Noname manuscript No.
(will be inserted by the editor)

An Analytical Survey on Recent Trends in High Dimensional Data Visualization

Alexander Kiefer · Md. Khaledur Rahman

the date of receipt and acceptance should be inserted later

Abstract Data visualization is the process by which data of any size or dimensionality is processed to produce an understandable set of data in a lower dimensionality, allowing it to be manipulated and understood more easily by people. The goal of our paper is to survey the performance of current high-dimensional data visualization techniques and quantify their strengths and weaknesses through relevant quantitative measures, including runtime, memory usage, clustering quality, separation quality, global structure preservation, and local structure preservation. To perform the analysis, we select a subset of state-of-the-art methods. Our work shows how the selected algorithms produce embeddings with unique qualities that lend themselves towards certain tasks, and how each of these algorithms are constrained by compute resources.

Keywords Data Visualization · Graph Visualization · Dimensionality Reduction

1 Introduction

According to the International Data Corporation (IDC), a leading market analytics company within the information technology industry, it is projected that "the Global Datasphere will grow from 33 Zettabytes (ZB) in 2018 to 175 ZB by 2025" (Rydning, 2018). With this significant increase in global information production come a number of opportunities for businesses to make more informed decisions using mathematics, statistics, and machine learning. However, with this increase in data, also come a number of challenges that must be addressed in order to reap its possible benefits, chief among those being the difficulty associated with the interpretation and understanding of raw data produced by technology. Data visualizations are capable of providing a great depth of information with high density, making them crucial tools in conveying the underlying meaning of data to people.

Data visualization is the process by which data of any size or dimensionality is processed to produce an understandable set of data in a lower dimensionality, allowing it to be manipulated and understood more easily by people. The typical dimension for visualization is 2D or 3D in picture/image format. Beyond 2D or 3D, the visualization is not easily perceivable by human beings. Good quality visualizations allow us to extract a wealth of meaningful information at first hand, as the famous saying goes"a picture is worth a thousand words" (Pinsky and Wipf, 2000).

There are a handful number of dimensionality reduction techniques that help us reduce the high dimensional data to 2D or 3D data for visualization. The dimensionality reduction algorithms which we will focus on typically fall into two groups, those which attempt to maintain the global structure of the data and those which maintain local distances over global distance (McInnes et al., 2018). In this paper, of the algorithms discussed, TriMap (Amid and Warmuth, 2019) falls into the first category, while UMAP (McInnes et al., 2018), LargeVis (Tang et al., 2016), and t-SNE (Maaten and Hinton, 2008) fall into the second. Other research fields, such as computational biology and graph embedding, are being significantly benefited from the visualization of high-dimensional data by using these methods (Becht et al., 2019; Kobak and Berens, 2019; Tsitsulin et al., 2018; Rahman et al., 2020b). The focus of our study will be on the effectiveness and applications of each of these algorithms for high-dimensional data visualization.

The dimensionality reduction techniques mainly work on the high-dimensional numeric data. However, if we have graphs/networks, we cannot apply them directly though they are also considered as high-dimensional structured datasets (Erdös et al., 1965). Thus, in close association with the previous process, graph visualization is a method by which data of low dimensionality (2D or 3D) is used to generate visual interpretations of

A. Kiefer, Indiana University Bloomington, E-mail: alkiefer@iu.edu · M. K. Rahman, Indiana University Bloomington, E-mail: morahma@iu.edu

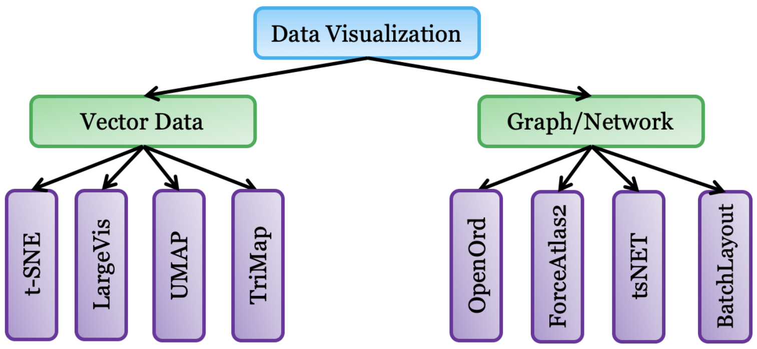


Fig. (1) Data visualization hierarchy

graphs or networks, making the relationships within the data more easily understood by users. These types of datasets are structured and we can extract valuable information from the visualizations. In this survey, we focus on analyzing the runtimes, memory consumption and the quality of layouts for some general purpose graph visualization techniques. Force-directed techniques are widely used for graph visualization. Thus, we will primarily focus on some state-of-the-art force-directed methods for large-scale graph visualization along with a dimensionality reduction technique for graph visualization.

The difference between the high-dimensional data point and graph visualization is that we only show the positions of the data points for the former whereas edges of a graph are also drawn in 2D/3D space for the latter. Though there are multiple survey papers in the literature for data visualization, an analytical study showing the performance comparison of existing methods is missing (Liu et al., 2014; Chen et al., 2019; Van Der Maaten et al., 2009). In this survey paper, we fill this gap by analyzing runtimes, memory consumption, and aesthetic quality scores of some representative visualization methods from the two categories. To make a fair comparison, we run all the methods in the same server machine, and carefully analyze the results. We summarize the main focus of our contributions as follows:

– We select a set of representative state-of-the-art visualization methods based on the popularity, and present the underlying methods in a concise but understandable format. Our survey covers two fundamental visualization areas: (i) vector data, and (ii) graphs or networks, which provide a global picture of the respective visualization field.
– We compare all the methods in terms of runtimes, memory consumption, and quality scores to analyze the advantage and disadvantage of using a respective method.
– Finally, we analyze the results to provide recommendations to users or researchers so that they can get an idea regarding which method to use for a given set of resources.

2 Preliminaries

We represent high-dimensional vector data by $X \in \mathbf{R}^{n\times d}$, where n is the number of entries and d is the dimension of each data entry. Hence, x_i represents d-dimensional vector of i-th entry of X. We represent the lower dimensional projection of X by $Y \in \mathbf{R}^{n\times 2}$, where y_i represents (x, y) coordinates of i-th entry in 2D space. Let $G(V, E)$ be a graph, where V is the set of vertices and E is the set of edges. Unless otherwise mentioned, we use the same definition of Y to represent the layout of graphs in 2D space.

Asymmetric high dimensional data points belong to vector data category when they have no explicit structural connection such as images, words, etc. On the other hand, data points having structural information belong to Network category. Without loss of generality, graph and network bear the same meaning in our study. Thus, we use these two terms interchangeably.

3 High Dimensional Data Visualization

Data can come from a variety of sources which can be structured or unstructured. Different methods have been proposed in the literature to visualize both types of data. In this survey, we discuss visualization methods focusing on two branches: (i) vector data, and (ii) graph data. We select a few methods to analyze both types of visualization techniques. Fig. 1 shows a hierarchy of our selected data visualization techniques.

3.1 Vector Data Visualization

Generally, dimensionality reduction techniques are used to project high dimensional data points to lower dimensions. This lower dimensional projection captures the intrinsic properties of the vector data as much as possible for readable visualization. Similar data points are clustered together whereas dissimilar points are placed apart. We discuss some of the state-of-the-art methods as follows.

3.1.1 t-SNE (Maaten and Hinton, 2008)

For high-dimensional vector data visualization, t-SNE is a pioneering tool and a progenitor of most modern dimensionality reduction tools. The algorithm significantly improved upon ones prior to it by allowing for much higher dimensionality in data sets and, overall, better visualization by reducing the tendency of points to cluster at the center of the graph, which is often referred to as the crowding-out problem. By employing a variation of Stochastic Neighbor Embedding (Hinton and Roweis, 2003), t-SNE is significantly easier to optimize, as compared to prior methods, increasing both the efficiency and quality of graphs, with a computational complexity modeled by $O(N^2)$.

To give a very general overview of t-SNE, high-dimensional data sets are converted into a pairwise similarities matrix to allow for construction of accurate and meaningful visualizations. Stochastic neighbor embedding begins by converting the high-dimensional Euclidean distances within the input data into a set of conditional probabilities, which represents the similarity between the data points. The conditional probability, $p_{j|i}$, can be modeled by Eqn. 1 as follows.

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

Where the similarity between two data points x_i and x_j is equal to the conditional probability that x_i is the neighbor of x_j, given that neighbors are picked under a Gaussian centered at x_i in proportion to their probability density. σ_i is the variance of the Gaussian on x_i. The values of $p_{i|i}$ is set to zero in order to constrain the model to pairwise similarities.

Next, to find the low-dimensional results, y_i and y_j of x_i and x_j another conditional probability, $q_{j|i}$ is calculated. However, in this instance, the variance of the Gaussian is set to $\frac{1}{\sqrt{2}}$. Thus, the similarity between the mapped points y_i and y_j is modeled by

$$q_{j|i} = \frac{exp(-||y_i - y_j||^2)}{\sum_{k \neq i} exp(-||y_k - y_l||^2)} \tag{2}$$

If the values of $q_{j|i}$ and $p_{j|i}$ are equal, then the mapped points y_i and y_j correctly model the original data points x_i and x_j. However, to improve efficiency, SNE allows for a small amount of variance between $q_{j|i}$ and $p_{j|i}$, which it optimizes by minimizing a Kullback-Leibler divergence between a joint probability distribution, P, in high-dimensional space and a joint probability distribution, Q, in low-dimensional space. The cost function of this method is given by the model

$$C = KL(P||Q) = \sum_i \sum_j p_{j|i} log \frac{p_{j|i}}{q_{j|i}} \tag{3}$$

Where P_i is the probability distribution, given data point x_i, over all other data points and Q_i is the same, but given data point y_i. Due to the unsymmetrical nature of the Kullback-Leibler divergence, if the mapped points used to represent nearby data-points are far apart, they will have a higher cost function associated with them. However, if the mapped points used to represent far apart data-points are close together, they will have a lower cost function associated with them.

The final parameter that must be set is the variance σ_i of the Gaussian. It is beneficial to pick a smaller value for σ_i when the region is more dense, and a larger one when it is more sparse. The value σ_i is found through a binary search with perplexity P_i defined by the user. One of the superior qualities of symmetric SNE, as compared to asymmetric SNE, is that the gradient function (Cook et al., 2007) is much simpler, decreasing computation time.

$$\frac{\delta C}{\delta y_i} = 4 \sum_j (p_{ij} - q_{ij})(y_i - y_j) \tag{4}$$

3.1.2 LargeVis (Tang et al., 2016)

Developed in 2016, LargeVis aims to solve the problem of the high computational cost associated with dimensionality reduction problem such as t-SNE. To give a very general overview of the LargeVis method, first, an accurately approximated k-nearest neighbor graph is generated from the input data. After this, the graph is then projected in a low dimensional space in a layout which is easily readable by the viewers. By using a principle probabilistic model, the dimensional reduction of the data can be optimized using an asynchronous stochastic gradient descent with a linear time complexity. Some advantages that LargeVis has over other methods are higher stability of its hyper-parameters over a large variety of datasets and greater efficiency and effectiveness as compared to t-SNE, with an overall computational complexity of $O(dsN)$, where d is the dimension of the low dimensional space, s is the number of negative samples, and N is the number of points in the dataset.

One of the primary challenges faced with dimensionality reductions is the construction of the k-nearest neighbor graph, which from now on will be referred to as the k-NN graph. In t-SNE, vantage point trees (Yianilos, 1993) are used to construct the k-NN graph. Though this is an effective method for relatively large data sets, the performance greatly reduces as the dimensionality of the data increases.

The computational complexity of an exact k-NN can be modeled by the formula $O(N^2 d)$, with N being the total number of data points and d the number of dimensions. This is considered very complex, so it is reduced using techniques for in-exact approximation such as space-partitioning trees (Charikar, 2002; Datar et al., 2004; Gionis et al., 1999), and nearest neighbor exploration (Dong et al., 2011). By partitioning the space, the data can be more easily organized into tree-structures while preserving the general structure of the data, with random projection trees (Datar et al., 2004) being used due to their high efficiency in nearest neighbor exploration. Locality sensitive hashing is then used to organize the data into "buckets" which contain data considered to be similar. Finally, the k-NN graph is refined and improved over a certain amount of iterations.

Going into greater detail, LargeVis begins by calculating the weights of the edges in the k-NN graph using the same method as t-SNE, modeled by Eqn. 1. The graph is then made symmetric by finding the weight between all inputs, modeled by

$$w_{ij} = \frac{p_{i|j} + p_{j|i}}{2N} \tag{5}$$

It is at this point that LargeVis begins to use a unique probabilistic model for visualizing the k-NN graph. With a goal of maintaining similarity between vertices in low dimensional space, the probability of observing a binary edge $e_{ij} = 1$ from a pair of vertices (v_i, v_j) is modeled as follows,

$$P(e_{ij} = 1) = f(||\vec{y_i} - \vec{y_j}||) \tag{6}$$

where $\vec{y_i}$ is the low dimensional embedding of v_i and $\vec{y_j}$ of v_j. Here, $f()$ is any probabilistic function calculated with respect to the distance between the embedded vertices. The type of probabilistic function used can be varied to optimize results based on the structure of the data. Generalizing this equation for weighted edges, the probability of a weighted edge $e_{ij} = w_{ij}$ can be modeled by

$$P(e_{ij} = w_{ij}) = P(e_{ij} = 1)^{w_{ij}} \tag{7}$$

Given these, the probability of the weighted graph, $G = (V, E)$, is modelled as

$$O = \prod_{(i,j)\epsilon E} p(e_{ij} = 1)^{w_{ij}} \prod_{(i,j)\epsilon \bar{E}} (1 - p(e_{ij} = 1))^{\gamma} \propto \sum_{(i,j)\epsilon E} w_{ij} \log p(e_{ij} = 1) + \sum_{(i,j)\epsilon \bar{E}} \gamma \log (1 - p(e_{ij} = 1)) \tag{8}$$

where $\bar{E}$ is the set of unobserved vertex pairs and γ the collective weight of the negative edges. The first component of this formula shows the probability of observed edges, which when maximized, clusters similar points together in the low dimensional space. The second component models the probability of negative edges which, when maximized, ensures differing points stay far apart in the low dimensional space.

However, maximizing both of these equations is inefficient and computationally difficult. In order to mitigate this, negative edges are randomly sampled using a noisy distribution $P_n(j)$, with every vertex i having randomly chosen vertices j, with (i, j) representing the negative edges. With a noisy distribution of $P_n(j) \propto d_j^{0.75}$, where d_j is the degree of the vertex j, and M being the number of negative samples per positive edge, a more easily computable objective function can be found where

$$O = \sum_{(i,j)\epsilon E} w_{ij} (\log p(e_{ij} = 1) + \sum_{k=1} E_{j_k \sim P_n(j)} \gamma \log (1 - p(e_{ij_k} = 1))) \tag{9}$$

Finally, Eqn. 9 can be optimized using an asynchronous stochastic gradient descent, accelerating training and improving performance on sparse data sets. An innovative feature of LargeVis, in order to maintain the norms of the gradient, edges are randomly sampled (Tang et al., 2015) with a probability proportional to their weights and then used as binary edges in the stochastic gradient descent.

3.1.3 UMAP (McInnes et al., 2018)

Uniform Manifold Approximation and Projection (UMAP) is a manifold learning algorithm used for the dimensionality reduction problem. The algorithm is rooted in Riemannian geometry and algebraic topology (May, 1992), allowing it to scale to much larger datasets, as compared to other algorithms, such as t-SNE. UMAP is commonly used in a variety of fields, including bioinformatics (Bagger et al., 2016; Becht et al., 2018; Diaz-Papkovich et al., 2018), material science (Fuhrimann et al., 2018; Li et al., 2018), and machine learning (Escolano et al., 2018; Blomqvist et al., 2018).

To give a general overview of the UMAP algorithm, local manifold approximations are combined to find their local fuzzy simplicial set representations (Goerss and Jardine, 1999). These representations are then used to construct a topological representation of the high dimensional data. Using this topological representation, a low dimensional representation of the data can be inferred. UMAP then optimizes the layout of the data representation in the low dimensional space in order to minimize the cross-entropy between the two topological representations.

Going into greater detail, we will discuss the specific computations in the UMAP algorithm. Being based heavily upon an in depth mathematical analysis of the problem, the motivation for certain steps in the algorithm can be found in Section 2 of (McInnes et al., 2018).

To begin, UMAP constructs a weighted k-nearest neighbors graphs using an input high-dimensional data-set $X = \{x_i, ..., x_n\}$ and a dissimilarity measure $m : X \times X \to \mathbb{R}_{\geq 0}$. Using a given hyper-parameter k, the set of k-nearest neighbors $\{x_{i1}, ..., x_{ik}\}$ is calculated according to the metric m using nearest neighbor descent (Dong et al., 2011).
Next, for every x_i, p_i and σ_i are calculated such that

$$p_i = \min\{m(x_i, x_{ij})|1 \leq j \leq k, m(x_i, x_{ij}) > 0\} \tag{10}$$

and

$$\sum_{j=1}^{k} \exp(\frac{-\max(0, m(x_i, x_{ij}) - p_i)}{\sigma_i}) = \log_2(k) \tag{11}$$

To construct the weighted-directed graph $\bar{G} = (V, E, w)$, we use $V = X$, $E = \{(x_i, x_{ij})|1 \leq j \leq k, 1 \leq i \leq N\}$, and

$$w((x_i, x_{ij})) = \exp(\frac{-\max(0, m(x_i, x_{ij}) - p_i)}{\sigma_i}). \tag{12}$$

Finally, the symmetric adjacency matrix B of the undirected, weighted graph output for UMAP, G, can be defined by

$$B = A + A^\top - A \circ A^\top, \tag{13}$$

where $\circ$ is the pointwise product and A is the weighted adjacency matrix of $\bar{G}$

To calculate the graph layout for G, UMAP utilizes a force directed graph layout algorithm in the desired low dimensional space which, using a set of attractive forces on edges and a set of repulsive forces on vertices, converges the graph and makes it aesthetically pleasing. With hyper-parameters of a and b, the attractive force between two vertices i and j at coordinates y_i and y_j is calculated as

$$\frac{-2ab||y_i - y_j||_2^{2(b-1)}}{1 + ||y_i - y_j||_2^2} w((x_i, x_j))(y_i - y_j) \tag{14}$$

In order to reduce the computational cost, edges are sampled from every vertex that has an attractive force applied to it and then one is repulsed. The repulsive force is calculated as

$$\frac{b}{(\epsilon + ||y_i - y_j||_2^2)(1 + ||y_i - y_j||_2^2)} (1 - w((x_i, x_j)))(y_i - y_j)), \tag{15}$$

where ϵ is an arbitrarily small number to prevent division by 0. By default, it is set to 0.001. G is initialized to a spectral layout which allows for quicker convergence and improved stability of the algorithm.

3.1.4 TriMap (Amid and Warmuth, 2019)

The TriMap method is one of the most recent tool for dimensionality reduction. With the majority of dimensionality reduction techniques aiming to preserve local neighborhood structure, TriMap focuses precisely upon providing an algorithm which preserves the global structure of the data. One of the major differences in the implementation of the TriMap method is the use of triplets to create an embedding, as compared to the more common pairwise method used by other algorithms. Here, they are represented as (i, j, k), with the interpretation being that point i is closer to point j than point k.

To give a general overview of the TriMap algorithm, using a low-dimensional representation of the data created using the PCA algorithm (Hotelling, 1933), triplets from the high dimensional representation are used to refine the quality of the low dimensional representation.

Going into greater depth, TriMap begins by sampling a subset of triplets $\tau = \{(i, j, k)\}$ and giving each a weight $\omega_{ijk} \geq 0$, where large values signify that (i, k) are more distant than (i, j). Using this, the cost function for a triplet (i, j, k) is defined as

$$l_{ijk} := \omega_{ijk} \frac{s(y_i, y_k)}{s(y_i, y_j) + s(y_i, y_k)}, \text{ where } s(y_i, y_j) = (1 + |y_i - y_j||^2)^{-1} \tag{16}$$

The unnormalized weight of a triplet in the high-dimensional space can be defined as

$$\tilde{\omega}_{ijk} = exp(D^2_{ik} - D^2_{ij}) \leq 0 \tag{17}$$

where D_{ij} and D_{ik} are the distances between points x_i, x_j and x_i, x_k in the high dimensional space, respectively. To find the euclidean distance, it is scaled by

$$D^2_{ij} = \frac{||x_i - x_j||^2}{\sigma_{ij}} \tag{18}$$

where σ_{ij} is the product of σ_i and σ_j. σ_i is the average Euclidean distance between x_i and the set of its nearest-neighbors, up to 6-th neighbors. In this way, depending upon the density of the data, σ_{ij} will dynamically change the scaling.

Finally, in order to improve the local accuracy of the algorithm, the previously calculated weights $\tilde{\omega}_{ijk}$ undergo a γ-scaled log-transformation, accentuating smaller weights, and pushing all other points farther away. Therefore, the final weight is defined as follows.

$$\omega_{ijk} = \zeta_\gamma(\frac{\tilde{\omega}_{ijk}}{W} + \delta) \text{ , where } \zeta_\gamma(u) := \log 1 + \gamma u \tag{19}$$

where $W = \max_{(i',j',k') \epsilon \tau} \tilde{\omega}_{i'j'k'}$, γ is a small constant, and $\delta > 0$ is the scaling factor. Both δ and γ can be set by the user.

Using all of this information, the embedding can be constructed. Considering a subset of all possible triplets (i, j, k), where j is one of i's nearest neighbors and k is one of i's most distant neighbors, we now define our hyper-parameters. Where $m = 10$ is the number of nearest neighbors for each point and $m' = 5$ is the number of triplets sampled for each nearest neighbor. This leaves us with 50 nearest neighbor triplets per point. In order to account for noise in the data and provide a more robust model, we allow for $r = 5$ randomly sampled, and ordered, triplets to be inserted, making the final count of triplets per point $m \times m' + r = 55$. Utilizing the ANNOY library for the construction of the approximate nearest neighbors tree, the initial embedding is set to the PCA embedding and scaled to improve convergence. By doing this, the algorithm provides superior global structure preservation in the embedding. The final cost function for the Trimap algorithm is represented as

$$l_{\text{ TriMap}} = \sum_{(i,j,k) \epsilon \tau} l_{ijk} \tag{20}$$

3.2 Graph Visualization

Graphs are considered as high-dimensional data (Erdös et al., 1965). Thus, visualization of graphs is similar to vector data provided that the graph has a given structural information i.e., connectivity of vertices by edges. There are various methods in the literature for graph visualization. We select a few of them for our study based on the popularity, runtime and quality of the visualizations. We discuss our selected methods as follows.

3.2.1 OpenOrd (Martin et al., 2011)

The OpenOrd method is a high-performance tool for graph visualization that can generate layout of graphs having millions of vertices. This method employs the spring-electrical model in its underlying optimization function. It uses a multi-level approach for graph layout generation. At each level, a force-directed approach is used to generate layout at each step. A graph is coarsened to reduce the size. In the coarsening step, vertices are merged using the average link clustering approach. When it reaches to the coarsest level, coarsened steps are reversed back to gain the final layout of the graph. Prolongation step and layout generation are performed at tandem so that it consumes less time. It optimizes the following objective function using simulated annealing heuristic approach.

$$min_{x_1,\ldots,x_n} \sum_i (\sum_j w_{ij} E(x_i, x_j)^2 + D_{x_i}) \tag{21}$$

In Eqn. 21, $E(x_i, x_j)$ represents the attractive force between vertices i and j using Euclidean distance, D_{x_i} represents the approximated repulsive force of vertex i, and w_{ij} represents the weight of the edge between vertices i and j.

OpenOrd has been implemented using parallel computing techniques. It also supports several other options e.g., edge cutting is to visualize the densely connected graph. OpenOrd has a few drawbacks which are listed below:

– One of the drawbacks of this approach is that it uses an adjacency matrix for graph representation. Thus, it easily reaches the available memory limit for large graphs.
– Parallel version has another drawback, for multiple cores, several runs results in different layouts which do not preserve consistency as required in scientific computing.

3.2.2 ForceAtlas2 (Jacomy et al., 2014)

The ForceAtlas2 (Jacomy et al., 2014) method is an engineering advancement that integrates several force-directed methods in Gephi software (Bastian et al., 2009) for general purpose network visualization. Authors represent attractive force F_a and repulsive force F_r of two vertices i and j in terms of Euclidean distance as follows: $F_a \propto E^a(i, j)$, and $F_r \propto E^{-r}(i, j')$, where $E^n(i, j)$ is defined as follows.

$$E^n(i, j) = \{(i_x - j_x)^2 + (i_y - j_y)^2\}^{1/n} \tag{22}$$

where, i_x and i_y are (x, y)-coordinates of vertex i. We can compute repulsive force of vertex i with respect to vertex j' similar to Eqn. 22. Now, changing the value of (a, r), we can generate different force-directed models (Hu, 2005; Kobourov, 2012). A repulsive force is computed when two vertices are not connected by an edge. For scale-free networks or graphs having low average degree, this computation can be asymptotically $O(n)$. Thus, the repulsive force computation is the most time-consuming part of force-directed models. In ForceAtlas2, the authors apply quadtree-based repulsive force approximation technique that reduce the overall runtime of the model (Barnes and Hut, 1986). Specifically, this technique reduces the runtime of repulsive computation from $O(n)$ to $O(\log n)$. Though ForceAtlas2 supports multi-threading, its Java-based implementation consumes high amount of memory and is not very effective to utilize the memory bandwidth optimally. Nevertheless, it generates readable layout for both connected and disconnected graphs. Later, this model has been implemented in GPU to generate layout of graphs having millions of vertices (Brinkmann et al., 2017).

3.2.3 tsNET (Kruiger et al., 2017)

The tsNET (Kruiger et al., 2017) is a version of the t-SNE (Maaten and Hinton, 2008) which has been formulated for graph layout problem. The attractive force is computed using a similar approach to t-SNE. First, the graph theoretic shortest path distance is calculated for all vertices. Then, the joint probability between the adjacent vertices is calculated using the t-SNE approach. For repulsive force, the logarithmic difference between non-adjacent vertices is computed for all vertices and then added to the average of it optimization function. The optimization function of tsNET is given below:

$$C = \lambda_a C_{KL} + \lambda_c \frac{1}{N} \sum_{i,j \in V} \parallel y_i - y_j \parallel + \lambda_r \frac{1}{N} \sum_{i,j \in V} \log(||y_i - y_j|| + \epsilon) \tag{23}$$

This method can generate good quality graph visualization. However, it has some drawbacks such as this method runs slower compared to other state-of-the-art graph visualization methods. Thus, it cannot generate layouts of large graphs. Since, choosing a good value of perplexity parameter is difficult in t-SNE (Wattenberg et al., 2016), the problem remains the same for tsNET. Sometimes, this method does not converge which results in an unreadable layout.

3.2.4 BatchLayout (Rahman et al., 2020a)

BatchLayout is a very recent method that focuses on minimizing runtime and memory consumption while maintaining a good quality of the layout (Rahman et al., 2020a). It employs the popular Fruchterman-Reingold (Fruchterman and Reingold, 1991) spring-electrical models to preserve the quality of the layout, quad-tree data structure to reduce the runtime to $O(n \log n)$, and a mini-batch approach to obtain a faster parallel implementation. The original spring-electrical model has a sequential dependency similar to Stochastic Gradient Descent (SGD) which obstructs a massively parallel implementation. In the mini-batch approach, the attractive

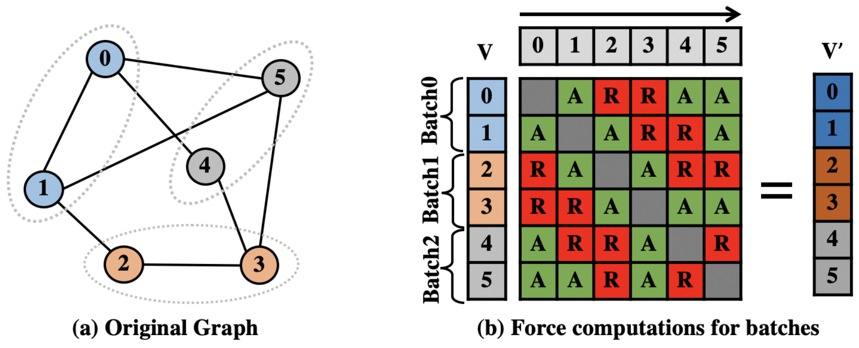


Fig. (2) Force computations in BatchLayout: (a) The original graph where different batches are shown by a gray-dotted area. (b) Each batch contains two vertices and attractive and repulsive forces within a batch are computed in parallel and then coordinates are updated. V and V' represents vertex sets of before and after updating coordinates, respectively. 'A' and 'R' in each cell of the matrix represent attractive and repulsive force, respectively.

and repulsive forces are calculated for a subset of vertices in parallel. Then, the corresponding coordinates are updated based on the combined forces (see Fig. 2). In Fig. 2, there are three mini-batches such as $\{0, 1\}$, $\{2, 3\}$, and $\{4, 5\}$. We can compute the forces of verteces 0 and 1 in parallel as they are in the same batch. We do not consider updated coordinates of vertices 0 and 1 while computing forces with respect to these vertices. Then, we move to the next minibatch and compute forces of vertices 2 and 3. Now, we consider the updated coordinates of the previous batches i.e., $\{0, 1\}$. Since, the updated coordinates of these vertices are not considered while computing forces in a batch, the convergences of this approach takes higher iterations to converge. However, the parallel computing approach makes it significantly faster that easily compensate to generate good quality layout. Authors empirically show that the scalable parallel implementation of BatchLayout can catch the total number of converging iterations within a very short time compared to the sequential approach. In summary, the BatchLayout method can generate readable quality layout very fast compared to other methods.

3.2.5 Other Graph Visualization Methods

Over the years, researchers have put efforts to reduce the runtime in force-directed models. Recently, a random sampling based repulsive force calculations approach has been introduced (Gove, 2019). Authors call it the Random Vertex Sampling (RVS) approach. The approach of this algorithm can be summarized as follows: select a subset of vertices $|V|^{\phi}$ to update the attractive force. Then, select $|V|^{\phi-1}$ randomly selected vertices for repulsive force calculations. Thus, the total time complexity will be $O(|V|^{\phi}.|V|^{\phi-1}) = O(|V|)$, where, ϕ is a real number such that $0 < \phi < 1$. Authors proposed to update the force from left to right based on a sliding window approach so that each vertex gets a chance to be updated by repulsive forces. This method has another variation which runs $T - k$ iterations using random sampling approach and then runs last k iterations using Barnes-Hut approach which produce good layout. Authors have tested their approach for graphs up to 100K vertices. Though this method introduces a fast sequential approach, it has some possible drawbacks which have not been considered in the model such as the randomly selected vertices can be neighbors (false negative) for repulsive force computation, and the suggested value of ϕ to be $\frac{3}{4}$ based on experiments, though graphs with a higher number of vertices will suffer as a result. It is obvious that for a small set of randomly selected vertices, the runtime for repulsive force calculations will always be low. However, it is also possible that the layout of such graph may not be readable. In the experimental results of the paper, the benchmark graph shows such negative results where RVS approach performs worse than Barnes-Hut approach.

There are various graph visualization softwares that can employ different graph visualization algorithms based on the users' expectation such as Gephi (Bastian et al., 2009), GraphVis (Ellson et al., 2001), OGBD (Chimani et al., 2013), etc. These tools provide either graphical user interface or command line interface while supporting multi-core computations. They provide users a great advantage such as flexibility of choosing a graph visualization algorithm.

4 Experiment

4.1 Experiment Setup

For our experiment, we analyze the runtime, memory performance, and aesthetic qualities of all four algorithms. In order to measure runtime, we use the *time* function [1] available in the bash shell, where the output "real" time is measured. To measure the memory usage of each algorithm over time, we will utilize the memory_profiler [2] package available for Python 3 .

For t-SNE, we use the original implementation distributed in the scikit-learn package [3] for Python 3. For LargeVis, we also use its original implementation in C++ [4]. For UMAP, we use the distribution available on conda-forge, which is a redistribution of the original algorithm's GitHub [5]. Finally, we use the TriMap implementation distributed on the Python Packed Index, which is sourced from the algorithms GitHub page [6].

We have run all tests on a system running RedHat Linux with an Intel(R) Xeon(R) CPU E5-2670 v3 CPU running at 2.30GHz. All of the tests have been run using the default parameters for each respective algorithm. For our vector data visualization experiments, we use the MNIST Digits dataset [7], a standard dataset used in a large variety of machine learning applications, composed of labeled images of handwritten numbers, and the Fashion MNIST dataset [8], a more modern and difficult dataset based on labeled images of clothing items.

For the analysis of our results, we measure the runtime and memory usage of each algorithm. We will also consider the quality of the produced embedding, analyzing the quality of the clustering, which is the tendency for similar points to group together, the delineation between clusters, which is tendency for clusters to be separate from other clusters (i.e., not overlap), and the retention of the local and global structure of the embedded data. Our quantitative measures[9] for these qualities are their Davies-Bouldin and Silhouette scores, and trustworthiness and global scores for their structures.

The Davies-Bouldin score (Davies and Bouldin, 1979) measures the quality of the clustering through the average ratio of within-cluster distances to between-cluster distances, with the best score being 0. The similarity between clusters R_{ij} is defined as

$$R_{ij} = \frac{s_i + s_j}{d_{ij}} \tag{24}$$

where s_i and s_j are the average distance between each point of cluster i and j and the centroid, or diameter, of that cluster, and dij is the distance between cluster centroids i and j. The final score is then calculated as

$$DB = \frac{1}{k} \sum_{i=1}^{k} \max_{i \neq j} R_{ij} \tag{25}$$

with k being the total number of clusters.

The silhouette score (Rousseeuw, 1987) measures the quality of the delineation between each point in each cluster on average, where positive values indicate superior delineation of values between clusters, values near 0 indicate overlapping clusters, and more negative values indicate points mapped to incorrect clusters. The score for one sample is calculated as

$$s = \frac{b - a}{max(a, b)} \tag{26}$$

where, a is the average distance between the current point and all other points in the same class and b is the average distance between the current point and all other points in the next nearest cluster. The overall score for an embedding containing multiple samples is calculated as

$$SC = \frac{\sum i = 1^k s(i)}{n} \tag{27}$$

where, i is the current sample and n is the total number of samples.

[1]https://ss64.com/bash/time.html

[2]https://github.com/pythonprofilers/memory_profiler

[3]https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html

[4]https://github.com/lferry007/LargeVis

[5]https://github.com/lmcinnes/umap

[6]https://github.com/eamid/trimap

[7]http://yann.lecun.com/exdb/mnist/

[8]https://github.com/zalandoresearch/fashion-mnist

[9]https://github.com/alexk101/dimensionality_reduction_measures

The trustworthiness score is a measure that expresses how well the local structure of the data has been preserved by comparing the original dataset to the embedded dataset, where the score ranges from 0 to 1, 1 being the best. The measure is calculated as

$$T(k) = 1 - \frac{2}{nk(2n-3k-1)} \sum_{i=1}^{n} \sum_{j \in \mathcal{N}_i^k} \max(0, (r(i,j) - k)) \tag{28}$$

where for every sample i up to the total number of samples n, $\mathcal{N}_i^k$ are its k-nearest neighbors in the embedding and each sample j is the $r(i,j)$-th nearest neighbor in the original space.

The global score, defined within the implementation of TriMap as a quality measure, is the minimum reconstruction error of the original dataset through the use of a linear inverse map derived from PCA, where the score ranges from 0 to 1, 1 being the best. With n data points such that $x_i \epsilon \mathbb{R}^m{}_{i=1}^n$, $X \epsilon \mathbb{R}^{m\times n}$ the ith column of the original dataset matrix corresponding to x_i, and $Y \epsilon \mathbb{R}^{d\times n}$ the embedded matrix corresponding to $y_i \epsilon \mathbb{R}^d{}_{i=1}^n$, the minimum reconstruction error (mre) for a given embedding is calculated as

$$\varepsilon(Y|X) := \min_{A \epsilon \mathbb{R}^{m\times n}} ||X - AY||_F^2 \tag{29}$$

where, $||\cdot||_F$ is the Frobenius norm of the matrix. Using this mre, the global score is then formulated as

$$GS(Y|X) := \exp(-\frac{\varepsilon(Y|X) - \varepsilon_{pca}}{\varepsilon_{pca}}) \epsilon [0,1] \tag{30}$$

where $\varepsilon_{pca} := \varepsilon(Y_{pca}|X)$, which is the ideal mre produced on the X dataset using the PCA embedding method.

We use the implementation of these measures found in the scikit-learn package, except for the global score, which is a part of the TriMap method implementation and can be found on the paper's respective Github page.[10,11,12]

4.2 Results

4.3 Comparison of High-Dimensional Data Visualization Tools

Table (1) Actual (minutes:seconds) and theoretical runtimes for reducing the dimensionality of the MNIST datasets from 784 to 2 using four techniques (from left to right): t-SNE, UMAP, LargeVis, and Trimap

Visualization Program Runtimes				
	t-SNE	UMAP	LargeVis	Trimap
MNIST Digits Actual	129:40.967	5:06.650	9:35.643	1:39.526
Fashion MNIST Actual	132:09.443	5:42.680	9:57.080	2:08.569
Theoretical Runtime	$O(N^2)$	$O(N^{1.14})$	$O(smN)$	$O(N)$

4.3.1 Runtime and Memory Consumption

To begin, we can see that all tested runtimes correspond with their respective theoretical runtimes, indicated in the bottom column of Table (1) for each algorithm. This tells us that the experiment ran as expected by the authors of all algorithms and is an accurate assessment of algorithmic runtime. Assessing the runtimes for all algorithms on the whole, we can see a general trend that the MNIST Digits dataset has a lower runtime for embedding than the Fashion MNIST dataset. This also follows our expectations, as the original intent of the Fashion MNIST dataset is intended and to be a more modern, drop-in replacement for the widely used, but dated, MNIST Digits dataset.

With the original implementation of t-SNE taking the longest out of all the algorithms, running for more than 2 hours on both datasets, it does not provide a realistic runtime for datasets of great size. Though the version of t-SNE used in this experiment was the slowest of all, there are various other implementations of the algorithm that significantly improve its runtime, such as multi-core Barnes-Hut t-SNE, CUDA t-SNE, and

[10]`https://github.com/eamid/trimap`

[11]`https://scikit-learn.org/stable/modules/generated/sklearn.metrics.silhouette_score.html`

[12]`https://scikit-learn.org/stable/modules/generated/sklearn.metrics.davies_bouldin_score.html`

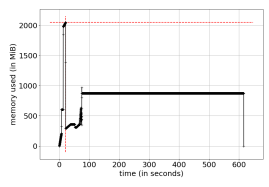
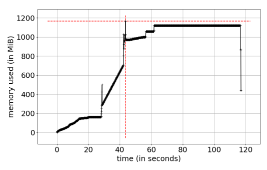
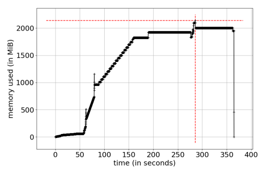
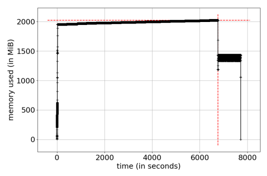
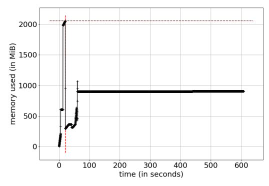
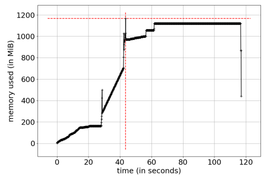
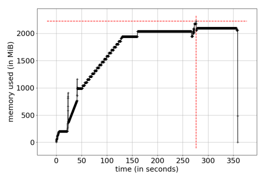
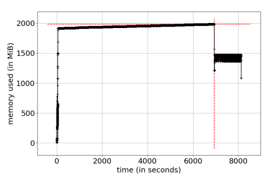
anchored t-SNE, decreasing runtime by up to 700 times (Van Der Maaten, 2014; Chan et al., 2018; Fu et al., 2019).

The next fastest algorithm for both data sets was LargeVis, with a runtime of approximately 10 minutes. Though this algorithm did not provide the quickest results, its runtime show that it can scale well to larger data-sets. Compared to t-SNE, the runtime for LargeVis is around 13 times lower. This is the only major jump that we see between the algorithms in terms of runtimes, with the remaining algorithms only providing minor speedups.

UMAP was the second quickest algorithm tested, with its unique approach of applying topology mathematics to dimensional reduction resulting in a runtime of less than 6 minutes for both data-sets. With this computational efficiency, UMAP can easily run on datasets over 70,000 data points in a realistic amount of time.

Finally, TriMap was the fastest of all algorithms by a significant margin, with it outperforming all other algorithms in terms of runtime by at least 2 times and, when compared to t-SNE, up to 60 times. With runtimes of around 2 minutes on both data-sets and a linear computational complexity, it can certainly scale far beyond the tested 70,000 member data-sets it was tested on without any trouble.

Table (2) Memory consumption graphs for the two varieties of MNIST datasets, Fashion MNIST and the MNIST Digits. The values obtained show the memory consumption for reducing the dimensionality of the datasets from 784 to 2 using four techniques (from left to right): LargeVis, TriMap, UMAP and t-SNE

	LargeVis	TriMap	UMAP	t-SNE
MNIST Digits				
Fashion MNIST				

Looking at the memory used for t-SNE, we can see that for both data-sets, the memory usage holds at around 2 gigabytes for the majority of the runtime, only dropping down to approximately 1.5 gigabytes in the last fifth of the runtime. With this memory usage, t-SNE has the highest sustained memory consumption of all algorithms tested, making it less useful in systems with a more limited capacity, or for very large datasets. Though this is true, it also has the most predictable and stable memory usage, as compared to the other algorithms, plateauing almost immediately at a quite constant memory usage, until dropping down slightly towards the end of the test.

For LargeVis, we see a peak memory usage of around 2 gigabytes occurring at the very start of the algorithm. Memory usage then quickly drops down and normalizes to a consistent value slightly less than 1 gigabyte. With a constant memory usage for the majority of its runtime, LargeVis is the second most stable algorithm according to memory usage.

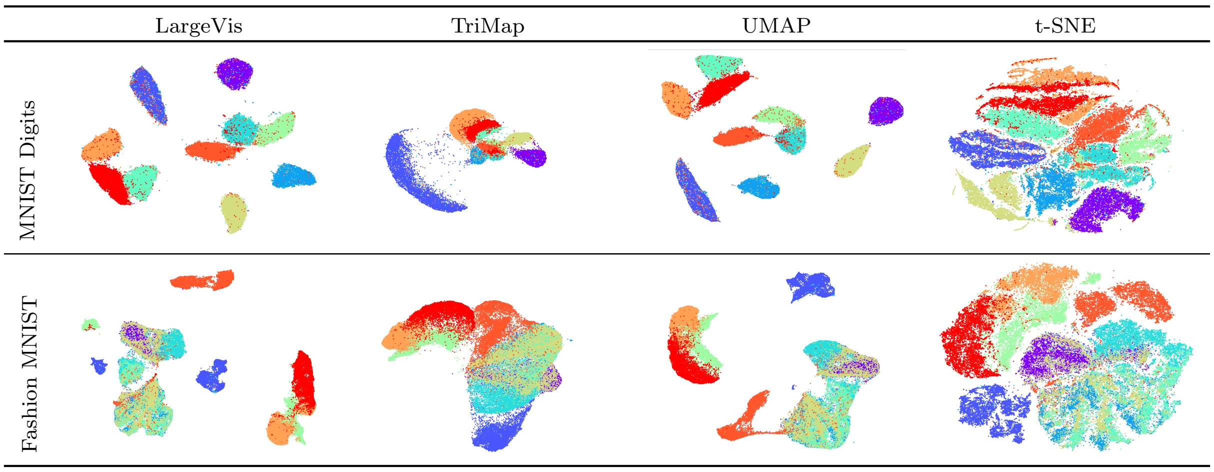
With a peak memory usage over 2 gigabytes, UMAP has the highest peak memory usage out of all algorithms tested. Furthermore, its memory usage is quite unstable, which several drops in usage paired with a series of sporadic climbs, only somewhat stabilizing at around half way through its runtime.

Finally, with a peak memory usage of about 1.2 gigabytes, TriMap is not only the quickest, but also the most memory efficient algorithm, allowing it to scale up to larger datasets without the need for a significantly larger amount of memory. However, the stability of its memory usage is among the worst of the algorithms tested, with it being comparable to that of UMAP with its frequent spikes in usage.

4.3.2 Qualitative Comparison

In Table 3, we present the final results of the each embedding algorithm. Comparing the quality of the embeddings between the datasets, we can see on the whole that the Fashion MNIST dataset is not as well separated between clusters as the MNIST Digits dataset, which is expected given the higher difficulty of the Fashion datasets. In terms of each specific algorithm, we can see similar qualities in the embeddings by LargeVis and UMAP, which tend to separate out the clusters more so than the other algorithms. This lends itself to the

Table (3) 2D representations of the two varieties of MNIST datasets, Fashion MNIST and the MNIST Digits. The 2D representations are obtained by reducing the dimensionality of the datasets from 784 to 2 using four techniques (from left to right): LargeVis, TriMap, UMAP and t-SNE



claim of global structure preservation presented by these algorithms. Looking at the results of TriMap, they are quite distinct from all the other algorithms, with clusters tending to concentrate about a centroid. Finally, we have t-SNE, which though the oldest of all the algorithms, seems to produce the most consistent results, with defined and well distributed clusters.

Table (4) Results of quantitative analysis of reduced dimensionality embeddings, focusing on the quality of certain clusters characteristics, as well as the preservation of neighborhood structure

Visualization Program Quality Measures					
		t-SNE	UMAP	LargeVis	Trimap
MNIST Digits	Silhouette	-0.04781	0.11363	0.05754	-0.01781
	Davies-Bouldin	2.92073	2.14496	78.92549	2.34602
	Trustworthiness	0.66168	0.71668	0.69962	0.67551
	Global Score	0.95202	0.94188	0.93215	0.94389
Fashion MNIST	Silhouette	-0.11374	-0.04145	-0.11132	-0.00672
	Davies-Bouldin	6.38772	6.22085	11.15644	2.97314
	Trustworthiness	0.66047	0.74379	0.66200	0.70229
	Global Score	0.80805	0.81065	0.68861	0.83558

4.3.3 Quantitative Comparison

To conduct a quantitative comparison of the final visualizations shown in Table 3, we will use the Davies-Bouldin and Silhouette scores as an assessment of clustering quality, and the Trustworthiness and Global Score metrics, as implemented in by the Trimap Python package(Amid and Warmuth, 2019). Of important note is the fact that the Davies-Bouldin and Silhouette scores do not assess whether or not the retrieval of information from the clusters is accurate, but rather, the inherent qualities of the clusters as they are in each embedding. We present these quantitative results in Table 4.

Beginning with the results of our clustering analysis, we were able to find quite consistent results between datasets. For the older MNIST Digits dataset, UMAP provided the best scores for both the Silhouette and Davies-Bouldin metrics, outperforming the other algorithms by a fair margin. For the newer Fashion MNIST Digits dataset, Trimap provided the best scores for both the Silhouette and Davies-Bouldin metrics, with other algorithms performing similarly by the Silhouette score, but much worse by the Davies-Bouldin index.

Highlighting the results of neighborhood structure preservation, we found that, for both datasets, UMAP was the most adept and maintaining the local structure of our datasets, as evidenced by its maximal Trustworthiness scores. However, in terms of the preservation of global structure, we saw superior performance from Trimap on the more challenging Fashion MNIST dataset, while for the older MNIST Digits dataset, global scores were all within 2 percentage points of one another, with t-SNE edging out Trimap just barely.

Table (5) Runtimes (seconds) of different graph visualization methods for different graphs shown in Table 6. BL - BatchLayout, BLBH - Barnes-Hut based BatchLayoout, FA2BH - Barnes-Hut based ForceAtlas2, OO - OpenOrd.

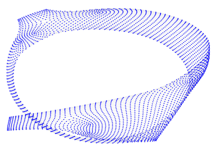
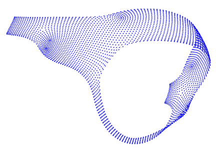
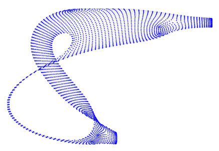
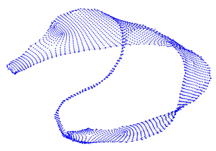
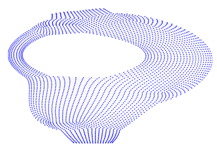
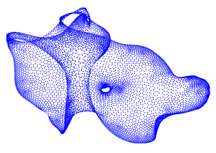
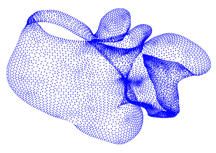
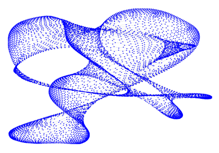
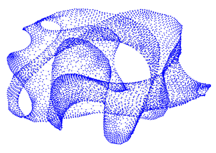
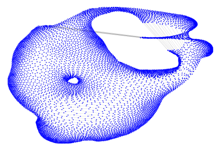
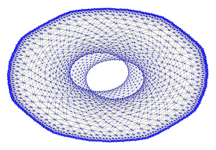
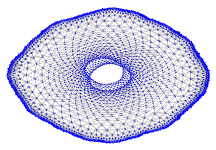
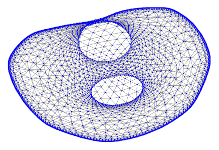
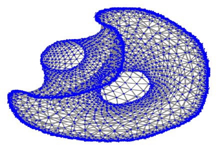
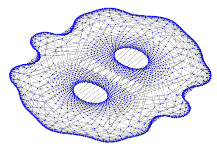
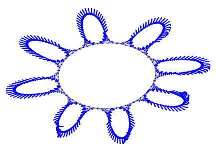
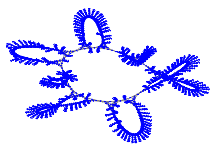
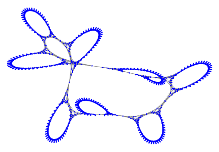
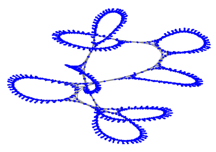
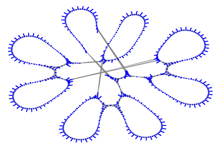
Graphs	#Vertices	BL	BLBH	FA2BH	OO	tsNET
grid2_dual	3,136	**3.31**	4.21	6.30	5.90	632.92
3elt_dual	9,000	16.67	**9.25**	19.40	16.50	3,396.66
tube2	21,498	81.23	**35.17**	60.27	43.69	32,162.03
finance256	37,376	234.66	**47.57**	81.54	69.19	163,532.71
Theoretical Runtime		$O(n^2)$	$O(n \log n)$	$O(n \log n)$	$O(n^2)$	$O(n^2)$

4.4 Comparison of Graph Visualization Tools

4.4.1 Runtime and Memory Consumption

With the emergence of bigdata, large-scale graph drawing has become an important aspect. To big graphs having the number of vertices more than one million, there exist several methods in the literature. Gephi has ForceAtlas2 model which can utilize the available cores in machine. OpenOrd also runs in parallel. Recently, BatchLayout method has been proposed in the literature to effectively utilize cache memory and speedup force computation in shared memory architecture. On the other hand, tsNET generates good quality layout but runs sequentially. Thus, it can not generate layouts for large graphs. We report the runtime of four methods in Table 5 along with their theoretical complexity. We observe that one version of BatchLayout always runs faster than other methods. For a graph having 9,000 vertices, tsNET takes around an hour to generate a layout whereas BatchLayout Barnes-Hut (BLBH) version can generate good quality layout within 10 seconds. For finance256 graph, tsNET took almost 2 days to generate a layout whereas BLBH can generate a readable layout within a minute. BatchLayout and OpenOrd consume less memory than Gephi's ForceAtlas2 and tsNET. BatchLayout uses compressed sparse row (CSR) representation of graph structure which significantly reduces the memory consumption cost for scale-free networks. On the other hand, Gephi's GML format consumes significant memory due to storing different node-link related information. Specifically, for grid2_dual graph, BatchLayout, OpenOrd, ForceAtlas2, and tsNET consume 2.6MB, 17MB, 632MB, and 710MB, respectively. Thus, for low memory consumption and runtime, BatchLayout is a probable choice to generate graph layouts.

Table (6) 2D layouts of four representative graphs with each graph shown in a row of the table. 2D layouts are obtained by embedding graphs directly on 2D space using five techniques (from left to right): BatchLayout, BatchLayoutBH, ForceAtlas2BH, OpenOrd and tsNET. BatchLayoutBH - Barnes-Hut approach of BatchLayout, ForceAtlas2BH - Barnes-Hut approach of ForceAtlas2.

	BatchLayout	BatchLayoutBH	ForceAtlas2BH	OpenOrd	tsNET
grid2_dual					
3elt_dual					
tube2					
finance256					

4.4.2 Qualitative Comparison

We show the quality of the generated layouts by different tools in Table 6. We observe that the layouts generated by tsNET is the most readable than others. The quality of layouts generated by BatchLayout are close to tsNET. Note that BatchLayout generates these layouts within a very short time compared to tsNET. The layouts of other methods are also comparable. Different people may have different perception for visual quality. Thus, in the next section, we also analyze the generated layouts using quantitative measures.

Table (7) Edge crossing and edge uniformity measures of different graph visualization methods for two representative graphs shown in Table 6. A lower value is better for both edge crossing and edge uniformity measures. BL - BatchLayout, BLBH - Barnes-Hut based BatchLayoout, FA2BH - Barnes-Hut based ForceAtlas2, OO - OpenOrd.

Graphs	Edge Crossing↓					Edge Uniformity↓				
	BL	BLBH	FA2BH	OO	tsNET	BL	BLBH	FA2	OO	tsNET
grid2_dual	1,043	515	596	2,549	0	0.40	0.35	0.58	0.55	0.26
3elt dual	2,400	3,851	5,385	7,354	692	0.41	0.42	0.64	0.53	0.62

Table (8) Stress and Neighborhood preservation measures of different graph visualization methods for two representative graphs shown in Table 6. A lower value is better for stress measure whereas a higher value is better for Neighborhood preservation measure. BL - BatchLayout, BLBH - Barnes-Hut based BatchLayoout, FA2BH - Barnes-Hut based ForceAtlas2, OO - OpenOrd.

Graphs	Stress↓					Neighborhood Preservation↑				
	BL	BLBH	FA2	OO	tsNET	BL	BLBH	FA2	OO	tsNET
grid2_dual	4.1E+5	2.4E+5	4.3E+5	4.5E+5	1.4E+5	0.50	0.60	0.53	0.43	0.73
3elt dual	3.7E+6	6.1E+6	6.6E+6	7.5E+6	3.1E+6	0.55	0.51	0.37	0.36	0.64

4.4.3 Quantitative Comparison

To assess the quantitative measure of visualization shown in Table 6, we choose grid2_dual and 3elt_dual graphs. We use four graph visualization measures (De Luca et al., 2019; Rahman et al., 2020a), namely, (i) the number of edge crossing, (ii) edge length uniformity, (iii) stress, and (iv) neighborhood preservation. A lower value means a better results for edge crossing, edge length uniformity and stress measures. On the other hand, a higher value means a better result for neighborhood preservation measure. All these measures are widely used in the literature to assess the layouts quantitatively. We report the results in Tables 7 and 8. We observe that tsNET wins in edge crossing, stress and neighborhood measures. One version of BatchLayout always shows runnerup results. For edge uniformity measure, BatchLayout shows competitive results. BatchLayout shows this competitive performance with a very low runtime compared to tsNET.

5 Discussions

Taking into account all of the data and analysis, we can see that all of the algorithms tested have certain strengths and weaknesses. With this, we can find a use case for all of the algorithms.

High-dimensional Vector Data. With the focus on solving the crowding-out problem, t-SNE is the best choice for reducing data-sets where a more uniformly distributed embedding is desired. Furthermore, with the most stable memory usage among algorithms, it is well applied to use cases where predictable memory usage is favorable. However, we recommend that the original serial version of t-SNE is not used, as it fails to scale to large data-sets, with its high memory consumption and high computational complexity. Therefore it is recommended that if it is used, that one of its more optimized versions be used. Being based upon t-SNE with the same KNN tree construction, LargeVis shares with it certain characteristics, such as it shows more stable memory usage. However, LargeVis is significantly less computationally complex than t-SNE, allowing it to scale to data-sets with greater data points. With the focus of LargeVis being to improve runtime as compared to t-SNE, it certainly delivers. Therefore, we believe that LargeVis should be used for applications where stable memory usage, high scalablity, and the preservation of the data's local structure are desired. Moving on to UMAP, we are given faster runtime than both t-SNE and LargeVis, but at the cost of less stable memory usage. Providing similar embedding quality as compare to LargeVis, UMAP should be used as an alternative for LargeVis use cases when the stability of memory is not a factor, as it provides similar results in nearly half the time. Finally, we are left with the fastest and most memory efficient of all algorithms tested, that being TriMap. With its significantly lower runtimes and memory usage TriMap certainly outclasses all other tested algorithms in all tests. However, TriMap is somewhat limited by its design specifications, as it is primarily geared towards maintaining the global structure of data, rather than local structure. With relatively unstable memory usage, the predictability of TriMap is less than that of algorithms like LargeVis and t-SNE. Therefore, we believe that the best use cases for TriMap are whenever maintaining the global structure of data is the goal or the size of the data-set is too great for any other algorithm to realistically process.

Graph Data. For graph data visualization, tsNET methods take a long time to generate a layout of a medium-scale graph and it also consumes a high amount of memory. Nevertheless, the layouts generated by tsNET are of good quality. If users have enough time and memory, then tsNET can be used to generate a good quality layout. OpenOrd is a high-performance graph visualization tool that can be run on a distributed system to generate the layout of a big graph. Thus, when we need to generate graph layout quickly, we can use OpenOrd though the quality may not be good. ForceAtlas2 is integrated with Gephi software which is user-friendly and scientists/researchers from other research domains can also use ForceAtlas2 without coding skill. Due to maintaining the structure of the Gephi framework, this method consumes a high amount of memory which reduces its usability on a large graph. Finally, the BatchLayout method generates good quality layout very quickly compared to other methods. It can also generate layouts within a reasonable time for graphs having millions of vertices. Thus, users can also consider using BatchLayout which simultaneously produces good quality layouts within a short time consuming less memory.

6 Conclusion and Future Work

Data visualization provides an important insight about the high-dimensional data in low-dimensional space. This is a precursor for data-mining and information retrieval. In this survey paper, we analyzed several state-of-the-art visualization techniques. To compare the results, we mainly focused on runtime, memory consumption and visual quality using benchmark datasets. From our analysis, we can recommend users to chose a visualization method for a given set of computing resources.

With the breadth and extensibility of established algorithms such as t-SNE and UMAP, the results for outputs can vary greatly based on the tuning modifications, making it a component worth investigating further. The quality of graph layouts can also vary based on the selection of different hyper-parameters where users should be careful and follow the guidelines of parameter sensitivity in the papers. Finally, an extension of our analysis to include modified implementations of algorithms already discussed, such as those of t-SNE and BatchLayout, may provide more depth to the strength of using more established algorithms.

References

Amid E, Warmuth MK (2019) Trimap: Large-scale dimensionality reduction using triplets 1910.00204

Bagger FO, Sasivarevic D, Sohi SH, Laursen LG, Pundhir S, Sønderby CK, Winther O, Rapin N, Porse BT (2016) Bloodspot: a database of gene expression profiles and transcriptional programs for healthy and malignant haematopoiesis. Nucleic acids research 44(D1):D917–D924

Barnes J, Hut P (1986) A hierarchical o (n log n) force-calculation algorithm. nature 324(6096):446–449

Bastian M, Heymann S, Jacomy M, et al. (2009) Gephi: an open source software for exploring and manipulating networks. Icwsm 8(2009):361–362

Becht E, Dutertre CA, Kwok IWH, Ng LG, Ginhoux F, Newell EW (2018) Evaluation of umap as an alternative to t-sne for single-cell data. bioRxiv DOI 10.1101/298430, URL https://www.biorxiv.org/content/early/2018/04/10/298430, https://www.biorxiv.org/content/early/2018/04/10/298430.full.pdf

Becht E, McInnes L, Healy J, Dutertre CA, Kwok IW, Ng LG, Ginhoux F, Newell EW (2019) Dimensionality reduction for visualizing single-cell data using umap. Nature biotechnology 37(1):38–44

Blomqvist K, Kaski S, Heinonen M (2018) Deep convolutional gaussian processes. 1810.03052

Brinkmann GG, Rietveld KF, Takes FW (2017) Exploiting gpus for fast force-directed visualization of large-scale networks. In: 2017 46th International Conference on Parallel Processing (ICPP), IEEE, pp 382–391

Chan DM, Rao R, Huang F, Canny JF (2018) t-sne-cuda: Gpu-accelerated t-sne and its applications to modern data. 1807.11824

Charikar MS (2002) Similarity estimation techniques from rounding algorithms. In: Proceedings of the Thiry-Fourth Annual ACM Symposium on Theory of Computing, Association for Computing Machinery, New York, NY, USA, STOC '02, p 380–388, DOI 10.1145/509907.509965, URL https://doi.org/10.1145/509907.509965

Chen Y, Guan Z, Zhang R, Du X, Wang Y (2019) A survey on visualization approaches for exploring association relationships in graph data. Journal of Visualization 22(3):625–639

Chimani M, Gutwenger C, Jünger M, Klau GW, Klein K, Mutzel P (2013) The open graph drawing framework (ogdf). Handbook of graph drawing and visualization 2011:543–569

Cook J, Sutskever I, Mnih A, Hinton G (2007) Visualizing similarity data with a mixture of maps. In: Meila M, Shen X (eds) Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics, PMLR, San Juan, Puerto Rico, Proceedings of Machine Learning Research, vol 2, pp 67–74, URL http://proceedings.mlr.press/v2/cook07a.html

Datar M, Immorlica N, Indyk P, Mirrokni VS (2004) Locality-sensitive hashing scheme based on p-stable distributions. In: Proceedings of the Twentieth Annual Symposium on Computational Geometry, Association for Computing Machinery, New York, NY, USA, SCG '04, p 253–262, DOI 10.1145/997817.997857, URL https://doi.org/10.1145/997817.997857

Davies DL, Bouldin DW (1979) A cluster separation measure. IEEE Transactions on Pattern Analysis and Machine Intelligence PAMI-1(2):224–227, DOI 10.1109/TPAMI.1979.4766909

De Luca F, Hossain I, Gray K, Kobourov S, Börner K (2019) Multi-level tree based approach for interactive graph visualization with semantic zoom. arXiv preprint arXiv:190605996

Diaz-Papkovich A, Anderson-Trocmé L, Gravel S (2018) Revealing multi-scale population structure in large cohorts. bioRxiv DOI 10.1101/423632, URL https://www.biorxiv.org/content/early/2018/09/23/423632, https://www.biorxiv.org/content/early/2018/09/23/423632.full.pdf

Dong W, Moses C, Li K (2011) Efficient k-nearest neighbor graph construction for generic similarity measures. In: Proceedings of the 20th International Conference on World Wide Web, Association for Computing Machinery, New York, NY, USA, WWW '11, p 577–586, DOI 10.1145/1963405.1963487, URL https://doi.org/10.1145/1963405.1963487

Ellson J, Gansner E, Koutsofios L, North SC, Woodhull G (2001) Graphviz—open source graph drawing tools. In: International Symposium on Graph Drawing, Springer, pp 483–484

Erdös P, Harary F, Tutte WT (1965) On the dimension of a graph

Escolano C, Costa-jussà MR, Fonollosa JAR (2018) (self-attentive) autoencoder-based universal language representation for machine translation. 1810.06351

Fruchterman TM, Reingold EM (1991) Graph drawing by force-directed placement. Software: Practice and experience 21(11):1129–1164

Fu C, Zhang Y, Cai D, Ren X (2019) Atsne: Efficient and robust visualization on gpu through hierarchical optimization. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp 176–186

Fuhrimann L, Moosavi V, Ohlbrock PO, Dacunto P (2018) Data-driven design: Exploring new structural forms using machine learning and graphic statics. 1809.08660

Gionis A, Indyk P, Motwani R (1999) Similarity search in high dimensions via hashing. In: Proceedings of the 25th International Conference on Very Large Data Bases, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, VLDB '99, p 518–529

Goerss P, Jardine J (1999) Simplicial Homotopy Theory. Progress in mathematics (Boston, Mass.) v. 174, Springer, URL https://books.google.com/books?id=xFwXQCtNcUoC

Gove R (2019) A random sampling o (n) force-calculation algorithm for graph layouts. In: Computer Graphics Forum, Wiley Online Library, vol 38, pp 739–751

Hinton GE, Roweis ST (2003) Stochastic neighbor embedding. In: Becker S, Thrun S, Obermayer K (eds) Advances in Neural Information Processing Systems 15, MIT Press, pp 857–864, URL http://papers.nips.cc/paper/2276-stochastic-neighbor-embedding.pdf

Hotelling H (1933) Analysis of a complex of statistical variables into principal components. Journal of Educational Psychology 24:498–520

Hu Y (2005) Efficient, high-quality force-directed graph drawing. Mathematica journal 10(1):37–71

Jacomy M, Venturini T, Heymann S, Bastian M (2014) Forceatlas2, a continuous graph layout algorithm for handy network visualization designed for the gephi software. PloS one 9(6):e98679

Kobak D, Berens P (2019) The art of using t-sne for single-cell transcriptomics. Nature communications 10(1):1–14

Kobourov SG (2012) Spring embedders and force directed graph drawing algorithms. arXiv preprint arXiv:12013011

Kruiger JF, Rauber PE, Martins RM, Kerren A, Kobourov S, Telea AC (2017) Graph layouts by t-sne. In: Computer Graphics Forum, Wiley Online Library, vol 36, pp 283–294

Li X, Dyck OE, Oxley MP, Lupini AR, Healy J, McInnes L, Jesse S, Kalinin S (2018) Dataset of the paper "Manifold Learning of Four-dimensional Scanning Transmission Electron Microscopy " DOI 10.6084/m9.figshare.7416317.v1, URL https://figshare.com/articles/Dataset_of_the_paper_Manifold_Learning_of_Four-dimensional_Scanning_Transmission_Electron_Microscopy_/7416317

Liu S, Cui W, Wu Y, Liu M (2014) A survey on information visualization: recent advances and challenges. The Visual Computer 30(12):1373–1393

Maaten Lvd, Hinton G (2008) Visualizing data using t-sne. Journal of machine learning research 9(Nov):2579–2605

Martin S, Brown WM, Klavans R, Boyack KW (2011) Openord: an open-source toolbox for large graph layout. In: Visualization and Data Analysis 2011, International Society for Optics and Photonics, vol 7868, p 786806

May J (1992) Simplicial Objects in Algebraic Topology. Chicago Lectures in Mathematics, University of Chicago Press, URL `https://books.google.com/books?id=QGjwV0gyQnIC`

McInnes L, Healy J, Melville J (2018) Umap: Uniform manifold approximation and projection for dimension reduction. arXiv preprint arXiv:180203426

Pinsky LE, Wipf JE (2000) A picture is worth a thousand words. Journal of general internal medicine 15(11):805–810

Rahman MK, Sujon MH, Azad A (2020a) BatchLayout: A batch-parallel force-directed graph layout algorithm in shared memory. In: Proceedings of PacificVis

Rahman MK, Sujon MH, Azad A, et al. (2020b) Force2Vec: Parallel force-directed graph embedding. In: Proceedings of ICDM

Rousseeuw PJ (1987) Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. Journal of Computational and Applied Mathematics 20:53 – 65, DOI https://doi.org/10.1016/0377-0427(87)90125-7, URL `http://www.sciencedirect.com/science/article/pii/0377042787901257`

Rydning DRJGJ (2018) The digitization of the world from edge to core. Framingham: International Data Corporation

Tang J, Qu M, Wang M, Zhang M, Yan J, Mei Q (2015) Line: Large-scale information network embedding. In: Proceedings of the 24th International Conference on World Wide Web, International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE, WWW '15, p 1067–1077, DOI 10.1145/2736277.2741093, URL `https://doi.org/10.1145/2736277.2741093`

Tang J, Liu J, Zhang M, Mei Q (2016) Visualizing large-scale and high-dimensional data. In: Proceedings of the 25th international conference on world wide web, pp 287–297

Tsitsulin A, Mottin D, Karras P, Müller E (2018) VERSE: Versatile graph embeddings from similarity measures. In: Proceedings of the 2018 world wide web conference, pp 539–548

Van Der Maaten L (2014) Accelerating t-sne using tree-based algorithms. The Journal of Machine Learning Research 15(1):3221–3245

Van Der Maaten L, Postma E, Van den Herik J (2009) Dimensionality reduction: a comparative. J Mach Learn Res 10(66-71):13

Wattenberg M, Viégas F, Johnson I (2016) How to use t-sne effectively. Distill DOI 10.23915/distill.00002, URL `http://distill.pub/2016/misread-tsne`

Yianilos PN (1993) Data structures and algorithms for nearest neighbor search in general metric spaces. In: Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, Society for Industrial and Applied Mathematics, USA, SODA '93, p 311–321