Accelerating t-SNE using a uniform grid-based approximation

Milan Otten[1]

Supervisor: Martin Skrodzki[1]**, Responsible Professor: Elmar Eisemann**[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 23, 2024

Name of the student: Milan Otten
Final project course: CSE3000 Research Project
Thesis committee: Elmar Eisemann, Martin Skrodzki, Gosia Migut

Abstract

Dimensionality reduction is an important task in high-dimensional data visualisation. Among the popular algorithms for achieving this is t-SNE, which aims to preserve local neighbourhoods in the lower-dimensional embeddings. While t-SNE traditionally works in Euclidean space, embedding in hyperbolic space offers several advantages, specifically for data of arbitrary size and exponential growth, such as tree-based structures. We propose a new solution to approximate and accelerate the calculation of t-SNE gradients using a uniform grid structure. This new method produces embeddings with better neighbourhood preservation than previous solutions, while also providing better runtime performance.

1 Introduction

Data from real life are often high-dimensional datasets. In order to get an overview of such data, we need to reduce the dimensionality, such that it can be displayed in 2 dimensions in an image. Using a popular linear algorithm such as Principal Component Analysis (PCA) has drawbacks. For instance, it is "not useful in capturing sample distributions spread on complex manifolds" [1]. A better, and widely used algorithm for dimensionality reduction is t-distributed Stochastic Neighbour Embedding (t-SNE), as it aims to preserve local neighbourhoods in the data points. However, t-SNE is designed for points in Euclidean space, which misses out on several positive properties of embedding into hyperbolic space instead. Hyperbolic space is a curved space which expands exponentially as you move away from the centre. It is especially well-suited for embedding arbitrary size data due to its essentially infinite area, while still being visualisable in a single image. Several methods have already been proposed to use hyperbolic space as a candidate for embedding trees into [2].

Modern Euclidean space embeddings use acceleration methods that make the process of creating an embedding much faster. However, hyperbolic space embeddings are hard to optimise, as accelerations for Euclidean space do not directly translate to hyperbolic space. This prevents the use of mean values and interpolation, which are important to Euclidean optimisations, such as the Barnes-Hut scheme [3] which accelerates t-SNE by building a quadtree on the data.

To extend Barnes-Hut to hyperbolic space, previous research [4] shows two things. First, it defines formulae to approximate the gradient for t-SNE in hyperbolic space. Second, it proposes the use of a polar quadtree [5] on the hyperbolic embedding of the data on the Poincaré disk. However, this is only one of many possible acceleration structures on the Poincaré disk space. We propose the use of a uniform grid structure on the Euclidean projection of the Poincaré disk for approximating the t-SNE gradient. Such a structure would provide a runtime linear in the number of points, at the cost of a more coarse approximation.

2 Background

This section introduces the concepts and algorithms that this research is based on. Specifically, we discuss the Poincaré Disk model, which we will use as our hyperbolic space for embedding data into. We introduce t-distributed Stochastic Neighbour Embedding as a way of embedding data, and Barnes-Hut as a method of optimising this process. Finally, we will discuss the idea of a uniform grid structure and its applications.

2.1 Poincaré Disk Model

We will be working in hyperbolic space, for which there are several models available. The research [4] that this paper is based on uses the Poincaré disk model, which is very suitable for embedding arbitrarily sized data, as it is conformal and maps all points to a 2-dimensional unit disk. Intuitively, distances get exponentially bigger closer to the edge of the disk. The Poincaré space has an infinite size and points on the edge of the disk would be 'at infinity'. This is what allows us to embed data of arbitrary size and exponential growth into a single 2-dimensional disk. Figure 1 shows how the hyperbolic distance increases with Euclidean distance from the centre of the disk.

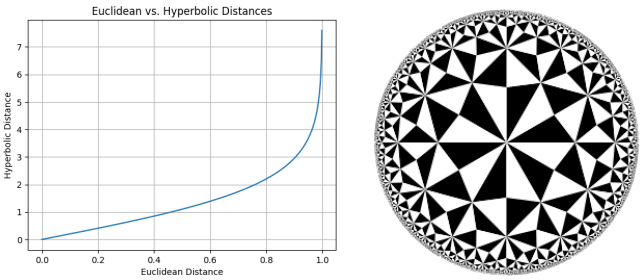

Figure 1: Left: Comparison of Euclidean distance from the centre of the disk compared to the hyperbolic distance at the same point. The hyperbolic distance would grow to infinity, but the Euclidean distance is stopped at 0.9999.
Right: Visualisation of the hyperbolic space using a constant pattern. Source: Weisstein, Eric W. "Poincaré Hyperbolic Disk." From *MathWorld*–A Wolfram Web Resource. https://mathworld.wolfram.com/PoincareHyperbolicDisk.html.

Formally, the Poincaré disk is defined as the space $\mathbb{D} = \left\{\mathbf{y} \in \mathbb{R}^2 : \|\mathbf{y}\| < 1\right\}$ with Euclidean metric:

$$g^{\mathrm{D}}_{\mathbf{y}} = \lambda^2_{\mathbf{y}} g^E, \quad \text{where} \quad \lambda_{\mathbf{y}} = \frac{2}{1 - \|\mathbf{y}\|^2} \tag{1}$$

Points where $\|\mathbf{y}\| = 1$ are not included since this would imply the point is at infinity, and the distance calculation would include a division by 0.

The hyperbolic distance between two points $\mathbf{a}$ and $\mathbf{b}$ is defined as:

$$d^{\mathcal{H}}(\mathbf{a}, \mathbf{b}) = \cosh^{-1}\left(1 + 2\frac{\|\mathbf{a} - \mathbf{b}\|^2}{\left(1 - \|\mathbf{a}\|^2\right)\left(1 - \|\mathbf{b}\|^2\right)}\right) \tag{2}$$

2.2 t-distributed Stochastic Neighbour Embedding

A popular algorithm for embedding high-dimensional data in a lower-dimensional space while preserving neighbourhoods is t-distributed Stochastic Neighbour Embedding (t-SNE) [6]. Intuitively, t-SNE first calculates an initial embedding of the data in the lower-dimensional space using principal component analysis (PCA) [7] after which it performs gradient descent on the Kullback-Leibler divergence. This can be interpreted as negative and positive forces between each pair of points. The negative forces push points apart, while the positive forces attract points that are close in the high-dimensional space, which aims to bring them close in the lower-dimensional embedding. Using these forces as a gradient, gradient descent is performed until a (locally) optimal solution is found.

More precisely, it takes high dimensional input data $\{x_1, x_2, ..., x_n\} \in \mathbb{R}^d$, which are interpreted as probabilities, with

$$p_{j|i} = \frac{\frac{\exp(-||x_i - x_j||^2)}{2\sigma_i}}{\sum_{k \neq i} \frac{\exp(-||x_i - x_k||^2)}{2\sigma_i^2}}, p_{ij} = \frac{p_{j|i} + p_{j|i}}{2}$$

Where $p_{i|i} = 0$ and σ_i is the variance of the Gaussian function centred on points x_i, which is a term derived from the user-provided 'perplexity' variable for this algorithm.

The gradient used for gradient descent is calculated per point as follows:

$$\frac{\delta C}{\delta \mathbf{y}_i} = 4 \sum_{j \neq i} (p_{ij} - q_{ij}) \left(1 + \|\mathbf{y}_i - \mathbf{y}_j\|^2\right)^{-1} (\mathbf{y}_i - \mathbf{y}_j). \tag{3}$$

Which can be written as

$$\frac{\delta C}{\delta \mathbf{y}_i} = 4 \left(\sum_{j \neq i} p_{ij} q_{ij} Z (\mathbf{y}_i - \mathbf{y}_j) - \sum_{j \neq i} q_{ij}^2 Z (\mathbf{y}_i - \mathbf{y}_j) \right) \tag{4}$$

with $Z = \sum_{k \neq \ell} \left(1 + \|\mathbf{y}_k - \mathbf{y}_\ell\|^2\right)^{-1}$. On the right side of the equation are two sums. The left sum are the 'positive forces' and the right sum are the 'negative forces', which pull and push the points.

2.3 Barnes-Hut

A widely used algorithm for speeding up n-body simulations is Barnes-Hut [8]. It can be used in many different simulation contexts [9], [10], [11]. In t-SNE, it can be used to speed up the calculation of the gradient, specifically the negative forces. It builds a quadtree on the data, creates a *summary* of the points in a quadrant for every level, and uses this summary to approximate forces for points that are far away, relative to the size of the quad. This way, the error from approximating is low, while still avoiding the exact calculation of many points [3]. Figure 2 illustrates the process of summarisation. Left is the exact method: the force is calculated between the green point and all other points. Right is the approximation: the centre in orange is calculated once, and the force is calculated between the green point and this centre.

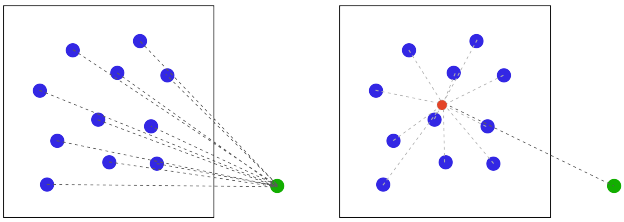

Figure 2: An illustration of the summarisation performed by Barnes-Hut, during the calculation of the forces acting on the green point. Left is the exact method, right is the approximation.

2.4 Uniform grid

A popular way of dividing up space for speeding up various calculations is the uniform grid [12]. It is widely used in simulation [13], [14], [15] for its simplicity, specifically in lower dimensions as is common in simulations of real-life phenomena. The main idea is that some space is divided into cells of equal size, which can then be used for limiting calculations to certain cells, or approximating underlying data.

A problem with the uniform grid is that it does not work well with sparse data, as it leaves many cells containing no points, which could result in less optimisation from using the uniform grid. A similar data structure would be a non-uniform grid, which aims to provide more cells in areas with more points. Another way of curbing the effects of this problem will be discussed in Section 4.1.

3 Related work

In the following sections, we highlight some solutions for dimensionality reduction and for accelerating t-SNE.

3.1 Dimensionality Reduction Techniques

Techniques for reducing the dimensionality of data can be split into linear and non-linear methods [16]. Linear methods are generally used for exploring global structures in the data, while non-linear methods aim to preserve local structures [17].

Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA) are very popular linear techniques. LDA is generally considered better, but PCA can outperform LDA with small datasets. PCA is also less sensitive to variations between datasets [18]. However, linear techniques generally suffer from information loss and the inability to handle non-linear data well.

t-SNE is a non-linear technique, useful to us for its neighbourhood preservation qualities [6]. Locally Linear Embedding (LLE) is another non-linear method focused on preserving local neighbourhoods which works by using linear reconstructions of local neighbourhoods [19]. A fast and scalable non-linear technique is uniform manifold approximation and projection (UMAP), which aims to preserve both local and global neighbourhoods [20].

3.2 t-SNE Acceleration

While t-SNE is precise and gives embeddings that effectively visualise high-dimensional datasets, it is naively $O(n^2)$ due to having to calculate the force between every possible pair of points. This has a significant impact on the runtime of the algorithm, making it infeasible for big datasets. There are many approaches for accelerating t-SNE. Most methods depend on the same Barnes-Hut idea: summarising multiple points as a single point to avoid many calculations [3]. For example, using a forest of balanced LSH (locality-sensitive hashing) trees [21]. LSH is a technique that aims to cluster similar items into the same "buckets" using a hashing function.

An important base for this research has been the implementation of Barnes-Hut for t-SNE in hyperbolic space [4]. This method involves the use of a polar quadtree on the Poincaré disk model for hyperbolic space. The polar quadtree used is similar to a regular quadtree, but creates cells in a radial manner from the centre of the disk. It also uses the properties of distance in hyperbolic space to keep the cells similarly sized when considered in hyperbolic space.

Research has been done to implement and thoroughly optimise Barnes-Hut running on the GPU using CUDA [22]. Much action was taken to ensure efficient memory access patterns and optimal use of all available resources on the GPU.

Another approach has been proposed which approximates the repulsive forces by splatting kernel textures on the data points. This allows this calculation to be performed as a series of tensor operations that can be performed efficiently on the GPU [23].

Finally, a method that involves building a pair of spatial hierarchies on the data which are then traversed simultaneously to approximate the forces has also been proposed and evaluated [24].

4 Uniform Grid for Approximating t-SNE

We propose a data structure that can be used as an alternative to the polar quadtree proposed in previous research [4] for approximating forces in t-SNE. We first formally define the data structure and subsequently show how it is used to accelerate t-SNE.

4.1 Uniform Grid

Our proposed data structure for approximating forces in t-SNE is the uniform grid, applied to the 2D embedding of all data points onto the Poincaré disk. This data structure consists of a uniform grid of non-overlapping and equally sized rectangles (grid cells) in Euclidean space, that in total covers the entire area of all data points. This means that each data point is in exactly one grid cell. We have data points $\{\mathbf{y}_0, \mathbf{y}_1, ..., \mathbf{y}_n\}$, each grid cell has a set of members $m_i = \{\mathbf{y}_j, \mathbf{y}_\ell, ..., \mathbf{y}_k\}$ and a centre $c_i \in \mathbb{R}^2$, which is the hyperbolic geometric mean of all points in the cell. We say that a data point is a member of a grid cell if it is in the area of the grid cell. Because each point is a member of exactly one grid cell, we have that $\sum_i |m_i| = n$ for n points. The centres are determined after all members have been found, using a formula that approximates the Fréchet-mean of the member points, which provides an accurate mean with regard to hyperbolic space. Then,

$$m(\{v_j\}) = \sum_j \left(\frac{\gamma(v_j)}{\sum_l (v_\ell)\gamma(v_\ell)} \right) v_j$$

With $\gamma(x) = \frac{1}{\sqrt{1-||x||^2}}$ and v the coordinates of x interpreted in the Klein model. Translation to and from the Klein model is shown below, with p the function to translate from the Poincaré model to the Klein model, and k the function to translate from the Klein model to the Poincaré model.

$$p(x) = \left(\frac{2 \cdot x_1}{1 + x_1^2 + x_2^2}, \frac{2 \cdot x_2}{1 + x_1^2 + x_2^2} \right)$$

$$k(x) = \left(\frac{2 \cdot x_1}{1 + \sqrt{1 - x_1^2 - x_2^2}}, \frac{2 \cdot x_2}{1 + \sqrt{1 - x_1^2 - x_2^2}} \right)$$

Setting up the uniform grid is done in $O(n+m)$ time, with n the number of points and m the number of grid cells.

Usage of the Uniform Grid

Previous research [4] proposes the following definition of an equation for the gradient in hyperbolic space:

$$\frac{\delta C^{\mathcal{H}}}{\delta \mathbf{y}_i} = 4 \left(\sum_{j \neq i} p_{ij} q_{ij}^{\mathcal{H}} Z^{\mathcal{H}} \frac{\delta d_{ij}^{\mathcal{H}}}{\delta \mathbf{y}_i} - \sum_{j \neq i} \left(q_{ij}^{\mathcal{H}} \right)^2 Z^{\mathcal{H}} \frac{\delta d_{ij}^{\mathcal{H}}}{\delta \mathbf{y}_i} \right) \tag{5}$$

$$q_{ij}^{\mathcal{H}} = \frac{\left(1 + \left(d_{ij}^{\mathcal{H}} \right)^2 \right)^{-1}}{\sum_{k \neq \ell} \left(1 + \left(d_{ij}^{\mathcal{H}} \right)^2 \right)^{-1}}, \tag{6}$$

$$d_{ij}^{\mathcal{H}} = d^{\mathcal{H}}(\mathbf{y}_i, \mathbf{y}_j) \tag{7}$$

$$\frac{\delta d_{ij}^{\mathcal{H}}}{\delta \mathbf{y}_i} = \frac{4 \left(\left(\left\| \mathbf{y}_j \right\|^2 - 2 \left\langle \mathbf{y}_i, \mathbf{y}_j \right\rangle + 1 \right) \mathbf{y}_i / \alpha - \mathbf{y}_j \right)}{\alpha \beta \sqrt{\gamma^2 - 1}}, \tag{8}$$

where $Z^{\mathcal{H}} = \sum_{k \neq \ell} \left(1 + d_{ij}^{\mathcal{H}^2} \right)^{-1}$

The uniform grid is used when calculating the negative forces, which is the second sum on the right side of Equation 5. In each of the following formulae used to describe the uniform grid functionality, a g is added to the superscript of previously defined variables, so as to differentiate these. Equation 9 shows the new gradient formula. This shows the operation of our new solution: instead of calculating the negative forces between each point, we calculate the negative force from each point to the centre c_i of each grid cell. This force is then multiplied by the number of members in the grid cell. Any grid cells with no members are not taken into account for the calculation. Where previously the distance between each point was used, $d_{ij}^{g\mathcal{H}} = d^{\mathcal{H}}(\mathbf{y}_i, \mathbf{c}_j)$ is used, which is the hyperbolic distance between point i and grid cell

j. This means that there is no exact force calculation between any two points, but only a much faster approximation.

$$\frac{\delta C^{g\mathcal{H}}}{\delta \mathbf{y}_i} = 4 \sum_{j \neq i} p_{ij} q_{ij}^{\mathcal{H}} Z^{\mathcal{H}} \frac{\delta d_{ij}^{\mathcal{H}}}{\delta \mathbf{y}_i} - 4 \sum_{1 \leq j \leq m} \left(q_{ij}^{g\mathcal{H}} \right)^2 Z^{g\mathcal{H}} |m_j| \frac{\delta d_{ij}^{g\mathcal{H}}}{\delta \mathbf{y}_i} \tag{9}$$

$$q_{ij}^{g\mathcal{H}} = \frac{\left(1 + \left(d_{ij}^{g\mathcal{H}}\right)^2\right)^{-1}}{\sum_{k \neq \ell} \left(1 + \left(d_{ij}^{g\mathcal{H}}\right)^2\right)^{-1}}, \tag{10}$$

$$\frac{\delta d_{ij}^{g\mathcal{H}}}{\delta \mathbf{y}_i} = \frac{4\left(\left(\|\mathbf{c}_j\|^2 - 2\langle \mathbf{y}_i, \mathbf{c}_j \rangle + 1\right) \mathbf{y}_i / \alpha - \mathbf{c}_j\right)}{\alpha\beta\sqrt{\gamma^2 - 1}} \tag{11}$$

Grid Size Optimisation

We fit the grid to be the bounding box of the points for each iteration, shown in figure 3. This is necessary because the initial embedding of t-SNE is concentrated in the centre of the disk, which would result in many cells containing no points. By fitting the grid to the points, the points are more evenly divided over the grid. A result of this is that the size of each grid cell is minimised which decreases the maximum error from summarising the points in a grid cell, which in turn increases accuracy.

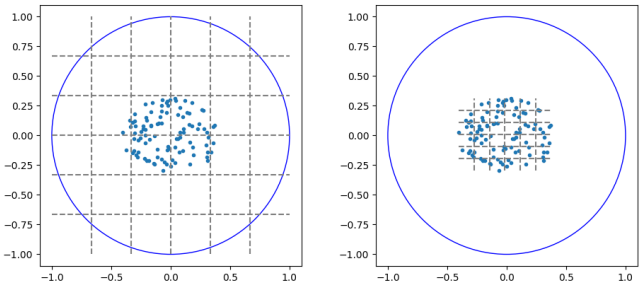

Figure 3: The uniform grid fitting to data concentrated near the centre. Left: before optimisation. Right: after optimisation.

4.2 Proof of Correctness

The uniform grid approximates the exact solution more closely as the number of grid cells increases. As this grows to infinity, the approximation approaches the exact solution. This is because as the number of grid cells increases, more points will be the only point in a grid cell. When this happens, calculating the force to the grid cell will be equivalent to calculating the force to the point directly, since the mean of the grid cell will be exactly equal to the point position. When the number of grid cells is sufficient such that each point is the only point in its cell, the force calculation is equal to the exact solution.

5 Evaluation

This section aims to evaluate different parts of the proposed solution and compare them to previous solutions using analysis and experiments. Specifically, we compare our uniform grid-accelerated t-SNE with the exact t-SNE calculation (no acceleration) and quadtree-accelerated t-SNE. We do this by first analysing and comparing the expected runtime complexity of our algorithm. Then, we will discuss experiments aimed at testing the accuracy and actual runtime of our algorithm and comparing the results with the other algorithms.

5.1 Experimental setup

For the following experiments, we have used four different datasets. The first is MNIST, which contains 70,000 handwritten numbers in the form of small images. The others all contain data obtained from single-cell RNA sequencing. The Planaria and C.Elegans datasets contain gene expression atlases. The Myeloid dataset contains synthetic data instead [25]. Table 1 includes more information about each dataset.

Name	Data Type	# Points	# Dim.	# Cl.
MNIST	images	70,000	784	10
MYELOID8000	single-cell	8,000	11	5
PLANARIA	single-cell	21,612	50	51
C ELEGANS	single-cell	89,701	20,222	37

Table 1: Dataset Information

The exact settings for the running setup are equal to the default settings used in the previous research [4]. This means that we perform 250 steps of early exaggeration, meaning the attractive forces between the points are multiplied by a constant value of 12. After this, we perform up to 750 steps of regular gradient descent, which is stopped earlier if a point moves too far from the centre of the disk.

All experiments are performed on a computer with an Intel Xeon E3-1240v3 at 3400 MHz, 32GB of DDR3 RAM at 1600 MHz and an NVIDIA GeForce GTX 1080 at 1607 MHz with 8GB of GDDR5X RAM and 2560 CUDA cores.

5.2 Accuracy of the Proposed Solution

To test the accuracy of the proposed solution, we use a precision metric which measures neighbourhood preservation. It gives a percentage of the points for which the $1 \leq n \leq 30$ closest neighbours in high-dimensional space are also in the 30 closest neighbours in the final embedding. For a perfect embedding, we would expect a precision of 1.0 for every n, but this is often not possible, which is why we compare to the precision of the exact calculation performed on the GPU and the previous quadtree solution. We use a GPU-based exact calculation since it is infeasible to run the previous CPU-based exact calculation for high point counts due to its runtime.

Effect of Grid Size on Accuracy

For this experiment, we look at our algorithm at different grid sizes, where we show the average of the precision values at each grid size. This will help us make an informed

choice about the number of grid cells we should use to get a good trade-off between runtime efficiency and accuracy. As can be seen in Figure 4, the additional precision gained by increasing the grid size levels out quickly. It also shows that an increasing number of points does not require an increasing number of grid cells to get optimal precision. Instead, around 10,000 to 15,000 grid cells seems to be optimal for all tested point counts, and more grid cells do not provide greater precision. Appendix B shows this experiment for the other datasets. Notable is that this number of grid cells seems optimal for these datasets too. A possible explanation for this is that around this level is where the grid gets so fine that additional granularity would not increase the precision more, as the error is already so low due to the small size of the grid cells. The optimal number of grid cells may still differ for other datasets, meaning that this number should be found per dataset for optimal performance and accuracy trade-off.

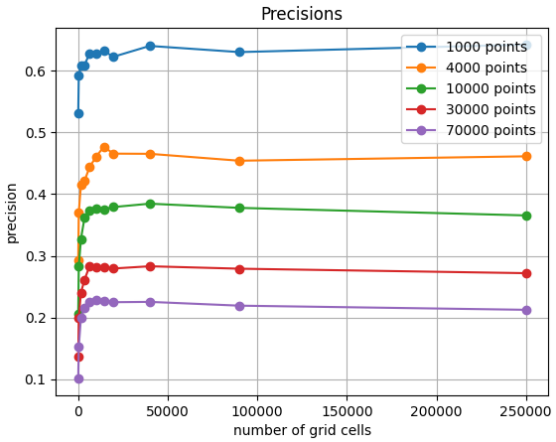

Figure 4: Average precision achieved of embedding different size datasets at different grid sizes.

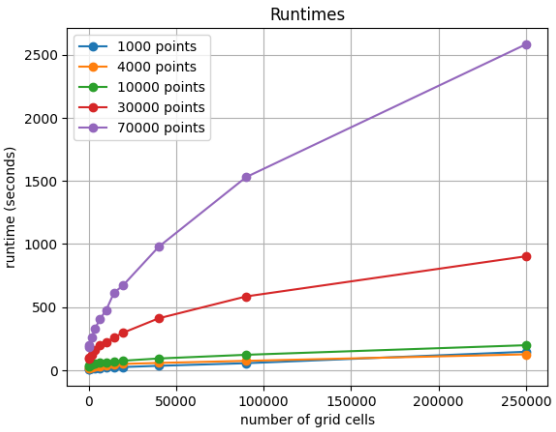

Figure 5: Runtimes of embedding different size datasets at different grid sizes.

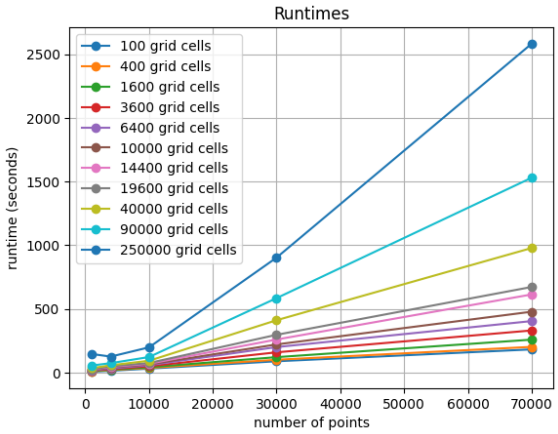

Figure 6: Runtimes of embedding using different numbers of grid cells.

Comparing the Accuracy of the Proposed Solution

This experiment aims to compare the accuracy of the uniform grid-based solution with the exact and quadtree-based solution in terms of accuracy. We do this by computing an embedding of 4 different datasets with every algorithm and comparing the recall/precision graphs of each. Figure 8 shows the actual embeddings created for the MNIST dataset. The other embeddings can be found in Appendix C. Figure 7 shows that our solution results in higher precision and recall for every single dataset that has been tested. This result is unexpected because the uniform grid is used as an approximation of the exact solution. It is possible that our solution is less susceptible to getting stuck in suboptimal local minima, as our rougher method of approximation may cause us to overshoot small local minima.

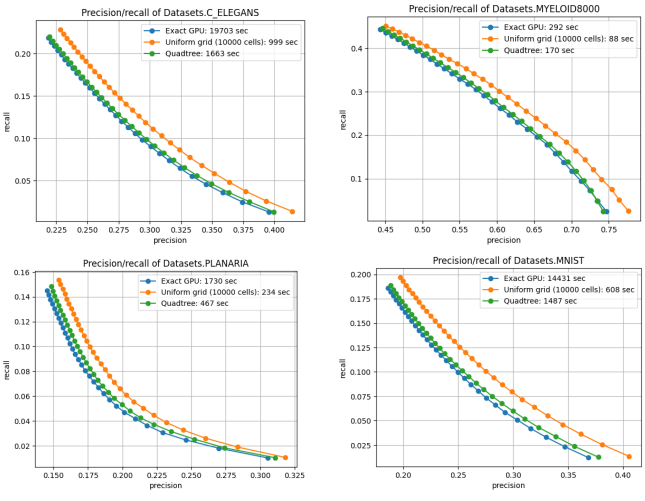

Figure 7: Comparison of the recall/precision graph of each algorithm with four different datasets.

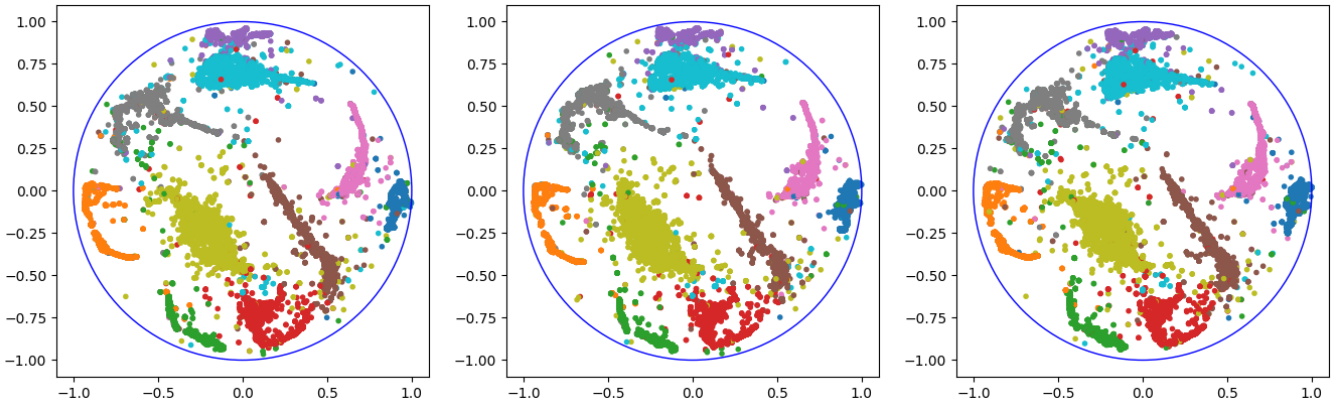

Figure 8: Embeddings of the MNIST dataset (70,000 data points) produced by the exact solution (left), quadtree-based solution (centre), uniform grid-based solution (right).

5.3 Time Acceleration

Now we will focus on comparing the theoretical time complexity of the proposed solution, and experimentally verifying the findings. We are interested in the total time taken to create a full embedding, as this is the closest to a realistic usage scenario.

Asymptotic Time Complexity

We will now discuss the asymptotic time complexity of our proposed solution, specifically the gradient calculation, as this is where our solution differs from previous work. The time complexity depends on two variables. Let n be the number of points for which we are computing an embedding, and m the number of grid cells used. The gradient calculation is performed in two main steps. First the generation of the uniform grid, and second the calculation that uses it.

Time Complexity of Grid Generation Generating the grid is done in $O(n + m)$ time. First, it requires looping over all points to find the bounding box of the data, which becomes the bounding box of the grid. Then, it requires looping over all points to find the grid cell each point belongs to, which is done in $O(n)$, as finding the grid cell for a single point is done in constant time. Secondly, it requires calculating the centres for each grid cell, which is done in $O(m)$ time.

Time Complexity of Gradient Calculation Finding the gradient for a single point is done in $O(m)$ time, as our solution loops over each grid cell and calculates the negative force to it, which is done in constant time. Since the gradient calculation must be performed for each point, we find that calculating the negative forces can be done in $O(n \cdot m)$ time. Previous work [4] has already shown that calculating the positive forces can be done in $O(n)$, provided the probabilities matrix is sparse.

In total, this means that the runtime complexity of the gradient calculation using the uniform grid is $O(n \cdot m)$.

Effect of Grid Size on Runtime

This experiment focuses on finding to what capacity the chosen grid size affects the runtime of the algorithm. Theoretically, the runtime should increase linearly with the number of grid cells. Figure 5 shows that for the grid sizes that were tested, the runtime seems to follow a rather logarithmic pattern. This may be because we are not able to leverage the full power of the GPU at low grid sizes, meaning part of the larger grid size will be compensated for by the GPU taking on more load.

Comparison to Previous Solutions

We experimentally compare the runtime of our solution with the GPU-based exact solution and the previous quadtree-based solution. We do this by running each on different numbers of data points and plotting the average runtime of 3 embeddings of each method at each point count. Figure 9 shows the results of this experiment. We can see that our method is much faster at every number of points and grows more slowly, meaning it will be relatively faster than the quadtree at higher numbers of points. This is because at a set number of grid cells, the runtime complexity is linear in the number of points, while the quadtree-based solution is log-linear in the number of points.

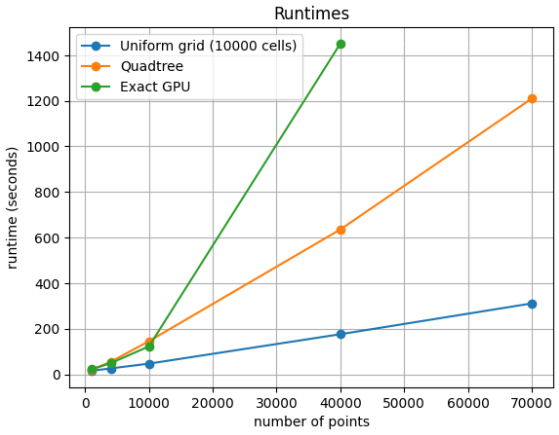

Figure 9: Runtime comparison between different methods at different numbers of data points.

6 Responsible Research

In this section, we will discuss the ethical aspect of this research. Specifically, we will go over transparency in the algorithm and its results in Section 6.1. We will also discuss the safety and privacy of the data used in Section 6.2. We show how we make sure that our research is reproducible by others in Section 6.3. Finally, we discuss the use of Artificial Intelligence (AI) and Large Language Models (LLMs) throughout the research and writing of this paper in Section 6.4.

6.1 Ethical Considerations

This section will discuss the ethical considerations regarding the understandability and interpretability of both the system and its results.

Transparency of Algorithm Operation

All steps of the algorithm are explained in this paper, and in more detail in the *readme* of the publicly available GitHub repository[1]. The code is also fully documented and elaborated on in comments in the code.

Iterative improvements of the algorithm and important decisions made in the process are always based on the results of extensive experiments that test any options. Through the use of GitHub, every iteration of the algorithm's development is publicly available, including a small description of each iteration.

Implications of Incorrect Usage

It needs to be clear to any user of the algorithm that generally no perfect embedding of higher-dimensional data can be made in 2 dimensions. This means that while the embedding is generally able to show neighbourhoods in higher-dimensional data, the embedding may not always be fully representative of the dataset. Incorrect interpretation of the embeddings could result in misinterpretation of the overall dataset, leading to possible incorrect conclusions. The algorithm should be used exclusively for exploratory data analysis and visualisation, such as finding underlying structures or relationships in the dataset.

The aforementioned *readme* file also includes a section about interpreting results from the algorithm which aims to reduce the impact and frequency of result misinterpretation.

6.2 Data Privacy and Confidentiality

This research and its experiments contain no data that is not publicly available, and no personal data has been gathered of any person. No data contains anything that can be traced back to any particular person.

6.3 Reproducibility of Methods

We have taken several steps to ensure that all results and conclusions in this paper can be verified by peers, as we believe this is very important to any scientific research.

Code and Data Availability

The code for this project including the experimental setup is available on *GitHub*[2]. The datasets are not included, but these can be downloaded using the links in the *readme*.

Reproducibility

Running the experiments requires a CUDA-compatible NVIDIA graphics card and an installation of CUDA, due to the use of CUDA for implementing GPU acceleration. The *readme* file of the GitHub shown above outlines the steps for running the program on any dataset, as well as how to run the experiments. All Python dependencies required can be installed by anyone for free, as shown in the *setup* section of the *readme*.

Reliability and Validity of Results

Each experiment that can vary in results due to external circumstances, such as the state of the computer or variations in the dataset sample, is run multiple times and the results are averaged. To curb the remaining influence of the computer state, we run no other programs than those required for running the experiments. When testing the difference between two or more options, we always make sure to only change the thing under testing between each experimental run.

The algorithms are fully tested on 4 full datasets, so that no algorithm is favoured due to higher suitability for a specific dataset. The result of this can be found in section 5.

6.4 Use of LLMs During Research and Writing

LLMs such as ChatGPT are very popular for writing and researching but come with risks such as uncited information. Transparency of the use of such models in research may hold researchers more accountable and prevent misuse.

Throughout this research, ChatGPT has been used in the following ways:

1. Helping with writing code.

   It has helped me write and debug code faster. This was particularly useful in finding correct syntax when working with new technologies, specifically Cython and CUDA. Note that in no instance has any LLM written a large piece of code to be used directly in the final codebase. Most use was for achieving a quick prototype or base of a part of the code, which is rewritten later. All code written by any LLM is thoroughly manually checked and verified.

2. Helping brainstorm ideas.

   ChatGPT has also been used to brainstorm ideas for improving and testing the code. For example, ideas on how to optimise the code or how the algorithm could be improved.

3. Helping with ideas in writing.

   While writing, I found the use of ChatGPT useful for coming up with important points for items that should be touched on in a section. This was only done for a select few sections, specifically the Abstract, the

[1] https://github.com/Milan7843/hyperbolic-tsne

[2] https://github.com/Milan7843/hyperbolic-tsne

Introduction, the Uniform Grid, and the Responsible Research sections. Under **no** circumstance has any text written by any LLM been used directly in the final text.

4. Writing BibTeX references.

   Some websites do not offer simple access to BibTeX references of an article, where ChatGPT can come in and generate it from the information on the page. After verification, we used such references in this paper.

5. Providing quick feedback on writing.

   Finally, we used ChatGPT for the purpose of getting feedback on a newly written section. This provided us with insight into the flow of the text, and if any important points were missing. This was not done for all sections.

A full list of all prompts made to an LLM for the purpose of this research, including any results from them can be found in Appendix A.

7 Conclusions and Future Work

In this paper, we have presented an alternative to the previous data structure used to accelerate embedding high-dimensional data in a 2-dimensional hyperbolic disk using t-distributed Stochastic Neighbour Embedding (t-SNE). We have shown that the uniform grid is a viable fit for accelerating the gradient calculation of this algorithm. Our experiments have shown that our solution not only accelerates this important step of t-SNE but also produces embeddings with better neighbourhood preservation. At a set number of grid cells, the runtime complexity of the proposed solution is linear in the number of points, while the previous quadtree-based solution is log-linear in the number of points.

A possible improvement could be to use a non-uniform grid instead, which may provide better accuracy at the cost of added complexity of setting up the grid. More research could be done into the application of this uniform grid structure to t-SNE in non-hyperbolic space. While the arguments regarding the increase of runtime performance will still hold, it would be interesting to see if the accuracy remains better than existing methods. It is possible that the increase in accuracy is only observed in hyperbolic space.

References

[1] Ayush Soni, Akhtar Rasool, Aditya Dubey, and Nilay Khare. Data mining based dimensionality reduction techniques. In *2022 International Conference for Advancement in Technology (ICONAT)*, pages 1–8, 2022.

[2] Radoslav Sarkar. Low distortion delaunay embedding of trees in hyperbolic plane. In Marc van Kreveld and Bettina Speckmann, editors, *Graph Drawing*, volume 7034 of *Lecture Notes in Computer Science*, pages 355–366. Springer, Berlin, Heidelberg, 2011.

[3] Laurens van der Maaten. Accelerating t-sne using tree-based algorithms. *Journal of Machine Learning Research*, 15(93):3221–3245, 2014.

[4] Martin Skrodzki, Hunter van Geffen, Nicolas F. Chaves de Plaza, Thomas Höllt, Elmar Eisemann, and Klaus Hildebrandt. Accelerating hyperbolic t-sne, 2024.

[5] Moritz von Looz, Henning Meyerhenke, and Roman Prutkin. Generating random hyperbolic graphs in subquadratic time. pages 467–478, 12 2015.

[6] Laurens van der Maaten and Geoffrey Hinton. Viualizing data using t-sne. *Journal of Machine Learning Research*, 9:2579–2605, 11 2008.

[7] Dmitry Kobak and George C. Linderman. Initialization is critical for preserving global data structure in both t-sne and umap. *Nature Biotechnology*, 39:156–157, Feb 2021.

[8] Josh Barnes and Piet Hut. A hierarchical o(n log n) force-calculation algorithm. *Nature*, 324(6096):446 – 449, 1986.

[9] Ilia Marchevsky, Evgeniya Ryatina, and Alexandra Kolganova. Fast barnes–hut-based algorithm in 2d vortex method of computational hydrodynamics. *Computers and Fluids*, 266, 2023.

[10] Salim Rukhsar and Anil Kumar Tiwari. Barnes–hut approximation based accelerating t-sne for seizure detection. *Biomedical Signal Processing and Control*, 84, 2023.

[11] E. Ryatina and A. Lagno. The barnes - hut-type algorithm in 2d lagrangian vortex particle methods. volume 1715, 2021.

[12] Christer Ericson. *Real-Time Collision Detection*. Morgan Kaufmann Publishers, San Francisco, CA, 2005. Chapter 7, Spatial Partitioning, pages 285–299.

[13] Sol Ha, Nam-Kug Ku, Myung-Il Roh, and Kyu-Yeul Lee. Cell-based evacuation simulation considering human behavior in a passenger ship. *Ocean Engineering*, 53:138 – 152, 2012.

[14] Rongzhen Xiao, Zhiping Wang, Changsheng Zhu, and Li Feng. Simulation of atypical dendrite growth in ni-cu binary alloy with phase-field method. *Journal of Computational and Theoretical Nanoscience*, 9(9):1495 – 1499, 2012.

[15] Tsz Ho Wong, Geoff Leach, and Fabio Zambetta. Virtual subdivision for gpu based collision detection of deformable objects using a uniform grid. *Visual Computer*, 28(6-8):829 – 838, 2012.

[16] Shuzhi Su, Gang Zhu, and Yanmin Zhu. An orthogonal locality and globality dimensionality reduction method based on twin eigen decomposition. *IEEE Access*, 9:55714 – 55725, 2021.

[17] Long Cheng and Chenyu You. Hybrid non-linear dimensionality reduction method framework based on random projections. page 43 – 48, 2016.

[18] Aleix M. Martinez and Avinash C. Kak. Pca versus lda. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(2):228 – 233, 2001.

[19] S.T. Roweis and L.K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323 – 2326, 2000.

[20] Etienne Becht, Leland McInnes, John Healy, Charles-Antoine Dutertre, Immanuel W H. Kwok, Lai Guan Ng, Florent Ginhoux, and Evan W. Newell. Dimensionality reduction for visualizing single-cell data using umap. *Nature Biotechnology*, 37(1):38 – 47, 2019.

[21] Marek Orliński and Norbert Jankowski. Fast t-sne algorithm with forest of balanced lsh trees and hybrid computation of repulsive forces. *Knowledge-Based Systems*, 206, 2020.

[22] Martin Burtscher and Keshav Pingali. An efficient cuda implementation of the tree-based barnes hut n-body algorithm. *GPU Computing Gems Emerald Edition*, 12 2011.

[23] Nicola Pezzotti, Julian Thijssen, Alexander Mordvintsev, Thomas Höllt, Baldur Van Lew, Boudewijn P.F. Lelieveldt, Elmar Eisemann, and Anna Vilanova. Gpgpu linear complexity t-sne optimization. *IEEE Transactions on Visualization and Computer Graphics*, 26(1):1172 – 1181, 2020.

[24] Mark Van De Ruit, Markus Billeter, and Elmar Eisemann. An efficient dual-hierarchy t-sne minimization. *IEEE Transactions on Visualization and Computer Graphics*, 28(1):614 – 622, 2022.

[25] Jan Krumsiek, Carsten Marr, Timm Schroeder, and Fabian J. Theis. Hierarchical differentiation of myeloid progenitors is encoded in the transcription factor network. *PLoS ONE*, 6(8):e22649, 2011. Received: March 16, 2011; Accepted: June 27, 2011; Published: August 10, 2011.

A Prompts Used to Interact with LLMs and Their Impact

1. *"how do i check if a pycuda resource handle is still valid so i can safely free it"*

   From this prompt I got no concrete answer but got inspired for a solution based on a try/catch statement that worked.

2. *"what else is there to mention about barnes hut [my section about Barnes-Hut]"*

   From this prompt I got a list of ideas for items to include in the section. I did not implement any of these.

3. *"is there any mathematical notation for the size of a set?"*

   Helped me find the notation: $|m|$.

4. *"what else can i mention [my section on the uniform grid]"*

   From this prompt I got a list of ideas for items to include in the section. From this I got the idea of talking about how the uniform grid is used for collision detection and comparing it to the non-uniform grid.

5. *"what else can i say here in my research? [my section on Data Privacy and Confidentiality]"*

   Nothing new was added fom this prompt.

6. *"I'm doing research into embedding high-dimensional data into a 2d poincare hyperbolic disk. specifically, im accelerating part of the t-SNE algorithm used for this by making a new data structure for the barnes-hut acceleration. Now, I need to write a 'Related research' section that notes 2 or so specific topics and references existing solutions for it. could you start off by giving me 2 topics? No need to write anything else yet, just the topics"*

   This prompt gave me the following two topics: "Hyperbolic Embedding Techniques" and "Barnes-Hut Algorithm in t-SNE". I ended up going with "Dimensionality Reduction Techniques" and "t-SNE Acceleration" instead.

7. *"write the subsections that should be in this section (only the titles)*
   \section{Responsible Research} Reflect on the ethical aspects of your research and discuss the reproducibility of your methods.
   Note that although in many published works there is no such a section (it may be part of some meta-information collected by the journal, or part of the discussion section), we require you to think (and report) about this as part of this course."

   Followed up with *"can you write some bullet points for each section that i would need to touch on"*

   From this, I got the ideas to write about Transparency of Algorithm Operation, Implications of Incorrect Usage, Data Privacy and Confidentiality, Reproducibility of Methods, Code and Data Availability, Reproducibility, Reliability and Validity of Results. Further follow up

gave me some ideas for what to write in each of these sections.

8. *"numpy generate integer from 0 to 99"*
   Gave me code to generate the integers [0, 99].
9. *"how do i adjust this function to load data of the following form? [data]*
   X = np.loadtxt(str(Path.joinpath(full_path, "amazon_data.txt")), delimiter=" ")"
   I needed to load a dataset in a different format than what the current loader could do. This prompt ended up giving me no real answers and I solved it myself.
10. *"when using two columns in latex, how do i make a figure go over both columns at the top of a page?"*
    Gave me LaTeX code to do this.
11. *"can you intuitively explain the initial embedding process of t-sne"*
    This prompt gave me more insight into t-SNE.
12. *"i have a gpu program that has a curve that first climbs fast from 90 to 120 with higher iteration count in a gpu program, but then increases much less fast from 2000 iterations onward, how could this be?"*
    This prompt gave me more insight into why this phenomenon could be occuring, but no definitive answers.
13. *"tell me why poincare disk hyperbolic space would be good for embedding arbitrary size and exponential data (e.g. a tree structure)"*
    This prompt gave me more insight, but no information from it has been used directly.
14. *"precision recall graph explain"*
    While short, this prompt gave me insight into the idea behind a precision recall graph, and how it can be interpreted. This is important, as such a graph is a good way of showing the performance of an embedding algorithm, and should be interpreted correctly.
15. *"how do i make the runtimes text spread over all columns [LaTeX code]"*
    This prompt gave me the right LaTeX code to format part of a table correctly.
16. *"how do i give this latex image a number and a caption \includegraphics[scale=0.6]{precision_40k.png}"*
    This prompt taught me how to properly place a figure into my paper.
17. *"define a python function that uses plt to plot some points connected by a line (x axis range from 1 to n with n data points)"*
    From this prompt, I got a decent first prototype for a function that I needed. I did need to refine it and to make it work correctly with my data.
18. *"im running this code on the gpu, but i think thread divergence makes it run really really slow. How could i make it run faster? [my GPU code]"*
    No new insights were gathered.
19. *"in latex, say i have x \in R^2, how would you denote taking the x coordinate or y coordinate?"*
    This prompt got me the idea to make x_1 and x_2 the x and y coordinate respectively.
20. *"can you help me figure out why this is so slow? [GPU code for calculating positive forces]"*
    From this prompt I gathered that the most probable reason for the slow code would be the thread divergence and irregular memory access patterns.
21. *"roughly how long should it take to transfer 1.2MB of data to the gpu?"*
    From this prompt, I got the insight that the runtime of the operation of transferring so little data from the CPU to the GPU should be negligible compared to other code. Therefore, I ruled this out as a potential bottleneck.
22. *"in cython, how can i just get the time in seconds"*
    After some follow-up questions, I gathered no new insight and decided on a different approach.
23. *"how to run nsight on a python scrip that uses pycuda"*
    No insights were gathered.
24. *"can you write some code that makes it so that the 'points' array are re-ordered according to the 'result_indices' array, you may assume that the 'result_indices' contains all indices exactly once"*
    From this prompt I got some code for a small experiment to identify a potential bottleneck. The code generated is since no longer in use.
25. *"here is the kahan summation algorithm for decreasing floating point errors [algorithm]*
    could you implement it on my algorithm, specifically for the summation of thread_sQ [my own code]"
    This prompt gave me code for a small experiment to see if this Kahan Summation algorithm could potentially solve a precision error I was encountering. It did not end up making a significant difference and the code generated is no longer in use.
26. *"i have an algorithm that calculate forces between some points and i also implemented in on the gpu using cuda, but theyre not giving the exact same values? what could be the cause? [values]"*
    This prompt gave me insight into the slight differences in computation on the CPU and the GPU and allowed me to keep searching for other solutions.
27. *"in cython, given a 2d array of points in poincare hyperbolic space, calculate the einstein midpoint of these points (and return the ansnwer in poicnare hyperbolic space)"*
    This prompt ended up giving me no particularly good solution and I ended up simply solving it myself.
28. *"give me the formula for converting from a point on poincare disk to euclidian space"*
    This prompt gave me no answer, since this is also not possible for the Poincaré disk model.

29. *"reserach plan review!! [my research plan]"*

    Ended up giving me some general guidelines for improving it, no specific points.

30. *"i have an array of n*n float values, and i want you to plot them in python with plt in 3f"*

    This prompt gave me good code for plotting such values, necessary for displaying the maximum distance values in the grid.

31. *"in python, given a 1d array containing alternating x and y points, make a 2D array that contains the points"*

    This gave me good code to perform this operation.

32. *"in numpy python, write a function that generates an array of n points with x and y in (-1, 1)"*

    This prompt gave me good code, necessary for an experiment.

33. *"write a cython function that takes an array of 2d points, and divides them over n*n grid squares. assume that the grid squares are in order from the top left and the points x and y are in (-1, 1)"*

    This prompt gave me a very rough outline for this function. In the final product, I rewrote every single part to make it compatible with my data and optimise it. It did give me much insight into what *Cython* syntax I could use for certain subtasks of this problem.

34. *"if i had a cuda program that takes some double pointer, and each seperate thread and block will add stuff to it, will that go right?"* followed up by *"i need to have a single value that multiple threads can write to (only +=), how would you do this?"*

    This prompt gave me the idea of using the 'addAtomic' function in CUDA to prevent mutiple threads attempting to access the same data at the same time.

35. *"how to time a function in python"*

    This prompt gave me good code for timing a function in Python.

36. *"i have a python script in hyperbolicTSNE, and another python script 'gpu.py' in hyperbolicTSNE/hyperbolic_barnes_hut . how do i import the gpu script"*

    After some back and forth I managed to get this to work, but not directly from an answer to any of the prompts, rather from further research outside of the LLM.

37. *"import python into cython"*

    This prompt showed me how to import Python code into a Cython file.

38. *"can you help me understand this code? [some Cython code from the previous codebase]"*

    This prompt gave me insight into the Cython code and helped me understand it and Cython in general better.

39. *"what could indptr be? [code that uses indptr as a variable]"*

    This prompt allowed me to understand what the code I gave was doing exactly. This code was from the project before I started working on adding my own implementation.

40. *"say i have a bunch of points, and i would like to use some data structure on it, such that i can find the nearest neighbours. i would also like to parallelize the data structure creation and traversal on the gpu. which structure should i pick, between k-d tree and uniform grid? consider all factors, also ease of implementation and speed (but also other factors not just these)"*

    This prompt gave me insight into the advantages and disadvantages of both methods and allowed me to make an informed decision between them when choosing the topic of my research.

B Additional Experiments Regarding Number of Grid Cells

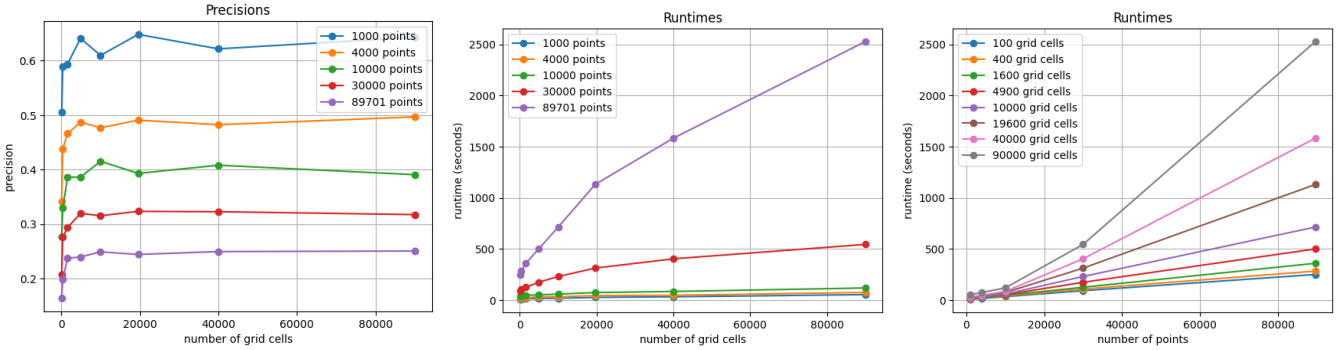

Figure 10: Left: Average precision of embeddings of the C_Elegans dataset (89,701 data points) at differing point counts. Centre: Runtimes of embeddings of the C_Elegans dataset at differing point counts. Right: Runtimes of embeddings of the C_Elegans dataset at differing numbers of grid cells.

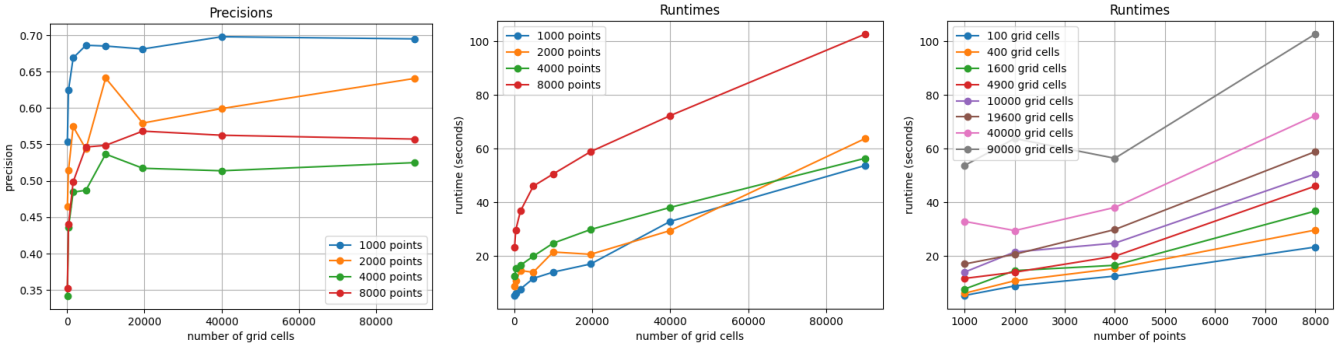

Figure 11: Left: Average precision of embeddings of the Myeloid dataset (8,000 data points) at differing point counts. Centre: Runtimes of embeddings of the Myeloid dataset at differing point counts. Right: Runtimes of embeddings of the Myeloid dataset at differing numbers of grid cells.

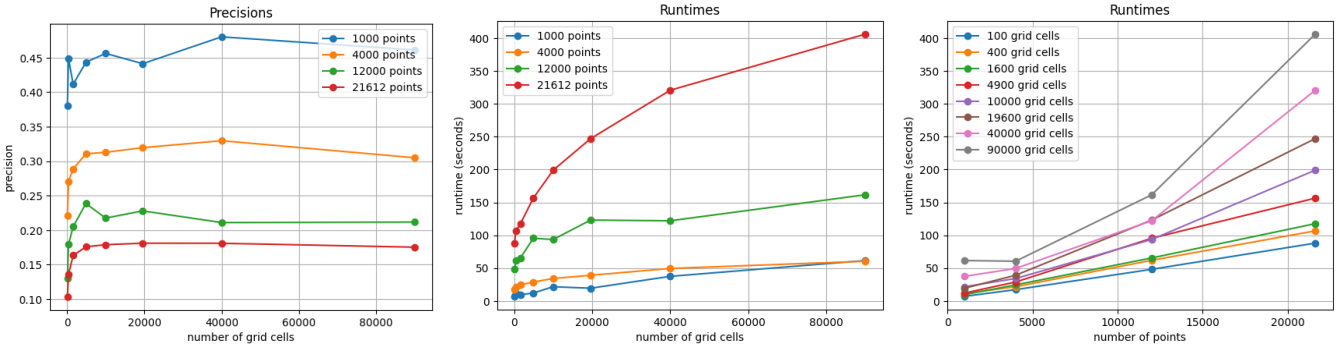

Figure 12: Left: Average precision of embeddings of the PLANARIA dataset (21,612 data points) at differing point counts. Centre: Runtimes of embeddings of the PLANARIA dataset at differing point counts. Right: Runtimes of embeddings of the PLANARIA dataset at differing numbers of grid cells.

C Additional Embeddings

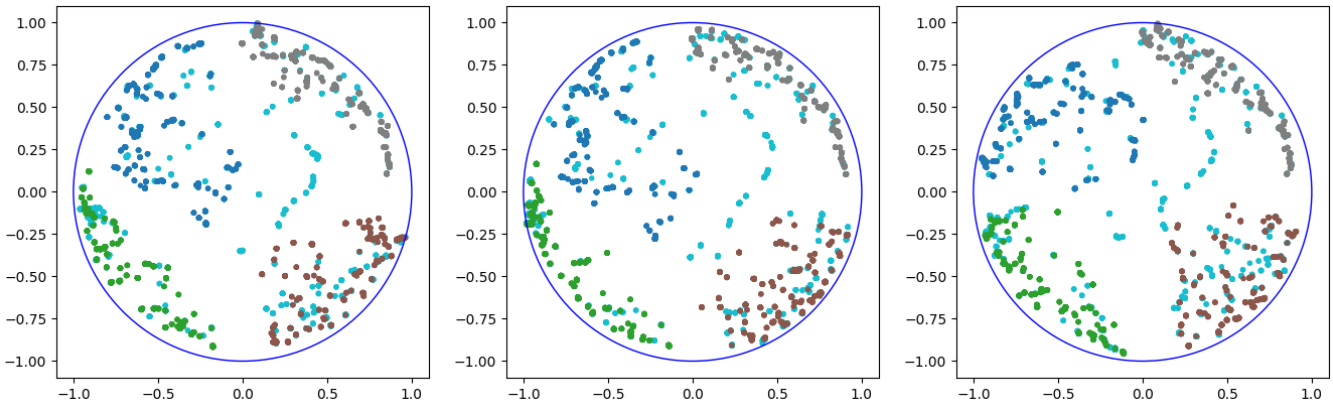

Figure 13: Embeddings of the Myeloid dataset (8,000 data points) produced by the exact solution (left), quadtree-based solution (centre), uniform grid-based solution (right).

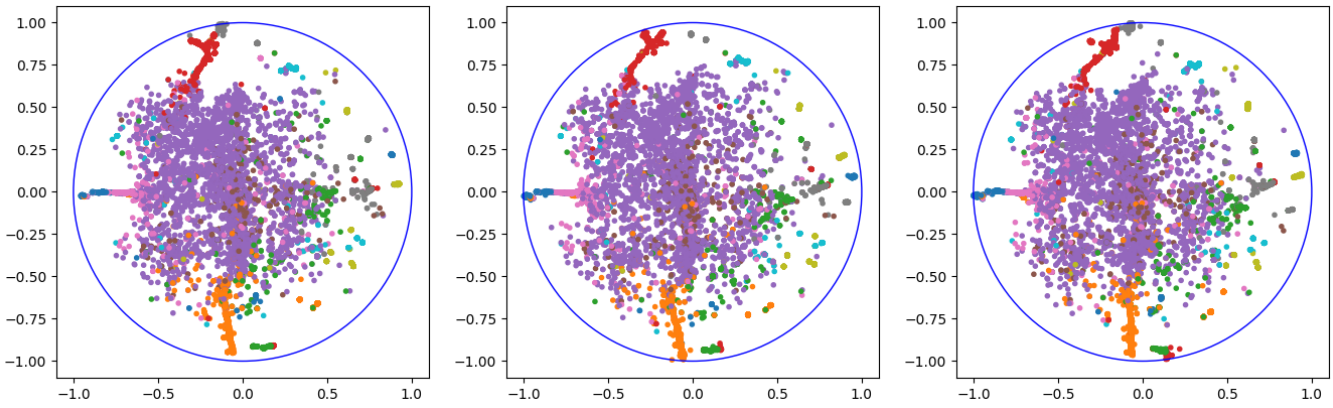

Figure 14: Embeddings of the PLANARIA dataset (21,612 data points) produced by the exact solution (left), quadtree-based solution (centre), uniform grid-based solution (right).

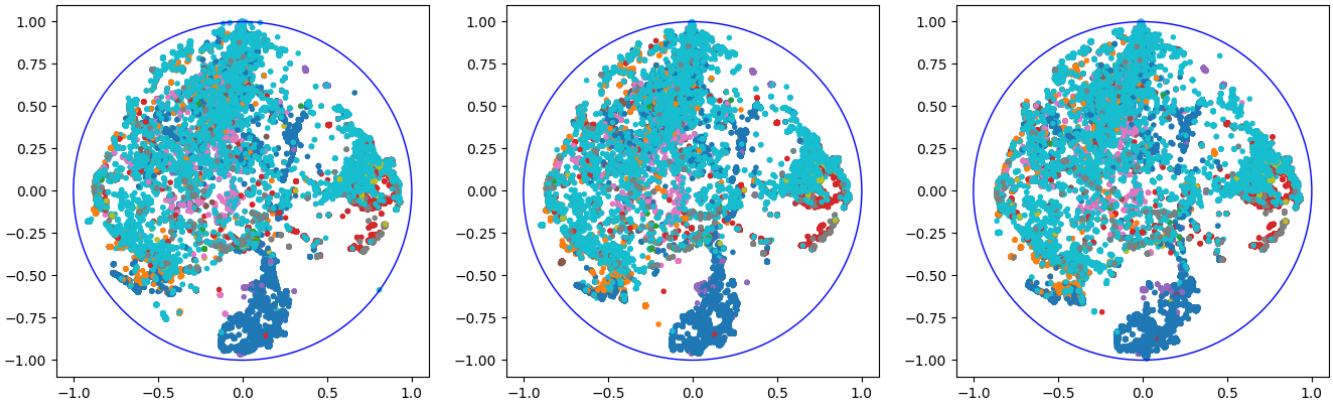

Figure 15: Embeddings of the C_Elegans dataset (89,701 data points) produced by the exact solution (left), quadtree-based solution (centre), uniform grid-based solution (right).