EFFICIENTLY VISUALIZING LARGE GRAPHS

DURF REPORT

Xinyu Li [*]
School of Art and Science
New York University Shanghai
Shanghai, China
xl3665@nyu.edu

Yao Xiao [*]
School of Art and Science
New York University Shanghai
Shanghai, China
yx2436@nyu.edu

Yuchen Zhou [*]
School of Art and Science
New York University Shanghai
Shanghai, China
yz7258@nyu.edu

ABSTRACT

Most existing methods for graph visualization based on dimension reduction face performance issues that limit them to relatively small graphs. In this work, we propose a novel dimension reduction method for graph visualization, called t-Distributed Stochastic Graph Neighbor Embedding (t-SGNE). t-SGNE is specifically designed to visualize cluster structures of the graph. As a variant of the standard t-SNE method, t-SGNE avoids the time-consuming computations of pairwise similarity. Instead, it uses the neighbor structures of the graph to reduce the time complexity from quadratic to linear, and can thus be used in larger graphs. Codes and data will be available at Yao Xiao's GitHub repository[2].

Keywords Large Graphs · Graph Layout · Graph Embedding · Dimension Reduction · Clusters · Neighbor Structure

1 Introduction

Graph data are widely used nowadays, and visualization of graph data is an important problem in various fields. For example, interactions of users on social media websites can be represented by graphs with users as nodes and their following relations as edges. Analysis on such social network graphs can give important information such as interpersonal ties, structural holes, and local online communities [5]. Various different types of methods for graph visualization have been proposed over the past few decades, many of which are reviewed and compared by Herman et al. in a survey on graph visualization and navigation [9]. Important techniques include force directed methods [11] and spectral drawing methods [12], etc. Most of these techniques can be easily distracted by noises when plotting cluster structures of graphs. They also run too slow or take too much space regarding large graph datasets consisting of over 10K nodes. These severely limits their applicability on large graph data and real-world data where thousands of noises may exist.

To handle large-scale data, we found that a faster approach of recent years can be applied. We can first compute a high dimensional embedding where each node in the original graph is marked by a vector. After that we use dimension reduction methods to convert this embedding into a 2D layout. There are various existing graph embedding methods, which can be found in the survey by Goyal and Ferrara [7]. Existing popular dimension reduction methods include PCA [21], t-SNE [15] and UMAP [16]. However, these methods are designed for data visualization and can not be directly applied to graph visualization. Moreover, although they can be combined with graph embedding to visualize graph, their runtime can still be improved. Also, their visualization quality is still a problem, especially regarding cluster structures.

In this paper, we focus on the simplest case of graph visualization: undirected, unweighted graphs without node attributes, and leave the generalization for future work. Motivated by the objective of preserving the cluster structures of graphs, we propose ShortestPath and ShortestPath Laplacian Eigenmaps Embedding (SPLEE) that take into account graph theoretical distances and layout clusters more clearly. Also, ShortestPath is a fast algorithm that can deal with

[*]The authors would like to thank Professor Jie Xue (email address: jx2389@nyu.edu) for helpful discussion and remarks, and are indebted to him for meticulous reading and constructive suggestions.

[2]https://github.com/Charlie-XIAO/embedding-visualization-test

larger datasets. We also put forward a fast and cluster-preserving dimension reduction technique called t-SGNE based on the original t-SNE, which takes advantage of the graph neighbor structures. Finally, in order to compare the results of different methods on different datasets, we propose two quantitative measures for testing, including Normalized Mutual Information (NMI) and Aesthetic Quality (AQ). These measures respectively evaluate the clustering accuracy and how well clusters are distributed in layout.

Organization. In the remaining part of this section, we outline the previous works related to our work. In Section 2, we discuss details of our methods, including graph embedding methods, dimension reduction methods and quantitative testing standards. In Section 3, we present the testing results of both previous methods and our methods on a variety of datasets, regarding visualization quality and running time. We also provide the repository of our codes on GitHub [2] In section 4, we discuss possible future improvements on our work.

1.1 Related Work

1.1.1 Graph Layout

A layout of a graph is a two dimensional embedding of the graph, where each node is assigned a coordinate on the 2D plane for visualization purpose. There are mainly three types of methods to compute the layout of a graph: force-directed methods, spectral methods, and dimension reduction methods.

- **Force directed methods** model nodes as particles that repel each other, and edges as springs that connect the particles. Thus, it can compute the graph layout by simulating the corresponding particle-spring system and minimizing the energy of the system [11]. It can produce aesthetically pleasing results with less edge crossing and uniform distribution of nodes. However, the optimization of this complex system is time-consuming and computation-intensive, which means we cannot directly apply force directed method to graphs of large scale. Moreover, the aesthetic criteria used in force directed method fail to reflect how the cluster structure of the graph is represented in the layout. In fact, empirically, for a graph with more than 10K nodes, force-directed methods usually produce a "hairy ball" with no identifiable clusters.
- **Spectral methods** use eigenvectors of some matrices related to the graph as the graph layout. Compared with force directed methods, spectral methods are much faster, most of which run quadratic time. Also, spectral methods are deterministic, presenting an exact mathematical formula that draws the layout. The most famous spectral drawing method is to compute the lowest eigenvectors of the Laplacian matrix of the graph [12]. The correctness of this method can be verified with an optimization problem. Spectral drawing methods tend to place each node at the centroid of its neighbors with some deviation. This preserves the graph-theoretical distance between nodes pretty well. However, this may also result in lots of crossings between edges and ambiguity of the borders of clusters. Different clusters may mix up a lot for large graphs.
- **Dimension reduction methods** have a wide range of usage. They can be applied to graph drawing with certain modifications. Dimension reduction methods aim to project the given high dimensional data to lower dimension while preserving some form of information of the original high dimensional data (typically represented by an objective function). To apply dimension reduction method to graph drawing, one can first compute a high dimensional embedding of the graph, then reduce the dimension of the embedding to two while minimizing some form of difference between the high and low dimensional embedding. The output can be treated as a graph layout. For each node, a typical choice of high dimensional embedding would be its graph-theoretic distance to all the nodes in the graph. Pivot MDS, proposed by Brandes and Pich, is a classic example [4]. It first samples some pivot nodes and uses the distances of each node to these pivots as high dimensional embeddings. Then, it applies Multi-Dimensional Scaling (MDS) to reduce the dimension to two, where the objective function is a so-called stress function that measures the discrepancy between the pairwise graph-theoretic distance in the graph and the pairwise Euclidean distance on 2D plane. Methods based on dimension reduction are suitable for the visualization of large-scale graph because there are various approximations of the objective functions of dimension reduction methods. However, layouts produced by dimension reduction methods tend to be less aesthetic compared to those produced by force-directed methods.

1.1.2 t-SNE

t-SNE is a nonlinear, statistical model for dimension reduction. It aims to map some known high dimensional data to low dimension in a way that preserves the neighbor structure in a probabilistic way, whereas traditional dimension reduction methods like MDS preserves the more intuitive distance structure. Loosely speaking, t-SNE assumes that similarity

[2].

between points are captured by their distance in high dimensional space. Thus, a point is similar to its neighbors and dissimilar to points far away. The low dimensional embedding is constructed so that, in the low dimensional space, points that are similar in high dimensional space are closer and dissimilar points are farther apart with high probability.

The primary use of t-SNE is to visualize data in high dimensional space, thus a common choice of the low dimensional space is $\mathbb{R}^2$. Compared to previous data visualization methods, t-SNE can preserve both local structure and global structure such as clusters at different scales [15].

To apply to large dataset, t-SNE uses an approximation based on random walk on neighborhood graph, where the neighborhood graph is constructed from the high dimensional embedding. Although this enable the application of t-SNE to larger dataset, the computation of neighborhood graph creates a computational bottleneck and restrict t-SNE to dataset of size about 100K [1].

1.1.3 Graph Embedding

Graph embedding is a powerful method for reducing the dimensions of graph data while preserving certain graph structures. More specifically, a graph embedding is a mapping that maps each node to a representation vector, whose dimension is much smaller than the number of nodes. In an effective graph embedding method, the representation vectors of nodes within the same community should be similar, so that closely-related nodes tend to lie close to each other after dimension reduction as well. There are various types of graph embedding methods.

DeepWalk [17] is a random walk based embedding method. It applies truncated random walks to walk through the nodes and obtain sampling. A truncated random walk starts at a certain node and randomly visits one of its neighbors and so on, until the length of the walk reaches a default value. This describes the cooccurrence relations of nodes, which is the key of this method. DeepWalk then uses word2vec [6] to create the representation vectors of each node. Word2vec is a common method for word embedding in NLP, which learns the cooccurrence relations among words from sentences and vectorize each word. DeepWalk is hence able to vectorize each node by learning their cooccurrence relations.

Node2Vec [8] is another random walk based embedding method. It is similar to DeepWalk but samples random walks differently. It introduces two parameters, which controls the probability of performing a breadth-first search or a depth-first search when randomly choosing the next node to visit. Breadth-first search better records the similarity of closely-related nodes. Depth-first search may preserve some global structures of the original graph.

Laplacian Eigenmaps [2] is a different type of graph embedding method. It constructs relations between nodes from a local perspective. Since the objective is to keep the closely-related nodes close to each other in the lower-dimensional space, Laplacian Eigenmaps tries to minimize $\sum_{i,j} \|y_i - y_j\|$, where y_i and y_j are data points in the lower-dimensional space. The problem becomes minimizing the trace of $Y^{\mathrm{T}} LY$ after some transformation, where $L = D - A$ is called the unnormalized Laplacian matrix, A is the adjacency matrix, and D is a diagonal matrix with entries $D_{ii} = \sum_j A_{ij}$. By trace derivative law, $LY = -DY\Lambda$ gives the optimized result. This can be rewritten as the generalized eigenvalue problem $Ly = \lambda Dy$. Hence, Laplacian Eigenmaps first computes the eigenvalues and eigenvectors of the Laplacian matrix of the graph, then takes the d eigenvectors corresponding to the d smallest nonzero eigenvalues as the m-dimensional output ($d \ll |V|$ is the dimension of the high dimensional embedding).

Geometric Laplacian Eigenmaps [20] is similar to Laplacian Eigenmaps while taking the eigenvectors corresponding to the m largest nonzero eigenvalues. The intuition is that these correspond to the best approximation to the Laplacian through singular value decomposition.

2 Method

The problem of graph drawing can be formulated as follows. Let $G = (V, E)$ be an undirected, unweighted graph where $|V| = n$ (for more general types of graphs, see Section 4). Graph Drawing defines a function $GD : G \mapsto Y$, where $Y = \{y_i \in \mathbb{R}^2 \mid i = 1, \cdots, n\} \subset \mathbb{R}^2$ is the two dimensional embedding of the graph G.

There are two steps to apply the method of dimension reduction to graph drawing. The first step is Graph Embedding (GE). Given a graph G, we want to find a function $GE : G \mapsto X$ where $X = \{x_i \in \mathbb{R}^d \mid i = 1, \cdots, n\} \subset \mathbb{R}^d$ is the high dimensional embedding of the graph G and $d \gg 2$. Next, we will perform Dimension Reduction (DR) on X, that is, to apply a function $DR : X \mapsto Y$ on X. Thus, we have the composition

$$GD = DR \circ GE.$$

2.1 t-SNE

t-SNE is a DR method. We will first state the original t-SNE algorithm, then describe its approximation based on random walk on neighborhood graph, as is done in the original paper. The original t-SNE consists of three parts: constructing a probability distribution $\mathcal{P}$ in $\mathbb{R}^d$ and approximate $\mathcal{P}$ with probability distribution $\mathcal{Q}$ in $\mathbb{R}^2$.

First, for $x_i, x_j \in X$, we can compute the probability $p_{j|i}$ that x_i would pick x_j as its neighbors as follows:

$$p_{j|i} = \frac{\exp(-\|x_i - x_j\|^2/2\sigma_i^2)}{\sum_{k\neq i} exp(-\|x_i - x_k\|^2/2\sigma_i^2)} \cdot \mathbb{1}_{\{i\neq j\}},$$

where χ_A is the indicator function of set A. Then, we can define the distribution $\mathcal{P}$ as

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n},$$

where σ_i is chosen using a binary search so that the perplexity $\text{Perp}(\mathcal{P}_i) = 2^{H(\mathcal{P}_i)}$ equals to a user-defined value ($\text{Perp}(\mathcal{P}_i)$ is typically between 5 and 50, scikit-learn choose30 as default), and $H(\mathcal{P}_i) = -\sum_j p_{j|i} \log p_{j|i}$ is the Shannon entropy as is suggested in SNE [10].

Next, we construct $\mathcal{Q}$ in a similar way, except we use the Student t-distribution instead of Gaussian distribution:

$$q_{ij} = \frac{(1 + \|y_i - y_j\|^2)^{-1}}{\sum_k \sum_{l\neq k}(1 + \|y_k - y_l\|^2)^{-1}} \cdot \mathbb{1}_{\{i\neq j\}}.$$

The Student t-distribution uses a degree of freedom equal to 1, which results in a Cauchy distribution. This distribution is heavy-tailed (infinite first moment), which helps alleviate the crowding problem of Stochastic Neighbors Embedding (SNE), that data points are all mapped to the center becoming indistinguishable [15].

To find $\mathcal{Q}$ that best approximates $\mathcal{P}$, we use the KL divergence of $\mathcal{P}$ from $\mathcal{Q}$ as an objective:

$$\min_Y \text{KL}(\mathcal{P}\|\mathcal{Q}) = \sum_{i,j} p_{ij} \log\left(\frac{p_{ij}}{q_{ij}}\right).$$

To apply t-SNE to larger datasets, we use an approximation of P based on neighborhood graph and random walk. First, we construct a neighborhood graph $G_{knn} = G_{knn,X}$ from X, where x_i is linked to its first k nearest neighbors, denoted as $N(i)$. We perform a fixed large number of random walks of fixed length on the graph for each node, where the probability of transiting from x_i to x_j is proportional to exp $\|x_i - x_j\|^2$. From the random walks, we can construct an approximation of $p_{j|i}$ as follows:

$$p_{j|i} = \frac{\text{number of random walks from } i \text{ to } j}{\text{number of random walks starting from } i}.$$

The rest is the same as what we stated above.

Time complexity. t-SNE mainly involves three steps. First, the construction of the $G_{knn,X}$ involves the computation of pairwise Euclidean distances, which has a time complexity of $O(d|V|^2)$. The reason is that for each node, we need to further compute and rank its distance to the rest of the nodes, and select the first k nearest nodes to construct the neighborhood graph. Second, the simulation of random walks runs $O(|V|)$ time. Finally, the optimization of $\text{KL}(\mathcal{P}\|\mathcal{Q})$ has a time complexity of $O(|V|^2)$ as is suggested in the original article [15].

2.2 t-SGNE

t-SGNE is a simple yet effective modification of t-SNE that can be applied to graph data with high dimensional embeddings. Given a graph G and its high dimensional embedding X, we perform t-SNE with nearest neighbor approximation, except that the neighborhood graph is constructed from G instead of X, i.e.

$$G_{knn} = G_{knn,G},$$

where $v_i, v_j \in G$ are connnected in $G_{knn,G}$ if v_i is of the first k nearest neighbors of v_j on graph G, where the nearest neighbor is computed by a breadth-first search of k steps.

To justify this simple modification, we need to answer a nontrivial question: is $G_{knn,G}$ a valid substitute for $G_{knn,X}$? The answer depends on our choice of GE method. The goal of t-SNE is to map points closer (farther) in $\mathbb{R}^d$ to closer (farther) positions in $\mathbb{R}^2$. When applied to large datasets, $G_{knn,G}$ with random walk approximates the distance relation of points in $\mathbb{R}^d$. If x_i and x_j are closer in $\mathbb{R}^d$, a random walk starting from x_i is more likely to reach x_j, which results in larger $p_{j|i}$. However, points closer in G are not necessarily closer in $\mathbb{R}^d$, and vice versa. For example, Structural Deep Network Embedding (SDNE) is a GE method that captures structural similarity instead of distance/neighbor relations between nodes in a graph [10]. Nodes are assigned to closer positions in $\mathbb{R}^d$ not because they are closer in the graph but because they have similar structural properties (e.g. both are central to a cluster, both are hubs between two clusters). In this case, simply comparing the ratio of random walk paths from v_i to v_j will not reflect the structural similarity and dissimilarity between these nodes.

To judge if t-SGNE is suitable, we need to ask another question: when we map a graph G to $X \subset \mathbb{R}^d$, what kind of points should be closer in the high dimensional space $\mathbb{R}^d$? Typically, t-SGNE can only be combined with GE methods that maps nodes of small geodesic distance (shortest path length) to closer embeddings. For GE methods that aim to preserve more complex structures like structural similarity, $G_{knn,G}$ is not a good approximation of $G_{knn,X}$. In other words, with an appropriate GE method, t-SGNE is suitable for visualizing the cluster structures of graph, which is captured by the geodesic distance relation in G.

Time complexity. The construction of $G_{knn,G}$ has a time complexity of $O(k|V|)$ since for each node, we need to perform a k-step BFS to determine its k neighbors. Compared with t-SNE which involves a quadratic-time computation of $G_{knn,X}$, t-SGNE only requires linear time to construct the neighborhood graph. The rest is the same as t-SNE.

2.3 ShortestPath

ShortestPath is a GE method that gives graph embedding of each node based on its shortest path length to certain target nodes. The motivation that we introduce such method is that we want the embedding result to have association with the distance between nodes in the original data, which is mostly represented by the shortest path lengths between nodes.

First, we pick the target nodes. Let d be the dimension of the high dimensional embedding. Through experiments, we found that randomly choosing d targets shows the overall best result.

Then, to calculate the length of shortest paths between each node in the graph and each target in X, we apply BFS starting from each node in X. To minimize the negative effect of nodes that are far apart in the final plot, we also apply a threshold l_0 here. If the length of the shortest path is less than l_0, we use it in the embedding. Otherwise, we regard the length as $l_0 + 1$. The default value of l_0 is $\sqrt{|E|}$ where $|E|$ is the number of edges in the graph. Such a threshold also reduces the number of computations thus increasing efficiency.

Hence, the ShortestPath embedding is defined as follows: we first obtain an embedding matrix

$$E_{ij} = \begin{cases} d_{ij} & \text{if } d \leq l_0 \\ l_0 + 1 & \text{if } d > l_0 \text{ or path does not exists between } v_i \text{ and } x_j \end{cases},$$

where d_{ij} represents the shortest path length between the nodes $v_i \in V$ and $x_j \in X$. Each row vector is then taken as the embedding vector of node v_i.

Time complexity. Since BFS is applied d times in total, and in each round it traverses each edge at most once, the total time complexity of ShortestPath is $O(d|E|)$.

2.4 SPLEE

ShortestPath Laplacian Eigenmaps Embedding (SPLEE) is a GE method that combines ShortestPath and Laplacian Eigenmaps. The motivation is to take into account graph-theoretically distances rather than just considering node connections when doing spectral embedding.

The original version of Laplacian Eigenmaps [2] can be applied on more general types of high dimensional data. However, they need to be transformed to weighted graphs before applying the method. Links exist if corresponding data points are close to each other, and weights are determined by some functions on the Euclidean distances between data points which will be mentioned later soon. However, when applying Laplacian Eigenmaps on graph data, no types of distances are taken into consideration. Hence, SPLEE uses shortest path lengths between nodes to make up for this.

The algorithm involves two main steps. The first step is to obtain a special distance matrix W. As in the original Laplacian Eigenmaps, a heat kernel is applied to the Euclidean distances to approximate the Gaussian [2]. Similarly, SPLEE applies the heat kernel to the shortest path lengths between nodes. Furthermore, $W_{ij} = 0$ if the shortest path length between the nodes v_i and v_j are beyond some threshold l_0 since nodes that are far apart should not be linked as in the original version of Laplacian Eigenmaps. Hence, the distance matrix W is defined as:

$$W_{ij} = \begin{cases} \exp(-\epsilon d_{ij}^2) & \text{if } d \leq l_0 \\ 0 & \text{if } d > l_0 \text{ or path does not exists between } v_i \text{ and } v_j \end{cases},$$

where d_{ij} represents the shortest path length between v_i and v_j. According to experimental results, ϵ is recommended to be around 5.0 to 7.0. The threshold l_0 can be chosen as the same default value $\sqrt{|E|}$ as in ShortestPath. The shortest path lengths are computed by BFS for unweighted graphs (or by Dijkstra's Algorithm for weighted graphs).

The second step is to compute eigenvectors to use as the high dimensional node embeddings. We generalize the original definition of Laplacian matrix L as:

$$L = D - W,$$

where W is the distance matrix above, and D is a diagonal matrix with entries $D_{ii} = \sum_j W_{ij}$. Then the eigenvectors corresponding to the smallest d eigenvalues of the generalized Laplacian matrix L are taken as the d-dimensional embedding ($d \ll |V|$).

Time complexity. SPLEE uses the same technique of computing shortest path lengths as ShortestPath, which takes $O(d|E|)$ time. To compute the lowest d eigenvectors of the Laplacian matrix, the state-of-art algorithm takes $O(k|V|^2)$ time, where k is the number of iterations. Hence, SPLEE runs a total time complexity of $O(d|E| + k|V|^2)$.

2.5 Normalized Mutual Information

This is a measure for clustering accuracy of the 2D layout. Normalized Mutual Information (NMI) is a measure that compares two network partitions. It ranges from 0 to 1, and the smaller the NMI, the more similar the two partitions. Here we do clustering for both the original graph and the 2D graph layout, and compare the clustering results. There are two main steps to achieve this.

First, we use Louvain's algorithm [3] to cluster the original graph. In this step, we output the clustering label for each node as well as the number of clusters created. One important reason to choose this algorithm is that it works directly on graph data structures and does not require the number of clusters as a parameter. In this way, we can obtain the optimal clustering result for the original graph to use as a standard community partition.

Second, we apply kNN clustering algorithm on the 2D graph layout, with the k chosen as the output number of clusters in the previous step. Hence, we can keep in accordance the number of clusters between the graph and the layout. In this way, the two sets of clustering labels have the same number of distinct values, thus the result will be more intuitive and accurate. Now we compute the NMI as follows:

$$\text{NMI}(PR; TR) = \frac{2 \cdot I(PR; TR)}{H(PR) + H(TR)},$$

where PR and TR denote the set of labels of the original graph and the 2D layout respectively, $H(\cdot)$ denotes the entropy, and $I(\cdot\,;\cdot)$ denotes the mutual information between the two arguments.

2.6 Aesthetic Quality

This is a measure for the aesthetic quality focusing on cluster structures of the 2D layout. It examines whether clusters are clearly divided. It ranges from 0 to 1, and the bigger the value, the better clusters are distributed in the layout. The basic idea is to divide the plot into $k \times k$ grids. For each grid, we calculate the portion that nodes of each cluster take up and we compare it with a user-defined threshold p. If the result is larger than p, we regard the grid a good grid with a clear dominating cluster, and otherwise we ignore it. Finally, we give an overall result by calculating the ratio of good grids.

Specifically, the x-length of each grid is given by

$$x_0 = (X_{\text{max}} - X_{\text{min}})/k,$$

where X_{max} is the largest value along the x-axis of the point position in the layout and X_{min} is the smallest. Similarly, we define the y-length of each grid as

$$y_0 = (Y_{\text{max}} - Y_{\text{min}})/k.$$

Thus, we can easily know the bounds for each grid. Now, by going over the position of every node in the layout, we can specify which grid it belongs to. Then by calculating the portion each kind of nodes takes up in the grid, we can examine whether it is a good grid. The result of this measure AQ is thus given by

$$AQ = (\text{number of good grids})/k^2.$$

Dataset	Method (GE / DR)		Time (GE / DR)		NMI	AQ
EMAILUNIV	DeepWalk	t-SNE	10.102	5.715	0.4951	0.50
	ShortestPath	t-SNE	0.394	5.848	0.2112	0.30
	SPLEE	t-SNE	3.231	5.916	**0.6106**	**0.57**
	DeepWalk	t-SGNE	9.530	6.219	0.4917	**0.57**
	ShortestPath	t-SGNE	0.298	6.442	0.4917	**0.57**
	SPLEE	t-SGNE	3.369	6.226	0.4917	**0.57**
WIKI	DeepWalk	t-SNE	24.279	17.410	0.6146	**0.59**
	ShortestPath	t-SNE	0.883	15.690	0.3627	0.24
	SPLEE	t-SNE	20.382	16.925	**0.6147**	**0.59**
	DeepWalk	t-SGNE	21.433	14.974	0.5988	0.56
	ShortestPath	t-SGNE	0.817	15.273	0.5883	0.53
	SPLEE	t-SGNE	17.759	14.508	0.5944	0.50
LASTFM	DeepWalk	t-SNE	65.456	41.130	0.6822	**0.67**
	ShortestPath	t-SNE	2.881	37.680	0.5210	0.56
	SPLEE	t-SNE	343.962	40.694	**0.6894**	**0.67**
	DeepWalk	t-SGNE	71.484	44.128	0.6746	0.62
	ShortestPath	t-SGNE	2.920	44.354	0.6746	0.62
	SPLEE	t-SGNE	343.272	44.723	0.6746	0.62

Table 1: Comparisons on running time (in seconds), NMI, and AQ scores of different combinations of GE and DR methods. The running time is experimented on a single machine with 2.3 GHz Intel Core i7 CPU and 16 GB of memory. While the running time gives an indication on the applicability of the methods on large graphs, the main focus of this experiment is on NMI and AQ scores which represent the visualization quality.

3 Experiments

In this section we present experimental analyses on our methods. We first compare different combinations of GE and DR methods to compare their visualization quality, then we take a specific combination (ShortestPath and t-SGNE) for testing on larger graph datasets. Testing codes, datasets, and part of the experimental results will be available at Yao Xiao's GitHub repository[3].

3.1 Visualization quality of different combinations

We evaluate our new GE methods ShortestPath and SPLEE and new DR method t-SGNE. We also compare with the existing GE method DeepWalk and DR method t-SNE as a baseline. This experiment focuses on comparing the visualization quality of different combinations of methods, hence only small graph datasets (< 10K nodes) are used. Larger datasets will be tested in the next experiment. We experiment on three different datasets from Network Data Repository [19].

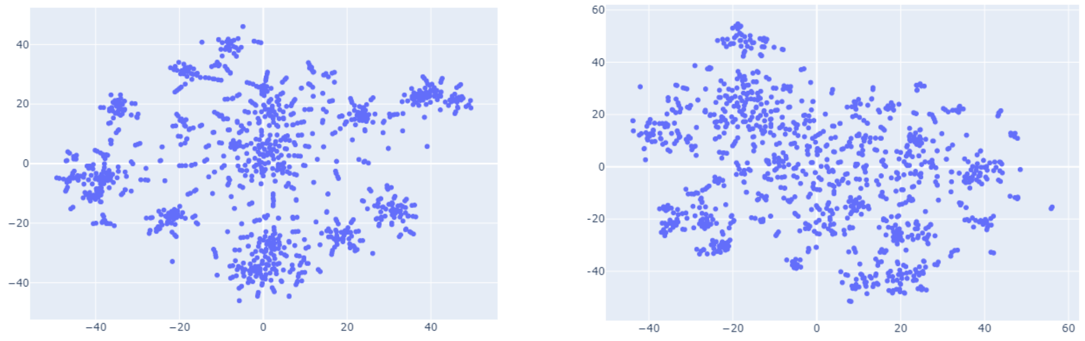
- EMAILUNIV is a network of a small collection of emails sent universally. It has 1133 nodes and 5451 edges, with an average degree of 9. It has clear cluster structures representing frequent sending and receiving of emails within certain communities.
- WIKI is a Wikipedia-based network constructed from different Wikipedia categories. It is a originally a labeled graph, but we will not take its labels into account. It has 2405 nodes and 12761 edges, with no clear borders of cluster structures due to the high interpenetration between the topics collected in this dataset.
- LASTFM is a heterogeneous network of user relations of the music website Last.fm. Each node represents an Asian user of Last.fm, and the edges represent the following relations between them. The network consists of 7624 nodes and 27806 edges, with a small density of 0.001. Cluster structures are clear in this network, representing the online communities of Asian Last.fm users.

Running time. As for running time of the GE methods, ShortestPath takes the lead, with a linear runtime with respect to the number of edges. DeepWalk takes longer than ShortestPath since it needs to train the random walk model, but it is less affected by the increase of graph size. Due to the quadratic runtime of SPLEE, it runs much slower than the other two GE methods when the size of graph gets bigger, so it may not be a good choice for large graphs consisting of

[3]https://github.com/Charlie-XIAO/embedding-visualization-test

10K nodes and more. As for the running time of the DR methods, t-SNE and t-SGNE do not differ much, since the size of the high dimensional embedding is set to be 128 here. However, if the size of the high dimensional embedding get larger, t-SGNE may outperform t-SNE since it takes advantage of graph neighbors rather than computing pairwise similarity of nodes.

Clustering accuracy. The NMI scores are used to test clustering accuracy of graph visualization. The combination of SPLEE with t-SNE shows the best quality performance in all three datasets. The baseline combination of DeepWalk and t-SNE also gives good results, but not as good as the previous combination. The ascendance of SPLEE combined with t-SNE is most clearly shown in the EMAILUNIV dataset, as can be seen in Figure 1. GE methods combined with t-SGNE fall below the baseline, but they are almost as good as the baseline combination, which is still pleasing with regard to quality. It is worth noticing that when the cluster structures in the original graph are clear (e.g. EMAILUNIV and LASTFM), different GE methods give completely the same layout combined with t-SGNE. This may be because t-SGNE takes both the high dimensional embedding and the original graph as inputs, which reduces the impact of the embedding to the layout. Another remarkable observation is that t-SGNE largely improves the NMI performance of ShortestPath. Combined with t-SNE, ShortestPath falls far behind the other two GE methods, but combined with t-SGNE, its performance is almost as good as the other two.



(a) SPLEE + t-SNE (b) DeepWalk + t-SNE

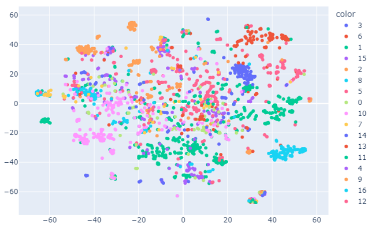
Figure 1: Graph layouts of EMAILUNIV dataset using SPLEE combined with t-SNE, compared with the baseline combination of DeepWalk and t-SNE.

Aesthetics quality. From the results, we can see that the AQ scores almost coincide with NMI scores, with the combination of SPLEE and t-SNE still taking the lead. Different GE methods combined with t-SGNE perform as well as this combination in some datasets (e.g. EMAILUNIV) and less satisfying in others, but overall their performances are pleasing for application. We also directly compare the layouts of these different combinations visually. Here we pick the WIKI dataset as is shown in Figure 2, since its cluster structures are ambiguous, so even slight differences in aesthetic performances can be clearly observed. SPLEE combined with t-SNE (Figure 2c) gives the best layout of WIKI, separating the cluster structures the most clearly among all combinations. As a baseline, DeepWalk combined with t-SNE (Figure 2a) separates some of the clusters on the periphery, but the borders are not as clear. When using t-SGNE as the DR method (Figure 2d, 2e, 2f), clusters are separated, but the clusters seem bloated and mixed together with no clear borders. Though below baseline, the results of t-SGNE are still acceptable since cluster structures can indeed be distinguished visually, and the such a problem will be covered up in graphs with clearer cluster structures (e.g. EMAILUNIV), as can be seen in Table 1 from the AQ scores.

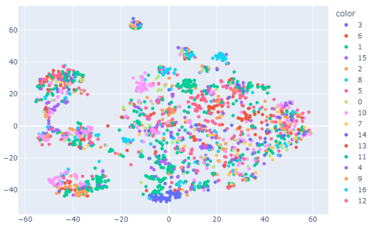
Hence, we can conclude that SPLEE combined with t-SNE produces layouts of the best quality. However, we also notice that SPLEE cannot deal with large datasets decently due to quadratic running time. Considering larger datasets, ShortestPath combined with t-SGNE is the best alternative choice. In the next experiment on large datasets, ShortestPath and t-SGNE will be the selected pair of GE and DR methods.

3.2 Runtime of t-SNE and t-SGNE

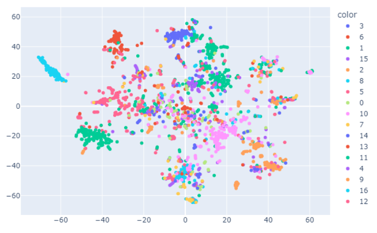
In this experiment, we compare the runtime of t-SNE and t-SGNE. Based on the results of Experiment 3.1, we fix our GE method to be ShortestPath and apply t-SNE and t-SGNE respetively on synthetic datasets of increasing size to compare their runtime. The datasets are generated using Lancichinetti–Fortunato–Radicchi (LFR) benchmark, which is an algorithm for generating synthetic networks with adjustable cluster structure [13]. LFR benchmark is typically used



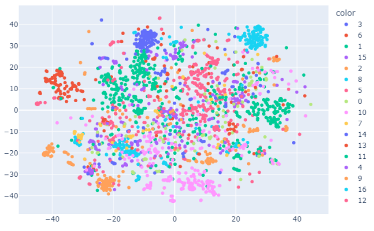
(a) DeepWalk + t-SNE



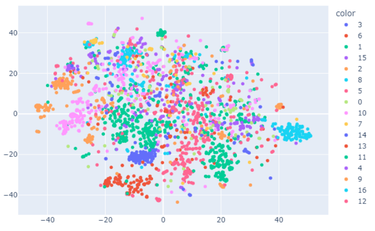
(b) ShortestPath + t-SNE



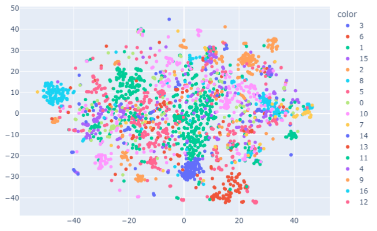
(c) SPLEE + t-SNE



(d) DeepWalk + t-SGNE



(e) ShortestPath + t-SGNE



(f) SPLEE + t-SGNE

Figure 2: Graph layouts of WIKI dataset using different combinations of GE and DR methods.

to compare between differnet community detection algorithm. It can generate datasets of arbitrary size with *a priori* known clusters. The data generation involves three parameters [13]:

- τ_1: power law exponent of the degree distribution of the graph;
- τ_2: power law exponent of the community size distribution of the graph;
- $\mu \in [0, 1]$: fraction of inter-community edges incident to each node (inter-community degree is $\mu \cdot \deg(u)$). Smaller μ values typically correspond to graphs with more separated clusters.

In this experiment, we use the implementation of LFR benchmark in the `scikit-learn` library of Python. We set $\tau_1 = 4$, $\tau_2 = 4$ and $\mu = 0.18$. The parameters are tuned so that the generated graphs will have clearly distinguishable cluster structures. Table 2 shows the results of this experiment[4].

Dataset	Size (Nodes / Edges)	ShortestPath / t-SNE / Total	ShortestPath / t-SGNE / Total
LFR_30K_0.18	30,000 / 75,643	0:00:04 / 0:01:33 / 0:01:37	0:00:04 / **0:00:23** / **0:00:27**
LFR_100K_0.18	100,000 / 253,036	0:00:17 / 0:08:12 / 0:08:29	0:00:17 / **0:01:21** / **0:01:39**
LFR_300K_0.18	300,000 / 756,664	0:00:57 / 0:32:43 / 0:33:40	0:01:03 / **0:03:54** / **0:04:57**

Table 2: Runtime comparison of t-SNE and t-SGNE on LFR-generated large datasets in the format of `h:mm:ss`. The machine we use for this experiment has a 32-core Intel Xeon Gold 6338 Processor CPU and 16GB of memory .

The results of this experiment are in accordance with our analysis in Section 2. The linear-time construction of the neighborhood graph in t-SGNE significantly reduces the running time as the size of the graph increases, compared with t-SNE in which the construction runs quadratic time. Hence, t-SGNE is a more suitable algorithm for visualizing large graph datasets.

3.3 Visualization of large graphs

In this experiment, we further push the limit of the combination of ShortestPath and t-SGNE to larger graph datasets. Based on the result of Experiment 3.1 and Experiment 3.2, this combination of GE and DR methods has the best runtime performance and a satisfying performance in the quality of the layout. We apply this combination to graph datasets involving both generated and real-world data, with sizes ranging from 10K to 4M as follows:

- LFR_30K_0.18, LFR_300K_0.18, and LFR_3M_0.18 are datasets generated by the LFR benchmark as in Experiment 3.2, with 30K, 300K, and 3M nodes respectively.

[4]During this experiment, there were other processes running on the same machine. To avoid their influences on the results, we perform each experiment three times and take the average.

- TWITCHGAMERS [18] is a network of Twitch users, where the nodes represent the users and the edges represent their following relationships. It has 168,114 nodes and 6,797,557 edges.
- DBLP [22] is a co-authorship network of researchers publishing papers of Computer Science. Nodes represent the authors and a link exists if two authors have at least one publication in common. The network has 317,080 nodes and 1,049,866 edges.
- YOUTUBECOMM [22] is a network of YouTube users, where the nodes represent the users and the edges represent the online friendship. It consists of 1,134,890 nodes and 2,987,624 edges, with 8385 user-defined communities.
- LIVEJOURNAL [14] is a network of LiveJournal users. Nodes represent the users and a link exists if a user declare the other member as a friend. The network has 3,997,962 nodes and 34,681,189 edges.

Type	Dataset	Size (Nodes / Edges)	ShortestPath	t-SGNE
Generated	LFR_30K_0.18	30,000 / 75,643	0:00:02	0:00:13
	LFR_300K_0.18	300,000 / 756,664	0:00:28	0:02:31
	LFR_3M_0.18	3,000,000 / 4,937,941	0:05:26	0:49:02
Real-world	TWITCHGAMERS	168,114 / 6,797,557	0:01:18	0:24:34
	DBLP	317,080 / 1,049,866	0:00:27	0:12:33
	YOUTUBECOMM	1,134,890 / 2,987,624	0:02:54	3:09:23
	LIVEJOURNAL	3,997,962 / 34,681,189	0:45:11	2:05:51

Table 3: Runtime of ShortestPath combined with t-SGNE on large datasets in the format of `h:mm:ss`. The machine we use for the experiment has an 8-core Apple M2 CPU and 16 GB of memory.

As can be seen from the results in Table 3, the runtime of ShortestPath combined with t-SGNE is overall satisfying on large graph datasets, finishing in less than a few hours as the size of the graph scales up to 4M nodes. No memory leak occurred at the million scale either. We do notice that the runtime of t-SGNE involves randomness in certain real-world datasets (e.g. YOUTUBECOMM). This may be due to randomness added to the stochastic machine learning algorithms, which guarantees different models for each training. However, such an abnormal increase in the runtime will not exceed an acceptable scale, since the maximum number of iterations is limited to a constant during the learning process.

4 Discussion

In this section, we briefly discuss some problems and deficiencies of our currently proposed methods and the potential future improvements to them.

General types of graphs. Although we restrict our discussion to undirected, unweighted graphs with no node attributes, t-SGNE can be applied in a more general setting. For weighted graphs, one possible modification is to construct a weighted neighborhood graph $G_{knn,G}$ from G, where the weight of the edge from x_i to x_j is the average of product of weights on the path of random walks from x_i to x_j. One can also come up with a more sophisticated function of the weights on the path to avoid the potential quick decay of w_{ij} as k grows.

Other GE methods. The reason why t-SGNE is not suitable for GE methods like SDNE is that $G_{knn,X}$ cannot approximate the distance structures in X. This raises a natural question: can we perform graph exploring algorithms more sophisticated than BFS, or more sophisticated analyses on the result of random walks, so that we can capture the more complex distance structure? For example, maybe a comparison of the degrees can capture the structural similarity of nodes.

Other combinations of GE and DR methods. ShortestPath combined with t-SGNE is currently the selected combination for visualizing large graphs due to its pleasing visualization quality and reasonable runtime. However, SPLEE combined with t-SNE is actually the best combination considering visualization quality. Is it possible to improve the quality of the combination of ShortestPath and t-SGNE, or reduce the runtime of the combination of SPLEE and t-SNE, in order to obtain a better layout in reasonable time?

References

[1] E. Amid and M.K. Warmuth. 2022. TriMap: Large-scale Dimensionality Reduction Using Triplets. `http://arxiv.org/abs/1910.00204` arXiv:1910.00204 [cs, stat].

[2] M. Belkin and P. Niyogi. 2003. Laplacian Eigenmaps for Dimensionality Reduction and Data Representation. *Neural Computation* 15, 6 (June 2003), 1373–1396. `https://doi.org/10.1162/089976603321780317`

[3] V.D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre. 2008. Fast Unfolding of Communities in Large Networks. (2008). `https://doi.org/10.48550/ARXIV.0803.0476`

[4] U. Brandes and C. Pich. 2007. Eigensolver Methods for Progressive Multidimensional Scaling of Large Data. In *Graph Drawing*, M. Kaufmann and D. Wagner (Eds.). Vol. 4372. Springer Berlin Heidelberg, Berlin, Heidelberg, 42–53. `https://doi.org/10.1007/978-3-540-70904-6_6` Series Title: Lecture Notes in Computer Science.

[5] A. Chakraborty, T. Dutta, S. Mondal, and A. Nath. 2018. Application of Graph Theory in Social Media. *International Journal of Computer Sciences and Engineering* 6, 10 (Oct. 2018), 722–729. `https://doi.org/10.26438/ijcse/v6i10.722729`

[6] K.W. Church. 2017. Word2Vec. *Natural Language Engineering* 23, 1 (Jan. 2017), 155–162. `https://doi.org/10.1017/S1351324916000334`

[7] P. Goyal and E. Ferrara. 2018. Graph Embedding Techniques, Applications, and Performance: A Survey. *Knowledge-Based Systems* 151 (July 2018), 78–94. `https://doi.org/10.1016/j.knosys.2018.03.022`

[8] A. Grover and J. Leskovec. 2016. Node2vec: Scalable Feature Learning for Networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, San Francisco California USA, 855–864. `https://doi.org/10.1145/2939672.2939754`

[9] I. Herman, G. Melancon, and M.S. Marshall. 2000. Graph Visualization and Navigation in Information Visualization: A Survey. *IEEE Transactions on Visualization and Computer Graphics* 6, 1 (March 2000), 24–43. `https://doi.org/10.1109/2945.841119`

[10] G.E. Hinton and S. Roweis. 2002. Stochastic Neighbor Embedding. In *Advances in Neural Information Processing Systems*, S. Becker, S. Thrun, and K. Obermayer (Eds.), Vol. 15. MIT Press. `https://proceedings.neurips.cc/paper/2002/file/6150ccc6069bea6b5716254057a194ef-Paper.pdf`

[11] S.G. Kobourov. 2012. Spring Embedders and Force Directed Graph Drawing Algorithms. `http://arxiv.org/abs/1201.3011` arXiv:1201.3011 [cs].

[12] Y. Koren. 2005. Drawing Graphs by Eigenvectors: Theory and Practice. *Computers & Mathematics with Applications* 49, 11-12 (June 2005), 1867–1888. `https://doi.org/10.1016/j.camwa.2004.08.015`

[13] A. Lancichinetti and S. Fortunato. 2009. Benchmarks for Testing Community Detection Algorithms on Directed and Weighted Graphs with Overlapping Communities. *Physical Review E* 80, 1 (July 2009), 016118. `https://doi.org/10.1103/PhysRevE.80.016118`

[14] J. Leskovec, K.J. Lang, A. Dasgupta, and M.W. Mahoney. 2008. Community Structure in Large Networks: Natural Cluster Sizes and the Absence of Large Well-Defined Clusters. `http://arxiv.org/abs/0810.1355` arXiv:0810.1355 [physics].

[15] L. Maaten and G. Hinton. 2008. Visualizing Data Using t-SNE. *Journal of Machine Learning Research* 9, 86 (2008), 2579–2605. `http://jmlr.org/papers/v9/vandermaaten08a.html`

[16] L. McInnes, J. Healy, and J. Melville. 2020. *UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction*. Technical Report arXiv:1802.03426. arXiv. `http://arxiv.org/abs/1802.03426` arXiv:1802.03426 [cs, stat] type: article.

[17] B. Perozzi, R. Al-Rfou, and S. Skiena. 2014. DeepWalk: Online Learning of Social Representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, New York New York USA, 701–710. `https://doi.org/10.1145/2623330.2623732`

[18] B. Rozemberczki and R. Sarkar. 2021. Twitch Gamers: a Dataset for Evaluating Proximity Preserving and Structural Role-based Node Embeddings. `http://arxiv.org/abs/2101.03091` arXiv:2101.03091 [cs].

[19] A.R. Ryan and K.A. Nesreen. [n.d.]. Network Data Repository | The First Interactive Network Data Repository. `https://networkrepository.com/`

[20] L. Torres, K.S. Chan, and T. Eliassi-Rad. 2020. GLEE: Geometric Laplacian Eigenmap Embedding. *Journal of Complex Networks* 8, 2 (April 2020), cnaa007. `https://doi.org/10.1093/comnet/cnaa007`

[21] S. Wold, K. Esbensen, and P. Geladi. 1987. Principal Component Analysis. *Chemometrics and Intelligent Laboratory Systems* 2, 1-3 (Aug. 1987), 37–52. `https://doi.org/10.1016/0169-7439(87)80084-9`

[22] J. Yang and J. Leskovec. 2012. Defining and Evaluating Network Communities based on Ground-Truth. `http://arxiv.org/abs/1205.6233` arXiv:1205.6233 [physics].