Improving Sampling-Based t-SNE Performance Using Dijkstra's Algorithm for Approximate Distance Computation

Filip Markov[1]

Supervisor(s): Klaus Hildebrandt[1]**, Martin Skrodzki**[1]

[1]**EEMCS, Delft University of Technology, The Netherlands**

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 22, 2025

Name of the student: Filip Markov
Final project course: CSE3000 Research Project
Thesis committee: Klaus Hildebrandt, Martin Skrodzki, Christoph Lofi

Abstract

T-SNE is widely used for visualising high-dimensional data in lower dimensions. To reduce the costs of parameter optimisation, t-SNE is performed on a sample of the original data. After sampling the points, the distances between them need to be calculated, which is expensive due to the high dimensionality of the data. We apply Dijkstra's algorithm to approximate the distances and similarities between the sampled points. This reduces the algorithm's execution time by up to 40% without degrading the quality of the projections.

1 Introduction

Data is typically represented as numerical matrices because that's how computers process information. But humans find it hard to get insight into high-dimensional data. In the bioinformatics field, there exist such challenges as described in [1; 2]. In the first example, when dealing with high-dimensional feature spaces in breast cancer detection, the data can be mapped to lower dimensions. This allows for easier visualisation of the data and for classification of breast mass lesions [1]. Another field where high-dimensional data needs to be analysed is geophysics. In one study [3], such data was embedded into a two-dimensional space, while preserving similarities of the original data. This allowed for effective visualisation and clustering of the data.

These are just a few of the use cases of the powerful dimensionality reduction algorithm, t-SNE (t-distributed Stochastic Neighbour Embedding). This algorithm visualises high-dimensional data in lower dimensions, while capturing non-linear data relations.

However, despite t-SNE's capabilities, it is computationally expensive. This becomes particularly problematic when you combine it with the challenge of tuning its parameters, requiring you to run several runs to achieve the desired visualisations [4].

Sampling-based t-SNE improves on this by working on subsets of the data and extrapolating optimal parameters to the full dataset [5]. This method builds a distance matrix (described in Subsection 2.1) on the full data, from which it generates matrices needed for the visualisation of the sample. After obtaining the desired parameters from the sample, we run the algorithm on the full dataset. However, during this step, the previously computed distance matrix on the full dataset is not utilised.

We propose a more efficient sampling-based t-SNE pipeline that reuses this distance matrix and applies a modified version of Dijkstra's algorithm to traverse it, approximating the distances between the sample points.

The question we answer with our paper is:

Can Dijkstra's algorithm be used to improve the speed of sample-based t-SNE?

We answer this question by breaking it down into the following research questions (RQs):

RQ1 Does our implementation produce similar matrices to the original sample-based implementation?

RQ2 Does it result in a lower runtime compared to the original sample-based implementation?

RQ3 How well do the visualisations compare to those of the original sample-based implementation?

To achieve this, our paper introduces our modified sampling-based t-SNE pipeline (detailed explanation in Section 4) integrated with Dijkstra's algorithm (Subsection 2.2) to reuse the global distance matrix. Then, in Section 5, we perform several experiments to compare our approach with the original sample-based implementation and present tables and figures of the results. Finally, we draw conclusions and suggest future work in Section 7.

2 Background

This section gives needed background information on the relevant algorithms for our research, namely t-SNE and Dijkstra's algorithm.

2.1 T-distributed Stochastic Neighbour Embedding

T-SNE [4] is an algorithm used to visualise high-dimensional data in lower dimensions (usually 2 or 3 dimensions). It attempts to embed close points in the original space to close points in the projected space. And the opposite, far points in the high-dimensional space correspond to distant points in the projection. First, it constructs a similarity matrix (also referred to as an affinity matrix) where the weights are modelled as probabilities. The similarity between two points x_i and x_j in the high-dimensional space is calculated with:

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

where $p_{i|i} = 0$.
After that, the values are symmetrised using:

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

Adapting to local density by setting the Gaussian kernels σ_i such that:

$$Perp(P_i) = 2^{H(P_i)} \tag{3}$$

where $H(P_i)$ is the Shannon entropy of the distribution Pi.
Then, the values from the affinity matrix are embedded into the low-dimensional space using:

$$q_{ij} = \frac{\left(1 + \|y_i - y_j\|^2\right)^{-1}}{\sum_{k \neq l} \left(1 + \|y_k - y_l\|^2\right)^{-1}} \tag{4}$$

where $q_{i|i} = 0$.
Finally, the locations y_i are calculated by minimising the Kullback-Leibler divergence between the high-dimensional distribution P and the low-dimensional distribution Q:

$$C = KL(P\|Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}} \tag{5}$$

In the tree accelerated version of t-SNE [6] we use, perplexity no longer determines the width of the Gaussian kernels σ_i. Instead, it determines the number of nearest neighbours $k = 3 * Perp$ which are considered when calculating similarities.

2.2 Dijkstra's Algorithm

Dijkstra's algorithm is a search algorithm performed on a weighted graph $G = (V, E)$, with vertex set V and edges E, with non-negative edge weights. It is used to find the shortest distance from a given vertex to all other vertices. It builds this distance matrix using a priority queue and runs in $O((V + E) \log V)$ time. In this paper, we make use of an improved version of this algorithm [7] with a time complexity of $O(|E| + |V| \log |V|)$ [8]. It is optimised for sparse graphs using a Fibonacci heap, explained in Subsection 2.3. We further modify this version to allow for early exit, further improving runtime (explained in Section 4).

2.3 Fibonacci heap

The Fibonacci heap is a data structure used for implementing priority queues. It uses a collection of min-heap trees, allowing for more flexibility and some lazy operations. A Fibonacci heap with size n performs all standard heap operations in $O(1)$ amortised time except for deleting an arbitrary element, which takes $O(\log n)$ amortised time [8].

3 Related Work

This section presents similar work that has been done before writing this paper. Furthermore, it shows the research gap between the current knowledge and our goal.

3.1 Similar Work

Several papers have discussed different ways to improve t-SNE. One example [9] uses a different way to compute pair similarities. Their implementation reduces the computation cost of random walk-based t-SNE by computing the LDL decomposition for the graph Laplacian. However, this is different to our goal, as we are looking to increase the performance of sample-based t-SNE using graph traversal.

Another paper [10] introduces an approximated solution, like our method, that aims to reduce computational time. Their approach incrementally constructs the neighbourhood graph by sampling edges, enabling users to see early results quickly. On the other hand, the sample-based approach, which we use and improve upon, focuses on vertex sampling.

Prior work [11] attempts to create a "shape-aware" version of t-SNE. The authors make use of graph distances to improve upon base t-SNE to achieve their goal of correctly representing hierarchical relationships between clusters. Furthermore, their goal is to improve the visualisations and robustness of the algorithm. This differs from our goal to improve the performance statistics of t-SNE.

Finally, in [12], they make use of a modified version of Dijkstra, similarly to our method. Another similarity is that they also use it to accelerate the calculation of distances. However, they apply it to the DIS-C algorithm, instead of t-SNE.

3.2 Research Gap

None of these papers attempts to solve the same issue as we do. Sampling-based t-SNE [5] tackled the problem of making parameter selection easier. However, one area where it could be improved, like most algorithms, is speed. The way sampling-based t-SNE works is as follows. First, it samples a portion of the original data. Then it builds a distance matrix on the full data, which is used to construct the distance matrix for the sampled data. Afterwards, it calculates a similarity matrix for the sample. Then, it uses that matrix to visualise the sampled data. However, instead of calculating the sample distance matrix, we could use Dijkstra's algorithm to traverse the initial distance matrix, obtaining the sampled distances.

4 Method

Instead of computing a new distance matrix for the sampled data, we utilise the distance matrix built on the full data.
We build the distances by traversing the full distance matrix with our modified version of Dijkstra's algorithm. The rest of this section gives a formal description of the problem at hand before giving a detailed explanation of our method.

4.1 Formal Problem Description

Let X be the full dataset of size n, and let $G = (X, E)$ be a distance matrix including at least $3 * k$ nearest neighbours per data point. Given a sample $S \subset X$, with $|S| = s \ll n$, existing methods recompute the distance matrix for S. We aim to avoid this by reusing the global graph G. The formal problem is:

> *Given a sample $S \subset X$, and the distance matrix $G = (X, E)$, efficiently construct an approximate distance matrix $G' = (S, E')$ without recomputing pairwise distances from scratch.*

4.2 Proposed Method

We propose the following solution (referred to as Dijkstra on Distances) to the problem outlined in the previous subsection. After computing the neighbourhood graph $G = (X, E)$ on the full dataset, we apply Dijkstra's algorithm. For each sample point in $S \subset X$, we run the modified Dijkstra's algorithm to find the shortest paths to the $3 * k$ nearest other sample points in S. These shortest path distances are then used, and an approximate neighbourhood graph $G' = (S, E')$ is constructed. Finally, we feed these new distances into the similarity calculations to continue the standard t-SNE pipeline. This approach allows the reuse of the global graph G while constructing a neighbourhood graph with distances.

During development, we found an alternative version (which we refer to as Dijkstra on Affinities). It uses the similarity matrix instead of the distance matrix (neighbourhood graph). Since Dijkstra's algorithm does not work with similarities directly, we use the reciprocals of the values from the similarity matrix. We perform the traversal using these new values and finally revert them to similarities for further use from the pipeline for visualisation.

Algorithm 1 Dijkstra Modified t-SNE

input $data, sample\_ratio, perplexity, mode$
$sampled\_data \leftarrow sample(data, sample\_ratio)$
$k \leftarrow 300$ # example value
if $mode ==$ "$distance$" **then**
  $distance\_matrix \leftarrow calculate\_distance\_matrix(data, k)$
elseif $mode ==$ "$affinities$"
  $affinity\_matrix \leftarrow calculate\_affinity\_matrix(distance\_matrix, k)$
  $reciprocal\_affinity\_matrix \leftarrow 1/affinity\_matrix$
if $mode ==$ "$distance$" **then**
  $sampled\_dijkstra\_distances \leftarrow modified\_dijkstra(distance\_matrix, sampled\_data, k)$
  $sampled\_data\_affinities \leftarrow standard\_affinity\_calculation(sampled\_dijkstra\_distances, perplexity)$
elseif $mode ==$ "$affinities$"
  $sampled\_data\_affinities \leftarrow 1/modified\_dijkstra(reciprocal\_affinity\_matrix, sampled\_data, k)$
$visualize(standard\_embedding(sampled\_data\_affinities))$

Pseudo Code

Algorithm 2 shows a high-level overview of the modified version of Dijkstra's algorithm that we use. It is based on the original version of the algorithm, while introducing an early stopping technique to save time. We keep track of how many sample points we have visited, and once we reach the desired number of $3 * k$, we stop the algorithm early.

Algorithm 1 with mode="distances" shows the modified pipeline for Dijkstra on Distances t-SNE. The new version relies on our new way to calculate the sample distance matrix.

Algorithm 1 with mode="affinities" shows the modified pipeline for Dijkstra on Affinities t-SNE. The new version relies on our new way to calculate the sample affinity matrix. In contrast to Dijkstra on Distances, here we need to calculate the affinity matrix and not just the distance matrix. Furthermore, in this algorithm, we need to turn the affinities into distances before running the Dijkstra algorithm.

Algorithm 2 Modified Dijkstra's Algorithm

input $graph, sample\_points, start, k \geq 1$
$dists \leftarrow \infty$
$dists[start] \leftarrow 0$
$heap \leftarrow start$
$samples\_visited = 0$
while $heap$ is not empty **do**
  $current \leftarrow heap.pop()$
  if $current$ is in $sample\_points$ **then**
    $samples\_visited++$
    if $samples\_visited \geq k$ **then**
      return $dists$
  for each $neigh$ of $current$ **do**
    $dist\_to\_neigh \leftarrow dists[current]$ + dist($neigh$)
    if $dists[neigh] > dist\_to\_neigh$ **then**
      $dists[neigh] \leftarrow dist\_to\_neigh$
      $heap.push(neigh)$
return $dists$

5 Experimental Setup and Results

In this section, we first describe the setup we used for our experiments. Then, we show the results obtained from the experiments.

Experimental Setup

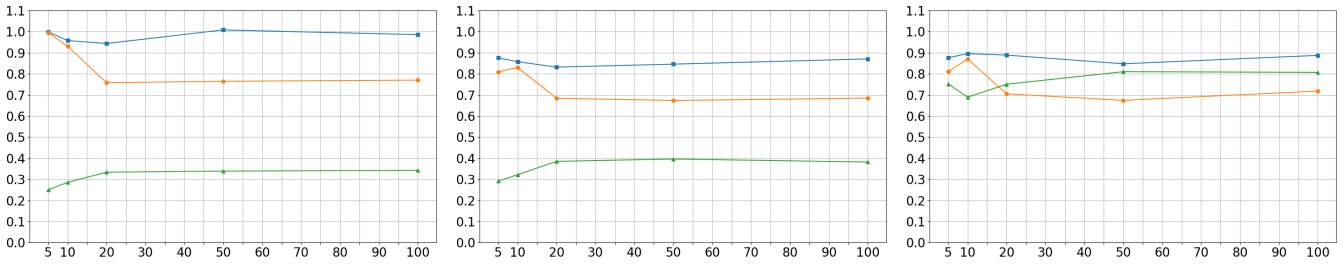
Our environmental setup consists of the following. First, the language and libraries used can be found in Subsection 6.2. The modified Dijkstra algorithm is based on SciPy's implementation of sparse Dijkstra [7]. The relevant compiled files and their usage instructions can be found in the repository mentioned in Subsection 6.2. The t-SNE methods we have used come from the openTSNE library [13]. The data that was used was first reduced to 50 by PCA (if it had more than 50 dimensions to begin with) [14]. This was done due to time and resource limitations. Furthermore, the embedding was done using the Fast Fourier transform (FFT) with early exaggeration, as part of the standard t-SNE strategy. Annoy was used for the distance matrix computations [15].

All experiments were conducted on a personal laptop equipped with an Intel Core i7-11800H CPU (8 cores, base 2.3GHz, scaling up to 4.6GHz under load), 16 GB of RAM, and running Windows 11.

5.1 Comparing Affinity Matrix Quality

To answer our first research question **RQ1**, we will analyse the similarity between the affinity matrices produced by the three different approaches by performing three pairwise comparisons. For this subsection, we have used a sample rate of 0.1, $k = 300$ nearest neighbours (used for initialisation of full distance and affinity matrices), relying on three metrics. Before defining these metrics, we introduce the following notation:

- n: the number of rows (data points).
- A_1: the first affinity matrix,
- A_2: the second affinity matrix,
- $N_i^{(j)}$: the set of nearest neighbours in row i of A_j,



(a) Normal t-SNE with Dijkstra sampled distances t-SNE

(b) Normal t-SNE with Dijkstra sampled affinities t-SNE

(c) Dijkstra sampled distances t-SNE with Dijkstra sampled distances t-SNE

Figure 1: Line graphs comparing the three methods based on the affinity matrices for different perplexity values for the MNIST dataset. On the Y axis, the three different measures are visualised. On the X axis lie the different perplexity values. The blue line with square markers represents the Ratio of Neighbour Count. The orange line with circle markers represents the Ratio of Shared Indices. The green line with triangle markers represents the Similarity.

The first metric is the ratio of the number of nearest neighbours per row between the second and first affinity matrices, averaged over all entries. The closer this value is to 1, the more similar the number of neighbours per data point.

$$\text{Ratio of Neighbour Count} = \frac{1}{n}\sum_{i=1}^{n}\frac{|N_i^{(2)}|}{|N_i^{(1)}|}$$

The second measure is again a ratio, per row, averaged over all rows. Its value is the size of the intersection of the indices of the nearest neighbours of the first and second affinity matrices, divided by the larger count of the two neighbour sets. This measure is in the range from 0 to 1, with higher values meaning more shared neighbours between the matrices per data point.

$$\text{Ratio of Shared Indices} = \frac{1}{n}\sum_{i=1}^{n}\frac{|N_i^{(1)} \cap N_i^{(2)}|}{\max\left(|N_i^{(1)}|,\, |N_i^{(2)}|\right)}$$

The last metric takes the indices which are already shared between the matrices per row, and calculates a similarity value between them. This value is obtained by taking the smaller value of the two and dividing it by the larger value. Finally, the average of all values is taken. This measure is in the range from 0 to 1, with higher values meaning more similar affinities.
The set of shared indices:

$$S_i = N_i^{(1)} \cap N_i^{(2)}$$

Then we calculate the similarity per row

$$\text{Similarity per row} = \frac{1}{|S_i|}\sum_{j \in S_i}\frac{\min\left(A_1[i,j],\, A_2[i,j]\right)}{\max\left(A_1[i,j],\, A_2[i,j]\right)}$$

Finally, we use it to reach one value for similarity:

$$\text{Similarity} = \frac{1}{n}\sum_{i=1}^{n}(\text{Similarity per row})$$

Effect of Changing Perplexity

For this comparison, we run the models with the set of perplexities [5, 10, 20, 50, 100].

Figure 1a compares the standard t-SNE approach with the first version of our algorithm, which builds the sampled distances by performing Dijkstra's algorithm on the full distance matrix. It shows that the average number of nearest neighbours is quite similar, with the value being close to 1. Furthermore, more than 75% of the nearest neighbours are the same. On the other hand, the shared neighbours have rather different values, with only around 0.3-0.35 similarity.

Figure 1b compares the standard t-SNE approach with the second version of our algorithm, which builds the sampled affinities by performing Dijkstra's algorithm on the full affinity matrix. Most numbers are similar to the other comparison graph in Figure 1a. However, here we can observe a slightly worse ratio of shared neighbours and number of neighbours and a slightly better similarity value.

Finally, in Figure 1c we compare our two approaches. Although they performed similarly in their comparisons with base t-SNE, here we see that they have differences. They produce rather similar matrices with a high ratio of shared neighbours and a high similarity value between them.

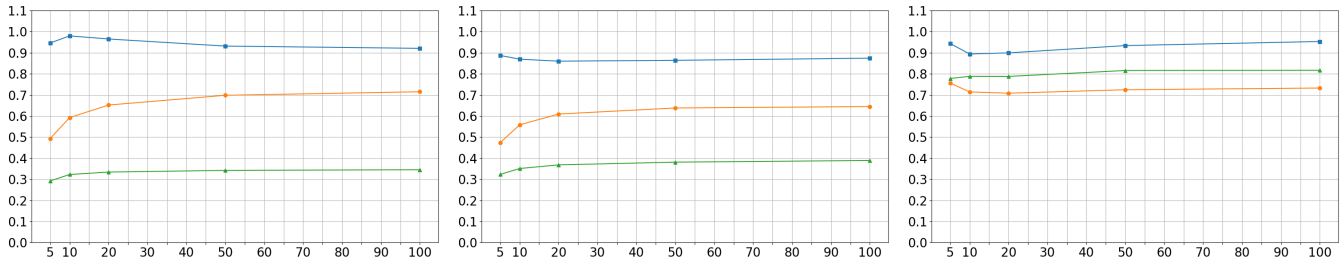
In Figure 1a, Figure 1b, and Figure 1c, all values stabilise with increasing perplexity.

To further support our results, we also perform the same experiments on the C. elegans dataset. These results (Figure 6) show similar trends to the graphs from the experiments on the MNIST data.

Effect of Changing Initialisation k

Now we will make the same comparison, but this time we will compare for different initialisation $k = 3 * Perplexity$. For example, if the perplexity is 5, for Normal t-SNE we will use a perplexity of 5. For the other two methods, we will use $k = 3 * 5$ for the initialisation steps, then use perplexity of 5.

In Figure 2 we see that our methods perform more similarly to the base method with increasing initialisation k. We derive this from the increasing similarity and ratio of shared indices.

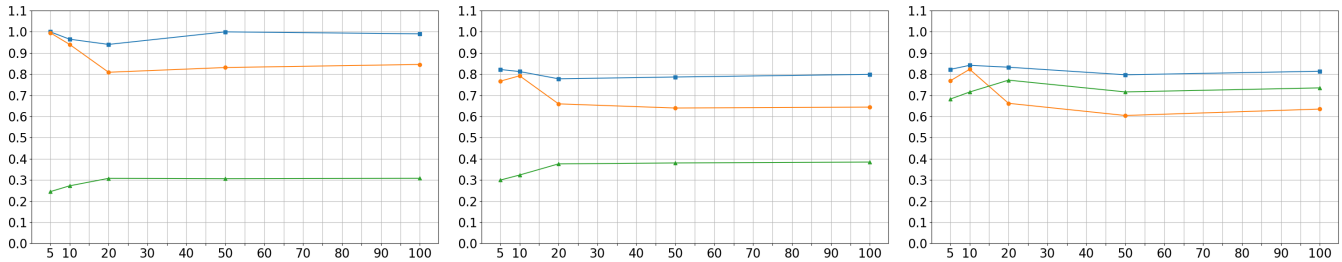


(a) Normal t-SNE with Dijkstra sampled distances t-SNE

(b) Normal t-SNE with Dijkstra sampled affinities t-SNE

(c) Dijkstra sampled distances t-SNE with Dijkstra sampled distances t-SNE

Figure 2: Line graphs comparing the three methods based on the affinity matrices for different values of initialisation k for the MNIST dataset. On the Y axis, the three different measures are visualised. On the X axis lie the different k values. The blue line with square markers represents the Ratio of Neighbour Count. The orange line with circle markers represents the Ratio of Shared Indices. The green line with triangle markers represents the Similarity.

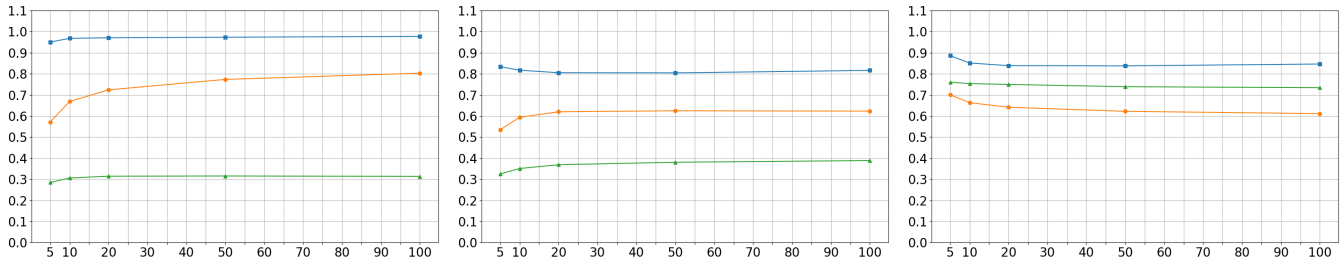


(a) Normal t-SNE with Dijkstra sampled distances t-SNE

(b) Normal t-SNE with Dijkstra sampled affinities t-SNE

(c) Dijkstra sampled distances t-SNE with Dijkstra sampled distances t-SNE

Figure 3: Line graphs comparing the three methods based on the affinity matrices for different perplexity values for the C. elegans dataset. On the Y axis, the three different measures are visualised. On the X axis lie the different perplexity values. The blue line with square markers represents the Ratio of Neighbour Count. The orange line with circle markers represents the Ratio of Shared Indices. The green line with triangle markers represents the Similarity.



(a) Normal t-SNE with Dijkstra sampled distances t-SNE

(b) Normal t-SNE with Dijkstra sampled affinities t-SNE

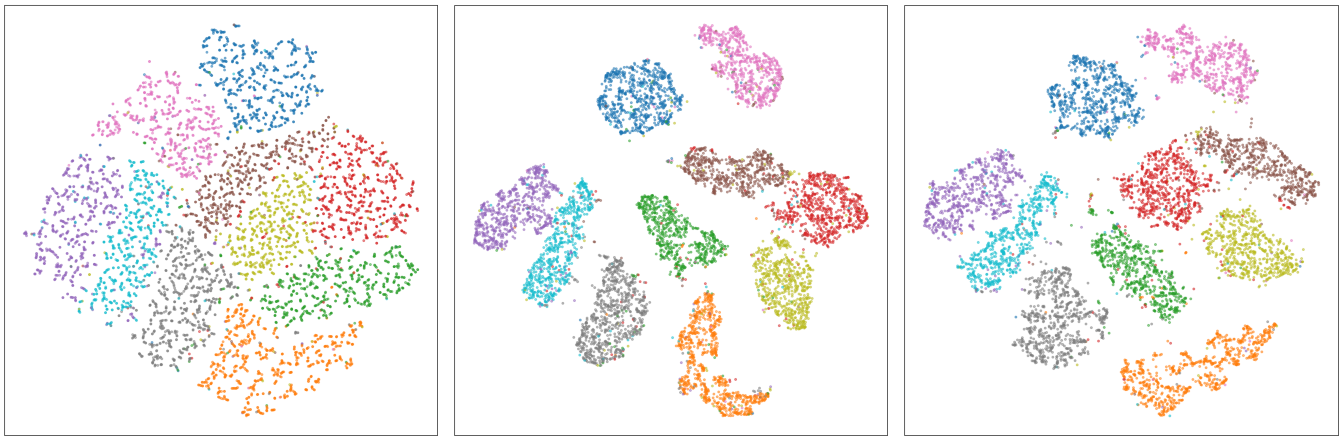
(c) Dijkstra sampled distances t-SNE with Dijkstra sampled distances t-SNE

Figure 4: Line graphs comparing the three methods based on the affinity matrices for different values of initialisation k for the C. elegans dataset. On the Y axis, the three different measures are visualised. On the X axis lie the different k values. The blue line with square markers represents the Ratio of Neighbour Count. The orange line with circle markers represents the Ratio of Shared Indices. The green line with triangle markers represents the Similarity.

Furthermore, we see that Dijkstra on distances is more similar to the normal method (Figure 2a) compared to Dijkstra on affinities (Figure 2b). Finally, our two methods perform differently as seen in Figure 2c, but their differences stay somewhat consistent over different k.

We repeat this experiment using C. elegans data to solidify our results.

From Figure 4c we see that for this dataset, our two methods differ more, compared to the MNIST dataset (Figure 2c). Furthermore, from Figure 4a we can observe that our Dijkstra on Distances method produces more similar results to the normal method, compared to MNIST (Figure 2a). However, the results from this dataset generally follow the same trends, except for the few outlined differences.



(a) Visualisation using normal t-SNE

(b) Visualisation using sampled distances with Dijkstra t-SNE

(c) Visualisation using sampled affinities with Dijkstra t-SNE

Figure 5: Comparison of visualisations of the three different methods analysed in our paper for the MNIST dataset. We can see that (b) and (c) have more distinct clusters than (a), but maintain the quality of the visualisations.

5.2 Comparing Runtime

To answer our second research question **RQ2**, we present the runtime statistics of the three methods being compared. We run each method 5 times with each perplexity from [5, 10, 20, 50, 100] and take the average runtime per perplexity. We have extracted the setup time as it usually needs to be done only once per dataset. In Dijkstra on distances, "setup" refers to the time it takes to create the distance matrix on the full data. In Dijkstra on affinities, it refers to the time it takes to calculate both the distance and affinity matrices on the full data.

In Table 1, we can see that the execution time for our two methods is lower than the execution time for the normal method, when looking at the lower perplexities. Dijkstra on Distances performs up to 40% faster than the Normal method. Another interesting point is that Dijkstra on Affinities takes longer to set up and generally performs slower than Dijkstra on Distances.

Method	**Setup**	**Perplexity**				
		5	**10**	**20**	**50**	**100**
Normal	-	137.1	118.0	94.0	69.9	52.8
Dijkstra on Distances	19.5	**80.4**	**71.7**	**53.4**	87.9	140.9
Dijkstra on Affinities	39.2	**82.3**	**70.6**	73.2	92.7	151.4

Table 1: Total execution time per method per perplexity in seconds for the MNIST dataset. Averaged over 5 runs, calculated using $k = 300$. We observe that our methods execute faster for lower perplexities.

These results are reinforced by the experiments performed on the C. elegans data (Table 2). These also show our methods executing faster than the normal method for lower perplexities. However, unlike the results from the MNIST data, here Dijkstra on Distances consistently performs faster than Dijkstra on Affinities.

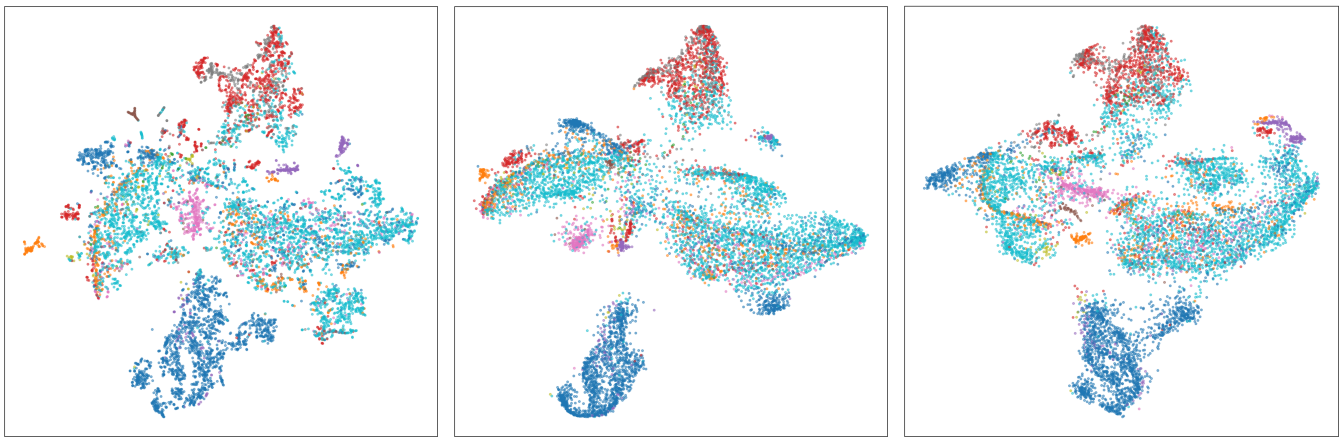
Method	**Setup**	**Perplexity**				
		5	**10**	**20**	**50**	**100**
Normal	-	130.4	120.4	93.1	71.9	63.5
Dijkstra on Distances	20.0	**77.7**	**69.4**	**67.0**	82.0	145.2
Dijkstra on Affinities	45.0	97.4	84.6	85.4	107.4	168.9

Table 2: Total execution time per method per perplexity in seconds for the C. elegans dataset. Averaged over 5 runs, calculated using $k = 300$. We observe that our methods execute faster for lower perplexities.

5.3 Qualitative Comparison

Finally, to answer our third and last research question **RQ3**, we show the visualisation made by the three versions of t-SNE we are comparing. We have made these visualisations using $k = 30$ and a perplexity of 5 on the MNIST dataset.

We can observe that in Figure 5a the data is more spread, compared to Figure 5b and Figure 5c, which are rather clustered. Another interesting thing we notice is that in Figure 5c there are significantly more points which are between clusters, compared to Figure 5b. Furthermore, the points within clusters seem to be slightly more spread out in the area of the cluster in Figure 5c, compared to Figure 5b.



(a) Visualisation using normal t-SNE

(b) Visualisation using sampled distances with Dijkstra t-SNE

(c) Visualisation using sampled affinities with Dijkstra t-SNE

Figure 6: Comparison of visualisations of the three different methods analysed in our paper for the C. elegans dataset. We can see that (b) and (c) group the data more. Sometimes resulting in better clusters, sometimes making it harder to distinguish clusters.

From this, we can deduce that with this setup, normal t-SNE captures better the global structure. So, to see how the different classes of data relate to each other, this would be the better choice. However, it is not that good at clustering the data. If we want to see more clearly separated clusters, we have to use one of the other methods. They both capture the local structure well. However, it seems like the Dijkstra on Affinities method leaves more points without clusters, so perhaps if misclassification is not an issue, we would use the Dijkstra on Distances method.

To solidify our results, we perform a similar experiment on the C. elegans data. This time we use $k = 300$ and a perplexity of 50, due to the increased complexity of the data.

In Figure 6 we observe similar results, our two methods (Figure 6b and Figure 6c) show somewhat similar visualisations, both different to normal t-SNE (Figure 6a). In Figure 6b, the clusters are similarly defined to Figure 6a, except for the smaller clusters. Figure 6c is more similar to Figure 6a. In general, all three visualisations seem quite similar, but our methods group the points more, even when not necessary.

6 Responsible Research

This section examines the ethical implications of our research and discusses the reproducibility of our methods.

6.1 Ethical Considerations

Intended Use

This research aims to improve the performance of sampling-based t-SNE using Dijkstra's algorithm. Its intended use is for visualising high-dimensional data.

Potential Misuse

Our method introduces a new level of approximation, which does not introduce new types of risks, compared to standard t-SNE. However, it could amplify existing ones as it may reduce transparency or accuracy, which could be a problem in applications where exact data is required. It is upon the user to ensure this approximation does not lead to making misleading conclusions. Furthermore, it should be considered that using this algorithm on sensitive data (e.g. medical or demographic datasets) could introduce further risks in the chain of use.

Data Ethics

All data used is publicly available and anonymised. It contains no personally identifiable information. The used data sets are MNIST [16] and C. elegans [17].

6.2 Reproducibility

To ensure our results are reproducible, our code is publicly available at https://github.com/Filipunder10/dijkstra_tsne. Furthermore, in Section 5 we outline more precisely the setup used to run the experiments. They were run using Python 3.9, with libraries such as NumPy and OpenTSNE listed in the environment.yml and README files. For all random components, a fixed $seed = 42$ was used to encourage reproducibility. Parameters and code are explained using comments and README files.

6.3 Use of LLMs

Large Language Models were not directly used in the writing of the report. However, the tool Grammarly was used to fix grammatical, spelling, punctuational, and other writing mistakes. While it wasn't used to directly create text, it was used to rewrite some sentences to enhance the reading experience. Since Grammarly uses AI, all suggested changes were reviewed, although it does not generate new content or ideas when suggesting them.

7 Conclusions and Future Work

This section first lays out the conclusions we have made from our research. Then it gives suggestions for related future work.

7.1 Conclusions

In summary, our paper analysed whether Dijkstra can be used to improve the performance of sampling-based t-SNE.

To answer **RQ1**, we compared the affinity matrices produced by our methods to those from the normal method. From this, we observed that they generally shared a lot of similarities, but also had differences.

For **RQ2**, we showed that our methods can improve the runtime of the algorithm for lower perplexity values. In some cases, the runtime was reduced by up to 40%.

Finally, we presented visualisations of our methods, compared to the standard pipeline, to answer the final **RQ3**. Those comparisons showed that, although the figures were different, the visualisations were still useful in certain situations.

On the other hand, our algorithm still falls short in some aspects when compared to standard sample-based t-SNE. One drawback is that our method introduces a new layer of approximation. This might be a problem in some fields, especially if precise data is required. Another area where our algorithm is lacking is speed. Although for some lower perplexities, the runtime can be reduced by up to 40%, for higher perplexities it is slower than the normal method by three times.

Our findings show that our method can be useful in certain situations. For example, when we want to visualise a sample of the data we have. We will need to perform several runs of the algorithm to achieve the desired results. Our algorithm would be faster than common methods, given that the perplexity we choose is lower than 20-50.

7.2 Future Work

More testing

We have tried to capture the general trends by testing with a wide range of variables. However, more testing can certainly bring more insights to our findings. Using more datasets and a wider set of parameters can accomplish that.

Optimising k

Currently, we adjust k so it matches the number of distances calculated by the normal implementation. This is done to make comparison easier, but maybe it is not necessary.

Different algorithm instead of Dijkstra

Dijkstra's algorithm is well-suited for our problem. However, other similar algorithms could also take its spot, such as A* or Bellman-Ford.

Attempt to use Djikstra to create strictly better visualisations and not improve speed

We see that visualisations are not degraded by our algorithm and, in fact, in some cases might be better. The goal of this paper is to trade accuracy for speed. However, the opposite might prove useful too.

References

[1] Andrew R. Jamieson, Maryellen L. Giger, Karen Drukker, Hui Li, Yading Yuan, and Neha Bhooshan. Exploring nonlinear feature space dimension reduction and data representation in breast Cadx with Laplacian eigenmaps and t-SNE. *Medical Physics*, 37(1):339–351, January 2010.

[2] Dmitry Kobak and Philipp Berens. The art of using t-SNE for single-cell transcriptomics. *Nature Communications*, 10(1):5416, November 2019.

[3] Mehala Balamurali, Katherine L. Silversides, and Arman Melkumyan. A comparison of t-SNE, SOM and SPADE for identifying material type domains in geological data. *Computers & Geosciences*, 125:78–89, April 2019.

[4] Laurens van der Maaten and Geoffrey Hinton. Visualizing Data using t-SNE. *Journal of Machine Learning Research*, 9(86):2579–2605, 2008.

[5] Martin Skrodzki, Nicolas F. Chaves-de Plaza, Thomas Höllt, Elmar Eisemann, and Klaus Hildebrandt. Navigating Perplexity: A linear relationship with the data set size in t-SNE embeddings, December 2024. arXiv:2308.15513 [cs].

[6] Laurens van der Maaten. Accelerating t-SNE using Tree-Based Algorithms. *Journal of Machine Learning Research*, 15(93):3221–3245, 2014.

[7] SciPy Developers. scipy.sparse.csgraph.dijkstra, api reference, v1.15.3. https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.csgraph.dijkstra.html. (accessed: 2025-04-28).

[8] Michael L. Fredman and Robert Endre Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM*, 34(3):596–615, July 1987.

[9] Yasuhiro Fujiwara, Yasutoshi Ida, Sekitoshi Kanai, Atsutoshi Kumagai, and Naonori Ueda. Fast Similarity Computation for t-SNE. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pages 1691–1702, Chania, Greece, April 2021. IEEE.

[10] Nicola Pezzotti, Boudewijn P. F. Lelieveldt, Laurens Van Der Maaten, Thomas Hollt, Elmar Eisemann, and Anna Vilanova. Approximated and User Steerable tSNE for Progressive Visual Analytics. *IEEE Transactions on Visualization and Computer Graphics*, 23(7):1739–1752, July 2017.

[11] Tobias Wängberg, Joanna Tyrcha, and Chun-Biu Li. Shape-aware stochastic neighbor embedding for robust data visualisations. *BMC Bioinformatics*, 23(1):477, November 2022.

[12] Rolando Quintero, Esteban Mendiola, Giovanni Guzmán, Miguel Torres-Ruiz, and Carlos Guzmán Sánchez-Mejorada. Algorithm for the Accelerated Calculation of Conceptual Distances in Large Knowledge Graphs. *Mathematics*, 11(23):4806, November 2023.

[13] Pavlin G. Policar, Martin Strazar, and Blaz Zupan. openTSNE : A Modular *Python* Library for t-SNE Dimensionality Reduction and Embedding. *Journal of Statistical Software*, 109(3), 2024.

[14] Andrzej Maćkiewicz and Waldemar Ratajczak. Principal components analysis (PCA). *Computers & Geosciences*, 19(3):303–342, March 1993.

[15] Wen Li, Ying Zhang, Yifang Sun, Wei Wang, Wenjie Zhang, and Xuemin Lin. Approximate Nearest Neighbor Search on High Dimensional Data — Experiments, Analyses, and Improvement (v1.0), October 2016. arXiv:1610.02455 [cs].

[16] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998.

[17] Junyue Cao, Jonathan S. Packer, Vijay Ramani, Darren A. Cusanovich, Chau Huynh, Riza Daza, Xiaojie Qiu, Choli Lee, Scott N. Furlan, Frank J. Steemers, Andrew Adey, Robert H. Waterston, Cole Trapnell, and Jay Shendure. Comprehensive single-cell transcriptional profiling of a multicellular organism. *Science*, 357(6352):661–667, August 2017.