM. From information distribution perspective, they are much closer metric wise

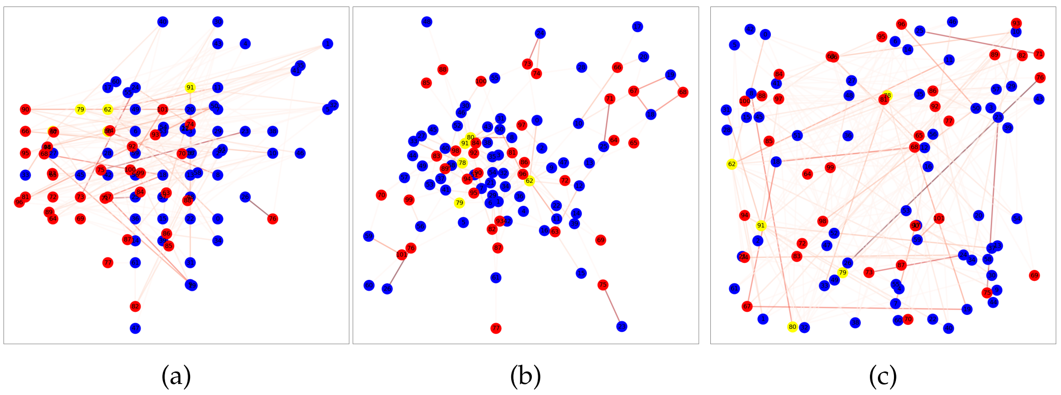


Figure 4.14: Hollywood Film Music graph drawing produced by each algorithm (a) GH4D: Clear clustering can be seen of each group (color) (b) Spring: Groups are not separated well, yet VQM wise its a better drawing (c) Random: No structure VQM and group separation wise

Algorithm	EC	EL	AR	NP	TT	FT
GH4D	0.781	0.604	0.179	0.018	0.514	**0.554**
Spring	**0.960**	**0.718**	**0.374**	**0.091**	**0.516**	0.516
Random	0.766	0.609	0.208	0.019	0.515	0.505

Table 4.7: Mean VQM and Information distribution metric results for each algorithm: VQM ise Spring has the best results. Information distribution wise results are closer.

Algorithm	Execution time
GH4D	55,98 ms
Spring	19,65 ms
Random	**0,09 ms**

Table 4.8: Average execution time for each algorithm: The best performing algorithm is spring, which has for each metric the highest value in *bold*

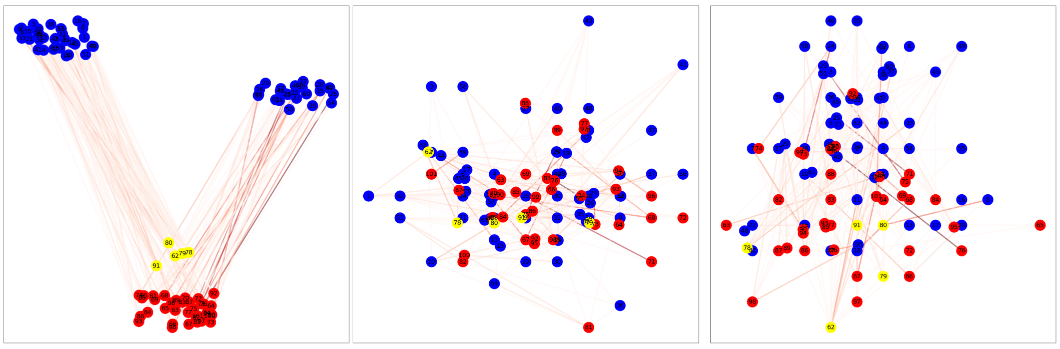
When we consider the results of the Hollywood Film Music dataset from a VQM perspective, in figure 4.13 and table 4.7, spring has the best results for each Visual quality metric (VQM). This is also visible in the graph drawing in figure 4.14 (b), when we compare it to the other two (a) and (c). We can clearly see less edge crossings, and it is also visible that edges length and angles are more balanced. VQM wise, GH4D and random are much closer. However, the drawing of GH4D has a slight edge over random, since we can see more structure in the drawing.

From information distribution metrics pov, for NP spring has an edge over GH4D, while TT spring and and GH4D are very close (0.514 vs 0.516). As for TT, when we look at the visuals in figure 4.14, spring is the better drawing topology wise (which is in fact represented in NP) but not so much in TT. Interestingly, FT wise GH4D is better than Spring (0.554 vs 0.516). This is well reflected in the drawing. We can see clear separation between the Composers (blue) and Producers (red) and top 1.5% producers (yellow) in figure 4.12 (a) compared to (b). As for edge weights, they are not very well clustered in any of the drawings.

Since we have seen that for Multivariate network Hollywood Film Music, we get from an attribute perspective, a better representation compared to the other algorithms. We want to take a deeper look into how incorporating node-attribute in the vector representation influences the graph drawing for GH4D.

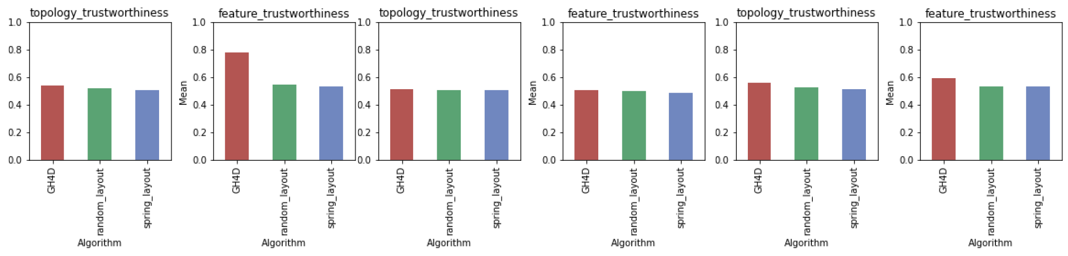
In addition, from a performance perspective, random is the fastest, spring is second and GH4D is roughly less than 3 times slower then spring.

4.1.5 Visual analysis of node-attribute impact in Hollywood film music dataset



(a) (b) (c)

Figure 4.15: Graph drawings produced by GH4D: (a) Purely based on node-attribute information (b) Purely based on topology information (c) Based on both attribute as well as topology



(a) (b) (c)

Figure 4.16: Bar-plots produced by GH4D (a) (a) Purely based on node-attribute information (b) Purely based on topology information (c) Based on both attribute as well as topology

In this part, we generate three drawings using GH4D for the Hollywood Film Music dataset: one that is purely based on an topological embedding, one that is purely based on a node-attribute embedding, and one that is drawn on both. We ran the GH4D algorithm on the Hollywood Film Music dataset, and created the three visualizations shown in figure 4.15. Metrics wise, in this part we focus on the information distribution metrics TT and FT, which are shown in figure 4.16.

As shown in the visualization where the node vectors only contain node-attribute in formation, in figure 4.15 (a), we can conclude that, for the given dataset GH4D can make a clear distinction between each group (composer, producer, top-producer). It further distinguishes the composers into two groups, where it seems that the higher weights are more clustered into the right. This is further confirmed by the FT metric, which shows that the drawing outperforms the other drawing by a large margin. In the drawing that encodes both topology as well as attributes shown in figure 4.15 (c) we can still see a clear distinction between the groups, which is also reflected by the corresponding FT metric result. While the drawing that encodes solely topology, shown in figure 4.15 (b), performs the worst FT metric wise. TT metric wise, all drawings are pretty close, while the drawing that encodes both topology and attributes has a slight edge.

5. Discussion

From the Visual quality metric (VQM) results of the Karate dataset we learn that they do not fall within our 10 percent margin. We hypothesised delta of 10 percent between the state of the art algorithm (Fruchterman Reingold) and GH4D, but in reality the margin is 15 % for the VQMs on average. However, from the visual evaluation of the drawings produced by GH4D, we learned that they are representative to encode structural proximity. This is also reflected by the Topology Trustworthiness (TT) metric (0.675 vs 0.717), but not by the Neighborhood preservation (NP) metric (0.125 vs 0.273). As for the VQM results of the Les Miserables dataset, they are slightly out of the 10 percent margin. We get a difference of 11,77 % on average between the VQM results of spring and GH4D. Just as with the karate dataset, we did see that the drawings encode several aspects of nodes structural proximity, just as the Fruchterman Reingold algorithm does. Yet, the information distribution metrics difference is larger compared to the Karate dataset.

The results of both datasets show us that GH4D can learn structural proximity, given the adjacency matrix as a vector representation. Performance of GH4D worse for both VQM and information distribution metrics than FR based on our selected metrics, but that is expected, since spring is optimized to visualize network structure, while GH4D does not consider optimal positions of nodes such that VQM are maximized. Using random as a sanity check, we have seen that it can create drawings that contains aspects of conserved structural proximity, where closely connected nodes are drawn closer in the drawing.

Next we considered network datasets that contain both topological information as well as node-attribute information, in the Football and Hollywood Film Music dataset.

The results of the Football dataset, which solely contains categorical nodes, we learned that GH4D can somewhat learn multivariate network structure. While the VQM metric results were a bit more divided in this dataset, the difference between spring and GH4D is 10 percent. This might be connected to the fact that Football is a ground truth network, in which encodes 11 groups that are heavily connected with each other. When we focus on the information distribution metrics (NP, TT, FT) spring performs best followed by GH4D. This is further shown by the graph drawings produced by both. We expected better results when feeding attribute information only, yet we got similar results. The Hollywood Film Music produced much better results from a Feature Trustworthiness perspective when only attributes were encoded into the vector. The results of Football dataset might be linked to the way the vectorization happens for the categorical node in Football.

The Hollywood Film Music dataset showed us that, while performing worse on VQM scale compared to spring (delta 23,8 %), it did in fact encode attributes better. Yet, must be stated that the VQM results of GH4D were more similar to the random algorithm. We came to the same conclusion by analysing the drawing produced by the algorithms. Except that we did see a better structure in the GH4D drawings compared to the random. In addition, the Hollywood Film Music dataset produced promising results, where higher attribute presence in the input matrix gives graph drawings in which attribute information is better represented. On the one extreme, a pure node-attribute representation matrix gives us a drawing where nodes are separated based on attribute value, and edge weights. This is also reflected by the Feature trustworthiness metric, which was highest. While on the other extreme, a pure topological drawing gives us a drawing where the feature trustworthiness is lowest. A combination of the two takes the middle ground, scoring in between. The results from the Hollywood dataset are at one hand in line with our hypothesis, where we stated that we want to create graphs where node-attributes are better represented compared to current state of the art methods. Yet this was not done within the delta 10 percent loss of VQM performance.

In addition, from a performance perspective, we seen a trend in that, the larger the dataset, the closer the execution time of spring and GH4D were.

5.1 Future work

We have identified several directions for future work. First of all, this work identified that there are opportunities for graph drawing algorithms aimed at Multivariate networks. At each of the four steps of our approach lie opportunities for future work.

The quality of the results of our approach depends on the vectorization step, in which graph data is represented as vectors. In Hollywood, we were able to get better classification based on node-attributes. Yet in Football, the classification based on node attributes was less accurate. In our approach the adjacency matrix was used together with one-hot encoding of the node-attributes to represent the graph data. An other method to use for vectorization is using random walk methods such as Deepwalk and Node2vec instead of the adjacency matrix. The benefit would be that they use less memory to represent nodes. This in turn is beneficial for the next steps, for example the embedding.

In this work we identified 27 network embedding methodologies that use node-attributes. As we have stated before, we are not aware of any works that uses network embedding for graph drawing in which attributes are incorporated. We focused on one particular class, graph hashing HashGNN. Yet, opportunities might lie in the other algorithms and classes. For example, the other graph hashing based methods such as Binarized Attributed Network Embedding (BANE) and w-Bit Quantization for Attributed Network Representation Learning (LQANR) or Locality sensitive graph hashing embedding implementations such as NearBucket-lsh. Or the neural network based methods such as Tri-party deep network representation model (Tri-DNR) and SEAL.

We used very sparse random projection (VSRP) to reduce the dimensionality of the produced embedding. There are several other dimensionality reduction techniques. For example, t-SNE and Principal component analysis (PCA). Or the dimensionality reduction based methods that are aimed at graph drawing, such as Tsnet and DRgraph. Since these methods are aimed at graph drawing, they might pose as good alternatives to draw graphs based on network embedding. A downside of using VSRP, from a graph drawing perspective, is that is results in an low dimensional representation in which multiple nodes can have the same value. The next step in this approach, would be to create a fully recursive implementation, in which buckets (sub-graphs) are recursively drawn using our approach.

An other direction for future work is to experiment with implementations that combine our approach with force directed graph drawing algorithms. We used network embedding for Multivariate graph drawing, that does not maximize Visual quality metrics, while force directed graph drawing algorithms do. What we have in mind for such a *hybrid strategy*, is that GH4D initially generates a drawing. As we have seen before, GH4D drawing contains groups of nodes with the same coordinates. The Force directed layout algorithm is in turn executed on those groups (bucket). This might be seen as a divide and conquer strategy, which takes advantage of GH4D to incorporate node-attributes into the drawing, and a Force directed maximizes the Visual quality metrics by drawing the sub-graphs. This might results in a graph drawing, that is a trade-off between performance in maximizing Visual quality metrics, while incorporating node attributes.

There are currently no metrics that measure attribute proximity in a Multivariate network drawing. Such metrics would, similar to the neighbourhood preservation metric, measure how well attribute-proximity is preserved, by comparing the neighbours in the raw graph data and the neighbours in the graph drawing. In this work we attempted something similar, yet we compared the high dimensional matrix representation, instead of sets of nodes, with the low dimensional coordinates.

An other direction of future research might be a live Multivariate graph drawing visualization tool with a slider, in which the user can experiment with attribute influence. Since, our approach is relatively fast from a visualization point of view (65 ms in football with roughly 100 nodes 600 edges), this might be feasible from a performance perspective, the user does not have to wait long for the result to be visible. Since, as we have noticed in the

Hollywood Film Music dataset, that a higher node-attribute presence in the vector encoding, resulted in higher results for the Feature-Trustworthiness metric. In the visualization tool the user has live influence on how much attribute information is injected into the graph drawing, from this we might learn how much attribute influence is preferred. Lastly, performance on bigger dataset has to be investigated. The largest dataset was only 115 nodes. This approach might be beneficial for large graphs from a execution time perspective.

5.2 Limitations

The comparison of GH4D to Spring from a execution time perspective has limitations. The implementation of HashGNN is based on the graph datasciene library and an internal implementation, in which the python library is used to execute HashGNN. Such that, in order to execute hashGNN, an API call is made to (in our case) a local database using a cypher query, which in turn returns the embedding. We compared our approach to Spring implementation from Networkx, which is implemented in a python module. To make a fair comparison from a execution time perspective, HashGNN must also be implemented in a python module. Unfortunately, there is no such implementation (yet) of HashGNN.

Because of time, we were limited our evaluation to 4 datasets. We would have like to test our implementation on more datasets, especially larger ones. Especially from a execution time perspective, executing GH4D on very large datasets might gives us insight into how is compares to Spring. Unfortunately, we had no automated way to import data. All data used in this work has been imported manually. For example, the Hollywood Film Music was in a Pajek format, which had to be manually transformed to a spreadsheet, which in turn could be imported into python.

6. Conclusion

This work is motivated by the fact that there are currently no multivariate graph drawing algorithms that generate graph drawings based on both attributes as well as structural proximity. We identified opportunities within network embedding, and summed up 27 state of the art network embedding methodologies and classified them based on whether they encode topology, attributes or both. We described an approach called Graph Hashing for Drawing (GH4D) which, inspired by dimensionality reduction graph drawing and network embedding for graph drawing methodologies, which takes advantages of a multivariate network embedding (graph hashing) technique called HashGNN to generate graph drawings in which both topology as well as attributes are represented. In addition, driven by the fact that there are no metrics that measure attribute proximity in a graph drawing, we reused trustworthiness metric to measure attribute preservation between the high dimensional graph representation matrix and the low dimensional node coordinates. We evaluated the approach using four datasets used within the graph drawing community, against a state of the art force-directed algorithm. We demonstrated that GH4D can learn both topological structures for small and middle sized datasets. In addition we did see, in some of the drawings that similar edges are encoded closer together. In addition, in the multivariate datasets while it was not very clear in the football dataset, in the Hollywood dataset is more clear that the attribute encoding does influence GH4D to make graph drawings based on both topology and attributes, where attributes are better represented compared to the Fruchterman Reingold algorithm. In addition, several directions for future work have been identified.

Appendices

A. A

A.1 GH4D graph drawing images

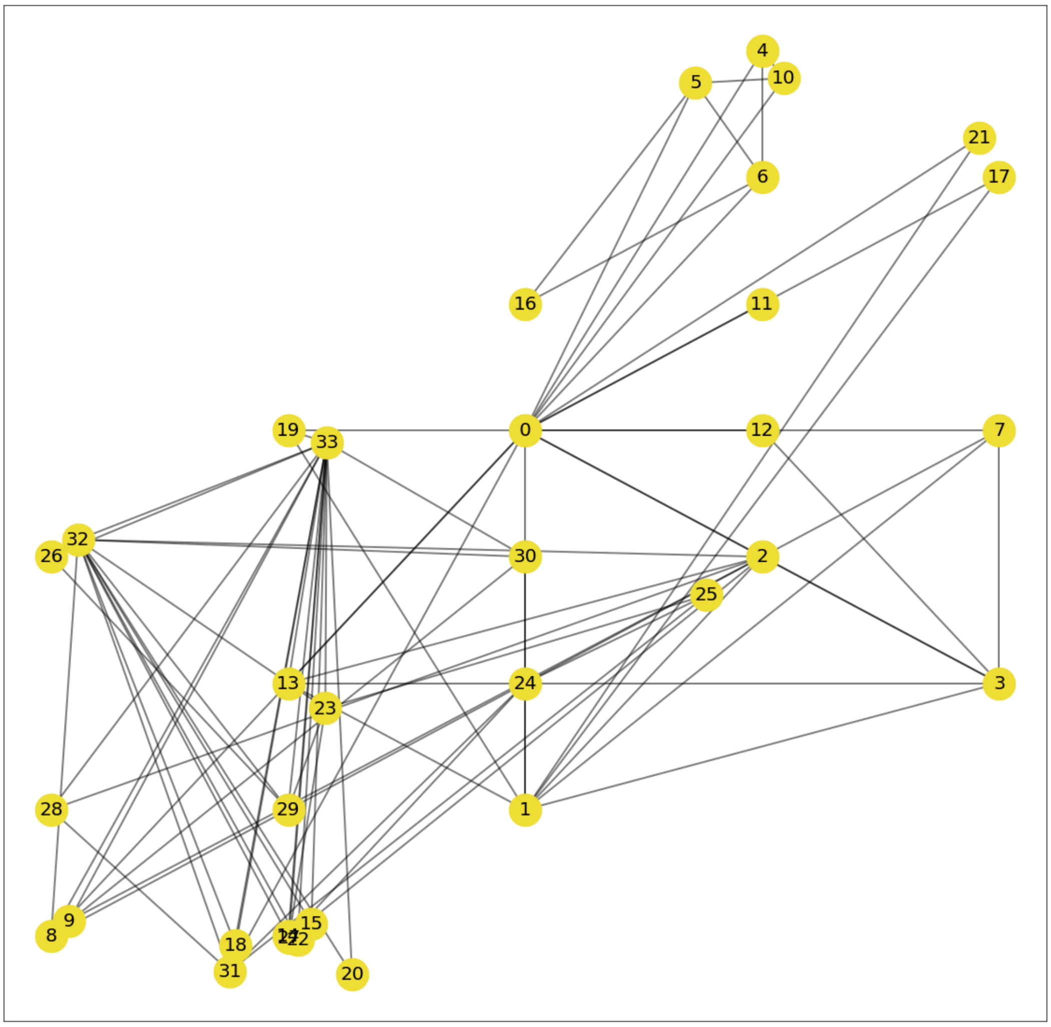


Figure A.1: GH4D drawing of the Zackary Karate Dataset, we can see clear structural proximity preservation, closely connected communities can be found similar to spring (for example 4,5,6,10,16).

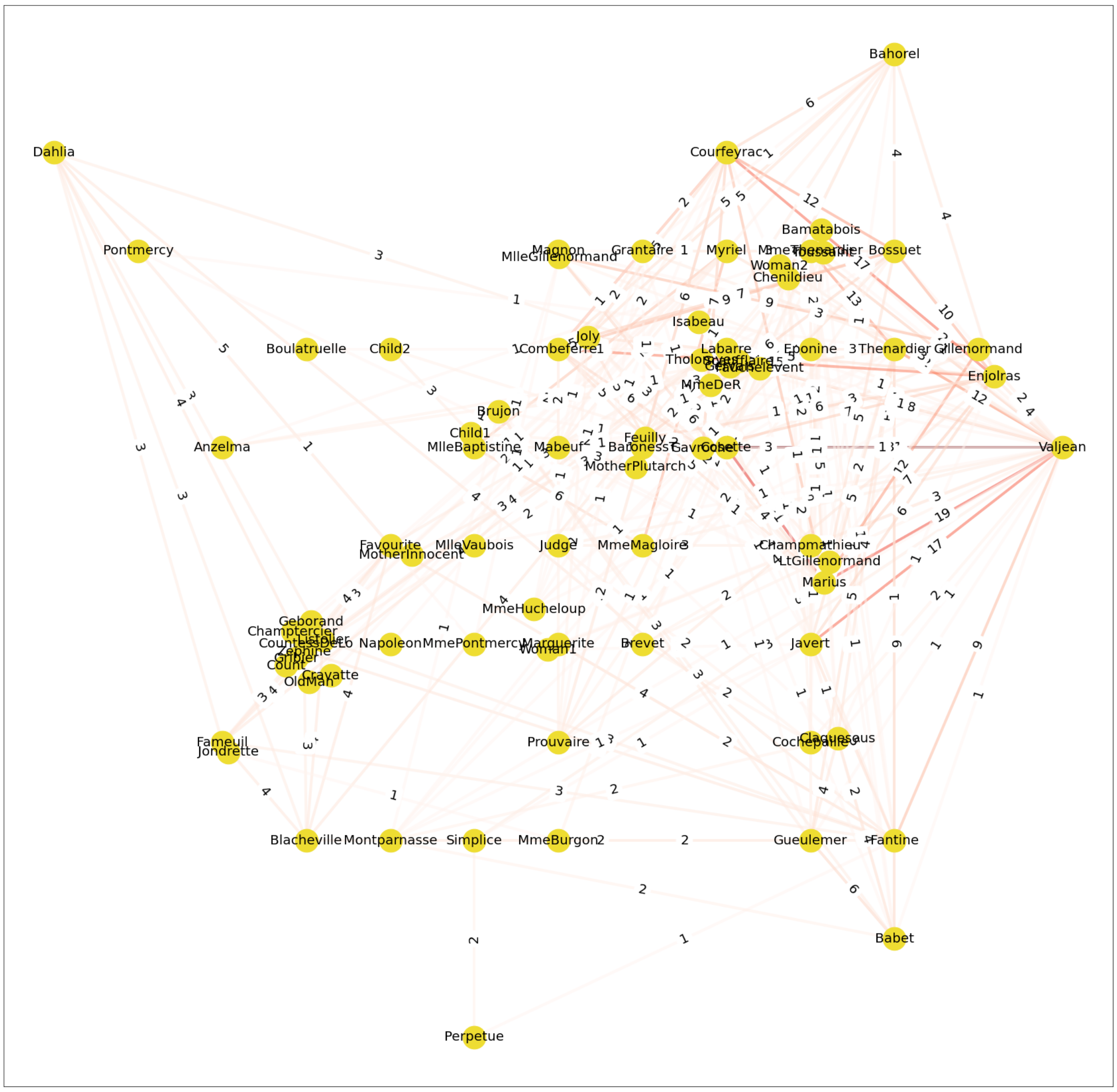


Figure A.2: GH4D drawing of the Les Miserables Dataset, we can see clear structural proximity preservation, closely connected communities can be found similar to spring (for example (Napoleon, Oldman, Count, Champtercier, Cravatte, countlessdelo, geborand))

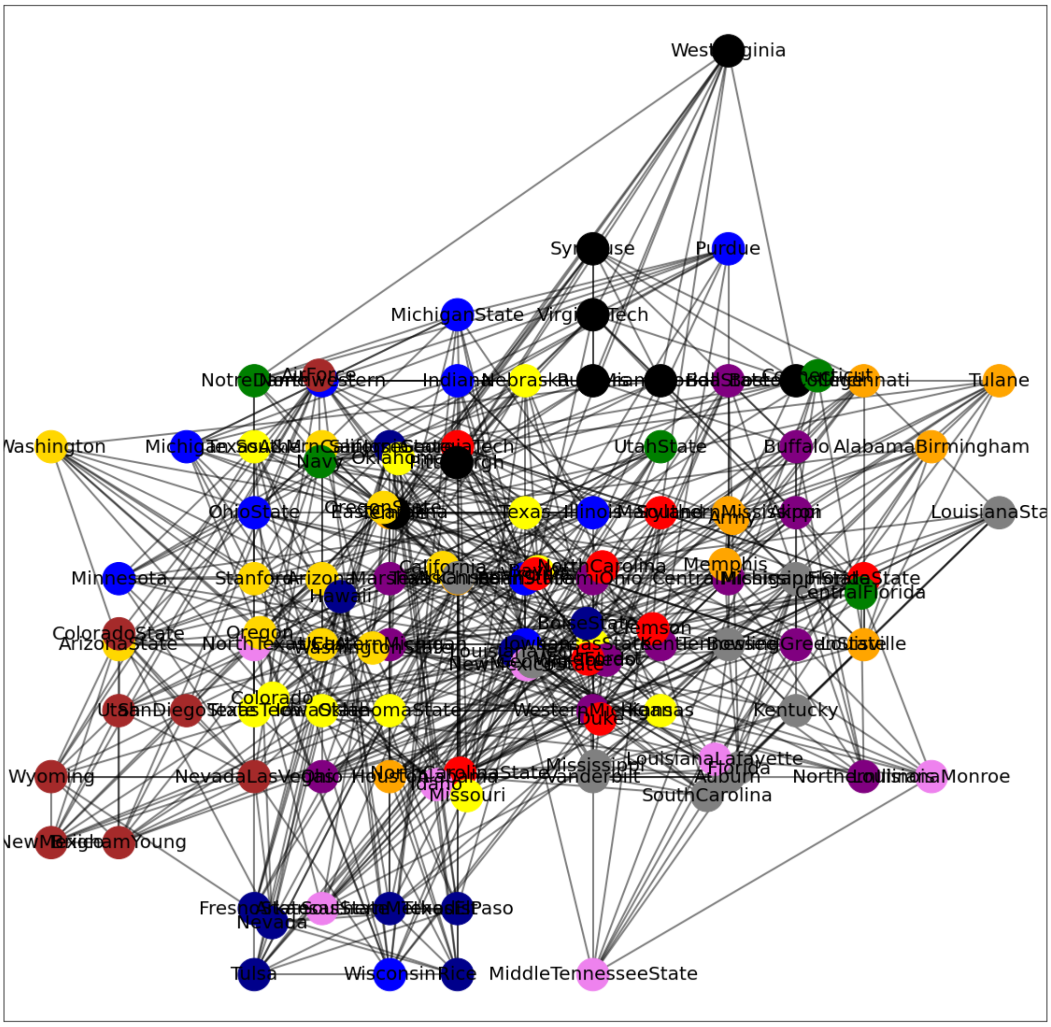


Figure A.3: GH4D drawing of the Football Dataset, there is visible clustering where similar nodes are drawn closer (color)

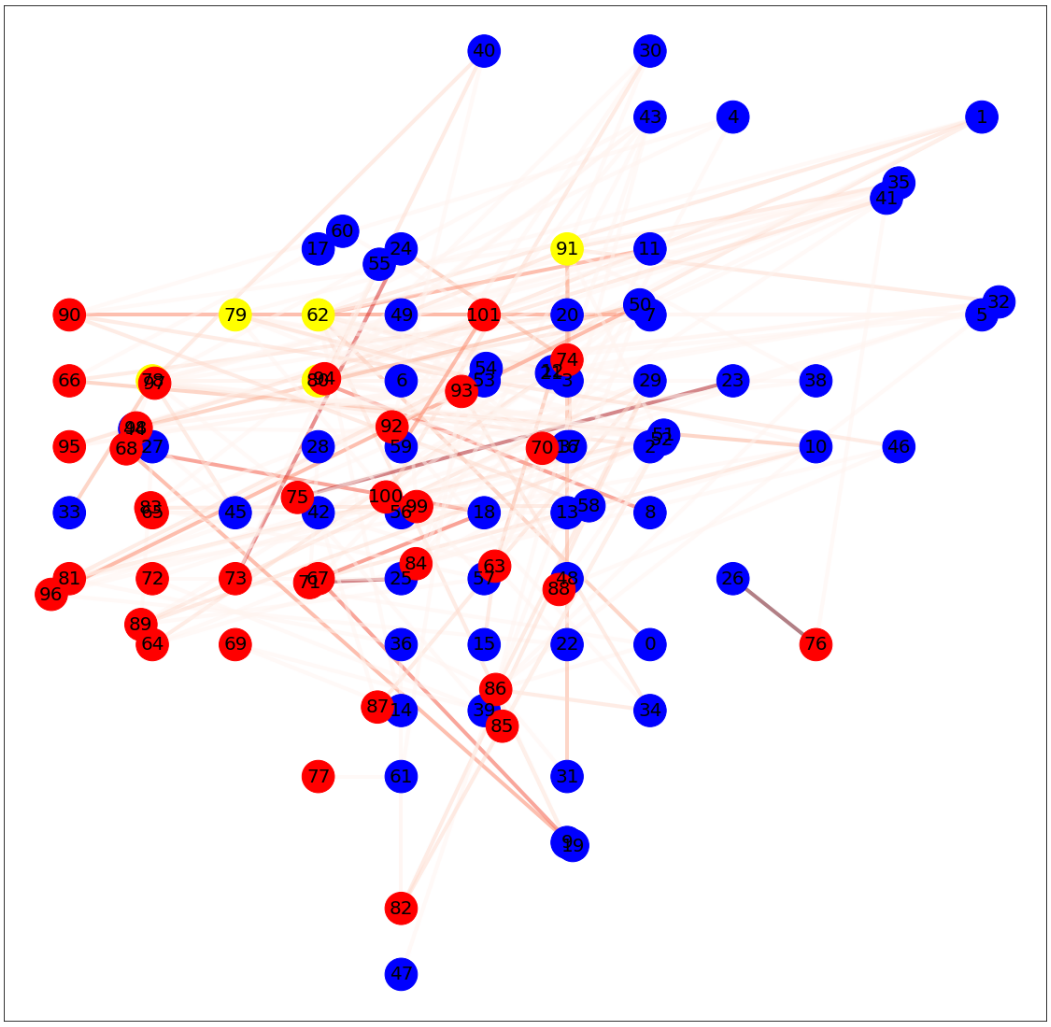


Figure A.4: GH4D drawing of the Hollywood Film Music Dataset, we can see attribute proximity preservation, as a clear separation is visible between the composers and producers (red and blue), as well as the top 1.5% grossing producers (yellow)

Bibliography

[1] C. Nobre, M. D. Meyer, M. Streit, and A. Lex, "The state of the art in visualizing multivariate networks," *Computer Graphics Forum*, vol. 38, 2019. [Online]. Available: `https://api.semanticscholar.org/CorpusID:199020476`.

[2] H. Neuweger, M. Persicke, S. P. Albaum, *et al.*, "Open access bmc systems biology visualizing post genomics data-sets on customized pathway maps by prometra – aeration-dependent gene expression and metabolism of corynebacterium glutamicum as an example." [Online]. Available: `https://api.semanticscholar.org/CorpusID:4039016`.

[3] N. Gehlenborg, S. I. O'Donoghue, N. S. Baliga, *et al.*, "Visualization of omics data for systems biology," *Nature Methods*, vol. 7, S56–S68, 2010. [Online]. Available: `https://api.semanticscholar.org/CorpusID:205419270`.

[4] D. Sun and K. Wong, "On evaluating the layout of uml class diagrams for program comprehension," in *13th International Workshop on Program Comprehension (IWPC'05)*, 2005, pp. 317–326. DOI: `10.1109/WPC.2005.26`.

[5] D. Reniers, L. Voinea, O. Ersoy, and A. C. Telea, "The solid* toolset for software visual analytics of program structure and metrics comprehension: From research prototype to product," *Sci. Comput. Program.*, vol. 79, pp. 224–240, 2014. [Online]. Available: `https://api.semanticscholar.org/CorpusID:29676923`.

[6] J. Trümper, A. C. Telea, and J. Döllner, "Viewfusion: Correlating structure and activity views for execution traces," in *TPCG*, 2012. [Online]. Available: `https://api.semanticscholar.org/CorpusID:2976046`.

[7] M. Lissandrini, D. Mottin, T. Palpanas, and Y. Velegrakis, "Data exploration using example-based methods," *Synthesis Lectures on Data Management*, 2018. [Online]. Available: `https://api.semanticscholar.org/CorpusID:70180158`.

[8] M. Lissandrini, T. B. Pedersen, K. Hose, and D. Mottin, "Knowledge graph exploration: Where are we and where are we going?" *SIGWEB Newsl.*, vol. 2020, no. Summer, Jul. 2020, ISSN: 1931-1745. DOI: `10.1145/3409481.3409485`. [Online]. Available: `https://doi.org/10.1145/3409481.3409485`.

[9] M. E. J. Newman, "Networks: An introduction," 2010. [Online]. Available: `https://api.semanticscholar.org/CorpusID:60557556`.

[10] A.-Ĺ. Barabási, "Network science," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 371,

2016. [Online]. Available: https://api.semanticscholar.org/CorpusID:41085411.

[11] J. S. Aguilar, A. Bassolas, G. Ghoshal, *et al.*, "Impact of urban structure on infectious disease spreading," *Scientific Reports*, vol. 12, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:247360820.

[12] S. Mimar, D. Soriano-Paños, A. Kirkley, *et al.*, "Connecting intercity mobility with urban welfare," *PNAS Nexus*, vol. 1, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:252127881.

[13] Y. Su, S. Yang, H. Sun, *et al.*, "Exploiting relevance feedback in knowledge graph search," *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2015. [Online]. Available: https://api.semanticscholar.org/CorpusID:7667829.

[14] T. Milo and A. Somech, "Automating exploratory data analysis via machine learning: An overview," *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:218982093.

[15] G. I. Diaz, M. Arenas, and M. Benedikt, "Sparqlbye: Querying rdf data by example," *Proc. VLDB Endow.*, vol. 9, pp. 1533–1536, 2016. [Online]. Available: https://api.semanticscholar.org/CorpusID:15983457.

[16] L. Liao, X. He, H. Zhang, and T.-S. Chua, "Attributed social network embedding," *IEEE Trans. on Knowl. and Data Eng.*, vol. 30, no. 12, pp. 2257–2270, Dec. 2018, ISSN: 1041-4347. DOI: 10.1109/TKDE.2018.2819980. [Online]. Available: https://doi.org/10.1109/TKDE.2018.2819980.

[17] G. D. Battista, P. Eades, R. Tamassia, and I. G. Tollis, *Graph Drawing: Algorithms for the Visualization of Graphs*, 1st. USA: Prentice Hall PTR, 1998, ISBN: 0133016153.

[18] C. Chen, "Information visualization: Beyond the horizon," 2006. [Online]. Available: https://api.semanticscholar.org/CorpusID:61131514.

[19] C. Bennett, J. Ryall, L. Spalteholz, and A. A. Gooch, "The aesthetics of graph visualization," in *International Symposium on Computational Aesthetics in Graphics, Visualization, and Imaging*, 2007. [Online]. Available: https://api.semanticscholar.org/CorpusID:15591569.

[20] H. Gibson, J. Faith, and P. Vickers, "A survey of two-dimensional graph layout techniques for information visualisation," *Information Visualization*, vol. 12, pp. 324–357, 2013. [Online]. Available: https://api.semanticscholar.org/CorpusID:18722099.

[21] S.-H. Cheong and Y.-W. Si, "Force-directed algorithms for schematic drawings and placement: A survey," *Information Visualization*, vol. 19,

pp. 65–91, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:59342122.

[22] S. van den Elzen and J. J. van Wijk, "Multivariate network exploration and presentation: From detail to overview via selections and aggregations," *IEEE Transactions on Visualization and Computer Graphics*, vol. 20, no. 12, pp. 2310–2319, 2014. DOI: 10.1109/TVCG.2014.2346441.

[23] D. Guo, "Flow mapping and multivariate visualization of large spatial interaction data," *IEEE Transactions on Visualization and Computer Graphics*, vol. 15, no. 6, pp. 1041–1048, 2009. DOI: 10.1109/TVCG.2009.143.

[24] B. Shneiderman and A. Aris, "Network visualization by semantic substrates," *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 5, pp. 733–740, 2006. DOI: 10.1109/TVCG.2006.166.

[25] Y. Wang, Y. Wang, Y. Sun, *et al.*, "Revisiting stress majorization as a unified framework for interactive constrained graph visualization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 24, pp. 489–499, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:1582708.

[26] F. van Ham and B. E. Rogowitz, "Perceptual organization in user-generated graph layouts," *IEEE Transactions on Visualization and Computer Graphics*, vol. 14, 2008. [Online]. Available: https://api.semanticscholar.org/CorpusID:10444314.

[27] H. Gibson and J. Faith, "Node-attribute graph layout for small-world networks," *2011 15th International Conference on Information Visualisation*, pp. 482–487, 2011. [Online]. Available: https://api.semanticscholar.org/CorpusID:16438208.

[28] M. Dörk, S. Carpendale, and C. L. Williamson, "Edgemaps: Visualizing explicit and implicit relations," in *Electronic imaging*, 2011. [Online]. Available: https://api.semanticscholar.org/CorpusID:357910.

[29] C. Ding, X. He, H. Zha, M. Gu, and H. Simon, "A min-max cut algorithm for graph partitioning and data clustering," in *Proceedings 2001 IEEE International Conference on Data Mining*, 2001, pp. 107–114. DOI: 10.1109/ICDM.2001.989507.

[30] F. Chen, Y. C. Wang, B. Wang, and C.-C. J. Kuo, "Graph representation learning: A survey," *APSIPA Transactions on Signal and Information Processing*, vol. 9, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:202541524.

[31] I. A. Chikwendu, X. Zhang, I. O. Agyemang, I. Adjei-Mensah, U. C. Chima, and C. J. Ejiyi, "A comprehensive survey on deep graph representation learning methods," *J. Artif. Int. Res.*, vol. 78, Jan. 2024, ISSN: 1076-9757. DOI: 10.1613/jair.1.14768. [Online]. Available: https://doi.org/10.1613/jair.1.14768.

[32] P. Cui, X. Wang, J. Pei, and W. Zhu, "A survey on network embedding," *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 5, pp. 833–852, 2019. DOI: 10.1109/TKDE.2018.2849727.

[33] A. C. Gilbert and K. Levchenko, "Compressing network graphs," 2004. [Online]. Available: https://api.semanticscholar.org/CorpusID:12451527.

[34] 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*, ser. KDD '14, New York, New York, USA: Association for Computing Machinery, 2014, pp. 701–710, ISBN: 9781450329569. DOI: 10.1145/2623330.2623732. [Online]. Available: https://doi.org/10.1145/2623330.2623732.

[35] 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*, ser. KDD '16, San Francisco, California, USA: Association for Computing Machinery, 2016, pp. 855–864, ISBN: 9781450342322. DOI: 10.1145/2939672.2939754. [Online]. Available: https://doi.org/10.1145/2939672.2939754.

[36] D. Duvenaud, D. Maclaurin, J. Aguilera-Iparraguirre, *et al.*, "Convolutional networks on graphs for learning molecular fingerprints," in *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, ser. NIPS'15, Montreal, Canada: MIT Press, 2015, pp. 2224–2232.

[37] M. Zhang and Y. Chen, "Link prediction based on graph neural networks," in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, ser. NIPS'18, Montréal, Canada: Curran Associates Inc., 2018, pp. 5171–5181.

[38] J. You, R. Ying, and J. Leskovec, "Position-aware graph neural networks," in *International Conference on Machine Learning*, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:174800449.

[39] T. Lei, W. Jin, R. Barzilay, and T. Jaakkola, "Deriving neural architectures from sequence and graph kernels," in *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ser. ICML'17, Sydney, NSW, Australia: JMLR.org, 2017, pp. 2024–2033.

[40] T. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *ArXiv*, vol. abs/1609.02907, 2016. [Online]. Available: https://api.semanticscholar.org/CorpusID:3144218.

[41] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17, Long Beach, California, USA: Curran Associates Inc., 2017, pp. 1025–1035, ISBN: 9781510860964.

[42] S. Pan, J. Wu, X. Zhu, C. Zhang, and Y. Wang, "Tri-party deep network representation," in *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, ser. IJCAI'16, New York, New York, USA: AAAI Press, 2016, pp. 1895–1901, ISBN: 9781577357704.

[43] 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*, ser. KDD '16, San Francisco, California, USA: Association for Computing Machinery, 2016, pp. 1225–1234, ISBN: 9781450342322. DOI: 10.1145/2939672.2939753. [Online]. Available: https://doi.org/10.1145/2939672.2939753.

[44] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "Line: Large-scale information network embedding," in *Proceedings of the 24th International Conference on World Wide Web*, ser. WWW '15, Florence, Italy: International World Wide Web Conferences Steering Committee, 2015, pp. 1067–1077, ISBN: 9781450334693. DOI: 10.1145/2736277.2741093. [Online]. Available: https://doi.org/10.1145/2736277.2741093.

[45] Y. Weiss, A. Torralba, and R. Fergus, "Spectral hashing," in *Neural Information Processing Systems*, 2008. [Online]. Available: https://api.semanticscholar.org/CorpusID:268118396.

[46] R. Salakhutdinov and G. E. Hinton, "Semantic hashing," *Int. J. Approx. Reason.*, vol. 50, pp. 969–978, 2009. [Online]. Available: https://api.semanticscholar.org/CorpusID:1501682.

[47] H. Xia, N. Gao, J. Peng, J. Mo, and J. Wang, "Binarized attributed network embedding via neural networks," in *2020 International Joint Conference on Neural Networks (IJCNN)*, 2020, pp. 1–8. DOI: 10.1109/IJCNN48605.2020.9206717.

[48] H. Yang, S. Pan, L. Chen, C. Zhou, and P. Zhang, "Low-bit quantization for attributed network representation learning," in *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, ser. IJCAI'19, Macao, China: AAAI Press, 2019, pp. 4047–4053, ISBN: 9780999241141.

[49] W. Wu, B. Li, L. Chen, and C. Zhang, "Efficient attributed network embedding via recursive randomized hashing," in *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, ser. IJCAI'18, Stockholm, Sweden: AAAI Press, 2018, pp. 2861–2867, ISBN: 9780999241127.

[50] I. T. Jolliffe and J. Cadima, "Principal component analysis: A review and recent developments," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 374, 2016. [Online]. Available: https://api.semanticscholar.org/CorpusID:20101754.

[51] L. van der Maaten and G. E. Hinton, "Visualizing data using t-sne," *Journal of Machine Learning Research*, vol. 9, pp. 2579–2605, 2008. [On-

line]. Available: https://api.semanticscholar.org/CorpusID:5855042.

[52] J. F. Kruiger, P. E. Rauber, R. M. Martins, A. Kerren, S. G. Kobourov, and A. C. Telea, "Graph layouts by t-sne," *Computer Graphics Forum*, vol. 36, 2017. [Online]. Available: https://api.semanticscholar.org/CorpusID:1235159.

[53] M. Zhu, W. Chen, Y. Hu, Y. Hou, L. Liu, and K. Zhang, "Drgraph: An efficient graph layout algorithm for large-scale graphs by dimensionality reduction," *IEEE Transactions on Visualization and Computer Graphics*, vol. 27, no. 2, pp. 1666–1676, 2021. DOI: 10.1109/TVCG.2020.3030447.

[54] C. Plant, S. Biedermann, and C. Böhm, "Data compression as a comprehensive framework for graph drawing and representation learning," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, ser. KDD '20, Virtual Event, CA, USA: Association for Computing Machinery, 2020, pp. 1212–1222, ISBN: 9781450379984. DOI: 10.1145/3394486.3403174. [Online]. Available: https://doi.org/10.1145/3394486.3403174.

[55] S. van den Elzen, D. Holten, J. Blaas, and J. J. van Wijk, "Reducing snapshots to points: A visual analytics approach to dynamic network exploration," *IEEE Transactions on Visualization and Computer Graphics*, vol. 22, no. 1, pp. 1–10, 2016. DOI: 10.1109/TVCG.2015.2468078.

[56] L. Giovannangeli, F. Lalanne, D. Auber, R. Giot, and R. Bourqui, "Deep neural network for drawing networks, (dnn)2," *ArXiv*, vol. abs/2108.03632, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:236956766.

[57] Y. Y. Leow, T. Laurent, and X. Bresson, "Graphtsne: A visualization technique for graph-structured data," *ArXiv*, vol. abs/1904.06915, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:119309096.

[58] C. Böhm and C. Plant, "Massively parallel graph drawing and representation learning," in *2020 IEEE International Conference on Big Data (Big Data)*, 2020, pp. 609–616. DOI: 10.1109/BigData50022.2020.9377976.

[59] M. Charikar, "Similarity estimation techniques from rounding algorithms," in *Symposium on the Theory of Computing*, 2002. [Online]. Available: https://api.semanticscholar.org/CorpusID:4229473.

[60] N. Kraus, D. Carmel, I. Keidar, and M. Orenbach, "Nearbucket-lsh: Efficient similarity search in p2p networks," *ArXiv*, vol. abs/1511.07148, 2015. [Online]. Available: https://api.semanticscholar.org/CorpusID:7624204.

[61] A. Gionis, P. Indyk, and R. Motwani, "Similarity search in high dimensions via hashing," in *Very Large Data Bases Conference*, 1999. [Online]. Available: https://api.semanticscholar.org/CorpusID:1578969.

[62] R. da Silva Villaça, L. B. de Paula, R. Pasquini, and M. F. Magalhães, "A similarity search system based on the hamming distance of social profiles," *2013 IEEE Seventh International Conference on Semantic Computing*, pp. 90–93, 2013. [Online]. Available: `https://api.semanticscholar.org/CorpusID:1243829`.

[63] Z. Wu and M. Zou, "An incremental community detection method for social tagging systems using locality-sensitive hashing," *Neural networks : the official journal of the International Neural Network Society*, vol. 58, pp. 14–28, 2014. [Online]. Available: `https://api.semanticscholar.org/CorpusID:2108996`.

[64] M. A. Abdulhayoglu and B. Thijs, "Use of locality sensitive hashing (lsh) algorithm to match web of science and scopus," *Scientometrics*, vol. 116, pp. 1229–1245, 2018. [Online]. Available: `https://api.semanticscholar.org/CorpusID:17863626`.

[65] O. Jafari, P. Maurya, P. Nagarkar, K. M. Islam, and C. Crushev, "A survey on locality sensitive hashing algorithms and their applications," *ArXiv*, vol. abs/2102.08942, 2021. [Online]. Available: `https://api.semanticscholar.org/CorpusID:231942424`.

[66] J. Johnson, M. Douze, and H. Jégou, "Billion-scale similarity search with gpus," *IEEE Transactions on Big Data*, vol. 7, pp. 535–547, 2017. [Online]. Available: `https://api.semanticscholar.org/CorpusID:926364`.

[67] W. Wu, B. Li, C. Luo, and W. Nejdl, "Hashing-accelerated graph neural networks for link prediction," *Proceedings of the Web Conference 2021*, 2021. [Online]. Available: `https://api.semanticscholar.org/CorpusID:235254058`.

[68] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher, "Min-wise independent permutations (extended abstract)," in *Symposium on the Theory of Computing*, 1998. [Online]. Available: `https://api.semanticscholar.org/CorpusID:53307334`.

[69] A. A. Hagberg, D. A. Schult, P. Swart, and J. Hagberg, "Exploring network structure, dynamics, and function using networkx," 2008. [Online]. Available: `https://api.semanticscholar.org/CorpusID:16050699`.

[70] C. R. Harris, K. J. Millman, S. van der Walt, *et al.*, "Array programming with numpy," *Nature*, vol. 585, pp. 357–362, 2020. [Online]. Available: `https://api.semanticscholar.org/CorpusID:219792763`.

[71] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[72] P. Virtanen, R. Gommers, T. E. Oliphant, *et al.*, "Scipy 1.0: Fundamental algorithms for scientific computing in python," *Nature Methods*, vol. 17, pp. 261–272, 2019. [Online]. Available: `https://api.semanticscholar.org/CorpusID:198229805`.

[73] H. Chen, S. F. Sultan, Y. Tian, M. Chen, and S. Skiena, "Fast and accurate network embeddings via very sparse random projection," in *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, ser. CIKM '19, Beijing, China: Association for Computing Machinery, 2019, pp. 399–408, ISBN: 9781450369763. DOI: 10.1145/3357384.3357879. [Online]. Available: https://doi.org/10.1145/3357384.3357879.

[74] S. S. Vempala, "The random projection method," in *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 2005. [Online]. Available: https://api.semanticscholar.org/CorpusID:41509000.

[75] P. Li, T. J. Hastie, and K. W. Church, "Very sparse random projections," in *Knowledge Discovery and Data Mining*, 2006. [Online]. Available: https://api.semanticscholar.org/CorpusID:7995734.

[76] Y. Wang, Y. Wang, Y. Sun, *et al.*, "Revisiting stress majorization as a unified framework for interactive constrained graph visualization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 24, no. 1, pp. 489–499, 2018. DOI: 10.1109/TVCG.2017.2745919.

[77] Y. Tanahashi, C.-H. Hsueh, and K.-L. Ma, "An efficient framework for generating storyline visualizations from streaming data," *IEEE Transactions on Visualization and Computer Graphics*, vol. 21, no. 6, pp. 730–742, 2015. DOI: 10.1109/TVCG.2015.2392771.

[78] A. Nocaj, M. Ortmann, and U. Brandes, "Untangling the hairballs of multi-centered, small-world online social media networks," *J. Graph Algorithms Appl.*, vol. 19, pp. 595–618, 2015. [Online]. Available: https://api.semanticscholar.org/CorpusID:37926089.

[79] K. Avrachenkov, A. Y. Kondratev, and V. V. Mazalov, "An information flow model for conflict and fission in small groups," *Journal of Anthropological Research*, vol. 33, pp. 452–473, 1977. [Online]. Available: https://api.semanticscholar.org/CorpusID:197843028.

[80] D. E. Knuth, "The stanford graphbase - a platform for combinatorial computing," 1993. [Online]. Available: https://api.semanticscholar.org/CorpusID:9370314.

[81] M. Girvan and M. E. J. Newman, "Community structure in social and biological networks," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 99, pp. 7821–7826, 2001. [Online]. Available: https://api.semanticscholar.org/CorpusID:2444655.

[82] J. Ryan and R. R. Faulkner, "Music on demand: Composers and careers in the hollywood film industry.," *Social Forces*, vol. 63, p. 589, 1984. [Online]. Available: https://api.semanticscholar.org/CorpusID:111371621.

[83] T. M. J. Fruchterman and E. M. Reingold, "Graph drawing by force-directed placement," *Software: Practice and Experience*, vol. 21, 1991. [Online]. Available: https://api.semanticscholar.org/CorpusID:31468174.

[84] D. Harel and Y. Koren, "Journal of graph algorithms and applications a fast multi-scale method for drawing large graphs." [Online]. Available: `https://api.semanticscholar.org/CorpusID:3981903`.

[85] Y.-X. Wang, Z. Li, L. Yao, W. Cao, and Z. Wang, "Two improved gpu acceleration strategies for force-directed graph layout," *2010 International Conference on Computer Application and System Modeling (ICCASM 2010)*, vol. 13, pp. V13-132-V13–136, 2010. [Online]. Available: `https://api.semanticscholar.org/CorpusID:17345678`.

[86] H. C. Purchase, "Metrics for graph drawing aesthetics," *J. Vis. Lang. Comput.*, vol. 13, pp. 501–516, 2002. [Online]. Available: `https://api.semanticscholar.org/CorpusID:5895591`.

[87] G. J. Mooney, H. C. Purchase, M. Wybrow, and S. G. Kobourov, "The multi-dimensional landscape of graph drawing metrics," *2024 IEEE 17th Pacific Visualization Conference (PacificVis)*, pp. 122–131, 2024. [Online]. Available: `https://api.semanticscholar.org/CorpusID:266739057`.

[88] J. Venna and S. Kaski, "Visualizing gene interaction graphs with local multidimensional scaling," in *The European Symposium on Artificial Neural Networks*, 2006. [Online]. Available: `https://api.semanticscholar.org/CorpusID:12239921`.