A Survey on Recent Trends in High Dimensional Data Visualization

Alexander Kiefer, M. Khaledur Rahman

January 2020

1 Introduction

As technology has progress through the 21st century, the importance of the data it produces has become increasingly essential. With business entities paying and selling data sets and information with an ever increasing fervor, technology has become dependent upon the data it produces in order to progress. Underlying this argument is the crux of why the field of data science has grown at such a quick rate, and that is the difficulty associated with the interpretation and understanding of the raw data produced by technology. Data and graph 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 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[20]. In this paper, of the algorithms discussed, TriMap falls into the first category, while UMAP, LargeVis, and t-SNE fall into the second. The focus of our paper will be on the effectiveness and applications of each of these algorithms.

In close association with this process, graph visualization is a method by which data of low dimensionality is used to generate visual interpretations of graphs or networks, making the relationships within the data more easily understood by people. In this survey, force-directed algorithms will be the primary focus in terms of graph visualizations [21].

2 Algorithms

2.1 Graph Visualization

2.1.1 OpenOrd

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 and coarsened for the next 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. It optimizes the objective function using simulated annealing heuristic approach. It has parallel implementation and several other options like edge cutting is to visualize the densely connected graph. OpenOrd has some drawbacks which are listed below:

- One of the drawbacks of this approach is that it uses an adjacency matrix for graph representation.
- Parallel version has another drawback, for multiple cores, several runs results in several layouts which do not preserve consistency as required in scientific computing.

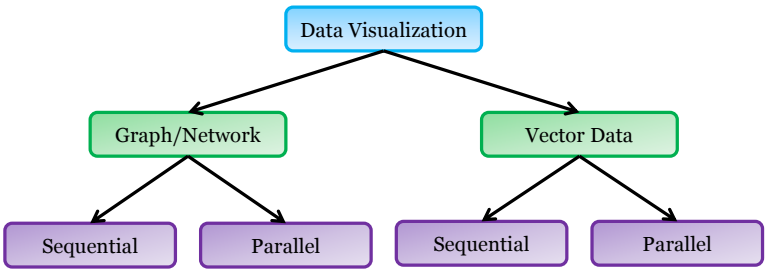


Figure (1) Caption

2.1.2 tsNET

tsNET [16] is a version of t-SNE which has been formulated for graph layout problem. Attractive force is computed using similar approach to t-SNE. First graph theoretic shortest path distance is calculated for all vertices. Then joint probability between adjacent vertices is calculated using t-SNE approach. For repulsive force, logarithmic difference between non-adjacent vertices is computed for all vertices and then add average of it optimization function. The optimization function of tsNET is given below:

$C = \lambda_{aKL} + \lambda_c \frac{1}{N} \sum_{i,j \in V} \parallel y_i - y_j \parallel + \lambda_r \frac{1}{N} \sum_{i,j \in V} \log(||y_i - y_j|| + \epsilon)$

The major disadvantage of this method is slower performance and choosing a good value for perplexity parameter is tricky. Sometimes, this does not converge which results in a unreadable layout.

2.2 Vector Data Visualization

2.2.1 t-SNE [18]

Developed in 2008, t-SNE is the progenitor of most modern dimensionality reducing graph visualization 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 [14], 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 graph 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 the equation

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

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

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

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 minimizes 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 of this method is given by the model

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

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 [6] is much simpler, decreasing computation time.

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

2.2.2 LargeVis [22]

Developed in 2016, LargeVis aims to solve the problem of the high computational cost associated with dimensionality reduction in methods such as t-SNE. To give a very general overview of the LargeVis program, 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 understood by the viewer. 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 $O(smN)$, where s is the dimension of the low dimensional space, m is the number of negative samples, and N is the number of points in the dataset.

One of the primary challenges faced with dimensional reductions is the construction of the K-nearest neighbor tree, which from now on will be referred to as the KNN graph. In t-SNE, vantage point trees [24] are used to construct the KNN 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 KNN can be modeled by the formula $O(N^2d)$, 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 [5, 7, 12], and nearest neighbor exploration [9]. 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 [7] 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 KNN graph is refined and improved over a certain amount of iterations. [1]

Going into greater detail, LargeVis begins by calculating the weights of the edges in the KNN tree using the same method as t-SNE, modeled by equation 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 KNN 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

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

where $\vec{y_i}$ is the low dimensional embedding of v_i and $\vec{y_j}$ of v_j. 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 found 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, 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 [23] with a probability proportional to their weights and then used as binary edges in the stochastic gradient descent.

2.2.3 UMAP [20]

Developed in 2018, UMAP, or Uniform Manifold Approximation and Projection, is a manifold learning algorithm used for dimensional reduction. The algorithm is rooted in Riemannian geometry and algebraic topology [19], 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 [2, 3, 8], material science [11, 17], and machine learning [10, 4]. To give a general overview of the UMAP algorithm, local manifold approximations are combined to find their local fuzzy simplicial set representations [13]. 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 [20].

To begin, UMAP constructs a weighted k-nearest neighbors graphs using an input data-set $X = \{x_i, ..., x_n\}$ and a dissimilarity measure $d : X \times X \to \mathbb{R}_{\geq 0}$. Using a given hyper-parameter k, the set of k nearest neighbors $\{x_{i1}, ..., x_{ik}\}$ is calculated according to the metric d using nearest neighbor descent [9].
Next, for every x_i, p_i and σ_i are calculated such that

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

and

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

To construct the weight 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

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

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 abitrarily 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.

2.2.4 TriMap [1]

Developed in 2019, TriMap is the most recent algorithm that we will be studying. 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 [15], 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

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

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

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

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

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

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 begin construction. Using a subset of all possible triplets (i, j, k). Finding $m = 10$ nearest neighbors for each point and sampling $m' = 5 triplets$ for every nearest neighbor, we are left with $m \times m' = 50$ nearest neighbor triplets per point. Adding $r = 5$ random triplets to ensure a more consistent distribution, the total triplets per point becomes $m \times m' + r = 55$. Utilizing the ANNOY library for the construction of the approximate nearest neighbors tree, the inital embedding is set to the PCA embedding and scaled to improve convergence. By doing this, the algorithm moves closer towards the goal of providing a better global structure in the embedding. With this, the final cost function is found to be

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

3 Experiment

3.1 Experiment Setup

For our experiment, we will be analyzing 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 will be using 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 will be using the TriMap implementation distributed on the Python Packed Index, which is sourced from the algorithms GitHub page [6].

We will be running 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 will be run using the default parameters for each respective algorithm. For our datasets, we will be using 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 will be measuring the runtime and memory usage of each algorithm. We will also consider the quality of the produced embedding, considering the quality of the clustering, which is the tendency for similar points to group together, and the delineation between clusters, which is tendency for clusters to be separate from other clusters (ie. not overlap).

3.2 Results

Visualization Program Quality Measures				
	TSNE	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	$O(N^2)$	$O(N^{1.14})$	$O(smN)$	$O(N)$

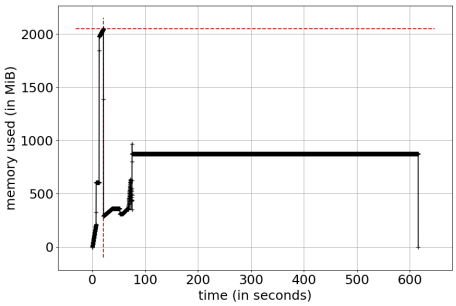
Table (1)

Visualization Program Quality Measures					
		TSNE	UMAP	LargeVis	Trimap
MNIST Digits	Silhouette	-0.04781	0.11363	0.05754	-0.01781
	Davies-Bouldin	2.92073	2.14496	78.92549	2.34602
Fashion MNIST	Silhouette	-0.11374	-0.04145	-0.11132	-0.00672
	Davies-Bouldin	6.38772	6.22085	11.15644	2.97314

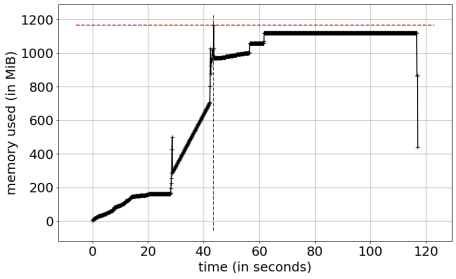
Table (2)

[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

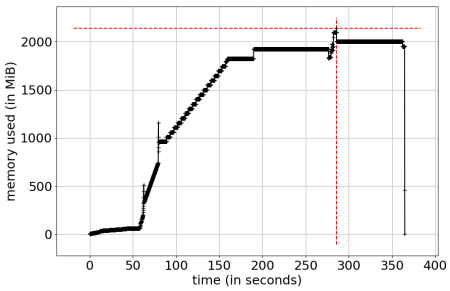
MNIST Memory Usage



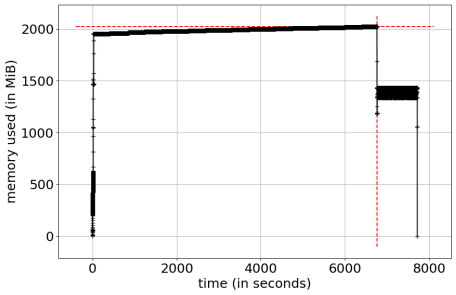
(a) LargeVis



(b) TriMap



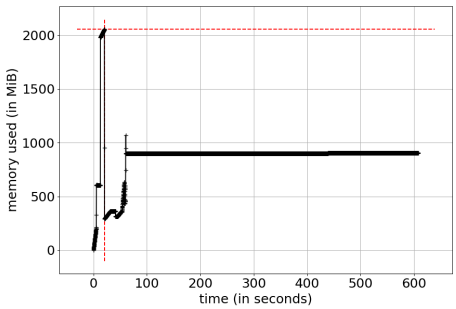
(c) UMAP



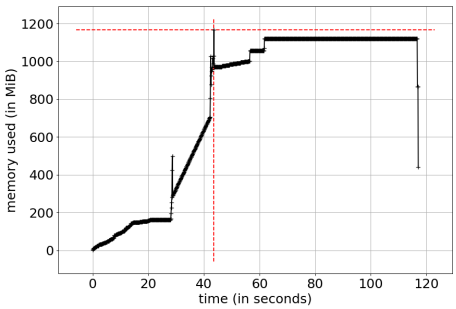
(d) TSNE

Figure (2)

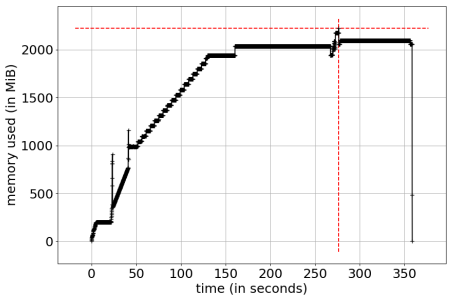
Fashion MNIST Memory Usage



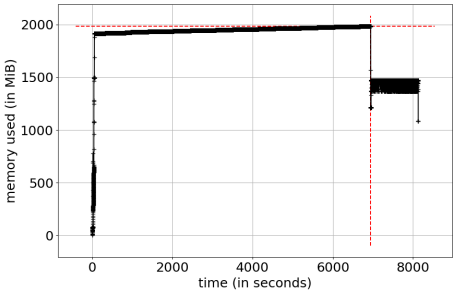
(a) Largevis



(b) TriMap



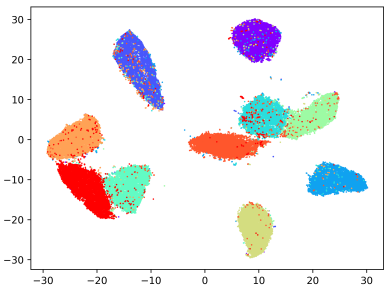
(c) UMAP



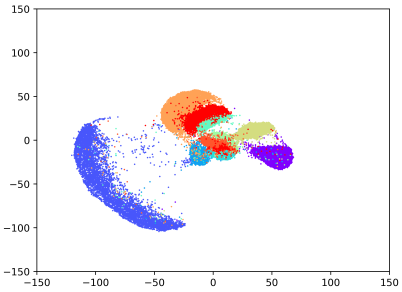
(d) TSNE

Figure (3)

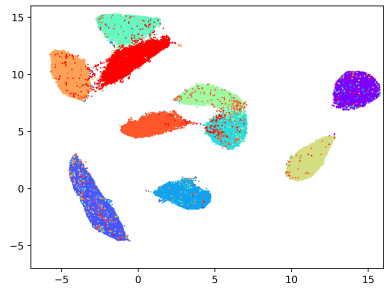
MNIST Digits 2D Embeddings



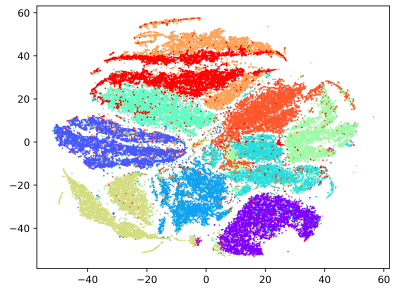
(a) Largevis



(b) TriMap



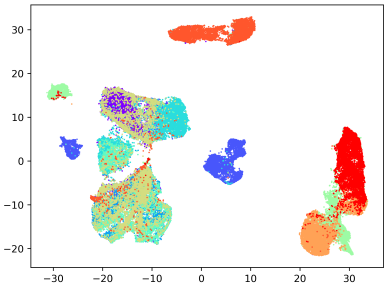
(c) UMAP



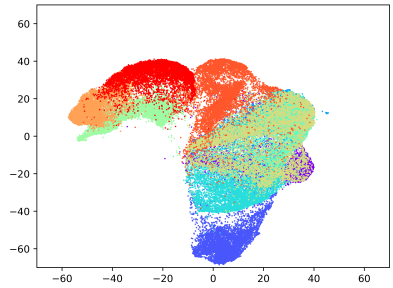
(d) TSNE

Figure (4)

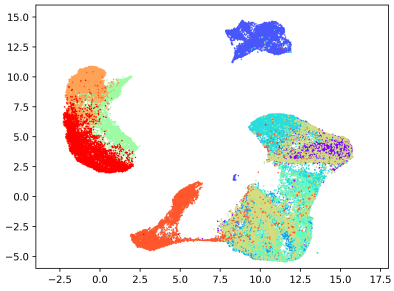
Fashion MNIST 2D Embeddings



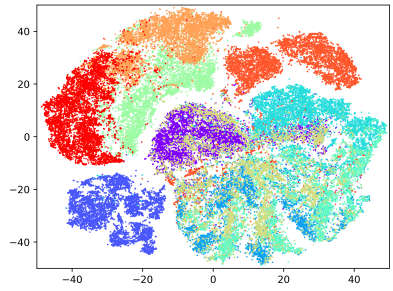
(a) Largevis



(b) TriMap



(c) UMAP



(d) TSNE

Figure (5)

3.3 Analysis

To begin, we can see that all actual 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 predictor of which algorithm will execute the fastest. This being said, we will analyze each individual algorithm according to their runtime, going from slowest to fastest.

3.3.1 t-SNE

With the serial version of t-SNE taking the longest out of all the algorithms, with a runtime of more than 2 hours on both the MNIST Digits and Fashion MNIST data-sets, it does not provide a realistic runtime for data-sets over 100000. 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 a multi-core Barnes-Hut t-SNE, CUDA t-SNE, and anchored t-SNE, decreasing runtime by up to 700 times.

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 1.5 gigabytes in the last fifth of the runtime. With this memory usage, t-SNE has the highest memory consumption of all algorithms tested, making it less useful in systems with a more limited capacity, though this, as before, can be mitigated by using other versions of the algorithm. Though this is true, it also has the most predictable and stable memory usage, as compared to the other algorithms.

Looking at the quality of the embeddings, for the MNIST digits data, the data is clustered very well, with the points being distributed more widely over the graph, as compared to the other embeddings, while still maintaining clear delineation between clusters. In the Fashion MNIST embeddings, we see similar qualities, with slightly less delineation among clusters than in the Digits data-sets. With this being said, it is clear that the goal of mitigating the crowding-out problem of dimensional reduction was successful for t-SNE.

3.3.2 LargeVis

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 reasonably to larger data-sets.

With a peak memory usage of around 2 gigabytes occurring at the very start of the algorithm, memory usage 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.

Finally, looking at the quality of the embeddings, we can see that for the MNIST Digits data-set, LargeVis provides very clear clustering among similar data points and a great amount of delineation between clusters. In the Fashion MNIST embedding, we also still see good clustering, however, with much less delineation between clusters, leading to a fair amount of overlap between data points.

3.3.3 UMAP

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.

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, assessing the quality of the embeddings generated by UMAP, we can see that on the MNIST Digits dataset that its output is comparable to that of LargeVis, with high quality clustering and delineation. Moving to the Fashion MNIST dataset, we see that, just like LargeVis, the quality of the embedding has decreased, with significantly more overlap and less delineation between clusters.

3.3.4 TriMap

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.

With a peak memory usage of about 1.2 gigabytes, TriMap is 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.

Finally, viewing the output embeddings, we can see that TriMap produces embeddings in both the MNIST Digits and Fashion MNIST data-sets that are quite different from all the others. This is most likely due to the fact that TriMap's main focus is maintaining the global structure of the data, as compared to LargeVis and t-SNE maintaining local structure and UMAP attempting to do find a medium between local and global structure. That being said TriMap produces embeddings for both data-sets which show clear clustering, with much less delineation between clusters than other algorithms and less overlap between data points.

4 Recommendations

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.

With its 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 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 to great for any algorithm to realistically process.

5 Future Work

The extent of this survey is limited in certain aspects and can definitely be improved in a few ways. One of these aspects is the scale of testing. Having only tested the algorithms on the MNIST Digits and Fashion MNIST datasets, we would like to include more data sets with greater variety and explore how various factors, such as original dimensionality and the number of data points, affect algorithm metrics. Some future aspects for analysis include the effect of algorithms' hyper-parameter settings on the quality and metrics of embeddings and the improvements and drawbacks of algorithm reimplementations, such as those of t-SNE.

References

[1] Ehsan Amid and Manfred K. Warmuth. Trimap: Large-scale dimensionality reduction using triplets. 2019.

[2] Frederik Otzen Bagger, Damir Sasivarevic, Sina Hadi Sohi, Linea Gøricke Laursen, Sachin Pundhir, Casper Kaae Sønderby, Ole Winther, Nicolas Rapin, and Bo T Porse. Bloodspot: a database of gene expression profiles and transcriptional programs for healthy and malignant haematopoiesis. *Nucleic acids research*, 44(D1):D917–D924, 2016.

[3] Etienne Becht, Charles-Antoine Dutertre, Immanuel W. H. Kwok, Lai Guan Ng, Florent Ginhoux, and Evan W. Newell. Evaluation of umap as an alternative to t-sne for single-cell data. *bioRxiv*, 2018.

[4] Kenneth Blomqvist, Samuel Kaski, and Markus Heinonen. Deep convolutional gaussian processes, 2018.

[5] Moses S. Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of the Thiry-Fourth Annual ACM Symposium on Theory of Computing*, STOC '02, page 380–388, New York, NY, USA, 2002. Association for Computing Machinery.

[6] James Cook, Ilya Sutskever, Andriy Mnih, and Geoffrey Hinton. Visualizing similarity data with a mixture of maps. In Marina Meila and Xiaotong Shen, editors, *Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics*, volume 2 of *Proceedings of Machine Learning Research*, pages 67–74, San Juan, Puerto Rico, 21–24 Mar 2007. PMLR.

[7] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the Twentieth Annual Symposium on Computational Geometry*, SCG '04, page 253–262, New York, NY, USA, 2004. Association for Computing Machinery.

[8] Alex Diaz-Papkovich, Luke Anderson-Trocmé, and Simon Gravel. Revealing multi-scale population structure in large cohorts. *bioRxiv*, 2018.

[9] Wei Dong, Charikar Moses, and Kai Li. Efficient k-nearest neighbor graph construction for generic similarity measures. In *Proceedings of the 20th International Conference on World Wide Web*, WWW '11, page 577–586, New York, NY, USA, 2011. Association for Computing Machinery.

[10] Carlos Escolano, Marta R. Costa-jussà, and José A. R. Fonollosa. (self-attentive) autoencoder-based universal language representation for machine translation, 2018.

[11] Lukas Fuhrimann, Vahid Moosavi, Patrick Ole Ohlbrock, and Pierluigi Dacunto. Data-driven design: Exploring new structural forms using machine learning and graphic statics, 2018.

[12] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Similarity search in high dimensions via hashing. In *Proceedings of the 25th International Conference on Very Large Data Bases*, VLDB '99, page 518–529, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[13] P.G. Goerss and J.F. Jardine. *Simplicial Homotopy Theory*. Progress in mathematics (Boston, Mass.) v. 174. Springer, 1999.

[14] Geoffrey E Hinton and Sam T. Roweis. Stochastic neighbor embedding. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15*, pages 857–864. MIT Press, 2003.

[15] Harold Hotelling. Analysis of a complex of statistical variables into principal components. *Journal of Educational Psychology*, 24:498–520, 1933.

[16] Johannes F Kruiger, Paulo E Rauber, Rafael Messias Martins, Andreas Kerren, Stephen Kobourov, and Alexandru C Telea. Graph layouts by t-sne. In *Computer Graphics Forum*, volume 36, pages 283–294. Wiley Online Library, 2017.

[17] Xin Li, Ondrej E. Dyck, Mark P. Oxley, Andrew R. Lupini, John Healy, Leland McInnes, Stephen Jesse, and Sergei Kalinin. Dataset of the paper "Manifold Learning of Four-dimensional Scanning Transmission Electron Microscopy ". 12 2018.

[18] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[19] J.P. May. *Simplicial Objects in Algebraic Topology*. Chicago Lectures in Mathematics. University of Chicago Press, 1992.

[20] Leland McInnes, John Healy, and James Melville. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*, 2018.

[21] Md. Khaledur Rahman, Majedul Haque Sujon, and Ariful Azad. Batchlayout: A batch-parallel force-directed graph layout algorithm in shared memory. 2020.

[22] Jian Tang, Jingzhou Liu, Ming Zhang, and Qiaozhu Mei. Visualizing large-scale and high-dimensional data. In *Proceedings of the 25th international conference on world wide web*, pages 287–297, 2016.

[23] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web*, WWW '15, page 1067–1077, Republic and Canton of Geneva, CHE, 2015. International World Wide Web Conferences Steering Committee.

[24] Peter N. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '93, page 311–321, USA, 1993. Society for Industrial and Applied Mathematics.