Fast Parallel Computation for Random Walk based t-SNE

Takeshi MIKI†, Yasuhiro FUJIWARA††, and Hideyuki KAWASHIMA††

† Keio University
5322 Endō, Fujisawa-shi, Kanagawa 252–0882, Japan
†† NTT Communication Science Laboratories,
1–5–1 Ootemachi, Chiyoda-ku, Tokyo, 100–8116, Japan
E-mail: †t.miki@keio.jp, ††yasuhiro.fujiwara@ntt.com, †††river@sfc.keio.ac.jp

Abstract Effective dimension reduction techniques are crucial in the era of big data. In this paper, we introduce an optimized version of the renowned dimension reduction technique, random walk-based t-SNE, by addressing its traditionally significant computational cost. Our proposed method incorporates parallel computing and Modified Cholesky factorization to enhance processing efficiency, without sacrificing the quality of dimension reduction. The experimental results demonstrate a substantial reduction in execution time while maintaining the high standard of dimension reduction quality. This method has the potential to accelerate the adoption of t-SNE across various data-intensive domains.

Key words t-SNE, Parallization, Efficient

1 Introduction

In our data-driven world, the volume of high-dimensional data that we handle is continuously expanding [2]. Consequently, the importance of data visualization has been steadily increasing. The practice of visualizing high-dimensional data often necessitates a reduction of its dimensions to two or three. Numerous dimension reduction algorithms have been developed for this purpose, including UMAP [12], PCA [9], and PLS [13]. Among those methods, t-SNE (t-Distributed Stochastic Neighbor Embedding) is one of the most common and widely accepted approaches [15].

t-SNE is a non-linear dimension reduction algorithm that portrays the similarity between data points by conditional probabilities. Subsequently, it tries to minimize the Kullback-Leibler (KL) divergence of the conditional probabilities between the high-dimensional and lower-dimensional data points. In doing so, t-SNE effectively preserves both the global and local structural features. Due to its remarkable performance and simplicity, t-SNE finds the following diverse applications:

Cancer Detection: Mass Spectrometry Imaging (MSI) is a technique that simultaneously provides the spatial distribution of biomolecules. This information is necessary for identifying diagnostic and prognostic biomarkers in the context of cancer research [1]. While MSI offers rich insights into the intricate molecular landscape within cancer, its data complexity poses significant challenges. t-SNE, with its ability to reduce data dimensionality while preserving both global and local structures, is a suitable solution. In practice, the application of t-SNE involves the detection of distinct clusters. This has proven particularly beneficial for identifying gastric and primary breast cancer patients [1].

Playlist Generation: In the domain of playlist generation, t-SNE has been introduced as a modeling approach. Specifically, each song is represented as a 34-dimensional data, and t-SNE is used to transform it into a two-dimensional data. The two-dimensional output of the model will be used for projecting and rendering a song catalogue onto a plane. In this context, it has outperformed previous methodologies, such as PCA [10].

As shown in previous examples, t-SNE is a popular technique used in various fields. However, its main drawback is the significant computational cost. The problem arises from the need to calculate pairwise similarities between all data points to compute the Kullback-Leibler divergence between the original data points and the embedded data points. Additionally, t-SNE computes pairwise similarities for all embedded data points in each iteration when updating the embedded data using the gradient descent method. This leads to a quadratic increase in computational complexity [14].

In order to solve this computational bottleneck, random walk-based t-SNE has been introduced. This method selects a subset of landmark points in a randomized manner, reducing the number of data points that need to be visualized. Additionally, random walk-based t-SNE uses the k-nearest neighbor graph to approximate distances between

Table 1 Definition of each symbols

Symbol	Definition
N	Number of data-points
M	Number of dimensions in the original data
m	Number of dimensions in the embedded data
n	Number of landmark points
$\boldsymbol{X}$	Matrix of the original data
$\boldsymbol{Y}$	Matrix of the embedded data
x_i	i-th data-point of the original data
y_i	i-th data-point of the embedded data
p_{ij}	Similarity of the original data
q_{ij}	Similarity of the embedded data
$\mathbf{L}$	Graph Laplacian
k	Number of neighbors in *knn* graph

these points efficiently.

However, despite these optimizations, time complexity still remains a substantial challenge. This complexity stems from the need to construct and utilize the k-nearest neighbor graph, which is a critical step in the random walk-based approach for estimating pairwise distances and similarities among data points.

In this paper, we propose a multi-core version of the modified random walk-based t-SNE. The implementation was done in C++ and leveraged OpenMP to execute the code in parallel across multiple cores. The parallelization was done with thread counts ranging from 1 to 64.

The paper is structured as follows: Section 2 provides an overview of the foundational concepts, offering a brief explanation of t-SNE and random walk-based t-SNE. Section 3 introduces the methodology developed for this research. Section 4 discusses the experiments conducted and the results earned. Sequentially, section 5 focuses on related work. The conclusion and discussion drawn from the research is provided in sections 6 and 7.

2 Preliminaries

2.1 t-SNE

This section will explain the background and mathematical definitions related to t-SNE. First of all, t-SNE is a dimensional reduction technique used for data visualization. It aims to reduce the dimension of high-dimensional data points and create a lower-dimensional representation while preserving their pairwise similarities. Specifically, t-SNE reduces the dimension of the original data $\boldsymbol{X}$, represented as $N \times M$ matrix, into a lower-dimensional matrix $\boldsymbol{Y}$, represented as $N \times m$ embedded matrix, where N, M and m are the number of data-points, number of dimension in the original data and the number of dimension in the embedded data, respectively. This is achieved by minimizing the cost function F, defined as follows:

$$F = \sum_{i \neq j} p_{ij} log \frac{p_{ij}}{q_{ij}} \tag{1}$$

where p_{ij} is the pairwise similarity between $\mathbf{x}_i$ and $\mathbf{x}_j$, where $\mathbf{x}_i$ and $\mathbf{x}_j$ represents the ith and jth element of $\boldsymbol{X}$, respectively. Additionally, q_{ij} represents the pairwise similarity of embedded data, $\mathbf{y_i}$ and $\mathbf{y}_j$, where $\mathbf{y}_i$ and $\mathbf{y}_j$ are the ith and jth element of $\boldsymbol{Y}$, respectively. Pairwise similarity in the high dimensional data, p_{ij} is defined as follows:

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2},\ p_{i|i} = 0 \tag{2}$$

where $p_{i|j}$ is the similarity of x_i to x_j using conditional probability. $p_{i|j}$ is given as follows:

$$p_{j|i} = \frac{exp(-||x_i - x_j||^2/2\sigma^2)}{\sum_{k \neq i} exp(-||x_i - x_k||^2/2\sigma^2)} \tag{3}$$

where σ is the variance of the Gaussian distribution.

Since x_i will not choose x_i as its neighbour, $p_{i|i}$ becomes 0. On the other hand, $p_{i|j}$ may not equal to $p_{j|i}$, however, joint probability of Equation (2) p_{ij} takes the average. Therefore, the pairwise similarity between x_i and x_j becomes symmetrized. The pairwise similarity of y_i to y_j is defined as q_{ij} which is given as follows:

$$q_{ij} = \frac{(1 + ||y_i - y_j||^2)^{-1}}{\sum_{k \neq l}(1 + ||y_i - y_l||^2)^{-1}},\ q_{ii} = 0 \tag{4}$$

Unlike p_{ij} which uses a Gaussian distribution to convert distance into probability [15], q_{ij} uses student-t distribution with single degree of freedom to effectively visualize data. Since the distance from y_i to y_i is 0, $q_{ii} = 0$.

Embedding original data is done by minimization of the cost function using gradient descent method with momentum. The gradient is as follows:

$$\frac{\partial F}{\partial y_i} = 4 \sum_{j \neq i} (p_{ij} - q_{ij})(y_i - y_j)(1 + ||y_i - y_j||^2)^{-1} \tag{5}$$

After obtaining the gradient, $\mathbf{Y}$ should be updated. Gradient update with momentum is given as follows:

$$Y^{(t)} = Y^{(t-1)} + \eta \frac{\partial F}{\partial Y} + \alpha(t) \left(Y^{(t-1)} - Y^{(t-2)} \right) \tag{6}$$

where t is the number of iterations when undergoing gradient descent method, α represents the momentum and η is the learning rate.

Since t-SNE visualizes all embedded data-points, it is required to compute the value of q_{ij} for each data-point pair. Consequently, the computational and memory costs becomes $O(N^2)$. These expensive costs are a major issue, since it becomes infeasible for t-SNE to deal with a big data-set.

2.2 Random walk-based t-SNE

One common solution for high computational cost is to use Random walk-based t-SNE, where it tries to reduce the cost by selecting a certain number of landmark points and only visualizing them [15].

Additionally, unlike original t-SNE, random walk-based t-SNE uses *knn* graph to calculate pairwise similarity analytically. this method starts off by creating *knn* graph for all N data-points with k as the number of neighbours. This *knn* graph is represented using graph Laplacian, denoted as $N \times N$ matrix $\mathbf{L}$, which is defined as

$$\mathbf{L} = \mathbf{A} - \mathbf{W} \tag{7}$$

where $\mathbf{W}$ represents adjacency matrix and $\mathbf{A}$ represents diagonal matrix. When obtaining $\mathbf{W}$, an assumption that the distance between data points x_i and x_j is equivalent to $e^{-||x_i - x_j||^2}$ is made. Additionally, diagonal matrix $\mathbf{A}$ is obtained by diag $(\sum_j W_{1,j}, \sum_j W_{2,j}, \cdots \sum_j W_{N,j})$.

Pairwise similarity p_{ij} can be redefined as the fraction of random walks that starts at landmark x_i and terminates at landmark x_j. Pairwise similarity between all data-points and landmark points could be obtained accurately by solving the following linear system:

$$\mathbf{LP} = -\mathbf{B^T} \tag{8}$$

In this equation, $\mathbf{B^T}$ is $N \times n$ matrix containing columns from adjacency matrix $\mathbf{W}$ corresponding to landmark points, where n is the number of landmark points. It is important to note that $\mathbf{P}$ is a $N \times n$ matrix where pairwise similarity between ith data point and jth landmark point is represented as $\mathbf{P}_{ij}$.

In order to solve Equation (8), Cholesky factorization [8] on Graph Laplacian $\mathbf{L}$ is first conducted, resulting in $\mathbf{L} = \mathbf{CC^T}$, where $\mathbf{C}$ represents the upper-triangular matrix. Sequentially, $\mathbf{Cy} = -\mathbf{B^T}$ is solved using forward substitution and $\mathbf{CP} = \mathbf{y}$ is solved using backward substitution [15].

Once pairwise similarities of landmark points from data points are obtained from $\mathbf{P}$, random walk-based t-SNE would compute the pairwise similarities between embedded data using Equation (4). However, it is important to note that only landmark points are used to visualize, meaning that there are only n embedded data. One should note that embedded data is denoted as $n \times m$ matrix $\mathbf{Y}$, where m is the number of dimension in the embedded space. Gradient descent method is used to update $\mathbf{Y}$.

While the random walk-based t-SNE method offers a potential for acceleration, the challenge involving computational complexity still remains. Specifically, the computation of pairwise similarities q_{ij} among the landmarks scales quadratically with the number of landmarks. Additionally the computational cost for updating embedded matrix $\mathbf{Y}$ has complexity of $O(tn^2)$ where t is the number of iterations in the gradient descent. When using very large datasets, where both the number of landmarks and the iteration count can be enormous, these challenges makes this technology infeasible. To address these issues, our approach focuses on optimizing three key areas: the acceleration of the k-nearest neighbors (*knn*) computation, the efficient calculation of the matrix P representing pairwise similarities, and the updating process for the matrix Y. These enhancements are designed to reduce the time complexity inherent in the original method, making it more applicable for the practical use of large-scale data.

3 Method

3.1 Parallelization of Graph Laplacian

Graph Laplacian $\mathbf{L}$ is calculated using Equation (7). The computational complexity of creating *knn* graph as graph Laplacian is $O(N^2M)$, where M is the number of dimension in the original data. Although this step is only done once, depending on the number of its data and dimension, it could be computationally expensive. Thus, this could be a major issue when dealing with a large data-set.

To address this challenge, we used OpenMP, a standard parallel programming API for C and C++. OpenMP allows us to compute the weight of the edges in parallel. Therefore, the construction of *knn* graph can be divided among multiple processors. Simple implementation is shown in Algorithm 1.

Each row of $\mathbf{L}$ corresponds to a node in *knn* graph. Determining which nodes are connected is done by comparing the weight of the edges and selecting k nearest nodes. In Algorithm 1, since determining each row of $\mathbf{L}$ can be computed independently, it is done in a parallel loop by distributing computation across multiple threads. It is important to note that the weight of the edges are calculated using $e^{-||x_i - x_j||^2}$, which is also computed in parallel. Sequentially, by comparing the weights, we could identify whether a node is k nearest neighbor or not. If a node is not a neighbor then those nodes are not connected, thus the weight will be 0.

In order to efficiently compare the weights, we would create a simple buffer with k spaces and a lock to avoid race condition. The buffer represents the k nearest neighbors. An entry value, which would be the weight W_{ij} of each nodes, would be placed in the buffer if it is empty. If not, by comparing the smallest value of the buffer, each node could either be swapped and replaced in the buffer or rejected meaning that the node is not k nearest neighbor. It is important to note that the smallest value in the buffer always comes in the beginning so that the comparison can be done simply.

Additionally, when a entry value wants to read or write to the buffer, it must acquires the lock in order to avoid race condition. Finally, once each row of $\mathbf{L}$ is computed, they are merged together to create one large matrix $\mathbf{L^T}$.

Algorithm 1: Parallelization of graph Laplacian

Data: $\mathbf{X} = \{x_1, x_2, \ldots, x_N\}$, k= Number of neighbours
Result: Graph Laplacian $\mathbf{L}$
♯ pragma omp parallel for
for *each* x_i **do**
  ♯ pragma omp parallel for
  for *each* x_j **do**
    $W_{ij} = e^{-||x_i - x_j||^2}$
    if x_j is not k-nearest neighbour **then**
      $W_{ij} = 0$
    end
  end
  ♯ pragma omp parallel for
  for *each* x_j **do**
    $A_{ii} += W_{ij}$
  end
end
$L^T = \begin{bmatrix} L_{1,;} & L_{2,;} & \cdots & L_{N,;} \end{bmatrix}$
Return $\mathbf{L}$

3.2 Solving Linear Systems

In order to find pairwise similarity p_{ij} using graph Laplacian, Equation (8) is solved. This is done by decomposing L using Cholesky factorization then using forward and back substitution to get the value of P which is a matrix containing pairwise similarity. However, computational cost for Cholesky factorization is cubic in the number of data points [5]. One solution is to use Modified Cholesky factorization.

Cholesky factorization decomposes positive symmetric matrix $\mathbf{L}$ into lower triangular matrix $\mathbf{C}$ and upper-triangular matrix $\mathbf{C^T}$. Starting from C_{11}, by using Equation (9) and Equation (10), Cholesky factorization solves each value of $\mathbf{C}$. The computational cost for Cholesky factorization would be $\frac{n^3}{3}$ [6].

$$C_{jj} = \sqrt{L_{jj} - \sum_{k=1}^{j-1} C_{jk}^2} \tag{9}$$

$$C_{ij} = \frac{L_{ij} - \sum_{k=1}^{j-1} C_{ik} C_{jk}}{C_{jj}},\ i > j \tag{10}$$

The proposed method uses Modified Cholesky factorization to accelerate the process. The core idea of modified Cholesky factorization is to decompose $\mathbf{L}$ in such way that $\mathbf{L} = \mathbf{CDC^T}$ where $\mathbf{D}$ is an additional diagonal matrix. It is important to note that diagonal component of $\mathbf{C}$ is all 1. This decomposition would reduce the computational cost for factorization but also support solving Equation (8). Similar to Cholesky factorization, starting off from C_{11}, using Equation (11) and Equation (12), Modified Cholesky factorization tries to reduce the total computation. It should be pointed out that, D_{11} would be equal to A_{11}. The total computational complexity would be $\frac{n^3}{6}$ [6], which is faster than normal Cholesky factorization.

$$C_{ij} = \frac{L_{ij} - \sum_{k=1}^{j-1} C_{ik} D_{kk} C_{jk}}{D_{jj}},\ C_{ii} = 1 \tag{11}$$

$$D_{ii} = L_{ii} - \sum_{k=1}^{i-1} C_{ik}^2 D_{kk},\ D_{11} = A_{11} \tag{12}$$

Modified Cholesky factorization guarantees a speedup by using its symmetric features to avoid unnecessary square root calculations.

Furthermore, solving linear system presented in Equation (8) is achieved by solving $\mathbf{CDH} = -\mathbf{B^T}$ using forward substituting and solving $\mathbf{C^T P} = \mathbf{H}$ using backward substitution [7].

Fast computation of solving linear systems is achieved by replacing Cholesky factorization with Modified Cholesky factorization and parallelization as presented in Algorithm 2.

Algorithm 2: Solving Linear Systems

Data: $\mathbf{L}$= Graph Laplacian, $\mathbf{B}$= Matrix containing columns from adjacency matrix $\mathbf{W}$, n= Number of landmark points, N= Number of data points
Result: $\mathbf{P'}$= Matrix containing the pairwise similarity between landmark points
for $i = 1$ *to* N **do**
  Solve D_{ii} with Equation (12)
  ♯ pragma omp parallel for
  for $j = i + 1$ *to* N **do**
    Solve C_{ji} with Equation (11)
  end
end
♯ pragma omp parallel for
for $i = 1$ *to* n **do**
  Solve $\mathbf{p}_i$ with Equation (13)
  ♯ pragma omp parallel for
  for $k = 1$ *to* N **do**
    if Encounter landmark **then**
      Append p_{ik} to $\acute{p}_i$
    end
  end
  $\mathbf{P'} = \begin{bmatrix} p'_1 & p'_2 & \cdots & p'_n \end{bmatrix}$
end
Return $\mathbf{P'}$

The first loop in Algorithm 2 endeavors to decompose $\mathbf{L}$ into $\mathbf{CDC^T}$ by solving Equation (11) and Equation (10) in

turn. This is because once D_{ii} is known, ith column of $\mathbf{C}$ can be solved. Therefore, computation of each element in the ith column is done in parallel. It is important to note that since $\mathbf{C}$ is a lower triangular matrix with diagonal component being 1, C_{ij} should only be computed if $j > i$.

In order to achieve efficient parallelization by distributing computation across multiple threads, we solved Equation (8) in such way that $\mathbf{P}$ is solved per column. Therefore, in each thread, we compute the following equation:

$$\mathbf{L}\mathbf{p}_i = \mathbf{b}_i \tag{13}$$

where $\mathbf{p}_i$ represents ith column of matrix $\mathbf{P}$. Similarly, $\mathbf{b}_i$ represents ith column of $-\mathbf{B^T}$.

Equation (13) can be solved by computing $\mathbf{CDh} = \mathbf{b_i}$ and $\mathbf{C^T p_i} = \mathbf{h}$ using forward and back substitution.

Since Equation (13) is computed in parallel, $\mathbf{p}_i$ would be computed in each thread. Since we are not interested in elements of $\mathbf{p}_i$ that corresponds to non landmark points, our algorithm would only store elements that corresponds to landmark points to a column vector $\mathbf{p'_i}$. To avoid excess operations, when all landmark points are selected, we would terminate the process.

The output for Algorithm 2 is $n \times n$ matrix $\mathbf{P'}$ where P'_{ij} represents pairwise similarity between ith and jth landmark points. It is important to note that $\mathbf{P'}$ is equivalent to selecting rows that corresponds to landmark points in $N \times n$ matrix $\mathbf{P}$.

3.3 Parallelization of Updating Embedded Data

Computational complexity for obtaining $\mathbf{Y}$ is $O(tn^2)$, where t is the number of iterations done when applying gradient descent method and n is the number of landmarks. Depending on the number of iterations, calculating $\mathbf{Y}$ can also be very computationally expensive. Parallelization of updating embedded data is shown in Algorithm 3.

Embedded data $\mathbf{Y}$ would be initialized by random manner [15]. Pairwise similarity between embedded data q_{ij} is obtained using Equation (4). After computing pairwise similarity, the gradient of cost function $\frac{\delta \mathcal{F}}{\delta \mathbf{Y}}$ is acquired using Equation (5). Since computation of q_{ij} for each point is computationally heavy, we would calculate this in parallel. Once the gradient is calculated, $\mathbf{Y}$ is updated using gradient descent method with momentum according to Equation (6).

3.4 Summary of the Proposed Method

Algorithm 4 provides a summary of the process to compute the proposed version of random walk-based t-SNE. There are 3 main steps to the entire implementation. Firstly, we need to compute the adjacency matrix $\mathbf{W}$ and graph Laplacian $\mathbf{L}$ which is indicated in Algorithm 1. Secondly, we would want to obtain matrix $\mathbf{P}$, which involves pairwise similarity

Algorithm 3: Parallelization of Updating Embedded Data

Data: $\mathbf{Y} = \{y_1, y_2, \ldots, y_n\}$, $\mathbf{P'}$ = Matrix with pairwise similarity between landmark points, t = The number of iteration to conduct gradient descent method, α = momentum, η = learning rate
Result: $\mathbf{Y} = \{y_1, y_2, \ldots, y_n\}$
Initialize $\mathbf{Y}$
for *i=1 to t* **do**
  ♯ pragma omp parallel for
  for *each* y_i **do**
    ♯ pragma omp parallel for
    for *each* y_j **do**
      Calculate q_{ij} using Equation (4)
    end
  end
  Update $\mathbf{Y}$ using Equation (5) and Equation (6)
end
Return $\mathbf{Y}$

p_{ij}, using Algorithm 2. However, before running Algorithm 2, we need to compute $-\mathbf{B^T}$ in order to solve Equation (8). This is done by extracting columns of adjacency matrix $\mathbf{W}$, which corresponds to landmark points. $\mathbf{Y}$ is then updated using Algorithm 3 where the computation of gradient is done in parallel.

Our approach improves the efficiency of random walk-based t-SNE focuses on parallel computation and algorithmic optimizations. By leveraging multi-core processors, we hope to accelerate the construction of the graph Laplacian, calculation of pairwise similarities and updating $\mathbf{Y}$.

Algorithm 4: Summary of Parallel and fast computation for random waslk-based t-SNE

Data: $\mathbf{X} = \{x_1, x_2, \ldots, P\}$, t= Number of iteration
Result: $\mathbf{Y} = \{y_1, y_2, \ldots, y_n\}$
if Dimension of $\mathbf{X} > 30$ **then**
  Reduce the dimension of X to 30 by PCA
end
$\mathbf{W}$, $\mathbf{L} \leftarrow$ Algorithm 1
$\mathbf{B^T} \leftarrow$ Columns of $\mathbf{W}$ corresponding to landmark points
$\mathbf{P'} \leftarrow$ Algorithm 2
Update $\mathbf{Y}$ using Algorithm 3
Return $\mathbf{Y}$

4 Experiments and Results

4.1 Experimental Setup

In order to examine the results of the proposed methods, we have conducted multiple experiments. Implementation was done using gcc and OpenMP on a x86-64 computer with 64 CPU cores with frequency of 2.10GHz and 1.5 TiB memory.

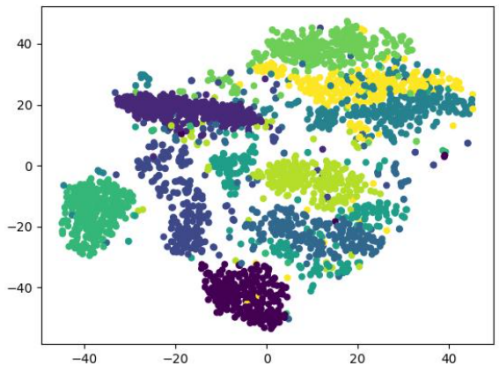

Figure 1 MNIST 2500 with Random Walk based t-SNE

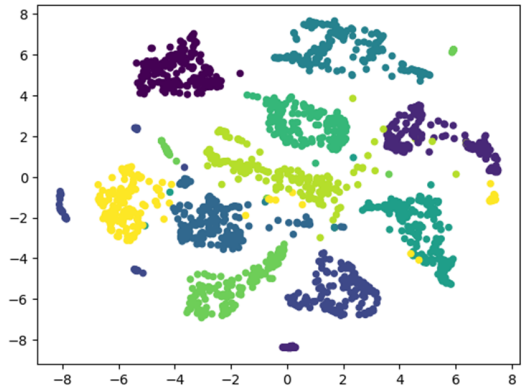

Figure 2 MNIST 2500 with proposed method

Following the original paper on t-SNE, if the number of dimensions of the original data was higher than 30, we conducted PCA to reduce it to 30 [15]. Sequentially, we used t-SNE in order to reduce the dimension into 2. Additionally, we limited the number of nearest neighbors to 20. The number of iteration t was set to 1000, the momentum was 0.5 if $t < 250$ and 0.8 if $t \geqq 250$ [15]. Referring to the original paper, we decided to use an adaptive learning rate, where the initial learning rate would be 100 but would get updated over time [15].

4.2 Visualization of datasets

In order to confirm the quality of this technique, we visualized multiple data using random walk-based t-SNE and the proposed method.

First dataset we used was MNIST 2500 [3] which has 2500 data with each containing 784 dimensions. For this common dataset, each dimension represents the pixel of a hand written number. There are 10 labels from 0 to 9. Figures 2 and 1 shows the result of applying random walk-based t-SNE and the proposed method. The separation of the clusters in Figure 2 indicates that the proposed method has effectively reduced the high-dimensional data into two dimensions while preserving the distinct features of each class. Although Figures 1 and 2 do not look exactly the same, they have both clustered different handwritten digits into a fairly distinct groups, meaning that the proposed method could conduct dimension reduction without the loss of quality. The reason why the two figures do not look exactly the same is due to randomness involving initialization of $\mathbf{Y}$.

The Iris dataset [4] was also used for visualization. The dataset contains 150 samples from three species of Iris flowers: setosa, Virginica and Versicolor. The four features in this dataset includes the length and the width of sepals and petals in centimeters. Figures 3 and 4 illustrate the results of visualization using random walk-based t-SNE and proposed method. The distinction between the three groups illustrates the success in visualization of the dataset. Both techniques achieve a clear visual separation of the species, which suggests that the proposed method maintains the quality of dimension reduction. The similarity in Figures 4 and 3 including the positioning of the embedded data points, is likely because of the straightforward structure of the Iris dataset. If we were using more complex datasets, it might be more challenging to achieve such consistency in results between different dimensional reduction methods.

In summary, when we visualized both the MNIST 2500 dataset and the Iris dataset, the outcomes suggested that the proposed method was capable of conducting dimension reduction without the loss of quality. Both methods were able to embed the data into a simpler form without losing the general structure of the data.

4.3 Processing Time

We have measure the execution time using MNIST 2500 [3] dataset. Figure 5 shows the execution time for varying numbers of threads. The experiment began with a single thread, and then additional threads were introduced in increments of 8, culminating in a total of 64 threads.

As shown in Figure 5, the parallelization was done perfectly as the execution time has dramatically dropped according to the number of threads. A significant decrease in processing time is observed as the number of threads increases. This steep decrease from a single thread to approximately 10 threads illustrates the initial benefits of parallel computation. After the 10 threads, the graph illustrates a fairly stable trend. This could originate from several factors in parallel computing, such as synchronization overhead, the communication time between threads etc.

Figure 6, 7, and 8 represents the execution time for Algorithm 1, 2, and 3, respectively,.

Figure 6 shows a steep decline in execution time as the number of threads increases, indicating significant gains from parallelization in the initial phase. However, it is important

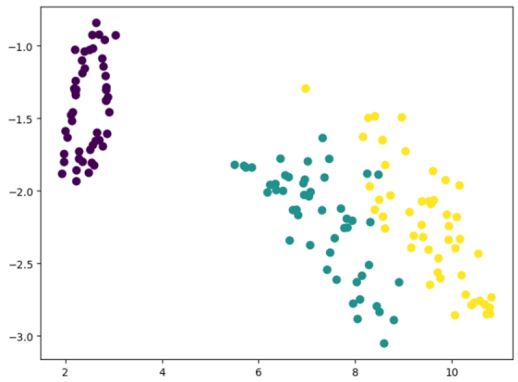

Figure 3 Iris dataset with Random Walk based t-SNE

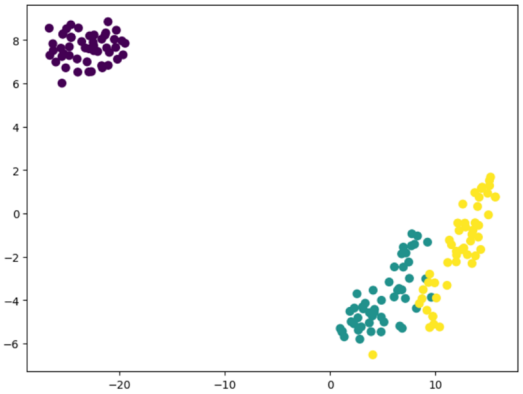

Figure 4 Iris dataset with proposed method

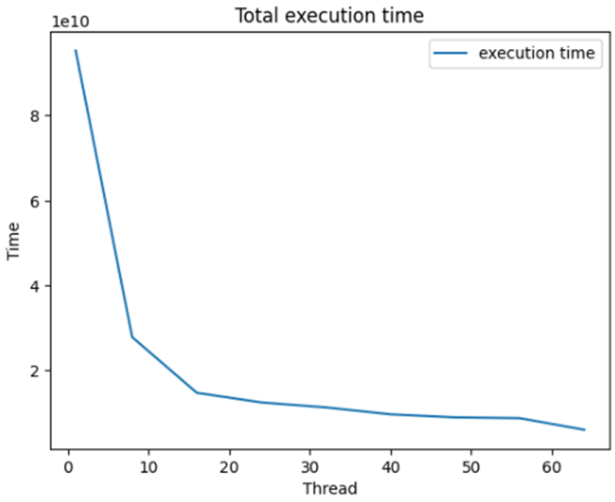

Figure 5 Execution time of Parallel t-SNE on MNIST 2500

to note that for this case, the actual impact of this step on the overall execution time was minimal. This implies that, although parallelization makes this step faster, it was not the critical factor in the total computation time. This maybe due to reduction in dimension in the prepossessing stage where the dimension of MNIST has been reduced to 30.

Figure 7 also shows a sharp reduction in execution time as more threads are utilized. This step has utilised parallel processing and modified choleky factorization. Given that this step is the primary contributor to the total execution time, the efficiency gains in Algorithm 2 are critical to the overall performance improvement of the method.

Figure 8 illustrates a less steep reduction in the execution time compared to the others. This could be due to factors like the iteration in gradient descent method that do not decrease linearly with the addition of more threads.

The substantial reduction in execution time was achieved mainly through the refinements in Algorithm 2. This was what decisively improved the method's efficiency. The other steps, while improved by parallelization, play secondary roles in the method's overall performance enhancement.

To sum up, by using modified cholesky factorization and parallel programming, we have successfully demonstrated that our method reduces computational time significantly, which has addressed one of the biggest disadvantages with original t-SNE.

5 Related Work

Previous efforts have explored enhancing t-SNE's efficiency through parallel processing. For example, Lopes et al. (2020) developed a parallel version of t-SNE to aid in visualizing smart city data [11]. Their work involved three main phases: prepossessing, joint probability calculations, and execution of t-SNE, all of which were parallelized using C and OpenMP.

In their prepossessing phase, parallel computing was applied to speed up the calculation of eigenvalues and eigenvectors as well as the computation of the distance matrix when undergoing PCA. For calculating joint probabilities, tasks such as Gaussian kernel calculations and entropy estimations were performed concurrently. Finally, in the execution phase, the joint probability calculations, the Student-t distribution computations, and the gradient calculations were all parallelized.

In contrast, our research advances the field by applying parallel computing specifically to random walk-based t-SNE for the first time, as far as the author is aware.

6 Conclusion

We have presented an advanced methodology for executing random walk-based t-SNE, utilizing parallel computing and Modified Cholesky factorization to enhance computational efficiency. Our approach has succeeded in significantly reducing processing times, with scalability that effectively corresponds with the addition of processing threads. The results demonstrate the feasibility of our method, as evidenced by the substantial decrease in execution time while maintaining the quality of data visualization. This approach not only offers a faster alternative to traditional random walk-based

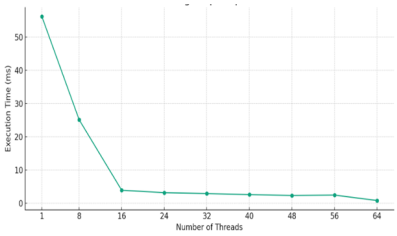

Figure 6 Creating graph Laplacian

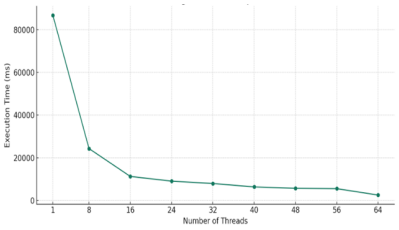

Figure 7 Computing pairwise similarity

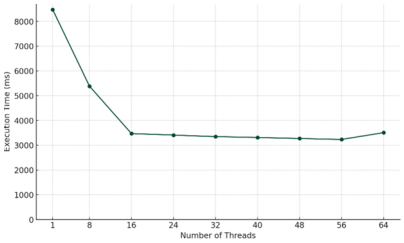

Figure 8 Updating embedded data

t-SNE methods but also provides a path for tackling larger and more complex datasets, which were previously impractical to process.

7 Discussion

A primary concern with the proposed method is its memory footprint, which expands proportionally with the dataset size, potentially limiting its application to extremely large datasets. With the proposed methodology, the computer stores at least three $N \times N$ matrices during Modified Cholesky factorization. As the number of data points grows significantly large, this requirement may become impractical. Future research will be directed towards optimizing memory usage, potentially through the refinement of the data structures employed. We aim to investigate techniques for partitioning the data processing, enabling our method to be both adaptable and memory-efficient. Such strategies might include mini-batching or streaming data processing, each of which can manage memory usage dynamically.

References

[1] Walid M Abdelmoula, Benjamin Balluff, Sonja Englert, Jouke Dijkstra, Marcel JT Reinders, Axel Walch, Liam A McDonnell, and Boudewijn PF Lelieveldt. Data-driven identification of prognostic tumor subpopulations using spatially mapped t-sne of mass spectrometry imaging data. *Proceedings of the National Academy of Sciences*, 113(43):12244–12249, 2016.

[2] Ejaz Ahmed, Ibrar Yaqoob, Ibrahim Abaker Targio Hashem, Imran Khan, Abdelmuttlib Ibrahim Abdalla Ahmed, Muhammad Imran, and Athanasios V Vasilakos. The role of big data analytics in internet of things. *Computer Networks*, 129:459–471, 2017.

[3] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[4] R. A. Fisher. Iris. UCI Machine Learning Repository, 1988. DOI: https://doi.org/10.24432/C56C76.

[5] 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. IEEE, 2021.

[6] PE Gill. W. murray, and mh wright. *Practical Optimization*, pages 212–229, 1981.

[7] Leo Grady. Random walks for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 28(11):1768–1783, 2006.

[8] Nicholas J Higham. Cholesky factorization. *Wiley interdisciplinary reviews: computational statistics*, 1(2):251–254, 2009.

[9] H Hotelling. Analysis of a complex of statistical variables into principal components. warwick & york. *Inc, Baltimore, Maryland*, 1933.

[10] Matteo Lionello, Luca Pietrogrande, Hendrik Purwins, and Mohamed Abou-Zleikha. Interactive exploration of musical space with parametric t-sne. In *15th Sound and Music Computing Conference (SMC 2018)*, pages 200–208. Sound and Music Computing Network, 2018.

[11] Maximiliano Araújo Da Silva Lopes, Adrião D Dória Neto, and Allan De Medeiros Martins. Parallel t-sne applied to data visualization in smart cities. *IEEE Access*, 8:11482–11490, 2020.

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

[13] William Robson Schwartz, Aniruddha Kembhavi, David Harwood, and Larry S Davis. Human detection using partial least squares analysis. In *2009 IEEE 12th international conference on computer vision*, pages 24–31. IEEE, 2009.

[14] Laurens Van Der Maaten. Accelerating t-sne using tree-based algorithms. *The journal of machine learning research*, 15(1):3221–3245, 2014.

[15] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.