university of groningen / faculty of mathematics and natural sciences / Joh. Bernoulli institute

Graph Layouts by t-distributed Stochastic Neighbour Embedding

MSc thesis in Computing Science

3rd September 2016

Student: J.F. Kruiger

First supervisor: Prof. dr. A.C. Telea

Second supervisor: P.E. Rauber

In this thesis, the use of the t-distributed Stochastic Neighbour Embedding (t-SNE) multidimensional projection technique for producing graph layouts is evaluated. t-SNE is typically used for visualizing high-dimensional data as two- or three-dimensional scatter plots. In our work, graphs are represented as high-dimensional datasets, and are projected to two-dimensional layouts using a suitably adapted t-SNE method. The novel graph layouts are visually compared with layouts obtained by current state-of-the-art methods. The comparison shows positive results that support the suitability claim of using t-SNE for producing graph layouts.

CONTENTS

1 INTRODUCTION

1.1 MOTIVATION

During the first few years of the 21st century, the world has seen an enormous growth in the amount of available data. The methods, tools, and knowledge to apprehend and getting insight into this data have seen a growth as well, but there is still much room for improvement.

There are different types of data. Examples include multivariate data, time-dependent data, relational data, or combinations of these. This thesis will focus mostly on relational data visualization, but uses approaches from multivariate data visualization.

In this thesis, the usage of the t-distributed Stochastic Neighbour Embedding (t-SNE) algorithm for the construction of visual representations of relational data, also known as graph layouts, is studied. This work has several motivations. First, as stated earlier, getting insight in various types of data, such as relational (graph) data, is an interesting and useful goal by itself. Secondly, and more specific to our work, the use of techniques for processing *multidimensional* data in order to treat *relational* data can lead to new ways of exploring data, by building bridges between the different classes of data processing and data visualization algorithms out there. Last but not least, our specific focus on t-SNE is motivated by the high prominence of this algorithm in the machine learning and data visualization communities. [1]

The rest of this chapter is structured as follows. Section 1.2 gives a quick overview of the context and scope of graph layouts. Section 1.3 gives an overview of the aims and scope of dimensionality reduction, the class of techniques we use to approach the graph layout problem. Section 1.4 makes the connection between graph layouts and dimensionality reduction, and poses the central research question.

1.2 GRAPHS

Relational data can be represented mathematically by using the concept of *graphs*. A graph is a structure that describes entities and relations between those entities.

The number of relations and entities in the graph can vary greatly. Especially for graphs with many entities and relations it can be hard to make a visualization of the graph in an insightful manner. For a node-link visualization of the graph to succeed, it needs an appropriate set of coordinates for the nodes in the network, which is part of the *graph layout*.

A graph G can be represented by a set of *vertices* V and a set of *edges* E.

$$G = (V, E)$$

Vertices are the entities in the network, and edges describe relations between entities in the network. The set of vertices is defined as follows:

$$V = \left\{ v_1, v_2, \ldots v_{|V|} \right\},$$

where |.| denotes the cardinality or size of a set.

For an undirected unweighted graph, set of edges is defined as follows:

$$E = \left\{ e_1, e_2, \dots e_{|E|} \right\} \qquad e_i = \left\{ v_j, v_k \right\}.$$

For a directed graph, the edges e_i would have to be defined as *tuples* to indicate the direction of the relation. For a weighted graph, a weight is associated to every edge, which specifies the importance or type of relation that the edge represents.

In this thesis, only undirected unweighted graphs are considered to keep the discussion simple. Naturally, the approach can be generalized to directed or weighted graphs.

1.2.1 *Toy example: Co-star network*

As an example of a graph, one could consider a graph that describes actors who co-star in movies. The vertices represent actors. An edge $e = \left\{ v_i, v_j \right\}$ represents that the two actors corresponding to v_i and v_j co-star in a movie.

Suppose we have four actors: Daniel, Emma, Rupert and Logan:

$$V = \{v_d, v_e, v_r, v_l\}. \tag{1}$$

Daniel co-stars with Emma and Rupert, Emma co-stars with Daniel, Rupert and Logan. Rupert co-stars with Daniel and Emma, while Logan only co-stars with Emma. So we have the following set of edges:

$$E = \{\{v_d, v_e\}, \{v_d, v_r\}, \{v_e, v_r\}, \{v_e, v_l\}\}.$$

All information from a graph can also be captured in a $|V| \times |V|$ matrix $A = \left(a_{ij}\right)$ called the adjacency matrix. For an unweighted graph, $a_{ij} = 1$ iff v_i and v_j are connected. For an undirected graph, $A^T = A$. For our co-star network, we have the following adjacency matrix:

$$A = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix},$$

where the rows and columns correspond to the vertices in the same order as in Equation (1).

1.2.2 *Graph layouts*

As explained, a graph can be represented as an adjacency matrix. However, for the purpose of understanding the structure of the graph, this representation is not optimal. Instead, graphs are most frequently visually represented by so-called node-link diagrams. Figure 1 shows the node-link diagram corresponding to the toy example discussed above. The structure of the graph, in terms of the connectivity pattern and the nodes it involves, is interpretable at a glance.

Constructing node-link diagrams, also called graph drawings, requires (in its simplest form) a way to assign 2D positions to all nodes in the graph, after which edges can be drawn as straight lines between the corresponding

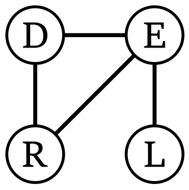


Figure 1: A drawing of the co-star network.

nodes. This process is referred to as *graph embedding* or producing a *graph layout*.

Many algorithms exist for computing layouts of graphs. These algorithms range from techniques focused on treating large graphs of hundreds of thousands of edges and nodes, up to very specialized techniques that treat smaller graphs but focus on creating specific drawing styles in terms of the arrangement of nodes in the visual space. Producing graph layouts is in itself a hard (and still open) problem, for several reasons. First, as the graph size increases, and assuming that we keep the visual space at a constant size, the density of information increases, which can easily lead to overlapping nodes and edges, or other areas of high-density information, referred to as *visual clutter*. Secondly, the computational complexity of several graph layout algorithms is superlinear in the graph size, which leads to methods which cannot efficiently be used for large graphs. Last but not least, several so-called aesthetic criteria can be thought of for a 'good' graph drawing; however, it is not evident how to encode these criteria into algorithms, and often it is also the case that several such criteria compete against each other.

1.3 DIMENSIONALITY REDUCTION

Multidimensional (or multivariate) data can be explained as a set of observations, where each observation has several measured quantities (also called dimensions, features or variables). Often, a multidimensional dataset is intuitively represented by a matrix where every row encodes an observation, and every column encodes a dimension.

When the number of dimensions is low, say under 5, such data can be explored fully by using direct visualization algorithms. For example, a 4-dimensional dataset can be visualized by a coloured 3D scatterplot, where the first three dimensions of the data are represented by the coordinates of the points, and the fourth dimension is represented by the colours of the points. As the dimensionality of the dataset becomes considerably higher, visualization becomes challenging, since there are insufficient direct visual channels to represent all data dimensions.

Dimensionality reduction (DR) is a class of data-processing techniques that, among other goals, aims to address the above problem by reducing the dimensionality of the data. DR takes as input a high-dimensional dataset, and gives as output a (typically much) lower-dimensional dataset with the same observations. The aim is to preserve relevant data patterns; such as groups, clusters, or correlations; during this process. Aside from addressing the visualization problem of high-dimensional data, DR is also useful for other tasks, such as reducing storage costs, or accelerating further data processing.

Within the class of DR algorithms, stochastic neighbour embedding (SNE), and its t-distributed variant (t-SNE), have gained high prominence and appreciation in the data processing, data mining, machine learning, and information visualization communities. [1] Particularly for visualization purposes, t-SNE is very successful in preserving the local structure of data within clusters of data, and in isolated observations. As such, and considering its relative novelty, t-SNE is an interesting technique to consider in the context of producing graph layouts by means of DR techniques.

Section 2.2 gives an overview of related work currently in the field of DR, and gives a more complete explanation of t-SNE.

1.4 RESEARCH QUESTION

From the previous sections, we can draw some interesting (and useful) parallels between dimensionality reduction and graph layouts, as follows.

First, both techniques can be thought of as an 'embedding' process which takes as input a dataset lying in some high-dimensional, or otherwise symbolic or abstract, space, and deliver a dataset whose observations are points in a low-dimensional (e.g., 2D) space. For dimensionality reduction, this is by definition. For graph layouts, we can think of a graph as a set of observations (nodes) lying in some symbolic data space, where edges are seen as encoding similarities, or distances, between the respective nodes. Separately, a 2D graph drawing is, obviously, a dataset having the same number of observations (nodes), and two dimensions per observation. Finally, in a graph drawing, connected nodes should be arguably placed close to each other so as to minimize drawing clutter, which can be seen as a parallel to DR's attempt to preserve the data structure.

Secondly, both techniques are subject to quality measures that attempt to encode desirable features of a visualization, as perceived by the user. For DR, a good embedding is one in which a user can easily perceive groups of strongly related entities and, separately, outliers. For graph layouts, a good layout is one where the user can easily perceive the structure of the graph in terms of strongly connected components. Separately, both techniques are based on heuristics which attempt to model what the user finds important to be delivered in the final drawing.

As such, it is interesting to consider whether such similarities between DR and graph layouting can be exploited. This can be done in two directions. First, one could consider using graph layout methods to reduce the dimensionality of datasets. We leave this direction out of our exploration. Secondly, one could consider using DR methods to produce good-quality layouts of graphs. This is the focus of our thesis.

Therefore, our central research question states: Can we use the t-SNE dimensionality reduction method to generate layouts of real-world graphs that have similar (or higher) quality, in terms of well-established graph-drawing quality aspects?

This question is approached in the remainder of the thesis as follows. Chapter 2 covers related work both in graph drawing and dimensionality reduction, and thereby establishes the context in which our research question operates. Chapter 3, the core of our work, introduces in detail a new graph layout method based on the t-SNE algorithm. Chapter 4 presents an extensive set of results (graph drawings) generated by our method for real-world graphs, and compares these with graph layouts obtained by other well-known layout algorithms. Chapter 5 discusses our results. Chapter 6 concludes the thesis and outlines directions for future work.

2 RELATED WORK

This chapter gives an overview of the current state of the fields in which this thesis operates. Section 2.1 covers the field of graph layouts, and Section 2.2 covers the field of dimensionality reduction. Section 2.3 connects the two fields, and shows that there is an unexplored gap that is the focus of this thesis.

2.1 GRAPH LAYOUTS

Perhaps the most crucial part of a drawing as in Figure 1 is the set of coordinates associated with the vertices. For straight-line graph layouts, where the edges are represented simply by lines, this set of coordinates fully determines the layout. For other types of graph layouts, edges can have control points which are also part of the layout. However, we will focus the discussion to straight-line graph layouts. In the case of Figure 1 the network was sufficiently small, so the coordinates have been manually determined.

The survey of the field of graph layouts presented here is quite minimal, and far from complete. The reason for this is that our approach is quite novel, and a complete taxonomy of the field does not bring much additional value to the discussion. For a more complete survey of the field, see [2].

In general, we can define the straight-line layout of a graph G as follows:

$$\Lambda(G) = \left\{\vec{y}_1, \vec{y}_2, \ldots \vec{y}_{|V|}\right\},$$

where $\vec{y}_i \in \mathbb{R}^m$ is the m-dimensional coordinate associated with vertex v_i. In this thesis only $m = 2$ is considered, but one could consider $m \neq 2$ without loss of generality.

The problem that is addressed in this thesis is how to determine $\Lambda(G)$. More specifically, to explore how t-SNE can be used to determine $\Lambda(G)$.

There has been a great deal of research on obtaining aesthetically pleasing graph layouts. For example, arc diagrams[3] as in Figure 2 have been around since the 1960s, and work has been done to find the most optimal ordering of the vertices that results in the least number of edge crossings. For these diagrams, the ordering of the vertices can be seen as a one-dimensional layout.

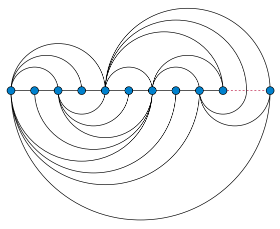


Figure 2: Arc diagram.

Force-based algorithms[4][5] model attracting and repelling forces between the vertices, and simulate a model of the system to obtain an equilibrium. Algorithms of this type can be reformulated as optimization problems, where an objective function is minimized. Users can change the parameters or add additional forces to the model in order to meet their criteria.

Many layout algorithms can benefit from the multilevel paradigm[6], where groups of vertices are represented by single placeholder vertices. The foremost advantages of this paradigm are better scalability[7], and fewer edge crossings caused by better local optima.

Moreover, there are algorithms that combine more of the previously mentioned ideas[8], and tools that enable users to experiment with many different algorithms.[9]

A graph can also be seen as an n-dimensional dataset with N observations, where $N = n$ is the number of vertices in the graph. Every observation describes the connectivity of a vertex to all other vertices. Dimensionality reduction can be used for visualizing such datasets. As we will discuss in Section 3.1, there are different options for the way of encoding connectivity.

2.2 DIMENSIONALITY REDUCTION

Dimensionality reduction (DR) is an important field in data visualization and machine learning. Given a set of n-dimensional observations or data points

$$X = \{\vec{x}_1, \vec{x}_2, \dots \vec{x}_N\}, \quad \vec{x}_i \in \mathbb{R}^n$$

the aim of DR is to find a set Y of m-dimensional data points with $m < n$ that represents X as well as possible:

$$Y = \{\vec{y}_1, \vec{y}_2, \dots \vec{y}_N\}, \quad \vec{y}_i \in \mathbb{R}^m .$$

Typically, for visualization purposes, $m = 2$ or $m = 3$ is considered.

Depending on the input data, and the purpose of the visualization, there are many strategies to consider.

A straightforward strategy is to simply eliminate certain features, and only retain a subset of the original features. The *wrapper* approach [10] is a forward selection approach used in machine learning for selecting relevant features in a feature space. Features are selected one-by-one, based on some objective function.

Arguably more interesting strategies do in general not use the same features as the input space, but rather (non-)linear combinations of the features. Principal Component Analysis (PCA)[11] is a well-known approach for DR that uses linear combinations of the original features, namely the eigenvectors of the covariance matrix.

We are often interested in preserving the pairwise distances between data points. Multidimensional Scaling (MDS) is a broad category of DR algorithms that aims to do just this. Torgerson MDS is an algorithm that minimizes an objective function called the *stress* to find an m-dimensional embedding. Minimizing this objective function corresponds to making the distances in $\mathbb{R}^n$ and $\mathbb{R}^m$ as similar as possible. In fact, when using Euclidean distances in the input space, Torgerson MDS is equivalent to PCA.[1]

ISOMAP[12] is a DR algorithm that first determines which data points are neighbours in the n-dimensional space, then determines the geodesic distances between the points in n-dimensional space, and finally performs Torgerson MDS with the pairwise geodesic distances.

Another DR algorithm, LSP[13], works by first projecting a subset of the n-dimensional data points using MDS, and subsequently using neighbourhood information from the n-dimensional space to reconstruct the rest of the data points in the m-dimensional space.

Moreover, the quality of DR techniques can be assessed by using techniques that visualize how well the neighbourhoods and pairwise distances in the input space are preserved in the output space.[14][15]

2.2.1 t-*distributed Stochastic Neighbour Embedding*

t-distributed Stochastic Neighbour Embedding [1], or t-SNE, is a visualization technique that is often used for creating scatter plots of high-dimensional data. This section briefly summarizes the method.

t-SNE is special, since it aims at preserving the local neighbourhoods in the data, while many other DR methods aim at preserving all pairwise

1 However, technically PCA deals with a different problem, because it has as input the n-dimensional data points, while MDS has as input only the pairwise distances.

distances. This aim at preserving local neighbourhoods is useful, since in many use-cases the goal is to reason about clusters of data, and not about the pairwise distances between points that are separated by large distances. For such use-cases, preserving all pairwise distances is not strictly needed. What is needed is a weaker constraint, namely to preserve neighbourhoods.

As mentioned above, t-SNE particularly focuses on accurately representing the *local* structure of the data; if the high-dimensional data points are close to each other, t-SNE aims to make sure that the low-dimensional points are also close to each other. However, if the high-dimensional data points are far away from each other, t-SNE does not enforce this as strictly in the low-dimensional embedding as some other methods do.

t-SNE achieves this by defining probabilities p_{ij} of picking a pair in the high-dimensional space and probabilities q_{ij} of picking a pair in the low-dimensional embedding.

p_{ij}, the probability of picking the pair $\{\vec{x}_i, \vec{x}_j\}$, is the symmetrized version of the conditional probabilities $p_{i|j}$ and $p_{j|i}$:

$$p_{ij} = p_{ji} = \frac{p_{i|j} + p_{j|i}}{2N}, \quad p_{ii} = 0,$$

where the conditional probabilities are given by the normalized Gaussian distribution:

$$p_{j|i} = \frac{\exp(-\frac{d(\vec{x}_i,\vec{x}_j)^2}{2\sigma_i^2})}{\sum\limits_{k \neq i} \exp(-\frac{d(\vec{x}_i,\vec{x}_k)^2}{2\sigma_i^2})},$$

which should be interpreted as the probability that $\vec{x}_i$ picks $\vec{x}_j$ as its neighbour. $d(\vec{x}_i, \vec{x}_j)$ is a distance measure, which is usually the Euclidean distance, but a pre-defined distance matrix is also possible.

σ_i, the standard deviations of the Gaussians, can be either hand-picked, or chosen (commonly with binary search) such that the *perplexity* κ for every data point matches some user-defined value. The perplexity is a measure of the effective number of neighbours a data point has. If $\vec{x}_i$ is in a dense region, the σ_i generally attains a lower value to match the perplexity. In sparser regions, σ_i generally attains a larger value.

q_{ij}, the probability of picking the pair $\{\vec{y}_i, \vec{y}_j\}$ in the low-dimensional space is given by:

$$q_{ij} = q_{ji} = \frac{(1 + \|\vec{y}_i - \vec{y}_j\|^2)^{-1}}{\sum\limits_{k \neq l} (1 + \|\vec{y}_k - \vec{y}_l\|^2)^{-1}}.$$

This is the normalized Student's t-distribution, which is known to be a heavy-tailed distribution.

The positions $\vec{y}_i$ in the low-dimensional embedding are determined by minimizing the Kullback-Leibler divergence between the probabilities of picking pairs of low- and high-dimensional data points with respect to $\vec{y}_i$:

$$C_{kl} = \sum_{\substack{i,j \\ j \neq i}} p_{ij} \log \frac{p_{ij}}{q_{ij}}.$$

The minimization is typically done using gradient descent.

Many results of t-SNE are applauded widely, because they often succeed to capture relevant groups in datasets, while also representing local variations within the groups.

2.3 DIMENSIONALITY REDUCTION FOR GRAPH LAYOUTS

As outlined earlier in Chapter 1, there are several similarities between the problem of computing graph layouts and that of performing dimensionality reduction. However, to our knowledge, there is a single attempt to use DR techniques to produce graph layouts: Martins has recently used LSP[13] and IDMAP[16] to process a similarity matrix that encodes the structure of a graph in terms of pairwise graph-theoretic distances between the graph nodes.[17, Chapter 7] The results show promising possibilities, in the sense that one is able to visually explore the resulting graph layouts to detect important structures such as node clusters and interconnection patterns in the graph.

Our method for computing graph layouts using DR techniques is related in spirit and at high level to the above approach of Martins. However, important differences exist. First, we use a neighbourhood-preserving DR technique (t-SNE), whereas Martins used distance-preserving techniques (LSP, IDMAP). As outlined earlier, these two classes of DR techniques have quite different properties. Moreover, as we argued, neighbourhood preservation appears to be a more useful criterion for generating an easy-to-interpret picture (of a graph or high-dimensional dataset) than distance preservation. Secondly, we explore different ways to encode the graph structure into a distance matrix. Thirdly, Martins' focus has been on using both the graph structure and data attributes of the nodes to compute the layout, whereas our focus is purely on computing layouts of unattributed graphs. Last but not least, we compare the results of our technique with state-of-the-art graph layout methods on an extensive set of graphs of different types. In contrast, Martins presented a less extensive evaluation.

3 METHOD

The approach that is used in this thesis consists of two steps. In the first step, the pairwise distances between all vertices is computed and stored in a matrix (d_{ij}). Subsequently, the matrix (d_{ij}) and the graph G are used to construct an objective function C, which is finally minimized with respect to the two-dimensional coordinates of the vertices $\{\vec{y}_i\}$. See Figure 3 for an illustration.

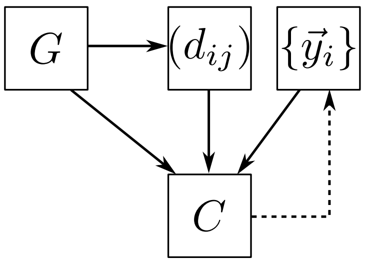


Figure 3: High-level illustration of the method. The dashed line represents that the source is minimized with respect to the target, and thus implicitly resulting in a value for the target.

Section 3.1 elaborates on the distance metrics that are considered in the input space. The cost function and the optimization process are further explained in Section 3.2. Finally, Section 3.3 provides information on how this method was implemented.

3.1 DISTANCE MATRIX

Formally, the input space for t-SNE is a distance matrix, but often the input space of a DR-problem is some high-dimensional space $\mathbb{R}^n$ and in that case the squared Euclidean distance is commonly used to construct a distance matrix.

In our case, the input space is the space of all graphs, which is of a more symbolic nature. Therefore, a convenient graph-theoretical distance metric must be chosen. The challenge is to find a distance metric that reflects the criteria of a good graph layout defined in previous chapters. Therefore, the distance metric should (i) assign a low distance between nodes that we want to place close to each other, and (ii) assign a larger distance between nodes that we want to place far away from each other, or nodes where the relative distance is unimportant. The first observation reflects that we want to make edges shorter, resulting in an easier interpretation of connected nodes, and a smaller chance of overlapping edges. The second observation reflects that we want to prevent overlap of (unrelated) nodes, and that there is some freedom in choosing embeddings for parts of the graph that have weaker relations.

Options that we consider for the distance matrix are the modified adjacency matrix, and the shortest path distance matrix, introduced in Sections 3.1.1 and 3.1.2, respectively. However, depending on the use-case, different, more exotic, distance metrics can be devised.

A comparison of the choice between the two distance matrices that are introduced here is presented in Section 3.1.3.

3.1.1 *Modified adjacency matrix*

As the name suggests, the modified adjacency matrix[17, Section 7.2.1] (*MAM*) is a modified version of the matrix introduced in Section 1.2.1. It is defined as $A' = \left(a'_{ij}\right)$ where $a'_{ij} = \frac{1}{a_{ij}}$ if $a_{ij} \neq 0$ and otherwise $a'_{ij} = k \max_{i,j} a_{ij}$, with a_{ij} the element from the regular adjacency matrix, and $k \geqslant 1$ is a fixed constant.

For unweighted graphs, the modified adjacency matrix reduces to the much simpler definition: $A' = \left(a'_{ij}\right)$ where $a'_{ij} = 1$ if $a_{ij} \neq 0$ and otherwise $a'_{ij} = k$.

This distance metric makes intuitive sense because two vertices that are directly connected by an edge have a small distance of 1, while vertices that are not directly connected have a larger distance k between them. A shortcoming of this distance matrix is that it does not provide much gradation; only the most local connectivity is encoded in this matrix.

3.1.2 *Shortest path distance matrix*

A choice that provides more granularity is the shortest path distance matrix (*SPDM*), where not only the direct neighbours of vertices are considered, but the lengths of paths (in the graph-theoretical sense) from vertices to all other vertices in the graph. The shortest path distance matrix is defined as $S = (s_{ij})$, where s_{ij} is the length of the graph-theoretical shortest path from vertex i to vertex j.

A drawback of the *SPDM* is that it can be expensive to compute the shortest paths for all pairs of vertices. However, there are efficient techniques [18] that can be used to determine it.

3.1.3 MAM *vs.* SPDM*: A comparison*

To compare layouts that use the *MAM* and the *SPDM*, layouts of multiple graphs have been produced with both distance matrices, using the same initial placement of vertices in the optimization. The results are shown in Figure 4. Note that these layouts only use the Kullback-Leibler divergence (introduced in Section 3.2.1) and not the other terms that are introduced (in Sections 3.2.2 to 3.2.4) to prevent unwanted artifacts in the layout.

In the drawings of the layouts, vertices are colour-coded based on the per-vertex cost function. The colourmap `hot` from the `matplotlib.cm`-module has been used; darker colours indicate low costs, and lighter colours indicate higher costs. The cost function is fully introduced in Section 3.2, but what is important here is that a low value (darker colour) indicates a locally better preservation of the neighbourhood, while a high value (lighter colour) indicates a locally worse preservation of the neighbourhood, and thus a worse layout.

Judging by the results in Figure 4, it is clear that the *SPDM* produces superior results. It seems that having only the most local connectivity encoded in the distance matrix is not sufficient to successfully untangle the networks; the *SPDM* produces better (but not perfect) results. Because of this reason, only the *SPDM* is considered from this point forward.

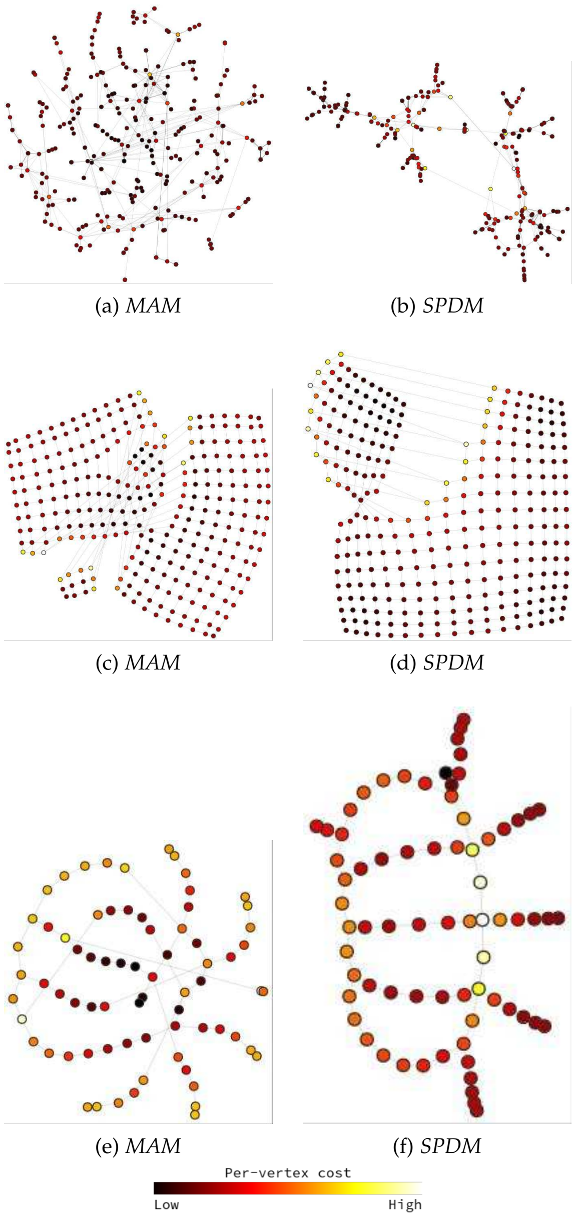


Figure 4: Layouts for different graphs with the *MAM* (left) and the *SPDM* (right) as distance matrix.

3.1.4 *Further modifications to the distance matrix*

As Figure 4 shows, using the SPDM distance produces better results, in terms of preserving the graph structure, than the MDM distance. However, this figure also shows that the SPDM is, by itself, not sufficient in generating a good-quality graph layout. For example, for the graph shown in Figure 4d, we observe a 'tear' in what appears to be a fully regular quad grid structure (which is what the graph encodes). Also, in Figure 4b, we see that several small-scale structures, formed by closely placed nodes, are hard to visually explore, as too little visual space is dedicated to them. As such, additional modifications of the t-SNE method are required to improve the layout's quality. These are described in the current section and the next one.

We start here by addressing the latter issue mentioned above – the presence of groups of nodes which are close to each other in the graph, and which are drawn too closely to each other to be able to visually explore their structure. To alleviate this, an easy way is to bias the SPDM distance function so as to artificially allocate more importance to small distances. To do this, we propose to use a transfer function, which is discussed in this section.

Before supplying the pairwise distance matrix to the t-SNE projection technique, it is normalized such that the maximum value is 1. This is done such that the transfer function can be applied more elegantly.

After the normalization, it is possible to apply a transfer function to increase the contrast in certain distance domains. Sometimes it may be desirable to have a larger contrast between smaller path lenghts. To take this into account, we can apply the following transfer function to the normalized pairwise distances:

$$f(x) = x^k \quad \text{for } k \leqslant 1.$$

Figure 5 illustrates that, indeed, the contrast is increased for smaller values of x.

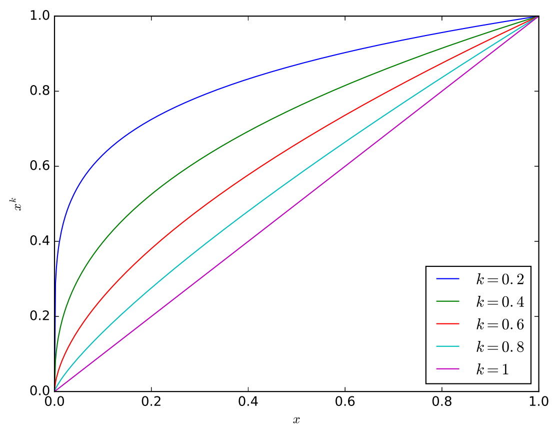


Figure 5: Transfer function for different values of k.

Transfer function on `dwt_72`

Figure 6 shows layouts of `dwt_72` with different values of k for the transfer function. Early compression (see Section 3.2.3) has also been used for these layouts to prevent overlapping edges. Considering that the vertices are colour-coded with the per-vertex cost function[1], we see that the cost function changes considerably while the layout only changes slightly. The two vertices encircled in blue in Figure 6a have an average cost in the first layout, while they have a lower than average cost in the layouts with higher k.

In the layout, the most notable thing that changes is that the two vertices encircled in blue are better distinguishable when k is lower. This is the effect of the increase in contrast for lower distances. Lowering k decreases the size of the effective neighbourhood of the vertices; the two encircled vertices in Figure 6a are not affected much by the vertices below them, while in the other layouts they seem more attracted to the vertices below them.

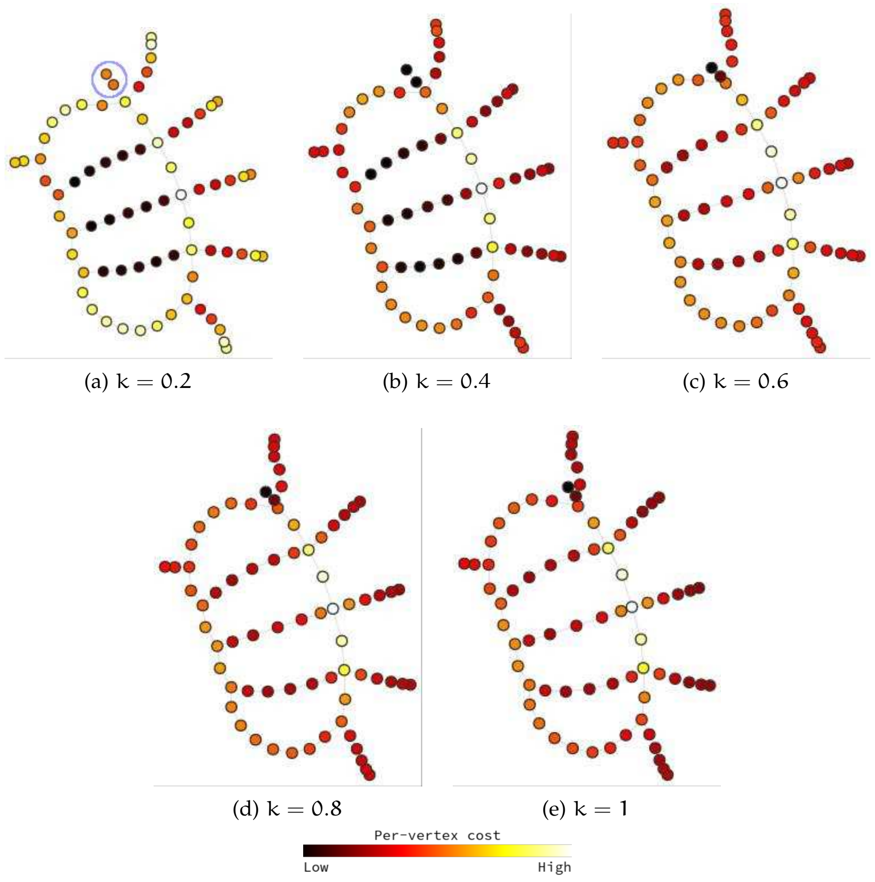


Figure 6: Layouts of `dwt_72` with increasing k for the transfer function. All layouts started with the same initial random placement of nodes.

Transfer function on `grid17`

Figure 7 shows layouts of `grid17` for different values of k. Early compression (see Section 3.2.3) has also been used for these layouts to prevent patching artifacts. The effects of the transfer function are minimal.

1 In these drawings, this is only the Kullback-Leibler divergence, since the early compression was only present in the first half of the optimization process. See Sections 3.2.1 and 3.2.5.

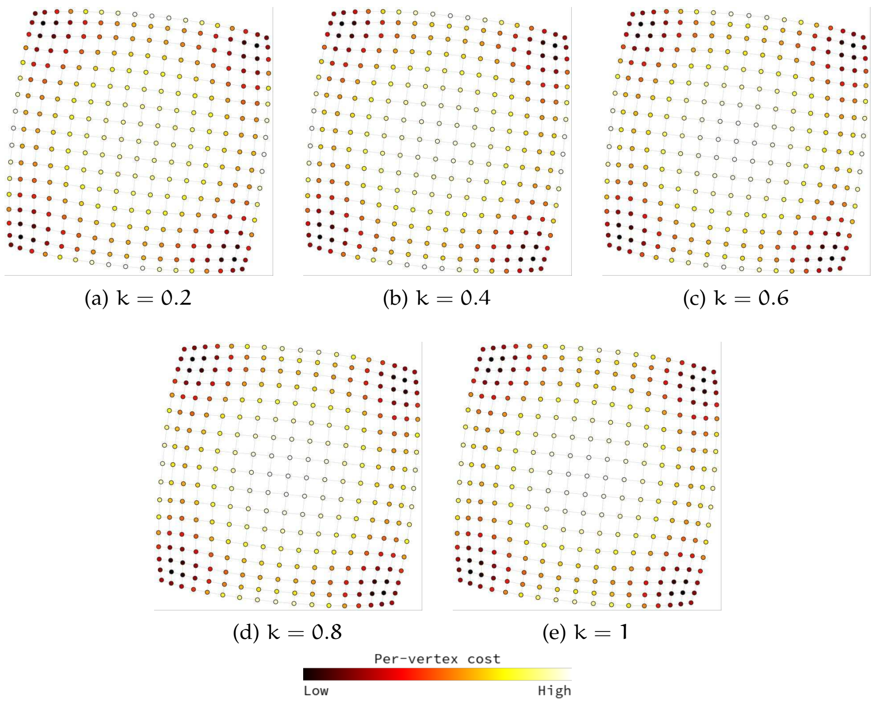


Figure 7: Layouts of `grid17` with increasing k for the transfer function. All layouts started with the same initial random placement of nodes.

Transfer function on `netscience`

Figure 8 shows layouts with different values of k for the `netscience` dataset. Because of the increased contrast for smaller distances, there is less overlap with smaller values of k.

The layouts with smaller k have a more square-like shape, while the layouts with larger k are more rectangular. This is because the smaller distances get transfered to larger distances in the layouts with smaller k. The increase in contrast for smaller distances comes at a cost; the contrast between smaller and larger distances becomes less distinct. This makes the general shape of the layout more square-like.

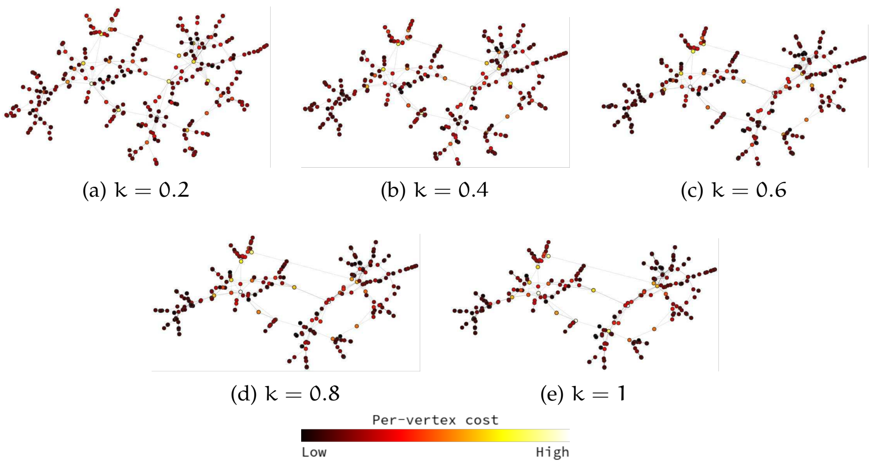


Figure 8: Layouts of `netscience` with increasing k for the transfer function. All layouts started with the same initial random placement of nodes.

3.2 COST FUNCTION

The cost function that is used in the final scheme is a summation of various terms. To determine the 2D positions of the vertices, $\{\vec{y}_i\}$, the cost function is minimized by using momentum-based gradient descent, with respect to $\{\vec{y}_i\}$.

The cost function is an objective function that (in the case of pure Kullback-Leibler divergence) reflects how well the neighbourhoods are preserved. A low cost means that the neighbourhoods are preserved well, while a high cost means that the neighbourhoods are preserved poorly, e.g., by placing nodes close to each other in the layout that are not close in the graph-theoretical sense. Therefore, a lower cost indicates a better layout, and therefore the cost function should be minimized.

During the development of the scheme, we started out with the Kullback-Leibler divergence, which is explained in Section 3.2.1.

Over the course of the scheme's development several terms were introduced with the aim of solving problems with the layouts. These are explained in Sections 3.2.2 to 3.2.4.

Finally, Section 3.2.5 discusses how the terms are combined to the final cost function.

3.2.1 *Kullback-Leibler*

The paramount term of the cost function comes from the t-SNE method, and is the Kullback-Leibler divergence from the probability of picking a pair in the two-dimensional space to the probability of picking a pair in the input space:

$$C_{kl} = KL(P, Q) = \sum_{\substack{i,j \\ j \neq i}} p_{ij} \log \frac{p_{ij}}{q_{ij}}.$$

Recall that p_{ij} is the probability of picking the pair of vertices $\{v_i, v_j\}$, based on the distance matrix that is supplied to the method, and the perplexity parameter κ. q_{ij} is the probability of picking a pair in the two-dimensional space, which is the normalized Student's t-distribution between the two two-dimenional positions $\vec{y}_i$ and $\vec{y}_j$.

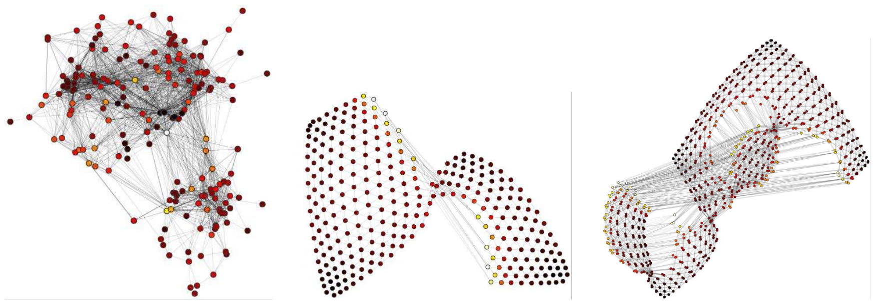
The Kullback-Leibler divergence from Q to P, i.e. $KL(P, Q)$ can be interpreted as *the amount of information lost when* Q *is used to approximate* P. [19, p. 51] Since we want to retain as much information as possible, we evidently want to minimize this term.

The gradient of C_{kl}, which is derived in [1], is given by:

$$\frac{\partial C_{kl}}{\partial \vec{y}_i} = 4 \sum_j (p_{ij} - q_{ij})(\vec{y}_i - \vec{y}_j)(1 + \|\vec{y}_i - \vec{y}_j\|^2)^{-1},$$

which has the pleasant property that if two nodes are embedded close together, while the distance in the input space is large, the gradient will repel the points strongly, while not going to infinity. Moreover, this is a term that is easily optimizable, since it often results in good local optima as a result of its long-range forces.

Figure 9 shows three layouts, using only the Kullback-Leibler divergence term. It can be seen that this produces suboptimal results for Figures 9b and 9c.



(a) Layout of `jazz` (b) Layout of `mesh3e1` (c) Layout of `bcsstk09`

Per-vertex cost
Low
High

Figure 9: Layouts with only the Kullback-Leibler term in the cost function.

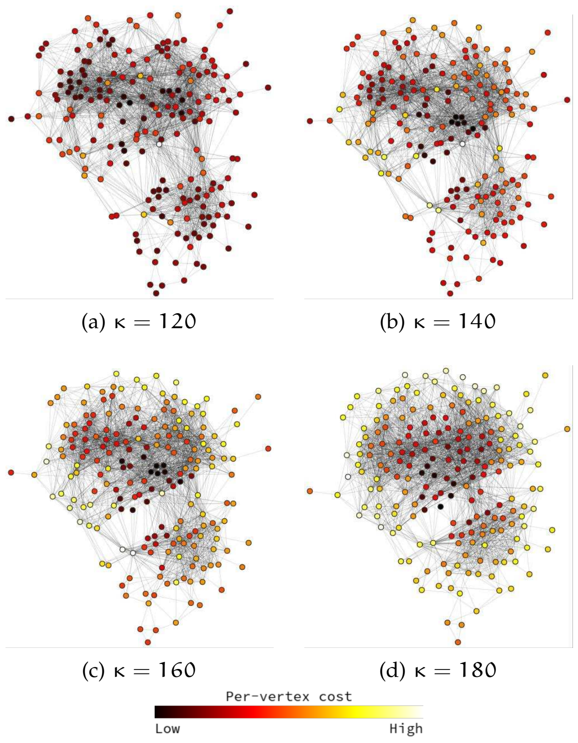
Perplexity

The perplexity, κ[2], which is a parameter for the symmetric probabilities p_{ij}, is the only remaining parameter for the Kullback-Leibler term. As mentioned before, it is a measure of how large neighbourhoods around nodes are.

Figure 10 shows layouts of `jazz` with different values of κ. Notice how more focus is directed to the local structure when using a lower perplexity; the nodes in the layouts with higher perplexity are spread more uniformly, while the nodes in the layouts with lower perplexity are more influenced by their local neighbourhood.

Figure 11 shows layouts of `netscience` using different values for κ. It can be seen that the optimization is quite stable with respect to the perplexity. The only major difference is that a part of the layout is twisted when using higher perplexities.

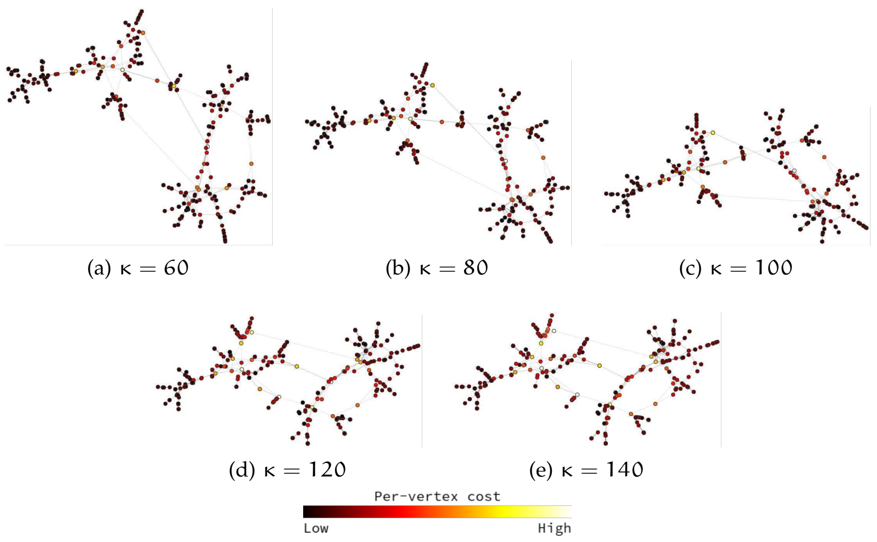
2 Notice how the perplexity (κ) is different from (but similar to) the exponent in the transfer function (k).



(a) $\kappa = 120$ (b) $\kappa = 140$

(c) $\kappa = 160$ (d) $\kappa = 180$

Figure 10: Comparison of layouts of `jazz` using different perplexities.



(a) $\kappa = 60$ (b) $\kappa = 80$ (c) $\kappa = 100$

(d) $\kappa = 120$ (e) $\kappa = 140$

Figure 11: Comparison of layouts of `netscience` using different perplexities.

3.2.2 *Edge contraction*

Figures 9b and 9c show artifacts that seem to tear the layout into multiple patches, even though the underlying graph has a regular structure. These artifacts are referred to as *patching artifacts*. In an attempt to remove such artifacts, an additional term was added to the cost function:

$$C_e = \frac{1}{2|E|} \sum_{(\nu_i,\nu_j)\in E} \left\|\vec{y}_i - \vec{y}_j\right\|^2,$$

where $\vec{y}_i$ and $\vec{y}_j$ are the positions of vertices ν_i and ν_j in the two-dimensional layout. It can be seen (by differentiating w.r.t. $\vec{y}_i$) that minimizing this distance is equivalent to embedding a spring with rest length 0 between the vertices that are connected with an edge. Because this term decreases the lenghts of edges, it is referred to as *edge contraction*. A similar term is used in existing spring-embedding layout techniques to enforce the spring forces.

This term is *only used in the first half* of the optimization process, since its goal is to remove the patching artifacts, which usually originate from the early phases of the optimization process. In this discussion, the term is weighted with a factor of 0.1, while the Kullback-Leibler term is weighted with a factor of 1.

The optimization of the cost function *with* edge contraction has been compared with optimization of the cost function *without* edge contraction. The comparison has been done on three graphs which suffered from patching artifacts. The high-level results are listed in Table 1.

	# artifacts in 30 layouts	
Graph	**no contr.**	**contr.**
`grid17`	25	6
`mesh3e1`	22	12
`bcsstk09`	29	7

Table 1: Overview of results of edge contraction on 30 optimizations of each graph

The edge contraction resulted in unstable optimizations for graphs with vertices that have many neighbours. The reason for this is that the edges that are connected to these vertices all exert forces on these vertices. This should not be a problem, but it makes the gradient of the cost function vary wildly with respect to these vertices, and therefore a lower learning rate must be chosen. Since this has a negative effect on the computation time, it was decided not to use edge contraction in the final method.

Edge contraction on `grid17`

The effect of the edge contraction term has been evaluated by comparing the number layouts with artifacts in 30 layouts. Figure 12 shows some layouts that had artifacts without the additional term. Two out of those three exhibited no artifacts with the additional edge contraction term. The edge contraction term is unable to improve the layout that is twisted in the middle.

As Table 1 shows, 25 out of 30 layouts exhibited artifacts without the edge contraction term, while only 6 layouts did so with the edge contraction term.

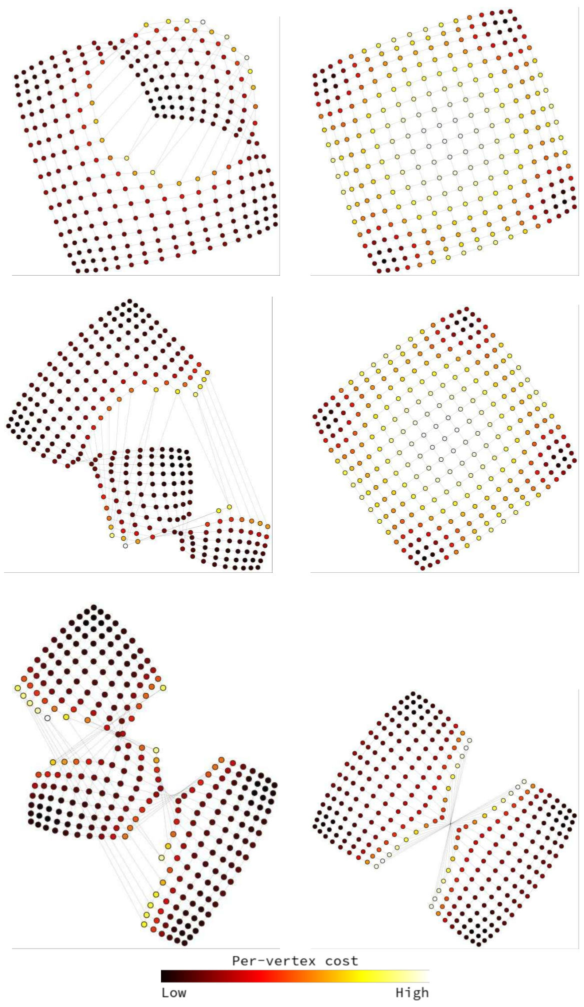


Figure 12: Layouts of `grid17` without (left) and with (right) edge contraction. All layouts that are on one row use the same initial configuration of vertices.

Edge contraction on `mesh3e1`

Figure 13 shows three layouts that had artifacts without the additional term. Two out of those three exhibited no artifacts with the additional edge contraction term. Again here, the additional term is unable to improve the layout that is twisted in the centre.

As Table 1 shows, 22 out of 30 layouts exhibited artifacts without the edge contraction term, while only 12 layouts did so with the edge contraction term.

Edge contraction on `bcsstk09`

Figure 14 shows three layouts of `bcsstk09` without and with edge contraction. Out of these three layouts, two were improved with edge contraction.

Out of the 30 layouts, 29 exhibited artifacts without the edge contraction term, while only 7 layouts did so with the edge contraction term.

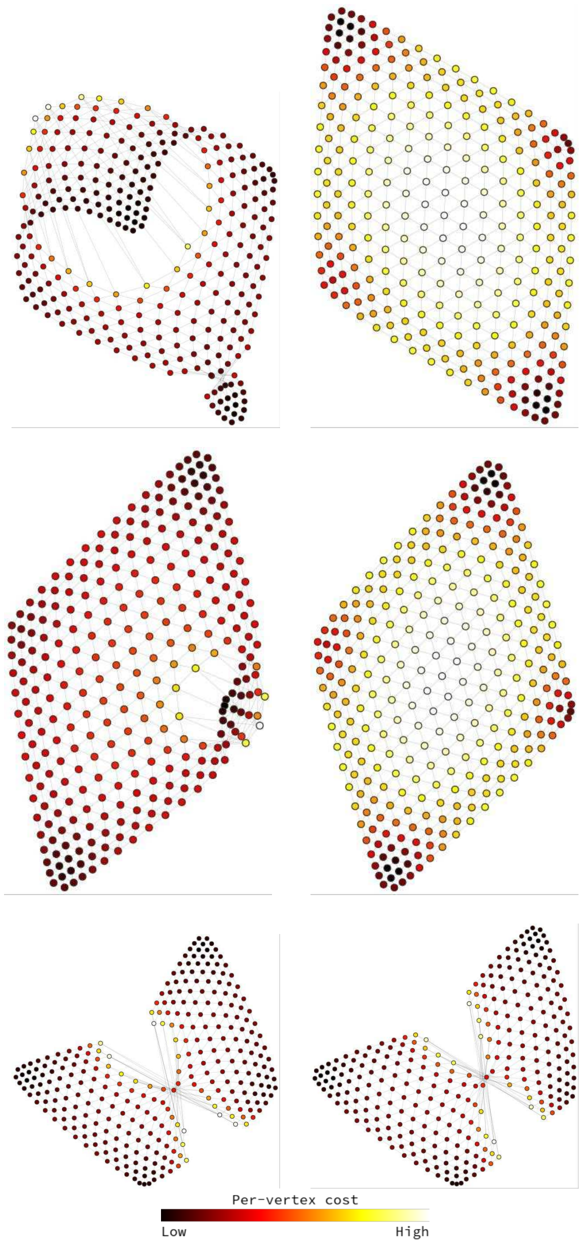


Figure 13: Layouts of `mesh3e1` without (left) and with (right) edge contraction. All layouts that are on one row use the same initial configuration of vertices.

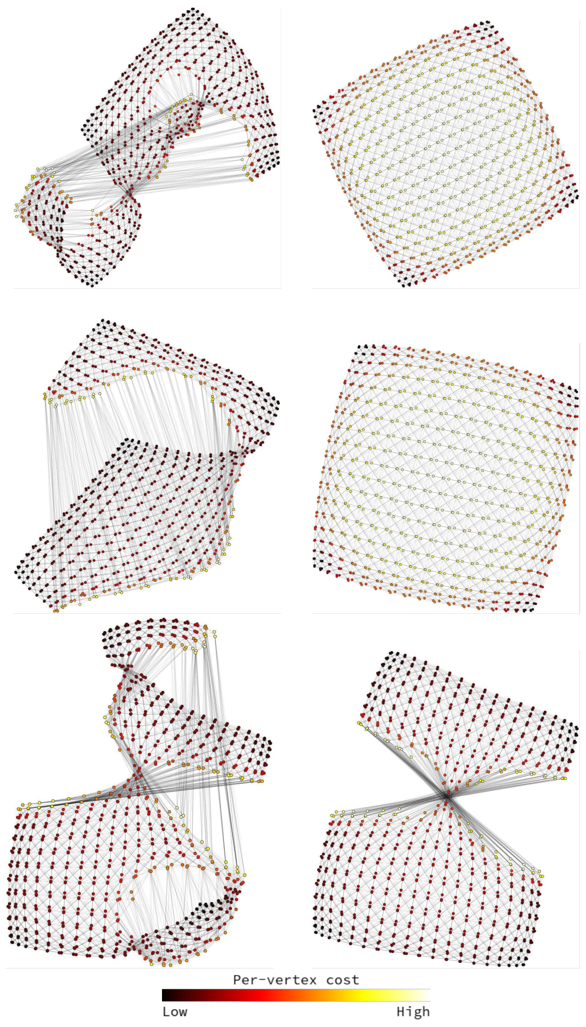


Figure 14: Layouts of `bcsstk09` without (left) and with (right) edge contraction. All layouts that are on one row use the same initial configuration of vertices.

3.2.3 *Early compression*

It is known that t-SNE produces better results when the projections are kept close to the origin in the early phases of the optimization. This is referred to as *early compression* in [1].

Early compression can be obtained by adding an additional term to the cost function, given by:

$$C_c = \frac{1}{2|V|} \sum_i \|\vec{y}_i\|^2, \tag{2}$$

where $\vec{y}_i$ are the two-dimensional projections of the vertices, and the notation $\|.\|$ indicates the Euclidean norm.

The early compression term is only nonzero in the first half of the optimization process. In this discussion, the compression term is weighted with a factor of 1.2, while the Kullback-Leibler term has a weight of 1.

Optimization of the cost function *with* early compression has been compared with that of the cost function *without* early compression. The results are summarized in Table 2. The following sections discuss the details of the comparison.

	# artifacts in 30 layouts	
Graph	**no compr.**	**compr.**
`grid17`	25	1
`mesh3e1`	22	9
`bcsstk09`	29	0

Table 2: Overview of results of early compression

Considering that these results are better than that of the edge contraction term, and that this term does not result in unstable optimizations, it was decided to use this term in the final method.

Early compression on `grid17`

30 random initializations of the `grid17` dataset have been used with and without early compression. Of these 30, 25 suffered from patching artifacts when *not* using early compression. All but one of these patching artifacts are shown to be preventable by using early compression. Figure 15 shows three layouts without and with early compression.

Early compression on `mesh3e1`

30 random initializations of the `mesh3e1` dataset have been used with and without early compression. 22 out of these 30 layouts suffered from patching artifacts when *not* using early compression. When using early compression, only 9 layouts suffered from patching artifacts. See Figure 16 for three layouts with and without early compression.

Early compression on `bcsstk09`

Early compression has also been evaluated on `bcsstk09`, where the best results were obtained. Of the 30 layouts, 29 suffered from patching artifacts when not using early compression. None of the layouts suffered from

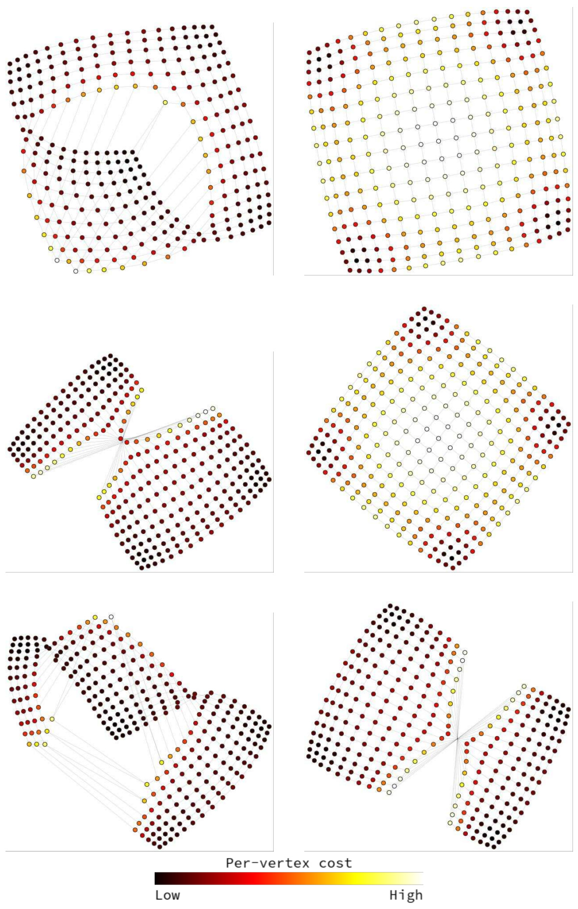


Figure 15: Layouts of `grid17` without (left) and with (right) early compression. All layouts that are on one row use the same initial configuration of vertices.

patching artifacts when using early compression. See Figure 17 for a visual comparison of some of the layouts.

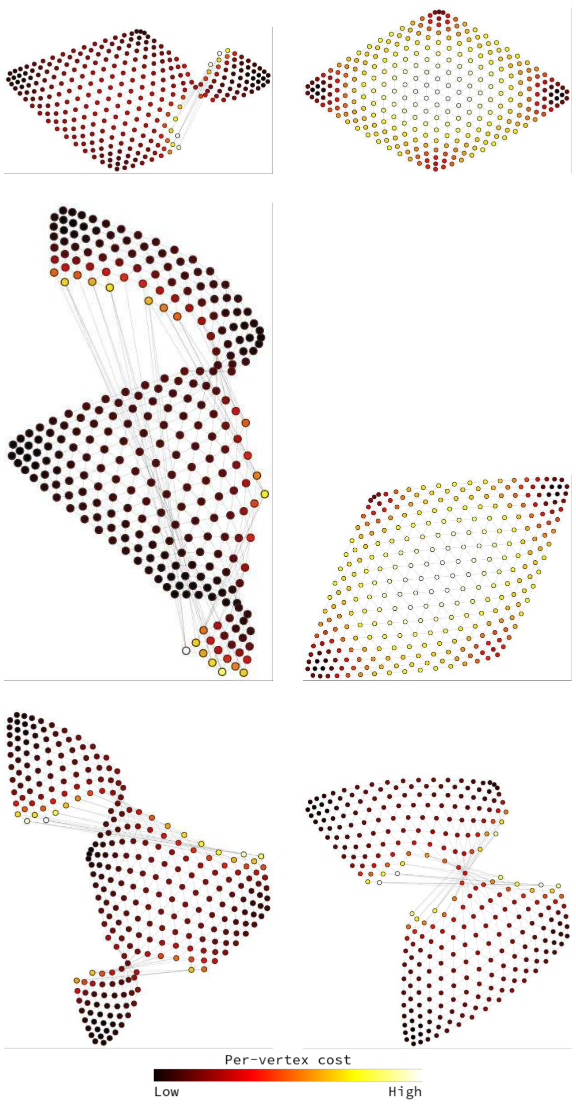


Figure 16: Layouts of `mesh3e1` without (left) and with (right) early compression. All layouts that are on one row use the same initial configuration of vertices.

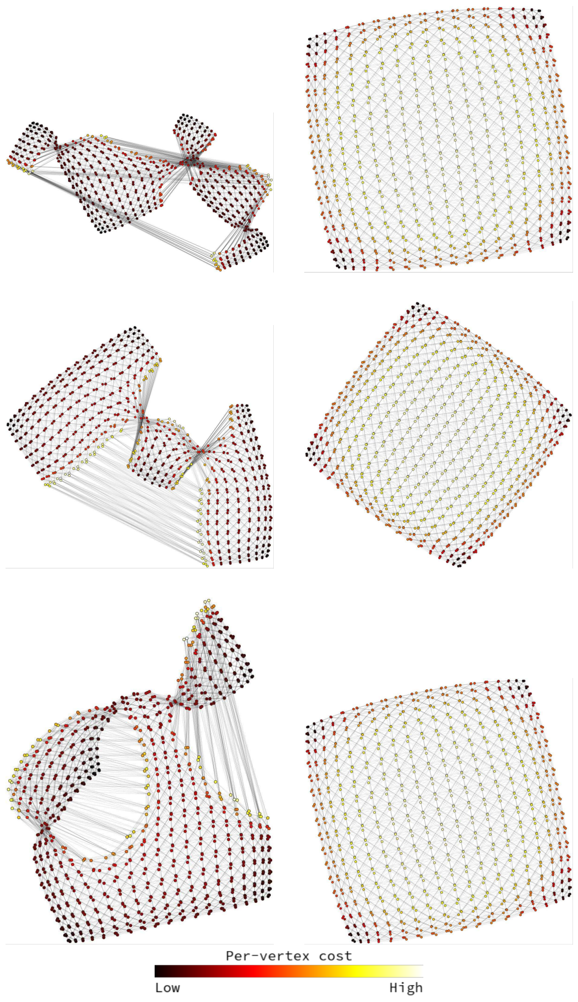


Figure 17: Layouts of `bcsstk09` without (left) and with (right) early compression. All layouts that are on one row use the same initial configuration of vertices.

3.2.4 *Repulsion*

To prevent clutter and overlapping vertices, a cost term was added to penalize vertices that are placed close to each other.

This cost term, referred to as the *repulsion* term, is given by:

$$C_r = \frac{1}{2|V|^2} \sum_{\substack{i,j \\ j \neq i}} \frac{1}{\epsilon_r + \left\|\vec{y}_i - \vec{y}_j\right\|}. \tag{3}$$

The repulsion term is also used in existing spring-embedders, also to prevent overlapping vertices.

The additional term, ϵ_r, has been added to the denominator to avoid the near-singularities in C_r (and its gradient) when $\left\|\vec{y}_i - \vec{y}_j\right\| \approx 0$. Without this additional term, vertices would often be thrown away from the origin resulting in unstable optimizations. Using a small value for ϵ_r can still result in unstable optimizations. Using larger values makes the gradient of C_r very small, and so this may diminish the effect of the repulsive term. The value $\epsilon_r = 0.2$ was found to be a good trade-off, and has been used unless stated otherwise.

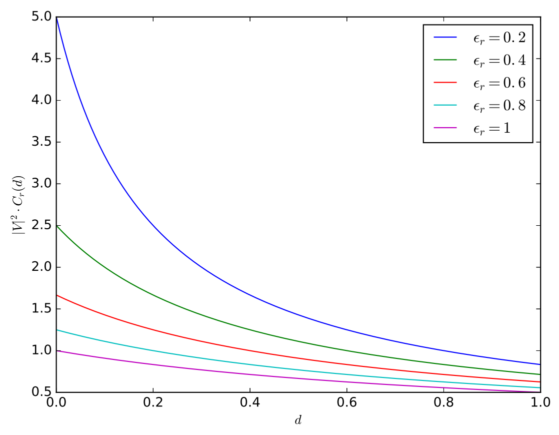


Figure 18: C_r of two vertices, as a function of the distance between the vertices, for different values of ϵ_r.

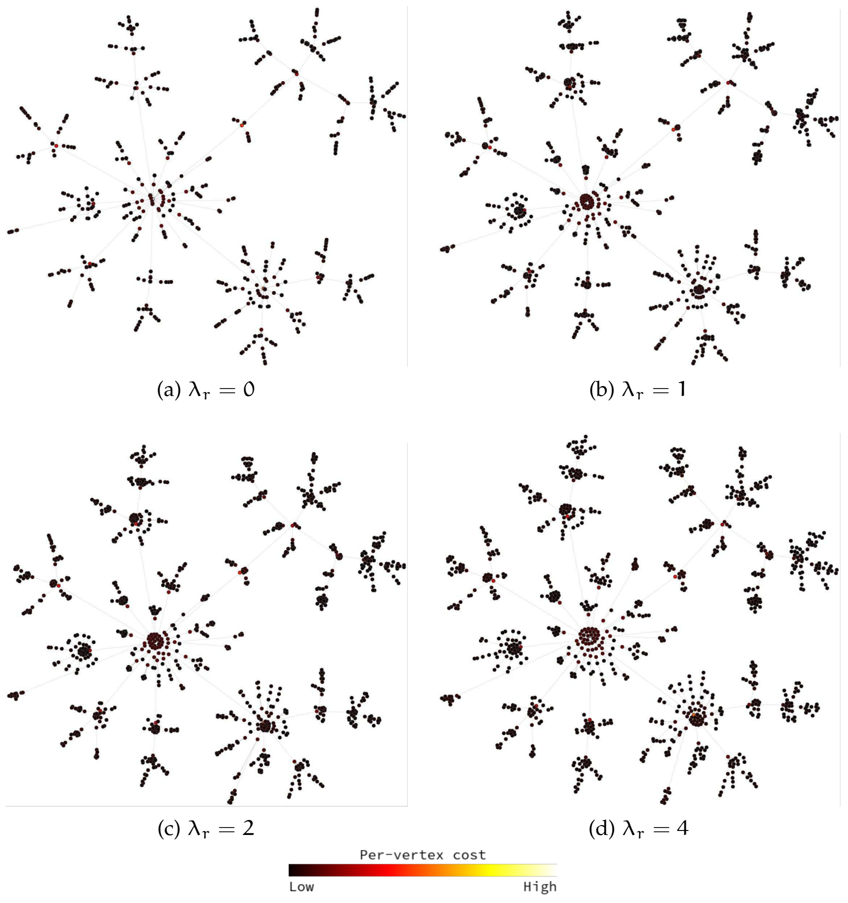
The repulsion term is only nonzero in the *second* half of the optimization, such that it does not interfere with the initial high-level layout of the graph.

For some graphs, the first half of the optimization resulted in vertices that are co-located. This resulted in the unexpected behaviour where the vertices *stay* co-located even during the repulsion phase. To prevent this, an additional epoch was added before the repulsion phase. In this additional epoch, co-located vertices are moved ever so slightly in random, but opposite directions. This caused the vertices' gradients to diverge and repel each other, as desired.

Repulsion on `price_1000`

The effect repulsion has been visually evaluated for `price_1000` in Figure 19. In the layout without repulsion (Figure 19a), many vertices are occluded by other vertices. Most notably the leafs of the graph overlap, since they only have a relatively small shortest path distance to one another, and therefore do not repulse substantially in the t-SNE projection.

The advantage of repulsion is that the user gets a better sense of how many vertices are in certain positions in the graph. A disadvantage of repulsion is that it is harder for the user to grasp the global structure of the graph.



(a) $\lambda_r = 0$ (b) $\lambda_r = 1$ (c) $\lambda_r = 2$ (d) $\lambda_r = 4$

Figure 19: Layouts of `price_1000` with different weights of the repulsion factor.

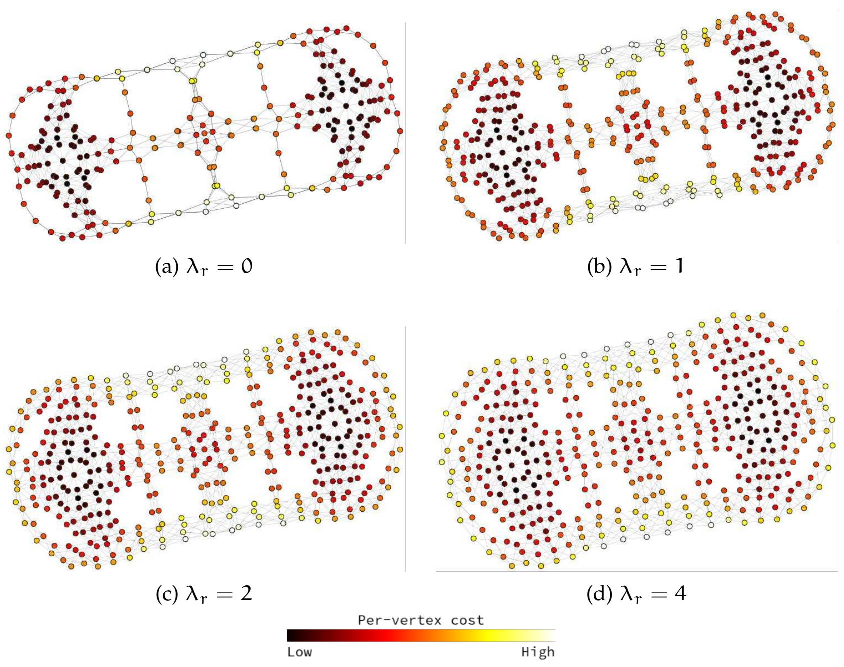
Repulsion on `dwt_419`

On `dwt_419`, the effect of repulsion has been visually evaluated in Figure 20. For this graph, again the layout without repulsion gives a better grasp of the overall structure of the graph. The layouts with repulsion give a better sense of the number of vertices. Especially in the upper and lower boundaries in Figure 20a it seems like there are single nodes, while in Figure 20b it is clear that there are multiple nodes.

It can be argued that in Figures 20c and 20d the repulsive term is too dominant, because the overall structure is not interpretable as easily.

Repulsion on `visbrazil`

The effect repulsion has been visually evaluated for `visbrazil` in Figure 21. As for `price_1000` and `dwt_419`, it is easier to grasp to global structure in the layout without repulsion. However, the layouts with repulsion gives a better sense of the sizes of the communities.



(a) $\lambda_r = 0$

(b) $\lambda_r = 1$

(c) $\lambda_r = 2$

(d) $\lambda_r = 4$

Figure 20: Layouts of `dwt_419` with different weights of the repulsion factor.

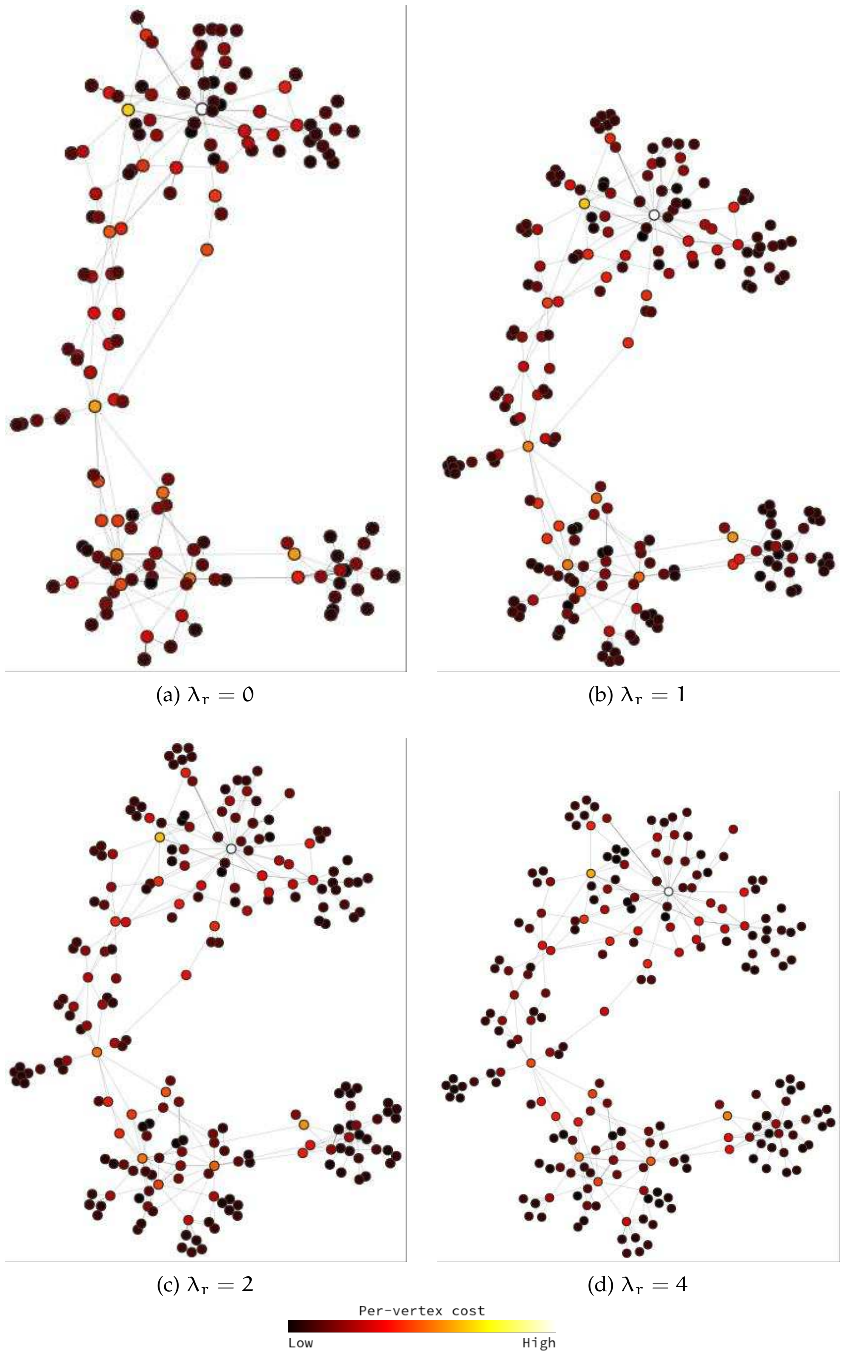


Figure 21: Layouts of `visbrazil` with different weights of the repulsion factor.

3.2.5 *Combining terms*

The four cost terms discussed in Sections 3.2.1 to 3.2.4 are combined to the final cost function by using a normalized weighted sum:

$$C(t) = \frac{1}{\sum_i \lambda_i(t)} \left(\lambda_{kl}(t) C_{kl} + \lambda_e(t) C_e + \lambda_c(t) C_c + \lambda_r(t) C_r\right). \tag{4}$$

The weights $\lambda_i(t)$ are functions of optimization time (in epochs):

$$\begin{aligned} \lambda_{kl}(t) &= 1 \\ \lambda_e(t) &= 0 \\ \lambda_c(t) &= \begin{cases} 1.2 & \text{if } t < \frac{n_e}{2} \\ 0 & \text{if } t \geqslant \frac{n_e}{2} \end{cases} \\ \lambda_r(t) &= \begin{cases} 0 & \text{if } t < \frac{n_e}{2} \\ 0.5 & \text{if } t \geqslant \frac{n_e}{2} \end{cases}, \end{aligned}$$

where n_e is the total number of epochs in the optimization.

That is, there are two phases:

PHASE 1 (first 50%): KL divergence and compression

PHASE 2 (last 50%): KL divergence and repulsion

The decision was made to eliminate the edge contracion term from the default cost function, because it often resulted in unstable optimizations, as discussed in Section 3.2.2.

The default weights are simply step functions or constants, but more involved contraptions could be devised. For example, a phase can be added in the middle where the repulsion weight increases linearly, and the compression weight decreases linearly.

3.3 IMPLEMENTATION

The method was implemented as a command-line utility in *Python 3.5.2*. Parameters can be passed as command-line arguments, or default values can be used.

The utility takes as input a (list of, or directory with) graph(s), and gives as output images of the layouts produces with our method. Optionally, an animation can be rendered that shows the layout over the course of the optimization process. This was useful as an exploratory tool during the development of the method. Also optionally, a file with the layout coordinates can be written.

A few software packages that the implementation has used, or relies on:

- `thesne` [20]: Originally used for dynamic t-SNE in [20], this implementation that uses `Theano` [21] (see below) has been altered to allow for the most essential module that minimizes the cost function used in the method.

- `Theano` [21] A *Python* symbolic mathematics toolkit for evaluating mathematical expressions. This toolkit allows for parallelized operations on the GPU. Especially the symbolic evaluation of gradients proved very useful for this project.

- `graph-tool` [22]: Used for internal graph representation, graph I/O, computing the shortest path distance matrix in parallel.

- `Graphviz` [23]: Used through a `graph-tool` interface for drawing graph layouts.

- `FFmpeg` [24]: Used for rendering animations of layouts that show the minimization of the cost function over time.

The source code of the implementation can be retrieved from [25]. The implementation was developed and tested on *ArchLinux*, and has a few dependencies and words of caution listed in the repository's `README` .

4 RESULTS

This chapter describes the results of our method. Section 4.1 introduces the benchmark, and argues why this benchmark is suitable. Section 4.2 discusses the results of our method on the benchmark one by one.

4.1 BENCHMARK

To provide a solid benchmark for evaluating the new layout approach, a set of graphs has been selected. During selection, the aim was to provide a broad range of types and sizes of graphs.

The benchmark, listed in Table 3, contains structural problems from the Harwell-Boeing collection[26], [27], real-world network data from Mark Newman's collection of network data[28], and other miscellaneous graphs. A reasonably large number of graphs (both real-world and synthetic) were included in the benchmark. The graphs have various topologies. Included in the benchmark are: trees, planar graphs, nonplanar graphs, communities, meshes, graphs with and without regular connectivity patterns. Altogether, this set of graphs covers a broad range of types and sizes, and is therefore a suitable benchmark.

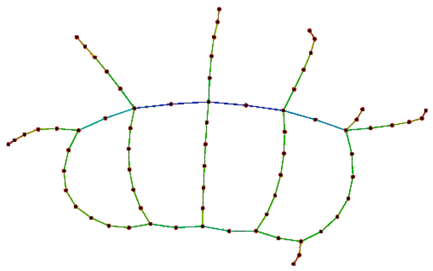
Most of the graphs were acquired through the *University of Florida* (UF) *sparse matrix collection* [29]. Others (marked with 'gen.' in the **Source** column) were synthesized using the `graph-tool` library[22]. The graphs marked with '*' are originally disconnected graphs that have been altered such that only the largest connected component is retained.

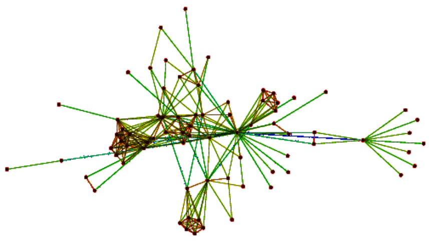
Our layouts are compared to the layouts drawn in Figures 22 to 24. Most of the layouts were also acquired from the UF sparse matrix collection website. The layouts from the UF sparse matrix collection have been generated using a multilevel force-based layout algorithm called *Scalable Force Directed Placement* (SFDP)[30]. An implementation[31] of the same algorithm has been used to generate layouts of the synthesized graphs. This implementation was not used for the other graphs, since the layouts by UF are arguably better, because finding the right parameter settings is not trivial.

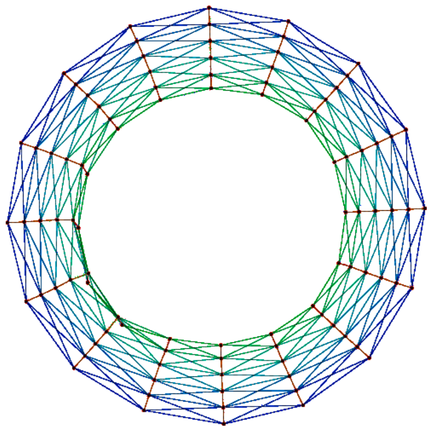
Almost all layouts are compared to the SFDP algorithm, and this is the only algorithm with which we compare our method (except for the `visbrazil` graph, which is compared to Martins' layout). This has several reasons. First, layouts that originate from other algorithms were not easily available. Second, it was out of the scope for this research to do a comparison with a larger number of layout algorithms, because this would also include finding the right parameter settings and running the layout algorithms many times for all the graphs.

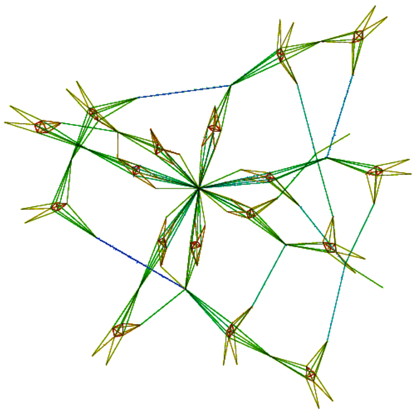
Name	Type	|V|	|E|	|E|/|V|	Source
dwt_72	planar	72	75	1.042	[32]
lesmis	communities	77	254	3.298	[33]
can_96	planar	96	336	3.500	[34]
rajat11	misc.	135	377	2.793	[35]
jazz	communities	198	2742	13.84	[36]
visbrazil *	communities	222	336	1.514	[17]
grid17	planar mesh	289	544	1.882	[37] (gen.)
mesh3e1	planar mesh	289	800	2.768	[38]
netscience *	communities	379	914	2.412	[39]
dwt_419	structural	419	1572	3.752	[40]
price_1000	tree	1000	999	0.999	[41] (gen.)
dwt_1005	structural	1005	3808	3.789	[42]
cage8	misc.	1015	4994	4.920	[43]
bcsstk09	structural	1083	8677	8.012	[44]
block_2000	communities	2000	9912	4.956	[45] (gen.)
CA-GrQc *	communities	4158	13422	3.228	[46]
EVA *	communities	4475	4652	1.039	[47]
us_powergrid	structural	4941	6594	1.335	[48]

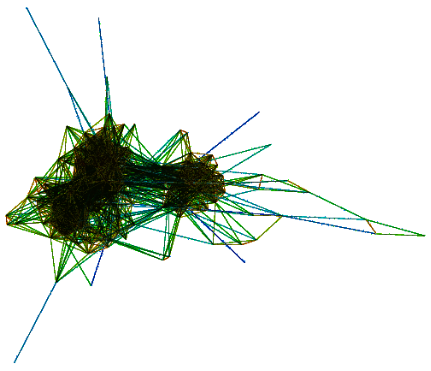
Table 3: Details of graphs in the benchmark.


(a) dwt_72 (UF layout)


(b) lesmis (UF layout)


(c) can_96 (UF layout)


(d) rajat11 (UF layout)


(e) jazz (UF layout)

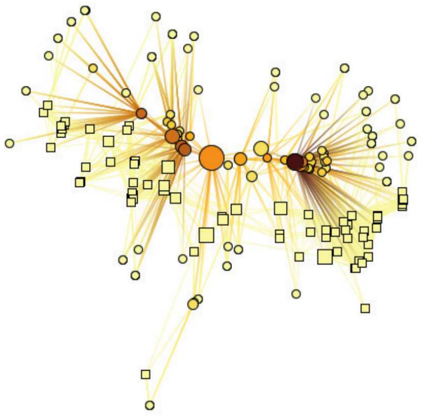
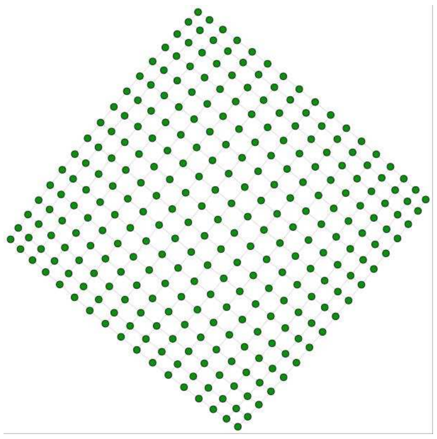
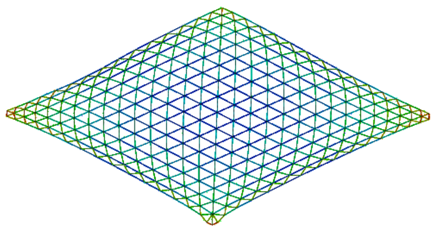
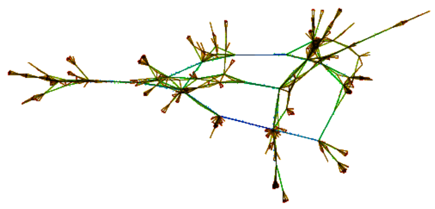
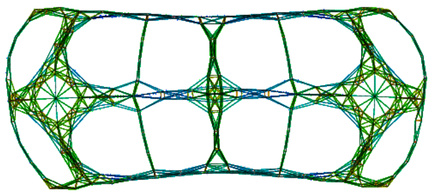

(f) visbrazil (layout from [17])

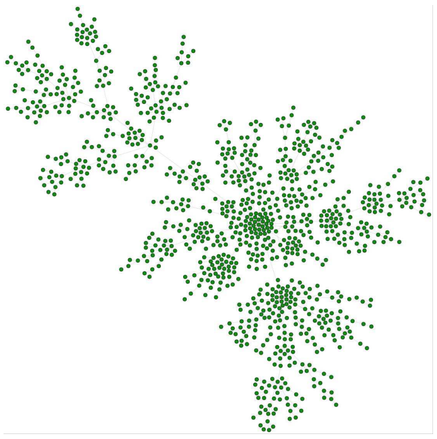
Figure 22: Overview of literature layouts of part of the benchmark.


(a) `grid17` (SFDP layout)


(b) `mesh3e1` (UF layout)


(c) `netscience` (UF layout)


(d) `dwt_419` (UF layout)


(e) `price_1000` (SFDP layout)

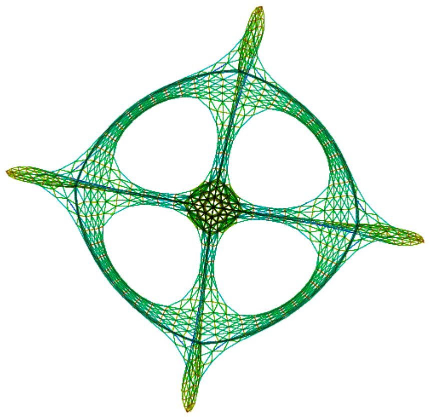
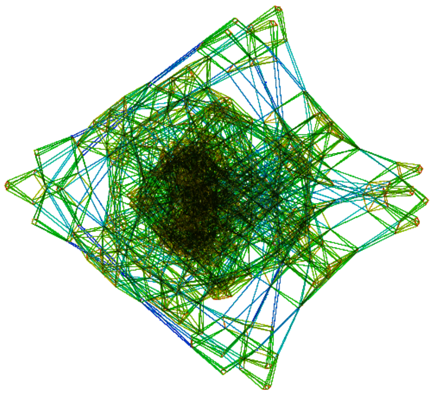
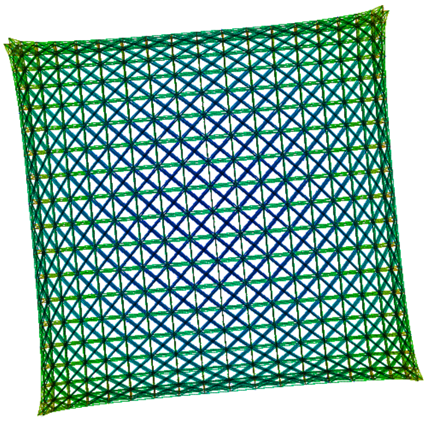
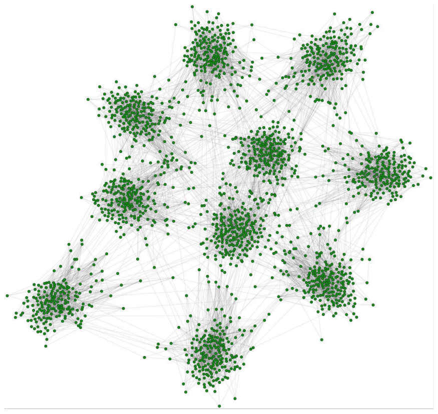
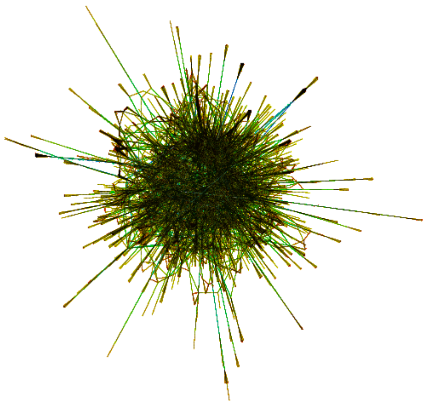

(f) `dwt_1005` (UF layout)

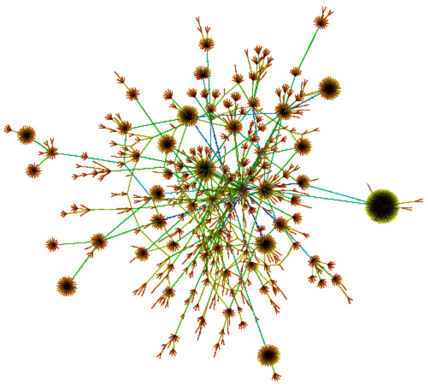
Figure 23: Overview of literature layouts of part of the benchmark.


(a) `cage8` (UF layout)


(b) `bcsstk09` (UF layout)


(c) `block_2000` (SFDP layout)


(d) `CA-GrQc` (UF layout)


(e) `EVA` (UF layout)

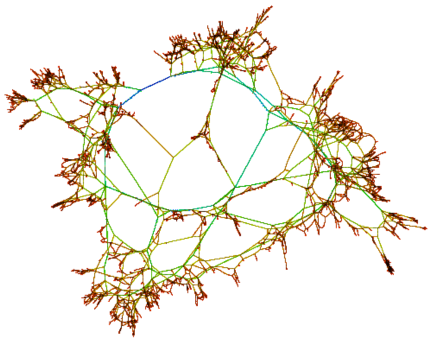

(f) `us_powergrid` (UF layout)

Figure 24: Overview of literature layouts of part of the benchmark.

4.2 RESULTS ON BENCHMARK

In this section, the t-SNE layouts are compared with the layouts shown in Figures 22 to 24.

Ideally, the comparisons of the layouts would have been made using the same drawing styles. However, neither the coordinates of the layouts or the drawing styles were always available, and so the comparison is made using our own drawing style. Our drawing style uses a simple green colour for the vertices, and a gray colour and for the edges. Moreover, alpha blending is used for the edges to improve visibility in edge-dense areas.

The layout drawings of UF have been manipulated using image editing software to (i) save ink, and (ii) make the drawings more alike and so the comparison fairer.

Two structural graphs (`can_96` and `dwt_1005`) produced satisfactory layouts only occasionally. For these graphs, both the satisfactory layouts and the unsatisfactory layouts are shown and discussed.

To investigate the optimization process, animations of the layouts over the course of the optimization have been rendered. These animations are linked in [25].

4.2.1 *dwt_72*

Figures 25a and 25b show the layout by UF and a t-SNE layout of `dwt_72` , respectively.

Some nodes are layed out in a more crooked fashion and not pushed outwards as much as the UF layout does. Aside from that, the t-SNE layout is successful in showing the structure of the graph.

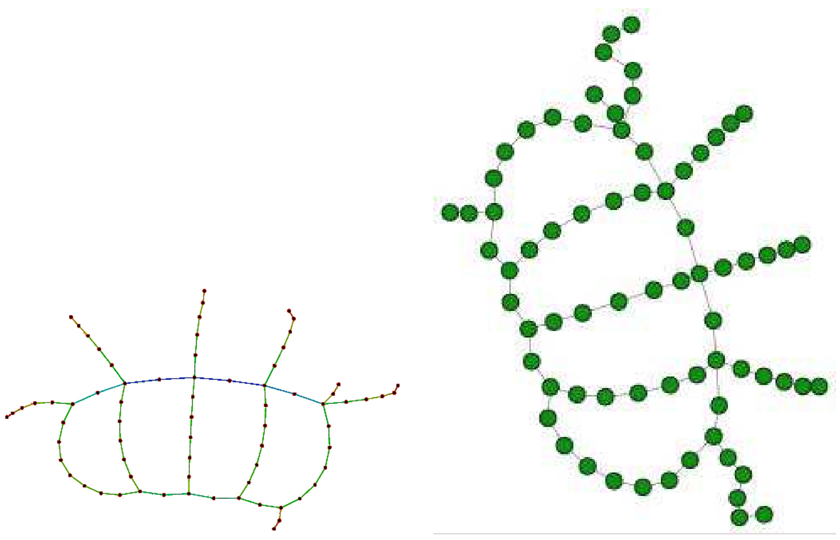

(a) UF layout (b) t-SNE layout

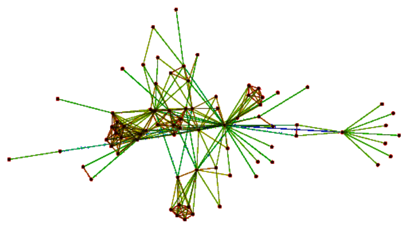
Figure 25: UF and t-SNE layouts of `dwt_72` .

4.2.2 `lesmis`

Figures 26a and 26b show the layout by UF and a t-SNE layout of `lesmis`, respectively.

Both layouts exhibit some tightly connected groups in the network. A notable difference is that the UF layout has more extended leafs that are further away from the centre of the layout.

Since the UF layout uses a large part of its layout space for the leafs, there is not much space left for the interior network. The t-SNE layout has a better focus on the interiors of the network, and makes it easier to follow paths in the central groups.


(a) UF layout

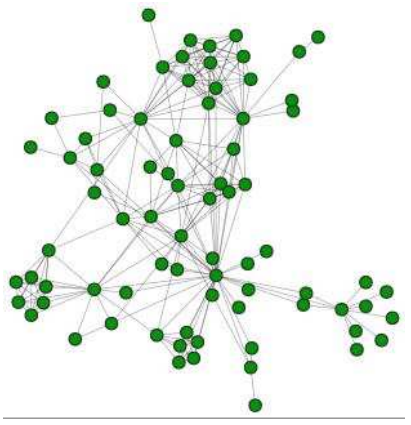

(b) t-SNE layout

Figure 26: UF and t-SNE layouts of `lesmis`.

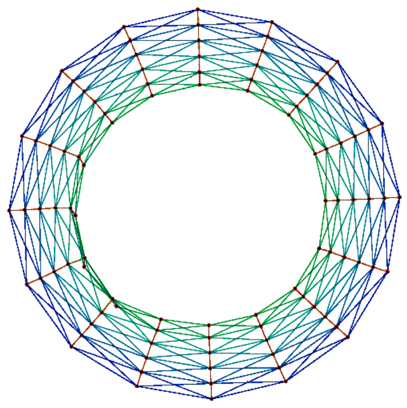
4.2.3 *can_96*

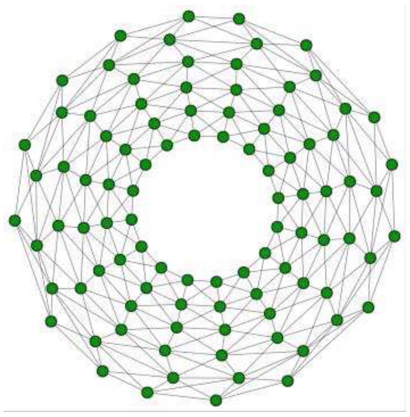
Figures 27a to 27c show the layout by UF and two t-SNE layouts of `can_96` , respectively.

It can be seen in the animation[1] that before the repulsion term becomes nonzero, the vertices are aligned in a circular fashion. When the vertices are repelled, the mesh can form in two correct ways: with the 'inner ring' on the inside, or the inner ring on the outside. It seems that if one part of the mesh happens to form in one way, and another part in the other way, both parts are not always able to 'convince each other' to form uniformly. This has happened in Figure 27c. In Figure 27b, the parts seem to agree when the repulsion becomes nonzero, and form a proper layout. Based on a sampling of 20 layouts, both proper and poor layouts seem to occur about half of the time.

The layout in Figure 27b is more crooked than the layout in Figure 27a. This is not desirable, because it suggest an absence of symmetry, despite the network being very symmetric.

Notice that although the UF layout in Figure 27a seems like a perfect layout of the graph, there are some irregularities in the lower-left part of the inner ring.


(a) UF layout


(b) Proper t-SNE layout

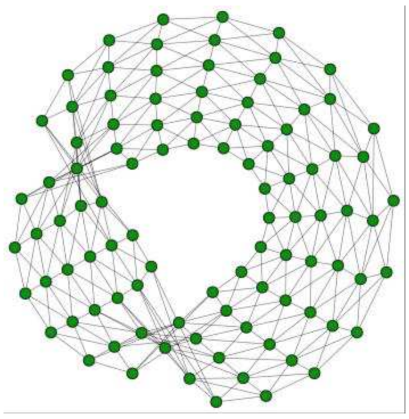

(c) Poor t-SNE layout

Figure 27: UF layout and t-SNE layouts of `can_96` .

[1] See the link in [25].

4.2.4 *rajat11*

Figures 28a and 28b show the layout by UF and a t-SNE layout of `rajat11`, respectively.

The graph is accurately represented in both layouts. A noticeable difference is that the layout by UF has very 'pointy' communities on the sides, while the t-SNE layout shows these communities in a less pointy fashion.

Another difference is that the triangles that the paths form between the little communities in the layout by UF are much more elongated than the ones in the t-SNE layout.

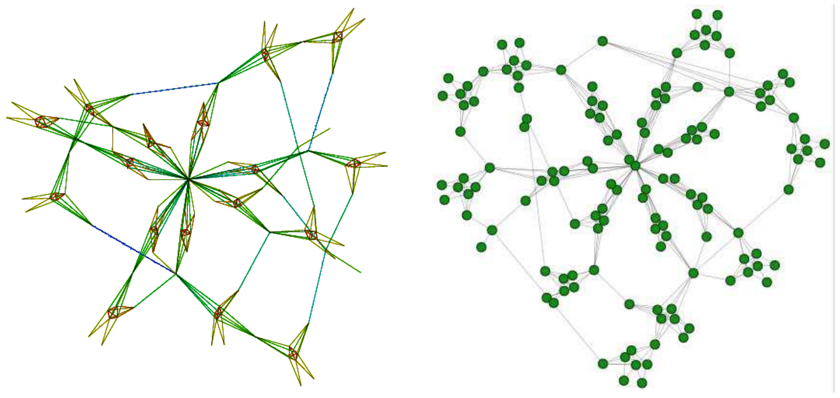

(a) UF layout (b) t-SNE layout

Figure 28: UF and t-SNE layouts of `rajat11`.

4.2.5 *jazz*

Figures 29a and 29b show the layout by UF and a t-SNE layout of `jazz`, respectively.

The layouts seem to suggest that the jazz community is very tightly connected. Both layouts show a similar structure: two large communities, and some smaller leaf-like communities.

In the UF layout, there seems to be more focus on the leafs, while the t-SNE layout reserves more space for the inner communities. As a consequence, the structure of the inner communities can be inspected better in the t-SNE layout. Still though, since the network is so tightly connected, mentally following individual paths in the two main communities remains almost impossible.

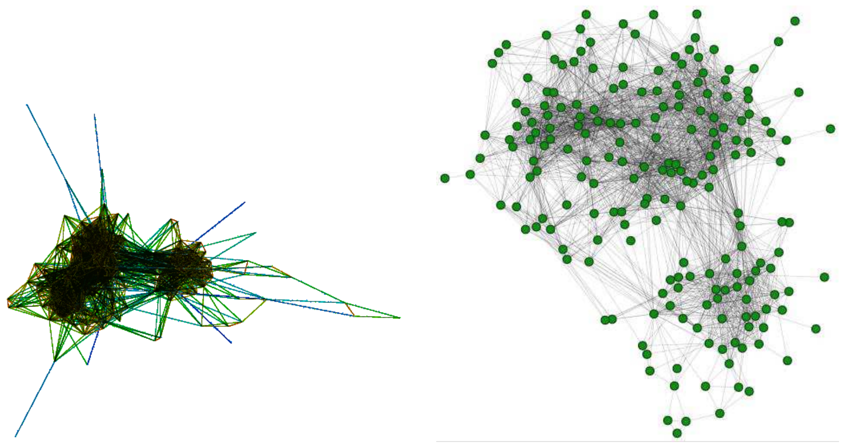

(a) UF layout (b) t-SNE layout

Figure 29: UF and t-SNE layouts of `jazz`.

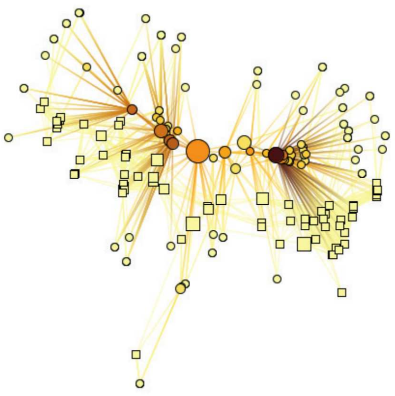
4.2.6 *visbrazil*

Figures 30a and 30b show the layout by [17] and a t-SNE layout of `visbrazil` , respectively.

Both layouts show that there is one very central vertex. The drawing in Figure 30a draws even more attention to the central vertex, because the node-sizes are determined by the betweenness centrality in that drawing.

A feature that is visible in Figure 30b, but not in Figure 30a is the 'shortcut' from the central vertex to the upper community in the drawing.

Moreover, since the nodes are placed very close together in the 'arms' of the central vertex in Figure 30a, the local structure is hard to interpret. The layout in Figure 30b reserves more space for the local structure, and so it is easier to mentally follow individual paths.


(a) Layout by [17]

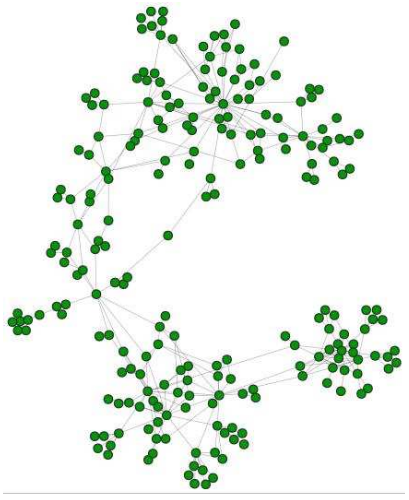
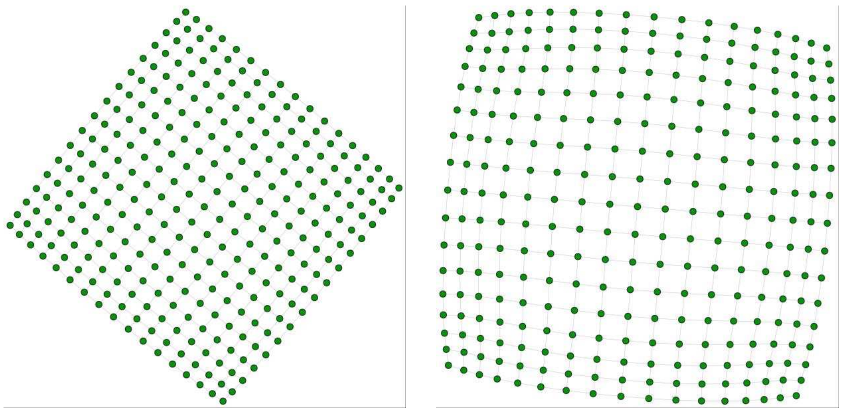

(b) t-SNE layout

Figure 30: Layout by [17] and t-SNE layout of `visbrazil` .

4.2.7 *grid17*

Figures 31a and 31b show a force-based layout and a t-SNE layout of `grid17` , respectively.

The layouts are almost identical, and both show the two-dimensional grid. The most notable difference is that the t-SNE layout allocates more space for the central vertices. This may be desirable, since central vertices are often considered more important. However for such a grid, a uniform layout is likely to be more desirable.



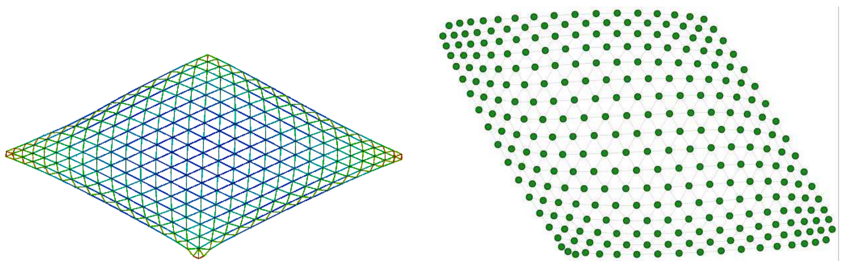
(a) SFDP layout (b) t-SNE layout

Figure 31: SFDP layout and t-SNE layout of `grid17` .

4.2.8 *mesh3e1*

Figures 32a and 32b show the layout by UF and a t-SNE layout of `mesh3e1` , respectively.

The layouts are almost identical. Both layouts accurately show the structure of the grid.



(a) UF layout (b) t-SNE layout

Figure 32: UF and t-SNE layouts of `mesh3e1` .

4.2.9 `netscience`

Figures 33a and 33b show the layout by UF and a t-SNE layout of `netscience` , respectively.

A notable difference between the layouts is that the layout in Figure 33a is more elongated, while the layout in Figure 33b is more circular. As seen in other layouts, the layout in Figure 33a has a more pointy appearance. Moreover, it can be argued that the layout in Figure 33b gives a better sense of the number of vertices.

Also, as seen before in other graphs, the layout in Figure 33b has a better focus on the more local structure of the network.

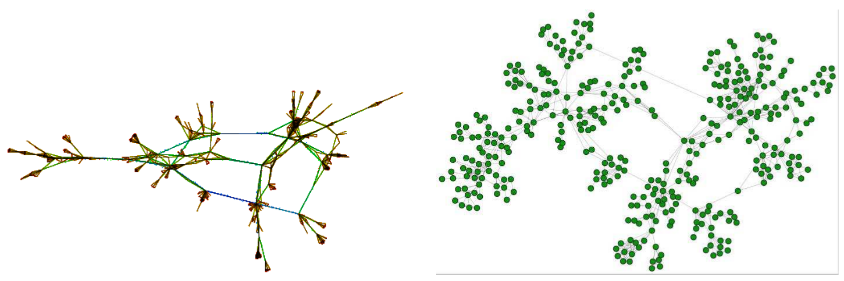

(a) UF layout (b) t-SNE layout

Figure 33: UF and t-SNE layouts of `netscience` .

4.2.10 *dwt_419*

Figures 34a and 34b show the layout by UF and a t-SNE layout of `dwt_419`, respectively.

The layout in Figure 34a makes it harder to distinguish the vertices in the upper and lower boundaries. As discussed in Section 3.2.4, Figure 34b allows for a better distinction between the vertices in those boundaries, because of the repulsion term.

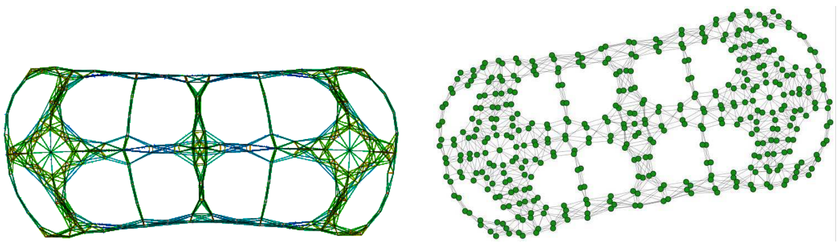

(a) UF layout (b) t-SNE layout

Figure 34: UF and t-SNE layouts of `dwt_419`.

4.2.11 *price_1000*

Figures 35a and 35b show force-based layout and a t-SNE layout of `price_1000` , respectively.

The t-SNE layout satisfactorily shows the seperation of central hubs, and does not grow large forests of nodes that may visually suggest various paths between the central hubs. However, it can be argued that the layout in Figure 35a gives a better sense of the number of vertices, since (despite the repulsion term) there are still some vertices that overlap in Figure 35b.

This difference is interesting, because in some other graphs the t-SNE layout had more focus on the *local structure* of the network, while here a result with a more *high-level structure* focus is obtained.

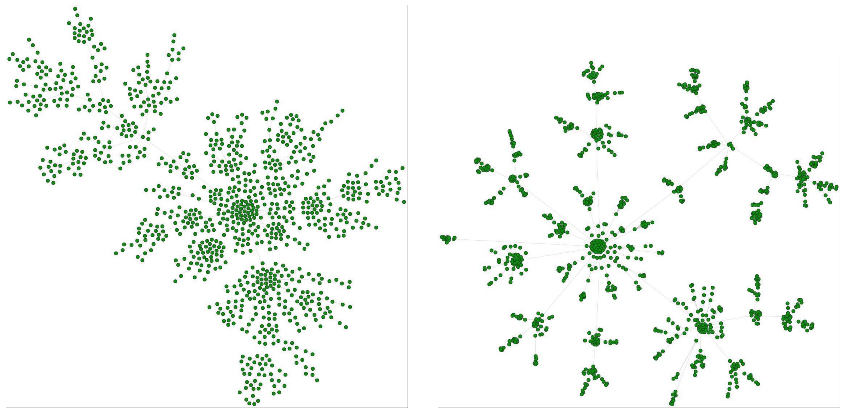

(a) SFDP layout (b) t-SNE layout

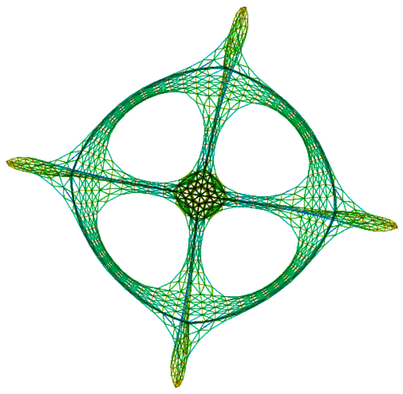
Figure 35: SFDP and t-SNE layouts of `price_1000` .

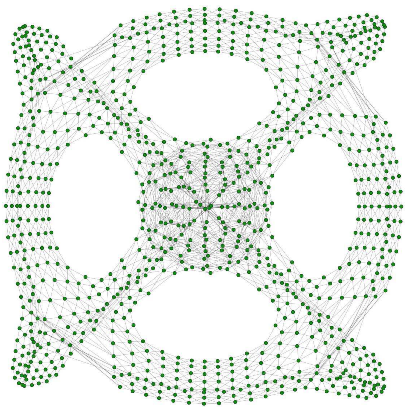
4.2.12 *dwt_1005*

Figures 36a and 36b show the layout by UF and a t-SNE layout of `dwt_1005` , respectively.

Both layouts capture the same structure, but the t-SNE layout displays less pointy bulges. The large number of overlapping edges around the bulges are a consequence of the intrinsic three-dimensional nature of the structural network.[49]

Figure 36c shows a t-SNE layout that used the same parameters as Figure 36b, but a different initial configuration of the vertices. This layout failed to capture the structure of the graph


(a) UF layout


(b) Good t-SNE layout

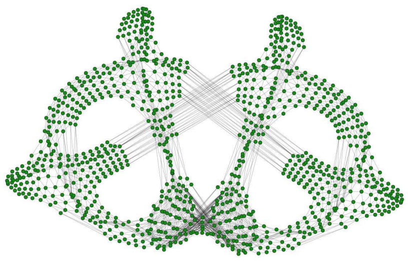

(c) Poor t-SNE layout

Figure 36: UF and t-SNE layouts of `dwt_1005` .

4.2.13 *cage8*

Figures 37a and 37b show the layout by UF and a t-SNE layout of `cage8`, respectively.

The layout by UF shows more symmetry than the t-SNE layout. However, since the central vertices of the t-SNE layout have more space allocated to them, the local structure is more easily observed in the t-SNE layout.

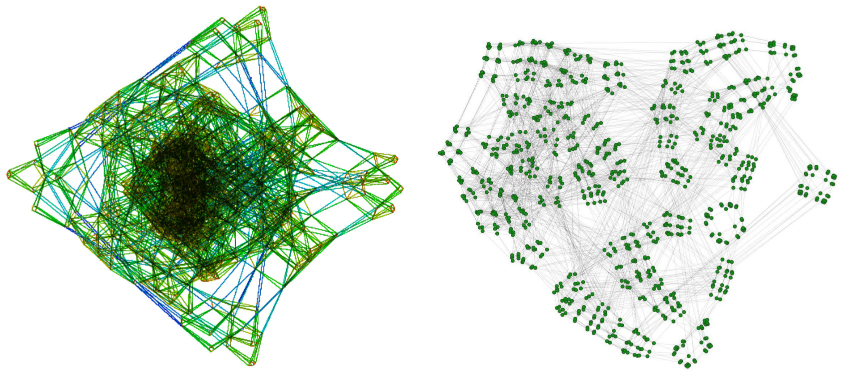
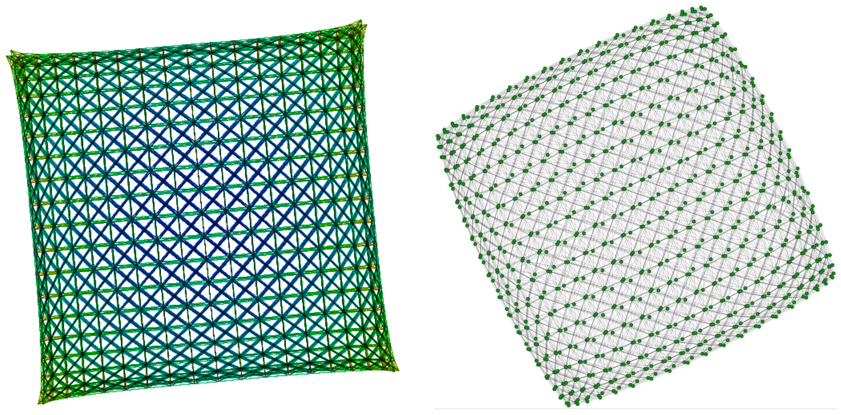

(a) UF layout (b) t-SNE layout

Figure 37: UF and t-SNE layouts of `cage8`.

4.2.14 *bcsstk09*

Figures 38a and 38b show the layout by UF and a t-SNE layout of `bcsstk09` , respectively.

The t-SNE layout has a convex shape, while the layout by UF exhibits concavities on the boundaries. Moreover, the internal nodes show less overlap in the t-SNE layout, while suggesting more structure in the UF layout. Apart from that, the layouts are mostly identical.



(a) UF layout (b) t-SNE layout

Figure 38: UF and t-SNE layouts of `bcsstk09` .

4.2.15 *block_2000*

Figures 39a and 39b show a force-based layout and a t-SNE layout of `block_2000` , respectively.

The layouts are similar, but the t-SNE layouts separates the communities more, while the SFDP layout has more focus on nodes that are outliers in a community. Effort has been made to tweak the parameters of the SFDP layout (in particular by increasing the repulsion constant and exponent), but a separation as in Figure 39b was not achievable.

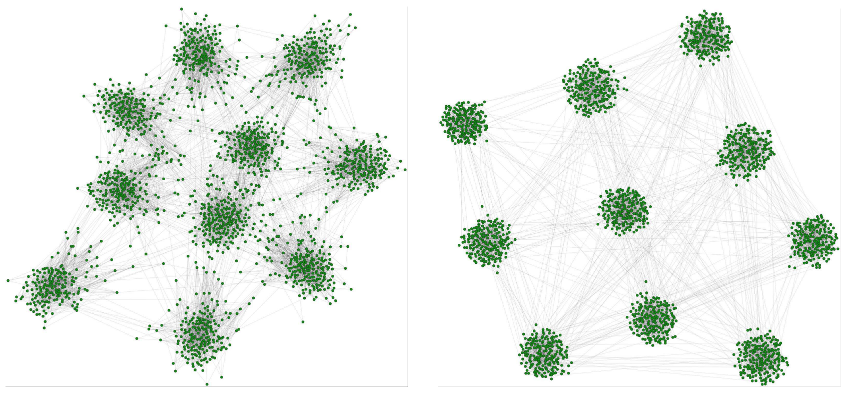

(a) SFDP layout (b) t-SNE layout

Figure 39: SFDP and t-SNE layouts of `block_2000` .

4.2.16 *CA-GrQc*

Figures 40a and 40b show the layout by UF and a t-SNE layout of `CA-GrQc`, respectively.

It can be seen that the layout by UF shows almost no structure in the central community. The UF layout has a very narrow focus on the leafs. As a result, the central community receives only little space in the layout.

The t-SNE layout show more structure in the inner community. In particular, there is a large community to be observed in the lower-left part of the t-SNE layout which is absent in the UF layout. However, it can be argued that this is due to the increase in resolution, and the differences in drawing styles.

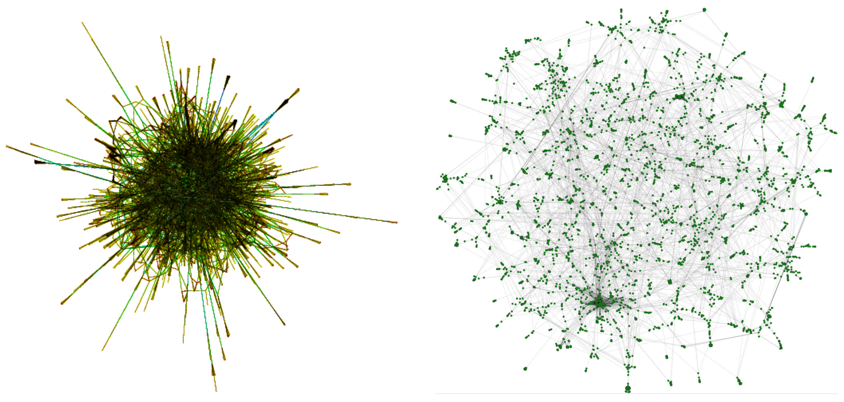

(a) UF layout (b) t-SNE layout

Figure 40: UF and t-SNE layouts of `CA-GrQc`.

4.2.17 *EVA*

Figures 41a and 41b show the layout by UF and a t-SNE layout of `EVA`, respectively.

The layout by UF is very appealing to look at, while the t-SNE layout is less aesthetically pleasing. However, considering only the layout, we see many overlapping edges in both layouts. The t-SNE layouts has some layouts that seem to be preventable, e.g. the two smaller communities above and below the large community on the right-hand side.

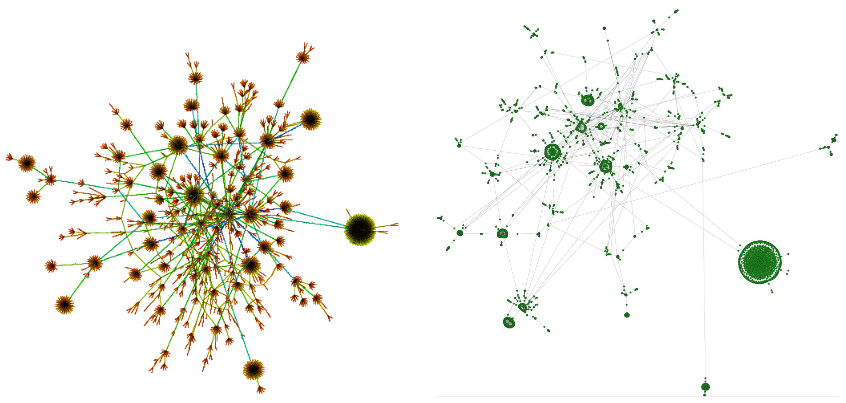

(a) UF layout (b) t-SNE layout

Figure 41: Layouts of `EVA`.

4.2.18 `us_powergrid`

Figures 42a and 42b show the layout by UF and a t-SNE layout of `us_powergrid` , respectively.

The layouts show many similarities; the large hubs are on the sides of the layout, and there is a sparser region in the centre that connects the hubs. One difference is that the t-SNE layout puts more emphasis on the local structure of the network. In the UF layout, the tightly connected parts are placed close together, while the t-SNE layout expands these parts of the network.

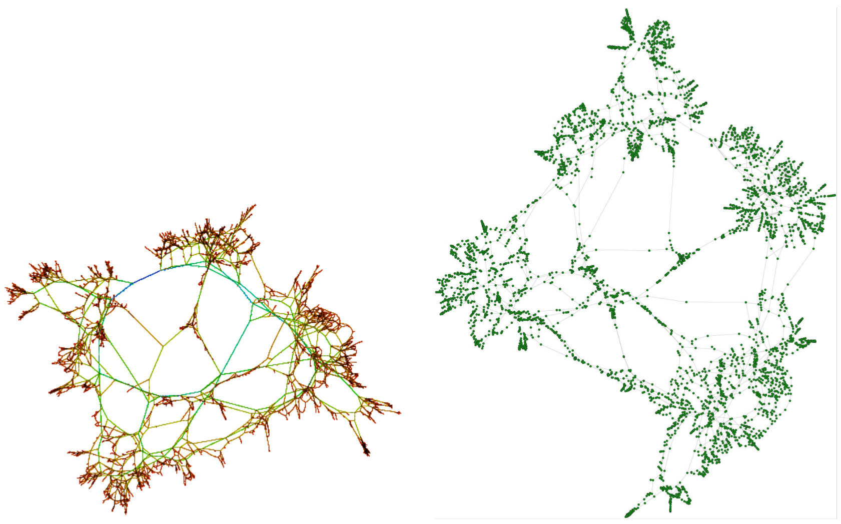

(a) UF layout (b) t-SNE layout

Figure 42: UF and t-SNE layouts of `us_powergrid` .

5 DISCUSSION

This chapter discusses the higher-level, more generalized results of our method in Section 5.1. Moreover, the challenges that were encountered in the method and research are discussed in Section 5.2.

5.1 GENERAL RESULTS

Considering the results presented in the previous chapter, we see that t-SNE can be useful for making graph layouts. The distinguishing shape of the t-SNE gradient results in some remarkable layouts.

t-SNE proved particularly useful for making layouts of collaboration networks, like `lesmis` , `jazz` , `visbrazil` , `netscience` and `CA-GrQc` . Also, the t-SNE layout of the synthesized `block_2000` graph is quite compelling. t-SNE also worked well for structural problems, especially `us_powergrid` in Figure 42a.

Less good results were obtained for `EVA` , which indicates that the method works poorly on near-tree networks, or networks with many leafs. Moreover, the methods that were used for solving most twisting artifacts did not work reliably on `can_96` and `dwt_1005` .

A recurring feature of the results is that the t-SNE layouts put more emphasis on the local structure of the networks. However, this happens not to such an extent that the overall structure becomes cluttered. The focus is a good trade-off between a high-level overview and a locally accurate representation.

5.2 CHALLENGES

Although good results were obtained, there were some challenges during this project that had to be overcome.

5.2.1 *Layout artifacts*

In the earlier phases of the development of the method, many layouts suffered from patching artifacts, where layouts were twisted and seemed to be stuck in local optima (or in a region with a very small gradient). This problem has been overcome by using early compression, as discussed in Section 3.2.3.

Another problem that was present until the later phases of the development of the method was that co-located vertices were not repelling each other. This has been solved by separating co-located vertices ever so slightly, causing the gradients of their repulsion terms to diverge.

5.2.2 *Parameter settings*

For some graphs, different parameters had to be used. The perplexity and the learning rate had to be hand-picked for every graph.

Values for the perplexity varied between 40 and 150. In general, the lowest possible perplexity for a graph had the best results. The perplexity could not be set arbitrarily low, because using a perplexity that is too low results in a diverging binary search for the standard deviations of the Gaussian distributions.

After seeing some results, it was sometimes desirable to increase the repulsion weight λ_r to have a larger separation between the vertices.

As a consequence of t-SNE's smooth gradient, it was possible to increase the learning rate significantly to reduce the computation time. For graphs with more vertices, generally a higher learning rate could be chosen. The learning rate is a hyperparameter, because it has to do with the optimization process.

5.2.3 *Differences in drawing style*

As mentioned earlier, the visual comparison of layouts where different drawing styles are used is not ideal. In particular, the UF layouts draw the vertices as very small (sometimes not even visible) nodes, while our drawings use small circles. Moreover, there are differences in the resolution of the images. However, effort has been made to try to compare only the layouts, e.g. by adjusting the colours of both layouts to be more similar.

5.2.4 *Computing time and scalability*

Although `Theano` was used for the optimization computations, GPU parallelization was not exploited, and some other optimizations were not utilised. (See Section 6.1.) As a result, performing the gradient descent optimization took several hours for the largest datasets. For the purpose of this thesis, this was acceptable, but for real-world implementations this requires additional work to speed up the algorithm.

5.2.5 *Comparison*

As discussed at several points during this thesis, the problem of computing a graph layout is not a well-defined one, in the sense that there is no single 'best layout' for most graphs. As such, assessing the quality of a graph layout can be done in practice in two ways. We comment below on the limitations of our work in both these respects:

- **Measuring quality criteria**: One way to assess the quality of a layout is to compute metrics that quantify several desirable properties. These include distance distributions between nodes; number and angles of edge crossings; and distributions of edge lengths. Doing this, however, requires that one has a well-defined set of values that indicate 'good quality' for specific graph types or classes. As we are not aware of such absolute values, we have not taken this path.

- **Visually comparing layouts**: The other way to assess the quality of a layout is a relative one. That is, the quality of a layout method is assessed by visually comparing its results (on the same graphs) with the results of several well-established layout methods. During the comparison, one effectively visually quantifies the quality criteria mentioned above. While less precise than formal measurement of quality criteria, this visual comparison is by far the most used comparison method in practice, arguably due to its simplicity and low cost. This is the path we have taken in our work. However, in this respect, our results are limited in comparing our method to two other main methods: SFDP [30] and the method of Martins [17]. Of course, many more layout algorithms exist out there; and an extensive comparison of our method should include several of these too. Due to time constraints, we however have decided to leave this comparison work out of the scope of this thesis.

6 CONCLUSION

In this thesis, we have explored the question of whether t-SNE, a well-known algorithm for dimensionality reduction, can be used to generate good-quality layouts for a wide set of real-world graphs. As we have shown in the previous two chapters, the global answer to this question is positive: For most of our studied graphs, we obtain results which are very similar, and in many cases of a higher visual clarity and lower visual clutter, than state-of-the-art methods used on the same graphs.

Besides the Kullback-Leibler divergence term that stems from t-SNE projection technique, other terms have been added to the cost function to prevent unwanted artifacts.

Especially for the collaboration datasets in the benchmark remarkable results were obtained. These results have a focus that is well balanced between the high-level structure and the local structure of the network.

Less positive results were obtained on a near-tree graph with many leafs. Also, despite the efforts, some structural graphs still occasionally exhibit unwanted patching artifacts.

6.1 FUTURE WORK

Considering the recent development of dynamic t-SNE [20] for visualizing time-dependent high-dimensional data, a similar approach can be explored for visualizing *dynamic graphs*. In the same way as dynamic t-SNE uses a cost term to decrease the frame-to-frame incoherence of a data point, dynamic t-SNE for dynamic graphs can introduce a term to decrease the frame-to-frame incoherence of a vertex.

As explored in [50] for t-SNE, tree-based optimizations like the Barnes-Hut algorithm can be used to accelerate the evaluation of gradients. Because of the similarity of the problem, it is expected that comparable results will be obtained for graph layouts, but this has not been verified.

Another strategy to accelerate the computation of layouts is to use a multi-level method. This is also left unexplored, and could yield promising results for the scalability of the method.

In the current implementation, a simple momentum-based gradient descent optimization is performed. Using different, more adaptive optimization techniques like (e.g. AdaGrad[51] or Adam[52]) could provide fruitful results. Moreover, exploiting parallelization through the GPU has also not been utilised.

Currently, the shortest path distance matrix is used as the pairwise distance matrix for the t-SNE technique. More sophisticated distance metrics can be explored, by incorperating e.g. the betweenness centrality in the distance metric.

Moreover, for a better evaluation of the method, our method (or derivations thereof) should be compared against more layout algorithms.

7 ACKNOWLEDGEMENTS

Firstly, I would like to thank my first supervisor, Alex Telea, for his advice and (elaborate!) feedback. Secondly, I want to thank my second supervisor, Paulo Rauber, for introducing me to his `thesne` implementation[20] and `Theano`, and for his great insights for improving the method. Moreover, I want to thank Rafael Messias Martins for staying in the loop, and providing the `visbrazil` dataset.

BIBLIOGRAPHY

[1] L. v. d. Maaten and G. Hinton, 'Visualizing data using t-sne', *Journal of Machine Learning Research*, vol. 9, no. Nov, pp. 2579–2605, 2008.

[2] T. Von Landesberger, A. Kuijper, T. Schreck, J. Kohlhammer, J. J. van Wijk, J.-D. Fekete and D. W. Fellner, 'Visual analysis of large graphs: State-of-the-art and future research challenges', in *Computer graphics forum*, Wiley Online Library, vol. 30, 2011, pp. 1719–1749.

[3] T. L. Saaty, 'The minimum number of intersections in complete graphs', *Proceedings of the National Academy of Sciences*, vol. 52, no. 3, pp. 688–690, 1964.

[4] P. Eades, 'A heuristics for graph drawing', *Congressus numerantium*, vol. 42, pp. 146–160, 1984.

[5] T. M. Fruchterman and E. M. Reingold, 'Graph drawing by force-directed placement', *Software: Practice and experience*, vol. 21, no. 11, pp. 1129–1164, 1991.

[6] C. Walshaw, 'A multilevel algorithm for force-directed graph drawing', in *International Symposium on Graph Drawing*, Springer, 2000, pp. 171–182.

[7] Y. Frishman and A. Tal, 'Multi-level graph layout on the gpu', *IEEE Transactions on Visualization and Computer Graphics*, vol. 13, no. 6, pp. 1310–1319, 2007.

[8] S. Martin, W. M. Brown, R. Klavans and K. W. Boyack, 'Openord: An open-source toolbox for large graph layout', in *IS&T/SPIE Electronic Imaging*, International Society for Optics and Photonics, 2011, pp. 786 806–786 806.

[9] M. Bastian, S. Heymann, M. Jacomy *et al.*, 'Gephi: An open source software for exploring and manipulating networks.', *ICWSM*, vol. 8, pp. 361–362, 2009.

[10] R. Kohavi and G. H. John, 'Wrappers for feature subset selection', *Artificial intelligence*, vol. 97, no. 1, pp. 273–324, 1997.

[11] H. Hotelling, 'Analysis of a complex of statistical variables into principal components.', *Journal of educational psychology*, vol. 24, no. 6, p. 417, 1933.

[12] J. B. Tenenbaum, V. De Silva and J. C. Langford, 'A global geometric framework for nonlinear dimensionality reduction', *Science*, vol. 290, no. 5500, pp. 2319–2323, 2000.

[13] F. V. Paulovich, L. G. Nonato, R. Minghim and H. Levkowitz, 'Least square projection: A fast high-precision multidimensional projection technique and its application to document mapping', *IEEE Transactions on Visualization and Computer Graphics*, vol. 14, no. 3, pp. 564–575, 2008.

[14] R. M. Martins, D. B. Coimbra, R. Minghim and A. Telea, 'Visual analysis of dimensionality reduction quality for parameterized projections', *Computers & Graphics*, vol. 41, pp. 26–42, 2014.

[15] R. M. Martins, R. Minghim, A. C. Telea *et al.*, 'Explaining neighborhood preservation for multidimensional projections', *EG UK Computer Graphics and Visual Computing*, 2015.

[16] R. Minghim, F. V. Paulovich and A. de Andrade Lopes, 'Content-based text mapping using multi-dimensional projections for exploration of document collections', in *Electronic Imaging 2006*, International Society for Optics and Photonics, 2006, 60600S–60600S.

[17] R. Messias Martins, 'Explanatory visualization of multidimensional projections', PhD thesis, 2016, ISBN: 978-90-367-8642-3.

[18] *"`graph-tool` documentation: All_shortest_paths"*, `https://graph-tool.skewed.de/static/doc/topology.html#graph_tool.topology.all_shortest_paths`, [Online; accessed 03-August-2016].

[19] K. P. Burnham and D. Anderson, 'Model selection and multi-model inference', *A Pratical informatio-theoric approch. Sringer*, 2003.

[20] P. E. Rauber, A. X. Falcão and A. C. Telea, 'Visualizing time-dependent data using dynamic t-sne', *Proceedings EuroVis Short Papers 2016*, 2016.

[21] T. T. D. Team, R. Al-Rfou, G. Alain, A. Almahairi, C. Angermueller, D. Bahdanau, N. Ballas, F. Bastien, J. Bayer, A. Belikov *et al.*, 'Theano: A python framework for fast computation of mathematical expressions', *ArXiv preprint arXiv:1605.02688*, 2016.

[22] T. P. Peixoto, 'The graph-tool python library', *Figshare*, 2014. DOI: `10.6084/m9.figshare.1164194`. [Online]. Available: `http://figshare.com/articles/graph_tool/1164194` (visited on 09/03/2016).

[23] J. Ellson, E. Gansner, L. Koutsofios, S. C. North and G. Woodhull, 'Graphviz - open source graph drawing tools', in *International Symposium on Graph Drawing*, Springer, 2001, pp. 483–484.

[24] S. Tomar, 'Converting video formats with ffmpeg', *Linux Journal*, vol. 2006, no. 146, p. 10, 2006.

[25] J. F. Kruiger, *Tsnetwork*, `https://github.com/HanKruiger/tsnetwork`, 2016.

[26] I. S. Duff, R. G. Grimes and J. G. Lewis, 'Sparse matrix test problems', *ACM Transactions on Mathematical Software (TOMS)*, vol. 15, no. 1, pp. 1–14, 1989.

[27] I. S. Du, R. G. Grimes and J. G. Lewis, 'Users' guide for the harwell-boeing sparse matrix collection (release i)', Report RAL-92-086, Atlas Centre, Rutherford Appleton Laboratory, Didcot, Oxon, UK, Tech. Rep., 1992.

[28] M. E. Newman, *Network data*, `http://www-personal.umich.edu/~mejn/netdata/`, [Online; accessed 27-July-2016].

[29] T. A. Davis and Y. Hu, 'The university of florida sparse matrix collection', *ACM Transactions on Mathematical Software (TOMS)*, vol. 38, no. 1, p. 1, 2011.

[30] Y. Hu, 'Efficient, high-quality force-directed graph drawing', *Mathematica Journal*, vol. 10, no. 1, pp. 37–71, 2005.

[31] *"`graph-tool` documentation: Sfdp_layout"*, `https://graph-tool.skewed.de/static/doc/draw.html#graph_tool.draw.sfdp_layout`, [Online; accessed 13-August-2016].

[32] G. Everstine and D. Taylor, *Hb/dwt_72 sparse matrix*, http://www.cise.ufl.edu/research/sparse/matrices/HB/dwt_72, [Online; accessed 09-March-2016].

[33] D. Knuth, *Newman/lesmis sparse matrix*, https://www.cise.ufl.edu/research/sparse/matrices/Newman/lesmis.html, [Online; accessed 30-June-2016].

[34] L. Marro, *Hb/can_96 sparse matrix*, https://www.cise.ufl.edu/research/sparse/matrices/HB/can_96.html, [Online; accessed 30-June-2016].

[35] Rajat, *Rajat/rajat11 sparse matrix*, https://www.cise.ufl.edu/research/sparse/matrices/Rajat/rajat11.html, [Online; accessed 30-June-2016].

[36] P. Gleiser and L. Danon, *Arenas/jazz sparse matrix*, https://www.cise.ufl.edu/research/sparse/matrices/Arenas/jazz.html, [Online; accessed 30-June-2016].

[37] *"graph-tool documentation: Lattice"*, https://graph-tool.skewed.de/static/doc/generation.html#graph_tool.generation.lattice, [Online; accessed 25-July-2016].

[38] NASA, *Pothen/mesh3e1 sparse matrix*, http://www.cise.ufl.edu/research/sparse/matrices/Pothen/mesh3e1, [Online; accessed 09-March-2016].

[39] M. E. Newman, 'Finding community structure in networks using the eigenvectors of matrices', *Physical review E*, vol. 74, no. 3, p. 036 104, 2006.

[40] G. Everstine and D. Taylor, *Hb/dwt_419 sparse matrix*, http://www.cise.ufl.edu/research/sparse/matrices/HB/dwt_419, [Online; accessed 30-June-2016].

[41] *"graph-tool documentation: Price_network"*, https://graph-tool.skewed.de/static/doc/generation.html#graph_tool.generation.price_network, [Online; accessed 09-March-2016].

[42] ——, *Hb/dwt_1005 sparse matrix*, http://www.cise.ufl.edu/research/sparse/matrices/HB/dwt_1005, [Online; accessed 30-June-2016].

[43] A. van Heukelum, *Vanheukelum/cage8 sparse matrix*, http://www.cise.ufl.edu/research/sparse/matrices/vanHeukelum/cage8, [Online; accessed 09-March-2016].

[44] J. Lewis, *Hb/bcsstk09 sparse matrix*, http://www.cise.ufl.edu/research/sparse/matrices/HB/bcsstk09, [Online; accessed 09-March-2016].

[45] *"graph-tool documentation: Random_graph"*, https://graph-tool.skewed.de/static/doc/generation.html#graph_tool.generation.random_graph, [Online; accessed 30-June-2016].

[46] *General relativity and quantum cosmology collaboration network*, https://snap.stanford.edu/data/ca-GrQc.html, [Online; accessed 09-March-2016].

[47] K. Norlen, G. Lucas, M. Gebbie and J. Chuang, *Pajek/eva sparse matrix*, http://www.cise.ufl.edu/research/sparse/matrices/Pajek/EVA.html, [Online; accessed 30-June-2016].

[48] D. J. Watts and S. H. Strogatz, 'Collective dynamics of small-world networks', *Nature*, vol. 393, no. 6684, pp. 440–442, 1998.

[49] G. Everstine and D. Taylor, *Matrix dwt 1005*, http://math.nist.gov/MatrixMarket/data/Harwell-Boeing/dwt/dwt_1005.html, [Online; accessed 12-Aug-2016].

[50] L. Van Der Maaten, 'Accelerating t-sne using tree-based algorithms.', *Journal of machine learning research*, vol. 15, no. 1, pp. 3221–3245, 2014.

[51] J. Duchi, E. Hazan and Y. Singer, 'Adaptive subgradient methods for online learning and stochastic optimization', *Journal of Machine Learning Research*, vol. 12, no. Jul, pp. 2121–2159, 2011.

[52] D. Kingma and J. Ba, 'Adam: A method for stochastic optimization', *ArXiv preprint arXiv:1412.6980*, 2014.