

Graph Layouts by t-SNE

J. F. Kruijger^{1,2}, P. E. Rauber^{1,3}, R. M. Martins⁴, A. Kerren⁴, S. Kobourov⁵, A. C. Telea¹

¹University of Groningen, the Netherlands

²École Nationale de l'Aviation Civile, France

³University of Campinas, Brazil

⁴Linnaeus University, Sweden

⁵University of Arizona, USA

Abstract

We propose a new graph layout method based on a modification of the *t*-distributed Stochastic Neighbor Embedding (*t*-SNE) dimensionality reduction technique. Although *t*-SNE is one of the best techniques for visualizing high-dimensional data as 2D scatterplots, *t*-SNE has not been used in the context of classical graph layout. We propose a new graph layout method, *tsNET*, based on representing a graph with a distance matrix, which together with a modified *t*-SNE cost function results in desirable layouts. We evaluate our method by a formal comparison with state-of-the-art methods, both visually and via established quality metrics on a comprehensive benchmark, containing real-world and synthetic graphs. As evidenced by the quality metrics and visual inspection, *tsNET* produces excellent layouts.

1. Introduction

Graph drawing (GD) is an important subfield of information visualization. Yet, making a ‘good’ drawing becomes more difficult for large and densely-connected graphs. Assuming edges are drawn as straight-line segments, the main goal of graph drawing is assigning appropriate coordinates to graph nodes, an operation known as *graph layout*. Many graph layout methods exist, *e.g.*, force-directed methods [Ead84,FR91], spectral methods [Hal70,BP07], and dimensionality-reduction methods [KS80,HK04,GKN05].

In contrast to traditional GD methods, dimensionality reduction (DR) methods typically map the graph connectivity information to a high-dimensional space [HK04], *e.g.*, via the graph-theoretic distances between all pairs of nodes [KS80,KKH89]. This high-dimensional data is then suitably mapped to 2D space to create the graph layout. While graph drawing by DR is elegant and simple, few methods exist in this class. This provides opportunities for improved graph layout methods that take advantage of advances in DR which provide increasingly more accurate, flexible, scalable, and easy-to-use techniques, as described in detail in Sect. 2.

In this paper, we propose a new DR-based method for graph drawing. We leverage *t*-SNE [vdMH08], one of the most acclaimed DR methods, extensively used in machine learning, computer vision, and information visualization [MKS*15,DJV*14]. In contrast to other DR methods used for GD—which aim to preserve *distances* between points when mapping these to 2D space—*t*-SNE aims to preserve point *neighborhoods*. As we argue later on, neighborhood preservation is desirable for GD. We describe a way to

encode the graph as a distance matrix, along with a suitable modification to the standard *t*-SNE cost function and iteration parameters, that allow us to map the graph’s connectivity to a 2D projection. Our method—called *tsNET*, a play on ‘*t*-SNE’ and ‘network’—is simple to implement and use, and has a single free parameter, perplexity, described in more detail in Sect. 5. Moreover, *tsNET* inherits the proven robustness and quality properties of *t*-SNE, and generates high quality layouts for a wide variety of graphs. We evaluate *tsNET* by comparing it quantitatively and qualitatively with several state-of-the-art methods on a representative family of real-world and synthetic graphs.

This paper is organized as follows. Sect. 2 discusses related work in DR and graph layouts. Sect. 3 presents our method. Sect. 4 shows our results on a comprehensive graph benchmark. Sect. 5 discusses *tsNET*, and Sect. 6 draws conclusions and outlines future work.

2. Background

Our work is related to both dimensionality reduction and graph drawing. We review related work in both of these areas below.

2.1. Dimensionality reduction

Dimensionality reduction (DR) considers the problem of reducing the number of dimensions of a dataset while retaining relevant data patterns. Given a set of N n -dimensional observations $\{\mathbf{x}_i \in \mathbb{R}^n\}_{i=1}^N$, a DR technique is a function $f: \mathbb{R}^n \rightarrow \mathbb{R}^m$ that maps $\{\mathbf{x}_i\}$ to a set of low-dimensional points $\{\mathbf{y}_i \in \mathbb{R}^m\}_{i=1}^N$,

where $m \ll n$ and typically $m \in \{2, 3\}$, so that the so-called ‘data structure’ is preserved [MCMT14]. DR methods can be classified into *distance-preserving* and *neighborhood-preserving* methods. Distance-preserving methods aim to minimize a cost such as the so-called aggregated normalized *stress*

$$\sigma = \sum_{i,j} \left(\frac{d(\mathbf{x}_i, \mathbf{x}_j) - \|\mathbf{y}_i - \mathbf{y}_j\|}{d(\mathbf{x}_i, \mathbf{x}_j)} \right)^2. \quad (1)$$

Here, $d(\cdot, \cdot)$ is a distance metric over the input space of f and $\|\cdot\|$ is usually the Euclidean 2D distance. Such methods are used when reasoning about inter-point distance ratios is important.

Neighborhood-preserving methods try to maximize overlap between k -nearest neighborhoods in the input (\mathbb{R}^n) and output (\mathbb{R}^m) spaces, and help find groups and outliers. Distance-preserving and neighborhood-preserving methods have related, but not identical, aims. In particular, distance-preservation gets hard when the dimensionality n is high [dST03, JPC*11]; neighborhood-preserving methods handle high n values better [vdMH08, PLvdM*15].

DR methods can also be classified into *projections* or *distance-based* methods. Projections require the actual \mathbb{R}^n coordinates of the N input points, and thus require $\mathcal{O}(nN)$ in space complexity [MCMT14]. Such methods are not directly applicable in a GD setting, as we do not typically have a way to generate n -dimensional coordinates of nodes in a graph. Distance-based methods, also called multidimensional scaling (MDS) methods, use only the pairwise distances between all \mathbf{x}_i , and require $\mathcal{O}(N^2)$ space complexity [Tor52, BSL*08]. MDS methods are more general than projections, as they do not require explicit \mathbb{R}^n coordinates for the input points.

In the MDS class, ISOMAP [TdSL00] first determines which data points are neighbors in \mathbb{R}^n , then determines geodesic distances between these points (in \mathbb{R}^n), and finally performs classical MDS with the pairwise geodesic distances. LSP [PNML08] works by first projecting a subset of the \mathbb{R}^n points via classical MDS, and next uses neighborhood information from \mathbb{R}^n to place the rest of the data points in \mathbb{R}^m . LAMP [JPC*11] is similar to LSP. It first projects a subset of points via classical MDS, yet it also allows the user to interact with the positioning of these points. IDMAP [MPd06] uses a distance metric based on cosine similarity, and subsequently makes a FastMap [FL95] projection which in turn uses a force-based placement approach. Landmark MDS [dST03] and PivotMDS (PMDS) [BP07] are sampling-based approximations of classical MDS. They place representatives from the input data with high accuracy, while remaining nodes are placed via linear combinations of the representatives. Both methods are very fast. PMDS typically produces better results as it takes into account distances to non-representatives when placing the representatives. For a complete overview of DR methods, we refer the reader to recent surveys in the area [SVPM14, CG15].

2.2. Graph layouts

General considerations: A graph $G = (V, E)$, also known as a network or relational dataset, consists of a set of nodes $V = \{\mathbf{x}_i\}_{i=1}^N$ and a set of edges $E = \{(\mathbf{x}_i, \mathbf{x}_j)\} \subseteq V \times V$ that map relations between nodes in V . Both nodes and edges can have additional data

attributes (such as weights), although many GD methods ignore weights.

A graph layout algorithm can be seen as a function L that takes as input a graph G and outputs a set of (typically 2D) node coordinates, i.e., $L(G) = \{\mathbf{y}_i \in \mathbb{R}^2\}_{i=1}^N$. We cannot describe all graph layout techniques here; for recent comprehensive surveys on the topic we refer to [Tam13, vLKS*11]. Next we provide details about graph layout methods that are particularly relevant for our goal—drawing large and potentially highly connected graphs.

A popular choice for graph layouts is SFDP [Hu05], which is available in GraphViz [ATT]. SFDP uses a multilevel approach and quadtree optimizations, which makes it usable for fairly large graphs. A drawback of SFDP is that it rigidly enforces uniform-length edges, which can be aesthetically pleasing but does not always result in the most insightful layout.

GRIP [GK01] is a nearly linear multilevel method that uses node filtrations based on maximal independent sets. A similar method uses a hierarchy of progressively less coarse graphs [Wal01]. Multilevel approaches make graph layout more scalable, but assume that pairs of nodes with small graph-theoretic distances also have small distances in the optimal layout [Noa07a].

The r -PolyLog class of energy models [Noa07a], which contains the LinLog energy model, aim to compromise between enforcing uniform edge-lengths and separating highly connected node clusters. LinLog can draw graphs that consist primarily of clusters very well, but is less successful for other graph types.

Graph layouts by DR: A particular class of graph layout methods is explicitly based on DR techniques and we discuss these in more detail, as they are most relevant to our method. A key to the DR-approach is suitably defining the distance d so that minimizing the stress σ (Eqn. (1)) reflects a good layout. For this, d can be set to the graph-theoretical shortest-path distance between nodes \mathbf{x}_i and \mathbf{x}_j . Minimizing stress for graph drawing was originally done by [KS80], and later by [KKH89]. NEATO [Nor04] is a popular implementation that minimizes the stress, and is available in GraphViz [ATT]. Graph drawing by stress majorization [GKN05] minimizes stress by means of *majorization*. A drawback of minimizing stress directly is that it fails to untangle many complex graphs because it aims to rigidly preserve distance.

ACE [KCH03] is a very fast method that uses a matrix-representation for the graph and several of its properties. It uses transformations to coarser, lower-dimensional matrices, and finds a layout that is projected back to the higher-dimensional representation where it is adjusted to solve the original problem. ACE works well for mesh-like graphs, but produces sub-optimal layouts for sparsely-connected graphs, such as trees. A related multilevel method is [FT07].

Graph drawing by high-dimensional embedding [HK04] is a very fast method that first makes an n -dimensional embedding, using n pivot nodes and the graph-theoretic distances from those pivot nodes to all other nodes. Subsequently, dimensionality is reduced from n to 2 by means of PCA. This method does not work well for non-mesh-like graphs. Also, PCA is well-known to poorly pre-

serve distances when the n -dimensional data does not lie on a linear subspace of \mathbb{R}^n .

[Kor04] builds on the high-dimensional embedding idea by additionally considering the high-dimensional subspace spanned by the eigenvectors of the Laplacian matrix of the graph, and projects this to 2D utilizing the graph's structure. This method works well for meshes but less so for other graph types.

ws-SNE [YPK14] proposes a general framework to theoretically unify DR methods and GD methods. ws-SNE also uses cost terms originating from t-SNE, but employs different distance measures and optimization strategies than we propose here (for details, see Sect. 3.2). More recently, s-SNE [LYC16] built further on ws-SNE by considering spherical projections. We did not incorporate these methods to our comparison due to the minimal implementation details of ws-SNE and the different layout space of s-SNE.

2.3. Contributions

Compared to other DR-based graph layout methods, our proposal is characterized by the following three aspects:

- we use the neighborhood-preserving t-SNE technique rather than the distance-preserving techniques prevalent in current methods;
- we use a modified cost function with terms based on t-SNE and classical force-based methods;
- we use the graph-theoretical shortest-path distances in the input space.

Additionally, we propose a refined method, tsNET*, which improves over tsNET by using suitable node pre-placement. Finally, a major contribution of our work is the extensive evaluation of several graph layout methods on a varied set of graphs.

3. Method

We next present our graph layout method. As our method exploits t-SNE, we first outline t-SNE (Sect. 3.1). Our method proper follows in Sect. 3.2.

3.1. t-SNE

t-SNE is a neighborhood-preserving MDS method that is often used for creating scatterplots of high-dimensional data [vdMH08]. It works by defining probabilities p_{ij} of picking a point-pair in the input space and probabilities q_{ij} of picking a point-pair in the output (2D) space. The probability p_{ij} of picking the pair $(\mathbf{x}_i, \mathbf{x}_j)$ 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 $p_{j|i}$ are given by the normalized Gaussian distribution

$$p_{j|i} = \exp\left(-\frac{d_{ij}^2}{2\sigma_i^2}\right) / \sum_{k \neq i} \exp\left(-\frac{d_{ik}^2}{2\sigma_i^2}\right), \quad p_{i|i} = 0,$$

which gives the probability that \mathbf{x}_i has \mathbf{x}_j as neighbor. Here, d_{ij} is the distance between \mathbf{x}_i and \mathbf{x}_j in input space. The Gaussian standard deviation σ_i can be either hand-picked or found *e.g.*, by binary

search so that the perplexity $\kappa_i = 2^{-\sum_j p_{j|i} \log_2 p_{j|i}}$ for every point \mathbf{x}_i matches a user-defined value. The perplexity measures the effective number of neighbors of a point. If \mathbf{x}_i is in a dense region, σ_i generally has a low value to match the perplexity. In sparser regions, σ_i typically gets a large value. Finally, q_{ij} , the probability of picking the pair $(\mathbf{y}_i, \mathbf{y}_j)$ in the output (2D) space, is given by

$$q_{ij} = q_{ji} = \frac{(1 + \|\mathbf{y}_i - \mathbf{y}_j\|^2)^{-1}}{\sum_{\substack{k,l \\ k \neq l}} (1 + \|\mathbf{y}_k - \mathbf{y}_l\|^2)^{-1}}, \quad q_{ii} = 0,$$

which is a normalized (heavy-tailed) Student's t -distribution.

Node positions \mathbf{y}_i in the 2D space are found by minimizing (with respect to \mathbf{y}_i) the Kullback-Leibler divergence between the probabilities of picking pairs of low- and high-dimensional data points:

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

This minimization is typically done using gradient descent. The gradient $\frac{\partial C_{KL}}{\partial \mathbf{y}_i}$ is easily expressed in analytic form and the minimization is therefore straightforward to implement.

t-SNE is among the best MDS methods that succeeds in emphasizing relevant point-groups while also capturing local variation within groups [MKS*15, DJV*14].

3.2. tsNET

Given a graph G , our layout method has three steps, as follows. We first compute the graph-theoretic shortest-path distances d_{ij} between all nodes \mathbf{x}_i and \mathbf{x}_j of G . If no path exists between \mathbf{x}_i and \mathbf{x}_j , d_{ij} is set to a large arbitrary value. This yields a symmetric $N \times N$ matrix $\mathbf{D} = (d_{ij})$. Next, we use \mathbf{D} to construct a cost function C as a sum of three terms:

$$C = \lambda_{KL} C_{KL} + \frac{\lambda_c}{2N} \sum_i \|\mathbf{y}_i\|^2 - \frac{\lambda_r}{2N^2} \sum_{\substack{i,j \\ i \neq j}} \log(\|\mathbf{y}_i - \mathbf{y}_j\| + \epsilon_r). \quad (2)$$

Here, term 1 is the Kullback-Leibler divergence from t-SNE (Sect. 3.1); term 2 denotes so-called compression, which is known to reduce the t-SNE optimization time [vdMH08]; and term 3 repulses nodes \mathbf{y}_i to prevent undesirable clutter and attain even dispersion (entropy in [GHN13]). The weights λ_{KL} , λ_c , and λ_r govern the impacts of the different terms, and do change during optimization. A small regularization constant $\epsilon_r = \frac{1}{20}$ is added to treat situations of nearly co-located nodes. We have also considered electrostatic repulsion, but finally settled on the entropy from [GHN13] motivated by empirical results and the simple analytic form of its gradient when $\epsilon_r = 0$: $\frac{\partial}{\partial \mathbf{y}_k} \left(\frac{1}{2} \sum_{\substack{i,j \\ i \neq j}} \log(\|\mathbf{y}_i - \mathbf{y}_j\|) \right) = \sum_{\substack{i \\ i \neq k}} \frac{\mathbf{y}_k - \mathbf{y}_i}{\|\mathbf{y}_k - \mathbf{y}_i\|^2}$.

The 2D node positions $\{\mathbf{y}_i\}_{i=1}^N$ are found by minimizing C by means of momentum-based gradient descent, as in classical t-SNE. For this, we propose a three-stage method. First, we randomly initialize \mathbf{y}_i . Secondly, we do momentum-based gradient descent optimization on C with $(\lambda_{KL}, \lambda_c, \lambda_r) = (1, 1.2, 0)$ until the summed displacement of nodes in the layout is smaller than a given threshold.

In the third stage, we use the node positions to do the same optimization, but with $(\lambda_{KL}, \lambda_c, \lambda_r) = (1, 0.01, 0.6)$. The second stage aims at accelerating the ‘untangling’ of the graph with the aid of compression. The third stage optimizes for the final layout where we want to prevent too closely placed vertices and the fisheye distortion introduced by high λ_c . We call the above method tsNET.

As a refinement of this method, we initialize \mathbf{y}_i using PivotMDS in stage 1, and then in stage 2 use the parameter settings $(\lambda_{KL}, \lambda_c, \lambda_r) = (1, 0.1, 0)$. Stage 3 remains unchanged. The weight of the compression term in stage 2 has been lowered to prevent it from distorting the PivotMDS layout. We call this method tsNET*. Section 4 will show the advantages of tsNET* over tsNET.

To better explain the effect of the individual cost terms, Fig. 1 shows layouts computed with and without certain cost terms on a graph of 1005 nodes and 3808 edges. We see that plain t-SNE ($\lambda_c = \lambda_r = 0$), shown in Fig. 1a, generates a layout with undesired occlusions, folds, an uneven node density, and has a slow convergence (requires many iterations). As such, plain t-SNE is not enough to generate a good graph drawing. Such aspects are removed by the additional components of tsNET and tsNET*, as Figs. 1b to 1e show. We observed similar effects on other graphs we tested (which are introduced further in Sect. 4.1).

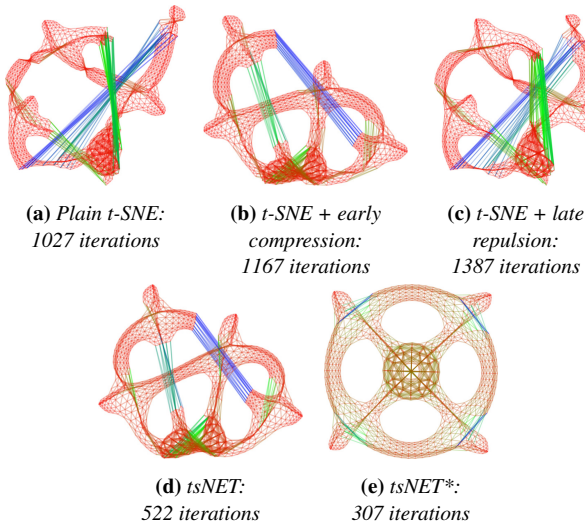


Figure 1: The effect of the additional cost terms, with the number of gradient descent iterations needed for convergence.

Important differences exist between ws-SNE and tsNET. First, ws-SNE uses different cost functions: Different repulsion terms are used (compare Eqn. (2), term 3, with $\sum_{ij} p_{ij} \|\mathbf{y}_i - \mathbf{y}_j\|^2 + \lambda \sum_{ij} M_{ij} \exp(-\|\mathbf{y}_i - \mathbf{y}_j\|^2)$ from [YPK14]). We use a compression term (Eqn. (2), term 2) to reduce optimization time. To our understanding, ws-SNE does not (and cannot) use such a term, as it relies on so-called separable divergences (sums of pairwise terms depending on data points i and j). Moreover, ws-SNE uses the degree centrality of nodes in its repulsion term (M_{ij} in the above expression), to favor good placement of central nodes. As shown next in Sect. 4, we obtain good results even without computing such a term, which can be expensive for large graphs. Finally, we compute

p_{ij} from shortest-path distances, whereas ws-SNE assumes that p_{ij} are computed from the k -nearest-neighbors of graph nodes; however, how this is precisely done is not specified in [YPK14], nor are shortest-path distances named in this context.

We have implemented tsNET in *Python 3*, using the *Theano* numerical library [BBB*]. Our open source is available online at [Kru16]. Its space and time complexities are $\mathcal{O}(N^2)$ and $\mathcal{O}(tN^2)$, resp., with t the number of gradient descent iterations—around several hundreds in our benchmark. If desired, massive accelerations are easily doable, see further Sect. 5.

4. Evaluation

We evaluate our layout method by comparing it to several state-of-the-art layout methods on a set of synthetic and real-world graphs with various sizes (Sect. 4.1). Sect. 4.2 compares the obtained graph layouts visually. Sects. 4.3 to 4.5 compare layouts via several quantitative metrics. Sect. 4.6 compares running times for our methods. Sect. 4.7 shows how our layouts can be further enhanced by edge bundling. Finally, Sect. 4.8 demonstrates that our method can also construct 3D layouts.

4.1. Benchmark

To evaluate tsNET(*), we used a wide set of graphs and layout methods, as follows.

Tab. 1 lists the graphs we used for benchmarking. These include real-world collaboration networks, structural problems from the Harwell-Boeing collection [DGL89a], mesh-like graphs, and tree-like graphs. Many of our graphs come from the Florida collection [DH11], a well-known source for graph drawing and graph analysis. Other sources are synthetic graphs computed with [Pei14, ATT] and graphs from [MAH*12] (a recent DR-based graph layout method). These graphs are used in additional papers beyond those listed in Tab. 1. However, these could not all be included for space reasons.

We compare tsNET(*) with the following layout methods:

- IDMAP** [MPd06], as implemented in VisPipeline [POM07];
- LSP** [PNML08], as implemented in VisPipeline [POM07], with number of control points set to 10% of the number of nodes;
- PMDS** [BP07], as implemented in Tulip 4.8.1 [Aub04], with number of pivots set to 10% of the number of nodes;
- SFDP** [Hu05], as implemented in GraphViz 2.38 [ATT];
- LinLog** [Noa07a], as implemented by the author [Noa07b];
- GRIP** [GK01], as implemented by the authors [Yus01]. The GRIP implementation in Tulip was also evaluated, but it produced visually inferior results, which are not shown here.
- NEATO** [Nor04], as implemented in GraphViz 2.38 [ATT];

All compared methods are well-known, and have open-source implementations, which makes it easy to replicate our experiments. We chose IDMAP and LSP, which are DR projections, specifically to show the advantages of our t-SNE-based method as opposed to other DR methods. We chose GRIP, PMDS, SFDP, and LinLog since these are well-known methods, present in many graph drawing papers. However, it is fair to note that these methods do not

Name	Type	V	E	Source	Also seen in
dwt_72	planar, structural	72	75	[DGL89b]	[HBFR14]
lesmis	collaboration network	77	254	[Knu93]	
can_96	mesh-like, structural	96	336	[DGL89b]	[HBFR14]
rajat11	miscellaneous	135	377	[Raj]	
jazz	collaboration network	198	2742	[GD03]	
visbrazil	collaboration network	222	336	[MAH*12]	
grid17	planar, mesh-like, structural	289	544	synthesized using [Pei14]	
mesh3e1	mesh-like, structural	289	800	[NAS]	[Tam13]
netscience	collaboration network	379	914	[New06]	[GJ12]
dwt_419	structural	419	1572	[DGL89b]	[PV13, HBFR14]
price_1000	planar, tree	1000	999	synthesized using [Pei14]	
dwt_1005	structural	1005	3808	[DGL89b]	[OKB16]
cage8	miscellaneous	1015	4994	[VHBB02]	
bcsstk09	mesh-like, structural	1083	8677	[DGL89b]	[FT07]
block_2000	clusters	2000	9912	synthesized using [Pei14]	
sierpinski3d	structural	2050	6144	synthesized using [ATT]	[GK01]
CA-GrQc	collaboration network	4158	13422	[New01]	[BG13]
EVA	collaboration network	4475	4652	[NLGC02]	
3elt	planar, mesh-like	4720	13722	[DP]	[Wal01, OKB16, FT07]
us_powergrid	structural	4941	6594	[WS98]	[OKB16, GHK13]

Table 1: Types and sizes of graphs in our benchmark. All graphs we used are publicly available, as indicated.

use a ‘full stress’ model, in contrast to ours. On the other hand, we compare also with NEATO, which does use such a model. For all graphs, we use the same method parameters, as given in Sect. 3.2, except $\lambda_r = 0.1$ for dwt_72, which gave better results.

4.2. Visual comparison

Tabs. 5 and 6 visually compare the obtained layouts; rows indicate graphs, and columns indicate layout methods. We see that tsNET* is of the same or, we argue, in some cases higher visual quality than the compared layouts, in terms of node clutter, minimizing unnecessary distortions, and preserving symmetry in the input graphs. Edge length distribution, encoded in edge color, shows that tsNET* slightly favors short edges (column 2 in Tabs. 5 and 6 shows relatively more red than the other columns). Yet, this does not adversely affect the visual quality of the layouts. Another interesting point is that tsNET* has a wider edge-length distribution than most other layouts. For instance, for dwt_1005, block_2000 and 3elt, our method chooses to draw a few long edges (blue) but keeps most other edges short (red). This selectively ‘pulls apart’ a few connected nodes so that the layout is nicely spread out over the 2D plane. Another finding is that our layouts appear more smooth and organic than others; structurally-similar graph regions, e.g., in mesh-like graphs (dwt_1005, bcsstk09, can_96, grid17, mesh3e1, and dwt_419) also appear structurally similar in our layouts, whereas some of the other methods produce unnecessary deformations. Finally, we see that tsNET* improves upon tsNET by removing artifacts where part of the tsNET layout is ‘folded over’, see e.g., can_96, dwt_1005 and 3elt.

4.3. Normalized stress metric

A compact way to assess the distance-preservation of a layout is to use the scalar normalized stress metric σ (Eqn. (1)), see e.g., [OKB16, GHN13]. Tab. 2 shows σ for all considered layouts. We see that tsNET(*) do not reach significantly better values than all other considered methods. In particular, NEATO, GRIP

and IDMAP perform better here. Yet, if we look at the actual layouts (Tabs. 5 and 6), IDMAP and GRIP deliver arguably worse results—lack of symmetry preservation (IDMAP on dwt_1005, bcsstk09), weak cluster preservation (block_2000), unnecessary distortions (3elt, IDMAP and GRIP on bcsstk09), and folding artifacts (NEATO on dwt_419, dwt_1005). Specifically, consider the low stress values of NEATO, IDMAP and GRIP on block_2000. It is evident that the stress metric does not capture the quality aspects that we require for this graph. This makes us hypothesize that pure distance preservation might not be the ideal aim of a good graph layout method. Rather, having groups of nodes topologically close in the graph be also geometrically close in the layout allows reading the drawing well [Tam13, Noa07a]. This neighbor preservation, discussed in Sect. 4.5, is precisely what tsNET aims at.

	tsNET	tsNET*	IDMAP	LSP	PMDS	SFDP	LinLog	GRIP	NEATO
dwt_72	0.048	0.048	0.039	0.118	0.072	0.061	0.201	0.038	0.043
lesmis	0.109	0.111	0.111	0.226	0.162	0.112	0.213	0.099	0.084
can_96	0.112	0.084	0.085	0.088	0.092	0.075	0.091	0.104	0.072
rajat11	0.096	0.097	0.097	0.098	0.107	0.096	0.194	0.074	0.064
jazz	0.127	0.128	0.126	0.155	0.158	0.137	0.387	0.114	0.110
visbrazil	0.098	0.098	0.083	0.113	0.157	0.081	0.474	0.089	0.068
grid17	0.021	0.021	0.016	0.026	0.025	0.023	0.210	0.018	0.014
mesh3e1	0.014	0.014	0.004	0.006	0.003	0.036	0.076	0.009	0.005
netscience	0.101	0.100	0.073	0.096	0.103	0.105	0.192	0.070	0.063
dwt_419	0.024	0.024	0.023	0.022	0.026	0.052	0.112	0.022	0.034
price_1000	0.165	0.160	0.117	0.159	0.248	0.133	0.190	0.126	0.093
dwt_1005	0.152	0.035	0.030	0.030	0.029	0.029	0.219	0.026	0.036
cage8	0.185	0.205	0.151	0.142	0.140	0.147	0.207	0.150	0.122
bcsstk09	0.022	0.022	0.037	0.027	0.066	0.024	0.096	0.021	0.015
block_2000	0.193	0.189	0.164	0.205	0.181	0.162	0.302	0.155	0.144
sierpinski3d	0.077	0.093	0.152	0.092	0.091	0.079	0.330	0.068	0.063
CA-GrQc	0.182	0.189	0.150	0.175	0.182	0.148	0.220	0.172	0.129
EVA	0.171	0.161	0.148	0.141	0.233	0.124	0.325	0.149	0.098
3elt	0.110	0.090	0.052	0.045	0.057	0.060	0.317	0.049	0.046
us_powergrid	0.150	0.101	0.074	0.080	0.090	0.094	0.271	0.091	0.058
average	0.103	0.094	0.083	0.097	0.106	0.085	0.219	0.078	0.069


Normalized stress (rel.)
low (good)  high (bad)
mean

Table 2: Normalized stress (Eqn. (1)). Layouts are uniformly scaled to give the minimal stress value. Cell colors encode the relative stress on the same row.

4.4. Distance preservation plots

Another measure used to evaluate the quality of a graph layout, or projection, is to look at the correlation between inter-point

tances in input vs. output space [BP09, GHN13, JPC* 11]. Fig. 2 shows such plots for our considered graphs and methods. Here, grayscale encodes the density of point-pairs (darker=more). Both input and output space distances are normalized by the largest distances in the individual spaces. Such plots help see how well graph-theoretic distances are preserved in the layout—for a perfect preservation, the plot is a diagonal line. Note that the x -axis is discrete, since \mathbf{D} contains graph-theoretic distances.

The plots for tsNET* do not show a larger spread (with respect to the diagonal) than those of compared methods. This is very interesting, since our underlying t-SNE method does not aim to preserve distances, but neighborhoods, as explained in Sect. 3. Moreover, the absence of dark regions close to the x -axis for tsNET(*) indicates that our layouts have few false neighbors, *i.e.*, nodes are well spread out in the available 2D layout space. The gaps in the plots for block_2000 by tsNET(*) and LinLog indicate that graph clusters are well separated. Yet, although LinLog can separate such clusters, it performs worse than our method for other types of graphs (see dwt_1005, 3elt and us_powergrid). Also, the cluster separation of LinLog is arguably of lower quality than ours, given the dark regions in the bottom-right part of its plot for block_2000, which indicate false neighbors. Another, more general observation from the plots in Fig. 2 is that the distribution of points for tsNET(*) tends to be more symmetric with respect to the diagonal, *i.e.*, there is a balance between the number of point-pairs in the layout that are false neighbors and those that are missing neighbors.

4.5. Neighborhood preservation metric

As argued in Sect. 4.3, we aim to provide a good graph layout by preserving neighbors rather than distances. To measure this, we use a neighborhood preservation metric similar to [GHN13]. While [GHN13] use a fixed neighborhood size, we change this size according to the position of a node in the graph. For every node \mathbf{x}_i in a graph $G = (V, E)$, we define a neighborhood $N_G(\mathbf{x}_i, r_G)$ as the nodes with a graph-theoretic distance of at most r_G from \mathbf{x}_i , *i.e.*,

$$N_G(\mathbf{x}_i, r_G) = \{\mathbf{x}_j \in V \mid d_{ij} \leq r_G\}.$$

Likewise, we define an equally sized neighborhood of \mathbf{x}_i in the layout $N_Y(\mathbf{x}_i, k_i)$ as the set of nodes corresponding to the points that are the k_i -nearest-neighbors of \mathbf{y}_i in the 2D layout space, with $k_i = |N_G(\mathbf{x}_i, r_G)|$. The neighborhood preservation ν is next defined as the fraction of nodes that are shared by these neighborhoods, averaged over all nodes in G , *i.e.*,

$$\nu = \frac{1}{|V|} \sum_i \frac{|N_G(\mathbf{x}_i, r_G) \cap N_Y(\mathbf{y}_i, k_i)|}{|N_G(\mathbf{x}_i, r_G) \cup N_Y(\mathbf{y}_i, k_i)|}.$$

This is the Jaccard similarity between the neighborhoods N_G and N_Y , averaged over G . Simply put, ν tells how well the 2D (layout) neighborhoods match the input (graph) neighborhoods [MMT15].

Tab. 3 shows ν for all our tests for $r_G = 2$. Evaluations for $r_G \in \{1, 3\}$ yielded similar results, omitted here for space reasons. The table shows very good results for tsNET(*): Not only does tsNET(*) often outperform other methods, it also does so by a large margin. In cases where tsNET(*) is below the median, the margins are small. Since the metric ν is an inverse measure of the number of false and missing neighbors, it is related to the observation about

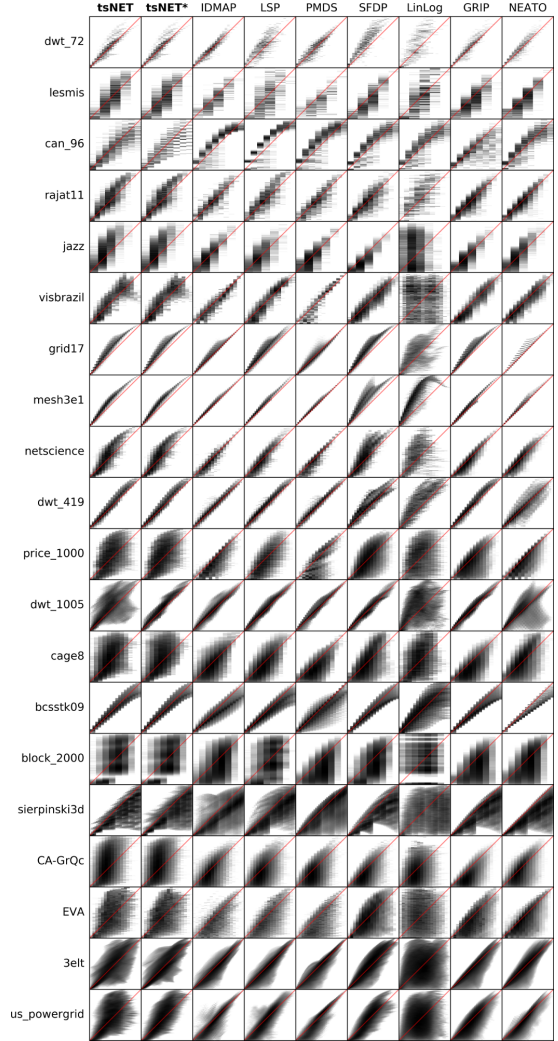


Figure 2: Plots of pairs of input (x -axis) and output (y -axis) distances. Gray value encodes the density of graph-distance vs. Euclidean-distance pairs.

false neighbors in the previous section. This good performance of tsNET(*) is directly related to the neighbor-preserving nature of t-SNE.

4.6. Running time

Tab. 4 gives the running time for tsNET(*) for the larger graphs in our benchmark, using the Python-based implementation, on a 3.4 GHz PC running Linux. We see that tsNET* provides a speed-up over tsNET of about 15% on average. Although our Python implementation is not fast, it is simple and easy to understand and use (see also our code [Kru16]). If running time is an important consideration, more efficient t-SNE implementations in C, using GPU acceleration [vdM] or quadtrees [vdM14, PLvdM* 15] can be directly used instead. Our method adds little overhead to t-SNE, as the PivotMDS preprocessor used for tsNET* takes less than a second on all tested graphs.

	tsNET	tsNET*	IDMAP	LSP	PMDS	SFDP	LinLog	GRIP	NEATO
dwt_72	0.855	0.855	0.770	0.676	0.732	0.714	0.692	0.914	0.828
lesmis	0.715	0.712	0.748	0.642	0.674	0.729	0.698	0.666	0.695
can_96	0.658	0.671	0.533	0.530	0.515	0.547	0.540	0.533	0.565
rajat11	0.716	0.717	0.675	0.638	0.624	0.661	0.637	0.634	0.655
jazz	0.805	0.804	0.842	0.791	0.827	0.840	0.777	0.824	0.817
visbrazil	0.589	0.584	0.476	0.449	0.414	0.471	0.542	0.452	0.425
grid17	0.785	0.785	0.812	0.750	0.727	0.751	0.369	0.804	1.000
mesh3e1	0.904	0.904	0.993	0.957	0.994	0.809	0.587	0.896	0.999
netscience	0.711	0.707	0.539	0.583	0.473	0.622	0.614	0.559	0.510
dwt_419	0.739	0.741	0.723	0.741	0.695	0.654	0.542	0.751	0.658
price_1000	0.639	0.639	0.483	0.469	0.422	0.528	0.594	0.216	0.284
dwt_1005	0.609	0.619	0.512	0.503	0.485	0.523	0.390	0.516	0.455
cake8	0.435	0.437	0.207	0.278	0.221	0.235	0.349	0.193	0.206
bcsstk09	0.867	0.867	0.767	0.795	0.602	0.835	0.565	0.856	0.973
block_2000	0.374	0.372	0.205	0.279	0.166	0.287	0.339	0.155	0.160
sierpinski3d	0.579	0.580	0.387	0.492	0.326	0.534	0.438	0.549	0.561
CA-GrQc	0.480	0.483	0.170	0.207	0.179	0.183	0.349	0.081	0.119
EVA	0.801	0.802	0.707	0.706	0.717	0.696	0.780	0.406	0.459
3elt	0.663	0.715	0.415	0.485	0.384	0.595	0.248	0.576	0.506
us_powergrid	0.454	0.457	0.231	0.353	0.253	0.429	0.409	0.233	0.215
average	0.842	0.852	0.469	0.426	0.296	0.541	0.332	0.386	0.399

Neighborhood preservation (rel.)
least preserving (bad) ■ ■ ■ ■ ■ ■ ■ ■ most preserving (good)
mean

Table 3: Neighborhood preservation metric ν for $r_G = 2$ (higher is better), indicating how well the input (graph) and output (layout) node-neighborhoods match. Cell colors encode ν on the same row.

	tsNET (s)	tsNET* (s)	tsNET* (% of tsNET)
dwt_72	3.2	3.1	96%
lesmis	4.2	4.7	110%
can_96	3.3	3.6	110%
rajat11	4.3	3.8	89%
jazz	5.1	4.6	91%
visbrazil	10.2	12.4	121%
grid17	5.5	5.3	97%
mesh3e1	5.9	5.5	94%
netscience	9.4	9.3	99%
dwt_419	9.7	7.5	78%
price_1000	182.7	100.8	55%
dwt_1005	63.9	39.4	62%
cake8	59.5	54.0	91%
bcsstk09	35.2	32.9	94%
block_2000	252.7	151.1	60%
sierpinski3d	332.9	206.6	62%
CA-GrQc	911.3	879.3	96%
EVA	1388.3	838.0	60%
3elt	1781.1	1303.1	73%
us_powergrid	1029.9	722.2	70%

Table 4: Running time in seconds of tsNET and tsNET*.

4.7. Bundled layouts

Sect. 4.2 shows that tsNET can retain neighborhoods successfully, at the expense of introducing a few long edges. We can reduce the clutter created by these with the help of edge bundling, [vdZCT16]. For this, we bundle the long edges, but keep the short ones unchanged (Fig. 3). This is easily done by modifying any existing general graph-bundling method to enforce a maximal edge-displacement δ as a function of the edge length, where we set $\delta = 0.25$. The result is a ‘hybrid’ graph-drawing in which short edges are straight lines (as in classical graph drawings) and long edges are bundled, thereby reducing clutter. To our knowledge, this is the first time that selective bundling has been applied in this way to declutter the drawing of graphs. This method helps one clearly see which parts of the graph layout have been ‘torn off’ by tsNET to achieve a globally optimal node placement. Of course, bundling is also applicable to tsNET* layouts. We chose the tsNET layouts (most notably 3elt) to illustrate this idea as they contained more long edges that could benefit from bundling.

4.8. 3D layouts

As the formulation of tsNET is independent of the output-space dimension, it is interesting to study its ability to produce 3D graph

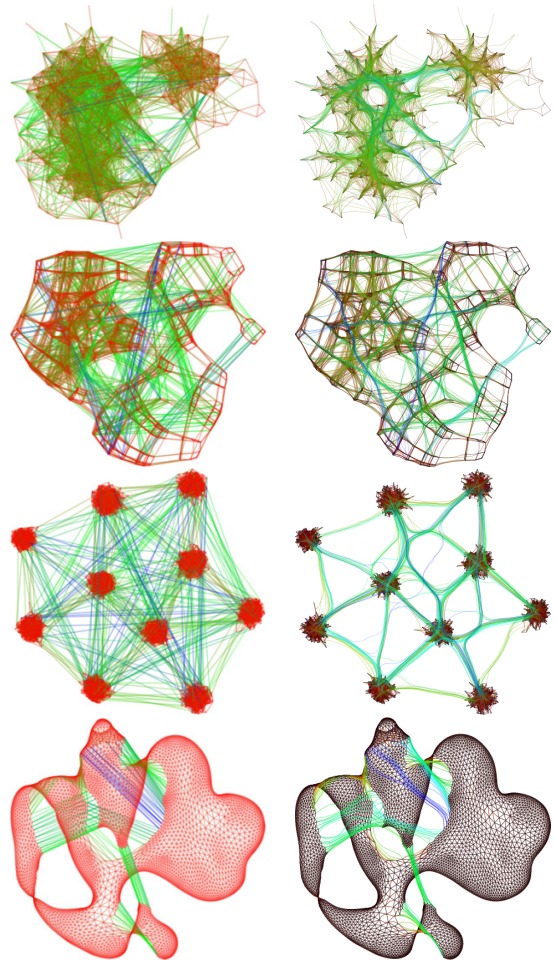


Figure 3: tsNET unbundled (left) and bundled layouts (right) for jazz, cage8, block_2000, and 3elt. Edge colors encode edge lengths ((dark) red = shortest, green = median, blue = longest).

layouts. To do this, we consider the problem of recovering geometric information from topological information present in 3D meshes, similar to work presented in [GK01, Wal01]: Given a mesh (Fig. 4, left), we consider the graph G given by its vertices V and cell-edges E . Next, we use tsNET to create a 3D layout of G (Fig. 4, right).

The tsNET reconstructions preserve the local regular structure of the input meshes (Fig. 4, right). This can be attributed to the fact that the underlying t-SNE technique is very well suited to preserve neighbors.

More importantly, we see that tsNET can reconstruct the high-level structure of the input shapes well, see *e.g.*, relative sizes and positions of the horse’s head and four limbs. However, reconstruction of the *exact* positions of mesh parts is not possible. This is not surprising, as Euclidean distances between nodes in the original mesh are not reflected by their graph-theoretical distances. *E.g.*, the back hooves of the horse are geometrically quite close, but graph-theoretically far away, and will therefore not be placed as close together. Also, certain shape parts, such as the horse’s limbs, can be articulated in the original mesh without significantly modifying

the graph-theoretical distance matrix. These shape parts will be positioned by tsNET in ways that may not match their original geometrical positions. Finally, let us stress that the aim of this illustration is not to claim that tsNET can be used as a standalone method for 3D geometry reconstruction from topology data; rather, we aim to show that tsNET can be used to construct 3D graph layouts as easily as for 2D layouts.

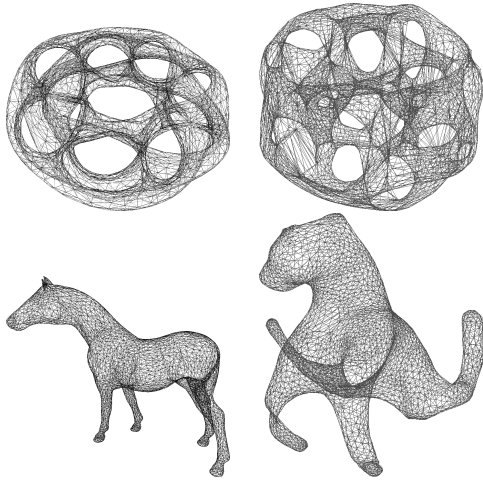


Figure 4: Original 3D meshes (left) and meshes reconstructed with tsNET in 3D (right).

5. Discussion

We discuss next several properties of our method.

Ease of use: Our method depends on a single parameter: the perplexity κ , inherited from t-SNE (Sect. 3.2), which was on average 80, with a standard deviation of 45, for all our tests, except EVA (Tab. 6), where t-SNE needed $\kappa = 600$ to converge. This sensitivity of t-SNE with respect to κ is also documented in other works, e.g. [WVJ16]. Apart from this, no other parameters are required, which is an advantage against other methods that rely on careful fine-tuning of multiple parameters.

Determinism: The tsNET algorithm is not deterministic, given the random point initialization of t-SNE [vdMH08]. However, tsNET*, which initializes 2D node positions using PivotMDS (Sect. 3), is deterministic, since PivotMDS is so. This important property guarantees that, given the same graph, tsNET* produces the same layout.

Robustness: We produce good layouts for different types of graphs, without fine-tuning parameters, whereas many other methods we are aware of handle only graphs of a certain type well.

Scalability: It is important to note that speed is not our main goal, but rather the *quality* of our layouts obtained by the novel use of t-SNE. As mentioned in Sect. 4.6, speed-ups can be obtained with the help of more advanced t-SNE implementations (at the expense of code complexity). In fairness, however, we should mention that such speed-ups do not accelerate the computation of the input distance matrix, which is $O(|V|^2)$.

Generalizability: We currently use the graph’s shortest-path distances to encode the graph information for the layout. More sophisticated distance metrics can be explored, by incorporating e.g., betweenness centrality [BG13] in the distance metric, or using Markov-chain models of random walks [BG14]. Also, approximations of shortest-path distances can be considered for larger graphs where computation of the full shortest-path distance matrix is expensive. Finally, given that our layout method is deterministic, it can be generalized for *dynamic* graphs, using recent developments for projecting time-dependent data with t-SNE [RFT16].

6. Conclusion

We presented the tsNET and tsNET* methods for computing 2D layouts of graphs. Our work shows that modern dimensionality reduction methods are a good alternative to classical graph layout techniques. From a practical standpoint, our methods are simple to implement, and have compact mathematical formulations, leveraging the proven t-SNE method for dimensionality reduction. Extensive visual and quantitative evaluations show that our methods result in excellent quality when compared to state-of-the-art methods.

There are several natural directions for future work. First, approximate versions of t-SNE can be used to significantly accelerate our methods. Second, node and edge weights can be incorporated in the distance metric to provide a more detailed representation of the given graph. Third, 3D layouts as in Fig. 4 can be further investigated. Finally, a direct extension of our methods to computing layouts of time-dependent graphs can be investigated, given recent developments in dynamic t-SNE [RFT16].

7. Acknowledgements

This work was partly supported by the project MOTO (H2020-SESAR-2015-1), grant 699379, offered by the European Commission.

References

- [ATT] Graphviz – graph visualization software. www.graphviz.org. [accessed 07-11-2016]. 2, 4, 5
- [Aub04] AUBER D.: Tulip - a huge graph visualization framework. In *Proc. Graph Drawing*. Springer, 2004, pp. 105–126. 4
- [BBB*] BERGSTRA J., BREULEUX O., BASTIEN F., LAMBLIN P., PASCANU R., DESJARDINS G., TURIAN J., WARDE-FARLEY D., BENGIO Y.: Theano: A CPU and GPU math compiler in python. 4
- [BG13] BAINGANA B., GIANNAKIS G. B.: Centrality-constrained graph embedding. In *Proc. IEEE Intl. Conf. on Acoustics, Speech and Signal Processing* (2013), pp. 3113–3117. 5, 8
- [BG14] BAINGANA B., GIANNAKIS G.: Embedding Graphs under Centrality Constraints for Network Visualization. *arXiv preprint arXiv:1401.4408* (2014). URL: <http://arxiv.org/abs/1401.4408>, arXiv: 1401.4408. 8
- [BP07] BRANDES U., PICH C.: Eigensolver methods for progressive multidimensional scaling of large data. In *Proc. Graph Drawing* (2007), pp. 42–53. 1, 2, 4
- [BP09] BRANDES U., PICH C.: An experimental study on distance-based graph drawing. In *Proc. Graph Drawing* (2009), pp. 218–229. 6

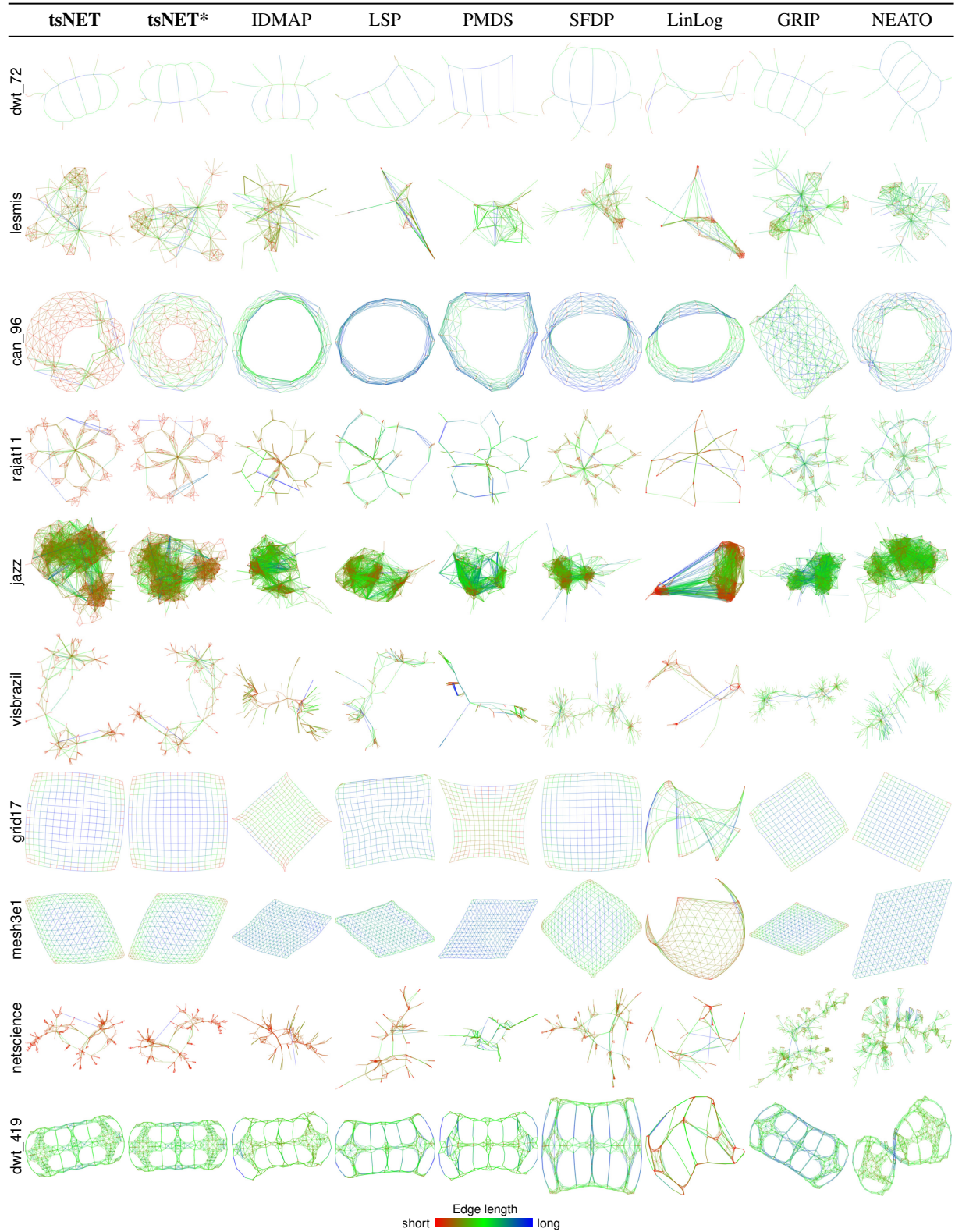


Table 5: Comparison of graph layouts (1/2). Rows and columns correspond to different graphs and layout methods. Edge colors encode their lengths.

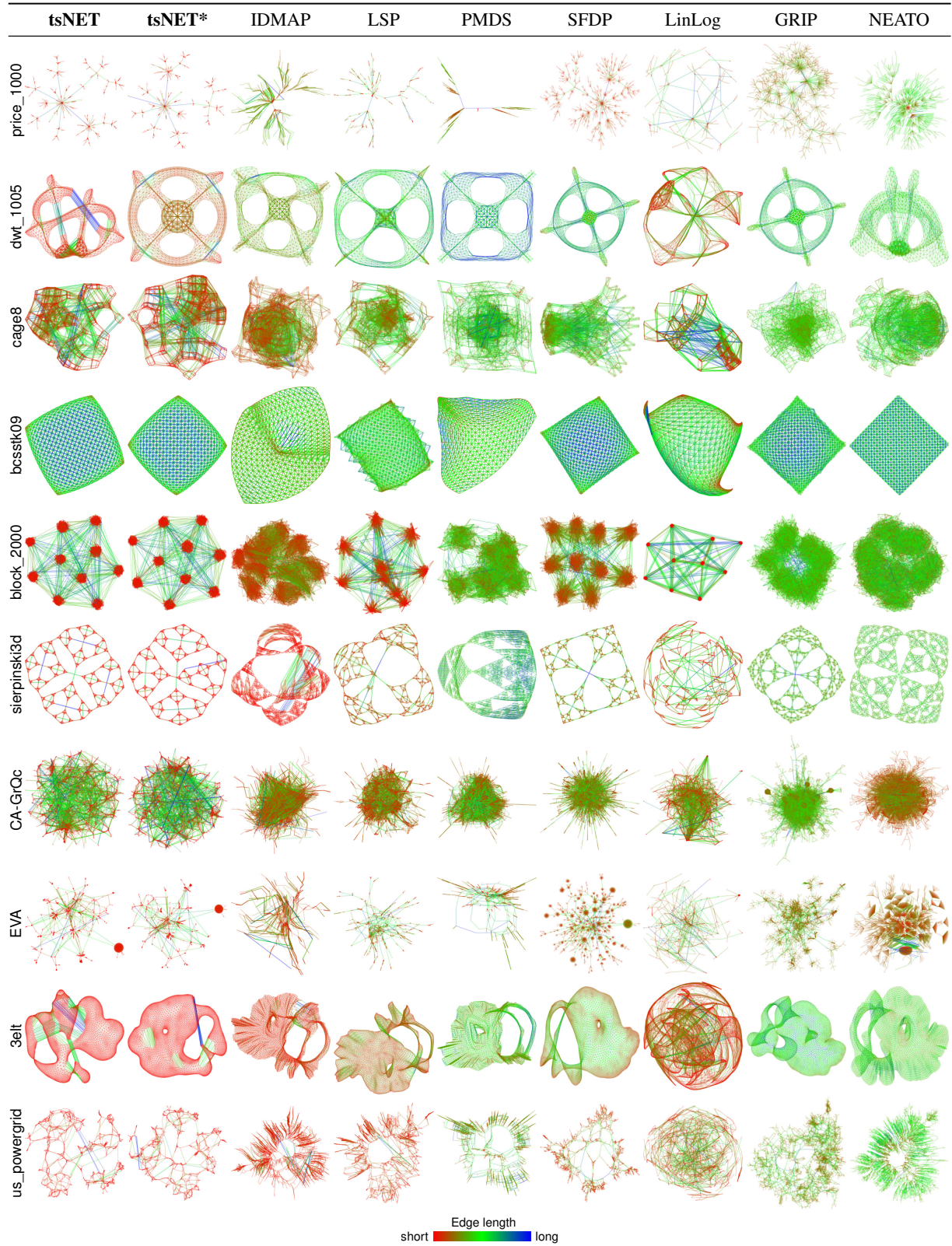


Table 6: Comparison of graph layouts (2/2). Rows and columns correspond to different graphs and layout methods. Edge colors encode their lengths.

- [BSL*08] BUJA A., SWAYNE D., LITTMAN M., DEAN N., HOFMANN H., CHEN L.: Data visualization with multidimensional scaling. *J Comp Graph Stat* 17, 2 (2008), 444–472. 2
- [CG15] CUNNINGHAM J. P., GHAHRAMANI Z.: Linear dimensionality reduction: Survey, insights, and generalizations. *JMLR* 16 (2015), 2859–2900. 2
- [DGL89a] DUFF I. S., GRIMES R. G., LEWIS J. G.: Sparse matrix test problems. *ACM Transactions on Mathematical Software (TOMS)* 15, 1 (1989), 1–14. 4
- [DGL89b] DUFF I. S., GRIMES R. G., LEWIS J. G.: Sparse matrix test problems. *ACM TOMS* 15, 1 (1989), 1–14. 5
- [DH11] DAVIS T. A., HU Y.: The university of Florida sparse matrix collection. *ACM TOMS* 38, 1 (2011), 1:1–1:25. 4
- [DJV*14] DONAHUE J., JIA Y., VINYALS O., HOFFMAN J., ZHANG N., TZENG E., DARRELL T.: DeCAF: A deep convolutional activation feature for generic visual recognition. In *Proc. ICML* (2014), vol. 32, pp. 647–655. 1, 3
- [DP] DIEKMANN T., PREIS R.: AG-Monien/3elt sparse matrix. www.cise.ufl.edu/research/sparse/matrices/AG-Monien/3elt.html. Accessed 12-10-2016. 5
- [dST03] DE SILVA V., TENENBAUM J. B.: Global versus local methods in nonlinear dimensionality reduction. *Advances in Neural Information Processing Systems 15* (2003), 705–712. 2
- [Ead84] EADES P.: A heuristics for graph drawing. *Congressus numerantium* 42 (1984), 146–160. 1
- [FL95] FALOUTSOS C., LIN K.-I.: FastMap: a fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. *Proc. ACM SIGMOD* 24, 2 (1995), 163–174. 2
- [FR91] FRUCHTERMAN T. M. J., REINGOLD E. M.: Graph drawing by force-directed placement. *Software –Practice and Experience* 21, 11 (1991), 1129–1164. 1
- [FT07] FRISHMAN Y., TAL A.: Multi-level graph layout on the gpu. *IEEE Transactions on Visualization and Computer Graphics* 13, 6 (2007), 1310–1319. 2, 5
- [GD03] GLEISER P. M., DANON L.: Community structure in Jazz. *Advances in complex systems* 6, 04 (2003), 565–573. 5
- [GHK13] GANSNER E. R., HU Y., KRISHNAN S.: COAST: A convex optimization approach to stress-based embedding. In *Proc. Graph Drawing* (2013). 5
- [GHN13] GANSNER E. R., HU Y., NORTH S.: A maxent-stress model for graph layout. *IEEE TVCG* 19, 6 (2013), 927–940. 3, 5, 6
- [GJ12] GRONEMANN M., JÜNGER M.: Drawing clustered graphs as topographic maps. In *Proc. Graph Drawing* (2012), pp. 426–438. 5
- [GK01] GAJER P., KOBOUROV S. G.: GRIP: Graph drawing with intelligent placement. In *Proc. Graph Drawing* (2001), pp. 222–228. 2, 4, 5, 7
- [GKN05] GANSNER E. R., KOREN Y., NORTH S.: Graph drawing by stress majorization. *Proc. Graph Drawing* (2005), 239–250. 1, 2
- [Hal70] HALL K. M.: An r-dimensional quadratic placement algorithm. *Management science* 17, 3 (1970), 219–229. 1
- [HBF14] HAMON R., BORGNAT P., FLANDRIN P., ROBARDET C.: Discovering the structure of complex networks by minimizing cyclic bandwidth sum. *arXiv preprint arXiv:1410.6108* (2014). 5
- [HK04] HAREL D., KOREN Y.: Graph drawing by high-dimensional embedding. *JGAA* 8, 2 (2004), 195–214. 1, 2
- [Hu05] HU Y.: Efficient, high-quality force-directed graph drawing. *Mathematica Journal* 10, 1 (2005), 37–71. 2, 4
- [JPC*11] JOIA P., PAULOVICH F., COIMBRA D., CUMINATO J., NONATO L.: Local affine multidimensional projection. *IEEE TVCG* 17, 12 (2011), 2563–2571. 2, 6
- [KCH03] KOREN Y., CARMEL L., HAREL D.: Drawing huge graphs by algebraic multigrid optimization. *Multiscale Modeling & Simulation* 1, 4 (2003), 645–673. 2
- [KKH89] KAMADA T., KAWAI S., HEHNER E.: An algorithm for drawing general undirected graphs. *Inform Process Lett* 31, April (1989), 7–15. 1, 2
- [Knu93] KNUTH D. E.: *The Stanford GraphBase: a platform for combinatorial computing*, vol. 37. 1993. 5
- [Kor04] KOREN Y.: Graph drawing by subspace optimization. *Proc. VisSym* (2004), 65–74. 3
- [Kru16] KRÜGER J. F.: tsNET. <https://github.com/HanKrueger/tsNET/>, 2016. 4, 6
- [KS80] KRUSKAL J. B., SEERY J. B.: Designing network diagrams. In *Proc. 1st General Conf. on Social Graphics* (1980), pp. 22–50. 1, 2
- [LYC16] LU Y., YANG Z., CORANDER J.: Doubly stochastic neighbor embedding on spheres. *CoRR abs/1609.01977* (2016). 3
- [MAH*12] MARTINS R. M., ANDERY G. F., HEBERLE H., PAULOVICH F. V., DE ANDRADE LOPES A., PEDRINI H., MINGHIM R.: Multidimensional projections for visual analysis of social networks. *Journal of Computer Science and Technology* 27, 4 (2012), 791–810. 4, 5
- [MCMT14] MARTINS R., COIMBRA D., MINGHIM R., TELEA A.: Visual analysis of dimensionality reduction quality for parameterized projections. *Computers & Graphics* 41, 1 (2014), 26–42. 2
- [MKS*15] MNIH V., KAVUKCUOGLU K., SILVER D., RUSU A., VENESS J., BELLEMARE M., GRAVES A., RIEDMILLER M., FIDJELAND A., OSTROVSKI G., PETERSEN S., BEATTIE C., SADIK A., ANTONOGLOU I., KING H., KUMARAN D., WIERSTRA D., LEGG S., HASSABIS D.: Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529–533. 1, 3
- [MMT15] MARTINS R., MINGHIM M., TELEA A.: Explaining neighborhood preservation for multidimensional projections. In *Proc. CGVC* (2015), Eurographics. 6
- [MPd06] MINGHIM R., PAULOVICH F. V., DE ANDRADE LOPES A.: Content-based text mapping using multi-dimensional projections for exploration of document collections. *Proc. SPIE 6060* (2006). 2, 4
- [NAS] NASA: Pothen/mesh3e1 sparse matrix. www.cise.ufl.edu/research/sparse/matrices/Pothen/mesh3e1. [accessed 09-03-2016]. 5
- [New01] NEWMAN M. E.: The structure of scientific collaboration networks. *PNAS* 98, 2 (2001), 404–409. 5
- [New06] NEWMAN M. E.: Finding community structure in networks using the eigenvectors of matrices. *Phys Rev E* 74, 3 (2006), 036104. 5
- [NLGC02] NORLEN K., LUCAS G., GEBBIE M., CHUANG J.: EVA: Extraction, visualization and analysis of the telecommunications and media ownership network. In *Proc. ITS* (2002). 5
- [Noa07a] NOACK A.: Energy models for graph clustering. *JGAA* 11, 112 (2007), 453–480. 2, 4, 5
- [Noa07b] NOACK A.: Lin-log layout. code.google.com/archive/p/linloglayout, 2007. [accessed 31-10-2016]. 4
- [Nor04] NORTH S. C.: Drawing graphs with NEATO. *NEATO User manual* (2004). 2, 4
- [OKB16] ORTMANN M., KLIMENTA M., BRANDES U.: A sparse stress model. In *Proc. Graph Drawing* (2016). 5
- [Pei14] PEIXOTO T. P.: The graph-tool python library. *figshare* (2014). URL: figshare.com/articles/graph_tool/1164194. 4, 5
- [PLvdM*15] PEZZOTTI N., LELIEVELDT B., VAN DER MAATEN L. J. P., HÖLLT T., EISEMANN E., VILANOVA A.: Approximated and User Steerable tSNE for Progressive Visual Analytics. *IEEE TVCG* 2626, 99 (2015), 1–15. 2, 6
- [PNML08] PAULOVICH F., NONATO L., MINGHIM R., LEVKOWITZ

- H.: Least square projection: A fast high-precision multidimensional projection technique and its application to document mapping. *IEEE TVCG* 14, 3 (2008), 564–575. 2, 4
- [POM07] PAULOVICH F., OLIVEIRA M., MINGHIM R.: The projection explorer: A flexible tool for projection-based multidimensional visualization. *Proc. IEEE SIBGRAPI* (2007), 27–34. 4
- [PV13] PAPADOPOULOS C., VOGLIS C.: Untangling graphs representing spatial relationships driven by drawing aesthetics. In *Proc. ACM PCI* (2013), pp. 158–165. 5
- [Raj] RAJAT: Rajat/rajat11 sparse matrix. <https://www.cise.ufl.edu/research/sparse/matrices/Rajat/rajat11.html>. [accessed 30-06-2016]. 5
- [RFT16] RAUBER P., FALCÃO A., TELEA A.: Visualizing time-dependent data using dynamic t-SNE. *Proc. EuroVis – Short papers* (2016). 8
- [SVPM14] SORZANO C., VARGAS J., PASCUAL-MONTANO A.: A survey of dimensionality reduction techniques, 2014. arxiv.org/pdf/1403.2877. 2
- [Tam13] TAMASSIA R.: *Handbook of Graph Drawing and Visualization*. CRC Press, 2013. 2, 5
- [TdSL00] TENENBAUM J. B., DE SILVA V., LANGFORD J. C.: Global geometric frameworks for nonlinear dimensionality reduction. *Science* 290, 5500 (2000), 2319–2323. 2
- [Tor52] TORGERSON W. S.: Multidimensional scaling: I. theory and method. *Psychometrika* 17, 4 (1952), 401–419. 2
- [vdM] VAN DER MAATEN L.: t-SNE: CUDA implementation. [lvdmaaten.github.io/tsne/code/TSNE_CUDA.zip](https://github.com/lvdmaaten/tsne/code/TSNE_CUDA.zip). [accessed 15-11-2016]. 6
- [vdM14] VAN DER MAATEN L.: Accelerating t-SNE using Tree-Based Algorithms. *JMLR* 15 (2014), 3221–3245. 6
- [vdMH08] VAN DER MAATEN L. J. P., HINTON G. E.: Visualizing high-dimensional data using t-SNE. *JMLR* 9 (2008), 2579–2605. 1, 2, 3, 8
- [vdZCT16] VAN DER ZWAN M., CODREANU V., TELEA A. C.: CUBu: Universal real-time bundling for large graphs. *IEEE TVCG* 22, 12 (2016), 2550–2563. 7
- [VHBB02] VAN HEUKELUM A., BARKEMA G., BISSELING R.: DNA electrophoresis studied with the cage model. *J Comput Phys* 180, 1 (2002), 313–326. 5
- [vLKS*11] VON LANDESBERGER T., KUIJPER A., SCHRECK T., KOHLHAMMER J., VAN WIJK J., FEKETE J.-D., FELLNER D.: Visual analysis of large graphs: State-of-the-art and future research challenges. *CGF* 30, 6 (2011), 1719–1749. 2
- [Wal01] WALSHAW C.: A multilevel algorithm for force-directed graph drawing. *Graph Drawing* 7, 3 (2001), 31–55. 2, 5, 7
- [WS98] WATTS D. J., STROGATZ S. H.: Collective dynamics of small-world networks. *Nature* 393, 6684 (1998), 440–442. 5
- [WVJ16] WATTENBERG M., VIÉGAS F., JOHNSON I.: How to use t-SNE effectively. *Distill* (2016). <http://distill.pub/2016/misread-tsne>. 8
- [YPK14] YANG Z., PELTONEN J., KASKI S.: Optimization equivalence of divergences improves neighbor embedding. In *Proc. ICML* (2014), vol. 2, pp. 1808–1839. 3, 4
- [Yus01] YUSUFOV R.: GRIP: Graph drawing with intelligent placement. <https://www.cs.arizona.edu/~kobourov/GRIP/>, 2001. [accessed 18-11-2016]. 4