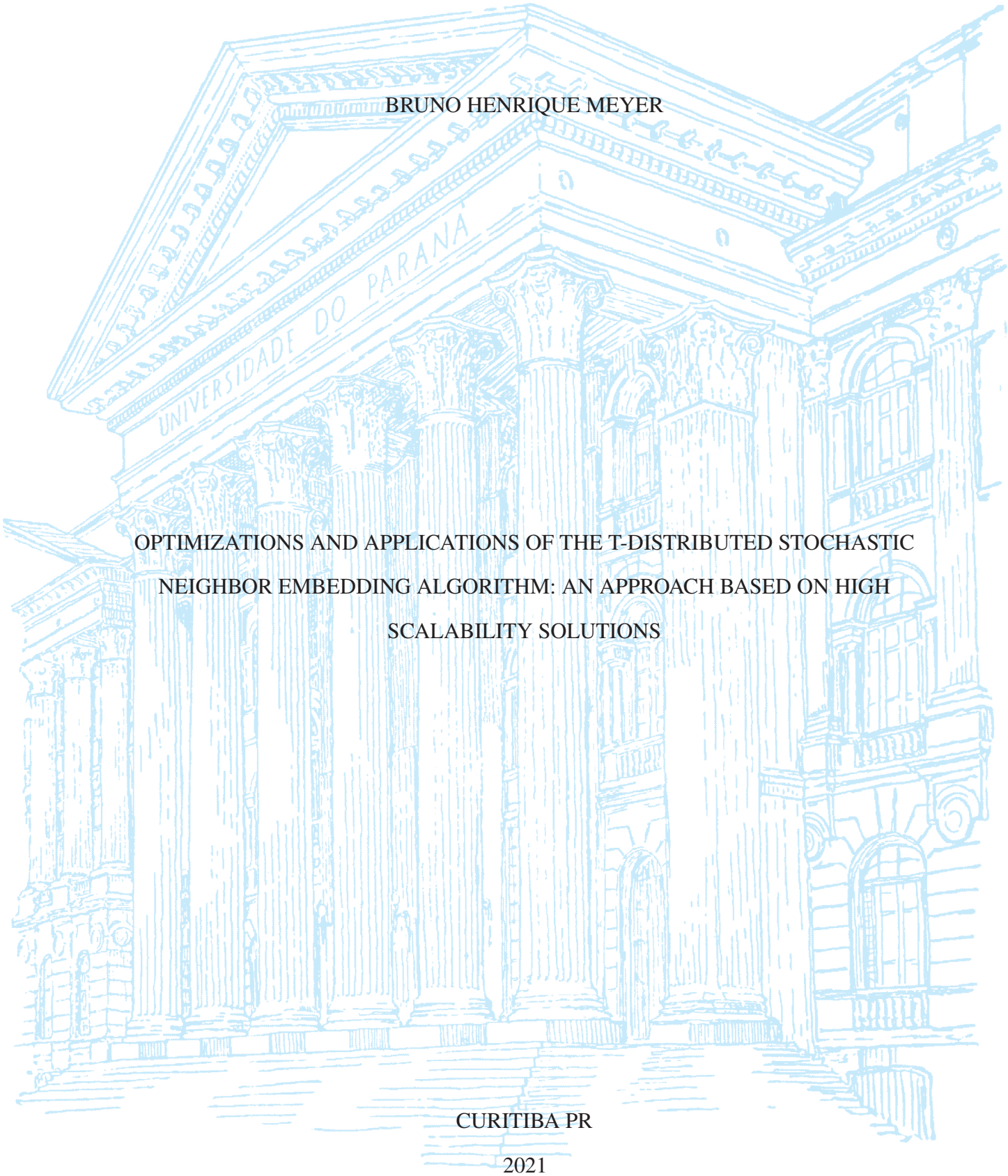
UNIVERSIDADE FEDERAL DO PARANÁ

BRUNO HENRIQUE MEYER



OPTIMIZATIONS AND APPLICATIONS OF THE T-DISTRIBUTED STOCHASTIC NEIGHBOR EMBEDDING ALGORITHM: AN APPROACH BASED ON HIGH SCALABILITY SOLUTIONS

CURITIBA PR

2021

BRUNO HENRIQUE MEYER

OPTIMIZATIONS AND APPLICATIONS OF THE T-DISTRIBUTED STOCHASTIC NEIGHBOR EMBEDDING ALGORITHM: AN APPROACH BASED ON HIGH SCALABILITY SOLUTIONS

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Informática no Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, da Universidade Federal do Paraná.

Área de concentração: *Ciência da Computação*.

Orientador: Wagner M. Nunan Zola.

Coorientador: Aurora Trinidad Ramirez Pozo.

CURITIBA PR

2021

Catalogação na Fonte: Sistema de Bibliotecas, UFPR
Biblioteca de Ciência e Tecnologia

M612o Meyer, Bruno Henrique
Optimizations and applications of the t-distributed stochastic neighbor embedding algorithm: an approach based on high scalability solutions [Recurso eletrônico] / Bruno Henrique Meyer – Curitiba, 2021.

Dissertação - Universidade Federal do Paraná, Setor de Ciências Exatas, Programa de Pós-graduação em Informática.

Orientador: Wagner Machado Nunan Zola
Coorientadora: Aurora Trinidad Ramirez Pozo.

1. Algorítmos. 2. Unidade de Processamento Gráfico (GPU). I. Universidade Federal do Paraná. II. Zola,Wagner Machado Nunan. III. Pozo, Aurora Trinidad Ramirez. IV. Título.

CDD: 006.6

Bibliotecária: Roseny Rivelini Morciani CRB-9/1585

TERMO DE APROVAÇÃO

Os membros da Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação em INFORMÁTICA da Universidade Federal do Paraná foram convocados para realizar a arguição da Dissertação de Mestrado de **BRUNO HENRIQUE MEYER** intitulada: **OPTIMIZATIONS AND APPLICATIONS OF THE T-DISTRIBUTED STOCHASTIC NEIGHBOR EMBEDDING ALGORITHM: AN APPROACH BASED ON HIGH SCALABILITY SOLUTIONS**, sob orientação do Prof. Dr. WAGNER MACHADO NUNAN ZOLA, que após terem inquirido o aluno e realizada a avaliação do trabalho, são de parecer pela sua APROVAÇÃO no rito de defesa.

A outorga do título de mestre está sujeita à homologação pelo colegiado, ao atendimento de todas as indicações e correções solicitadas pela banca e ao pleno atendimento das demandas regimentais do Programa de Pós-Graduação.

CURITIBA, 20 de Abril de 2021.

Assinatura Eletrônica
22/04/2021 14:38:11.0
WAGNER MACHADO NUNAN ZOLA
Presidente da Banca Examinadora (UNIVERSIDADE FEDERAL DO PARANÁ)

Assinatura Eletrônica
22/04/2021 08:17:59.0
RICHARD ADERBAL GONÇALVES
Avaliador Externo (UNIVERSIDADE ESTADUAL DO CENTRO-OESTE)

Assinatura Eletrônica
22/04/2021 09:31:20.0
ANDRÉ LUIZ PIRES GUEDES
Avaliador Interno (UNIVERSIDADE FEDERAL DO PARANÁ)

I dedicate this work to all the professors and people who somehow helped me in my studies.

ACKNOWLEDGEMENTS

This work was partially supported by Brazilian Research Council – CNPq. Also, I want to thank all people in the Bio-inspired Computation lab (C-Bio) for the support and advice. Specifically i would like to acknowledge the guideness and patient of my advisors Wagner M. Nunan Zola and Aurora Pozo.

RESUMO

O t-Distributed Stochastic Neighbor Embedding (t-SNE) é uma técnica amplamente usada para redução de dimensionalidade, mas é limitada por sua escalabilidade quando aplicada a grandes conjuntos de dados. Uma aproximação bem-sucedida do t-SNE chamada BH-tSNE foi recentemente proposta, a qual transforma uma etapa do algoritmo original em um problema de simulação de N-Corpos, que pode ser resolvido pelo algoritmo Barnes-Hut. No entanto, essa melhoria ainda tem limitações para processar grandes volumes de dados (milhões de registros). Estudos posteriores como t-SNE-CUDA usaram GPUs para paralelizar a execução do BH-tSNE. A pesquisa desta dissertação desenvolveu uma nova implementação em GPU do BH-tSNE que produz resultados em duas e três dimensões. Examinamos os problemas de escalabilidade em duas das etapas mais caras do GPU BH-tSNE usando estratégias eficientes de acesso à memória, técnicas de aceleração recentes para GPU e uma nova abordagem para calcular a estrutura de grafos de K-Vizinhos mais próximos (*KNN Graph*) usada no GPU BH-tSNE. Nosso design permite uma aceleração do tempo de execução em até 460% quando comparado à implementação t-SNE-CUDA. Considerando as tecnologias emergentes de Inteligência Artificial (IA) aplicadas a conjuntos de dados em grande escala, vários estudos focam ou usam a redução de dimensionalidade como t-SNE para visualizar os dados. A literatura conta com vários métodos de redução de dimensionalidade para processar grandes conjuntos de dados. Esta pesquisa também comparou diferentes técnicas para realizar a redução de dimensionalidade usando conjuntos de dados em grande escala obtidos de aplicações do mundo real. A comparação enfocou na relação entre as características dos algoritmos, a qualidade dos resultados e a interpretação dos pontos de dados de baixa dimensão. Nossos experimentos concluíram que estratégias como o método denominado AtSNE podem melhorar a qualidade da redução de dimensionalidade, considerando a preservação da informação global. No entanto, não pode obter resultados melhores do que outras práticas, como usar a Análise de Componentes Principais na inicialização do t-SNE. Ainda assim, as idéias de ambos os métodos podem ser combinadas em uma única técnica por estudos futuros. Comparamos sete métodos considerando duas aplicações de IA: Aprendizagem por Reforço e Redes Adversariais Gerativas (GAN). As principais contribuições desta pesquisa consistem na proposta de duas técnicas denominadas SWW-tSNE (Simulated Wide-Warp t-SNE) e SWW-AtSNE (Simulated Wide-Warp AtSNE) para realizar a redução da dimensionalidade em duas ou três dimensões. Esta dissertação também propôs um algoritmo denominado RSFK (Random Sample Forest KNN) que utiliza GPU para calcular uma estrutura denominada *Approximate KNN Graph*, necessária no algoritmo BH t-SNE. A preservação de estruturas globais foi medida com uma nova métrica chamada Preservação de Vizinhança Média.

Palavras-chave: t-SNE. Dados de larga escala. GPU. Redução de dimensionalidade.

ABSTRACT

The t-Distributed Stochastic Neighbor Embedding (t-SNE) is a widely used technique for dimensionality reduction but is limited by its scalability when applied to large datasets. A successful approximation of t-SNE called BH-tSNE was recently proposed, which transforms a step of the original algorithm into an N-Body simulation problem that a modified Barnes-Hut algorithm can solve. However, this improvement still has limitations to process large data volumes (millions of records). Late studies such as t-SNE-CUDA have used GPUs to implement highly parallel BH-tSNE. The research of this thesis has developed a new GPU BH-tSNE implementation that produces the embedding of multidimensional data points into three-dimensional space. We examine scalability issues in two of the most expensive steps of GPU BH-tSNE by using efficient memory access strategies, recent acceleration techniques, and a new approach to compute the KNN graph structure used in BH-tSNE with GPU. Our design allows an acceleration of the execution time in up to 460% when compared to the t-SNE-CUDA implementation. Considering the emergent technologies of Artificial Intelligence (AI) applied to large-scale datasets, numerous studies focus on or use dimensionality reduction like t-SNE to visualize the data. The literature counts with various dimensionality reduction methods to process large datasets. This research also compared different techniques to perform dimensionality reduction using large-scale datasets obtained from real-world applications. The comparison focused on the relation between the characteristics of the algorithms, the quality of the results, and the interpretation of the low dimensional data points. Our experiments conclude that strategies like a method called AtSNE could improve dimensionality reduction quality, considering global information preservation. However, it cannot achieve better results than other practices like using the Principal Component Analysis in the initialization of t-SNE. Still, the ideas of both methods could be merged into a unique technique in future studies. We have compared seven methods considering two AI applications: Reinforcement Learning and Generative Adversarial Networks (GAN). The major contributions of this research consist in the proposal of two techniques named SWW-tSNE (Simulated Wide-Warp t-SNE) and SWW-AtSNE (Simulated Wide-Warp AtSNE) to perform dimensionality reduction in two or three dimensions. This thesis also proposes an algorithm named RSFK (Random Sample Forest KNN) that uses GPU to compute a structure called Approximate KNN Graph, required in BH t-SNE algorithm. The preservation of global structures was measured with a new metric called Medium Neighborhood Preservation (MNP).

Keywords: t-SNE. Large scale data. GPU. Dimensionality reduction.

LIST OF FIGURES

1.1 Visualization of a dataset consisting of 500005 points with 1536 dimensions, where each point represents an image generated with a Generative Adversarial Networks method. The figure illustrates the representation of these images using dimensionality reduction obtained by the execution of the Simulated Wide-Warp t-SNE technique using Principal Component Analysis as initialization. Each color represents a different class. 17

1.2 3-dimensional embedding generated for different datasets. Each color represent a class for the original instance.. 18

2.1 Example of a quadtree representation in the Barnes-Hut algorithm. Internal nodes of the trees represents the centers of mass and leafs nodes the input data points. . 22

4.1 Comparison between the representation of a quadtree using sparse and implicit and sparse structures. 32

4.2 Illustration of the difference between a normal traverse in the implicit quadtree structure (bottom) and the same traversal using the Simulated Wide-Warp technique (top).. 32

4.3 Correlation between the quality of approximate KNN graph and the respective t-SNE projection quality using the MNIST dataset for 1000 t-SNE iterations.. . . 42

4.4 Comparison between IVFFLAT, FLATL2, and the proposed RSFK algorithm. The vertical axis represents the total of points divided by the time required to compute the approximate KNN graph (in log scale) and the horizontal axis respective accuracy. Each curve represents the execution of the algorithms with different configurations and by varying the parameters that control the trade-off between computational time and quality. The time of FLATL2 (exact result), which accuracy is 1.0, is used as the baseline and is dashed in the chart. The labels MNBS ($minBucketSize$) and MXBS ($maxBucketSize$) represent the minimum and maximum of points inside each leaf, respectively, in each configuration of the RSFK algorithm. 43

4.5 Execution time breakdown of different GPU t-SNE versions applied to Amazon Electronics dataset for 1000 t-SNE iterations. *Original* represents the standard t-SNE-CUDA, *Original+RSFK* represents the standard t-SNE-CUDA using our RSFK implementation instead of the FAISS library to compute the approximate KNN graph, and the other versions represent our implementations using Implicit Tree data structure and Simulated Wide-Warp with different WarpWidth sizes. . 44

4.6 Result of the 3-dimensional embedding and convergence generated for MNIST datasets using different versions and fixed random seed for 1000 t-SNE iterations. 47

5.1 2-Dimensional embedding result of the MNIST using different dimensionality reduction approaches. Note that plots (a) to (h) are in increasing order of Global Structure Preservation, as evaluated by our proposed MNP(16) metric. 55

5.2 Comparative of different dimensionality reduction techniques for different datasets. The global structure preservation were measured using 1000 samples of the MNP(16) metric with sample size $C = 100$. The Nearest Neighborhood Preservation was measured using the $R_{\mathrm{NX}}(16)$ metric. 57

5.3 Representation of images of the BigGAN dataset using dimensionality reduction obtained by the execution of PCA and AtSNE. Each color represents a different class. 59

5.6 Phoenix-V0 visualization with SWW-tSNE using PCA initialization. The points color represent the normalized Q-Value related to the action that leads the agent to that state. The frames at top of each image represent samples of states with different Q-Values. The color of each frame were inverted in order to obtain better contrast in the image. 62

A.1 Metodologia utilizada para buscar trabalhos relacionados.. 71

LIST OF TABLES

4.1 Dataset sizes used in experiments. 38

4.2 Computational time (seconds) for 1000 t-SNE iterations with the breakdown of each step, for different methods, to create 3-dimensional embeddings of the GoogleNews300 dataset. WarpWidth=n is abbreviated as WW_n. 45

4.3 Computational time (seconds) for 1000 t-SNE iterations with the breakdown of each step, for different methods, to create 3-dimensional embeddings of the Amazon Electronics dataset. WarpWidth=n is abbreviated as WW_n. 45

4.4 Average speedup achieved and (standard deviation) in the **Barnes-Hut Tree Traversal (*Repulsive Forces Calculations*)** in the different versions of t-SNE-CUDA for 1000 t-SNE iterations. WarpWidth=n is abbreviated as WW_n.. 45

4.5 Average speedup achieved and (standard deviation) of the **total time** execution in the different versions of t-SNE-CUDA for 1000 t-SNE iterations. WarpWidth=n is abbreviated as WW_n. 45

4.6 Average $R_{\mathrm{NX}}(32)$ and (standard deviation) achieved for 1000 t-SNE iterations. . 46

4.7 Comparison of execution time and local neighborhood preservation of different methods to create 2 and 3-dimensional embeddings using the GoogleNews300 dataset for 1000 t-SNE iterations.. 46

5.1 Information of the different stages where the agent model were saved and their respective performance measured by the average reward for all steps from 600 episodes. 50

5.2 Dataset useds in the experiments. Deep Neural Network is abbreviated as DNN.. 51

A.1 Artigos de controle utilizados na metodologia para buscar trabalhos relacionados 71

A.2 Categorias e descrições das abreviaturas utilizadas na Tabela A.3. 72

A.3 Características dos trabalhos selecionados. O símbolo ✓ representa a confirmação da característica em um trabalho, ✗ representa a não presença e valores em branco indicam características não identificadas.. 73

A.4 Tamanho da maior base de dados utilizada para aplicar o t-SNE dentre os trabalhos selecionados. 76

LIST OF ACRONYMS

DINF	Departamento de Informática
PPGINF	Programa de Pós-Graduação em Informática
UFPR	Universidade Federal do Paraná
SGD	Stochastic Gradient Descent
SNE	Stochastic Neighbor Embedding
t-SNE	t-distributed Stochastic Neighbor Embedding
PCA	Principal Component Analysis
UMAP	Uniform Manifold Approximation and Projection
BH-tSNE	Barnes-Hut t-SNE
SWW-tSNE	Simulated Wide-Warp t-SNE
GAN	Generative Adversarial Networks
DQN	Deep Q-Network
AtSNE	Anchor-t-SNE
SWW-AtSNE	Simulated Wide-Warp AtSNE

LIST OF SYMBOLS

u	Parameter called perplexity used in SNE e t-SNE
σ	Standard deviation in a gaussian function
θ	Parameter that controls the approximation in the Barnes-Hut algorithm
η	Step size (or learning rate) of SGD algorithm
α	Momentum of SGD algorithm

CONTENTS

1 INTRODUCTION 14
2 BACKGROUND 19
2.1 BARNES-HUT T-SNE 20
2.2 K-NEAREST NEIGHBORS GRAPH (KNN GRAPH) 23
2.2.1 Approximate Nearest Neighbors Search 23
2.3 T-SNE-CUDA 24
2.4 ATSNE 25
2.5 THE QUALITY OF DIMENSIONALITY REDUCTION 26
3 STATE OF THE ART 28
4 IMPROVING THE SCALABILITY OF BARNES-HUT T-SNE WITH GPU 31
4.1 SIMULATED WIDE-WARP T-SNE (SWW-TSNE) 32
4.1.1 Implicit Tree 33
4.1.2 Simulated Wide-Warp (SWW) 33
4.1.3 GPU Random Sample Forest KNN 34
4.2 METHODS 35
4.2.1 Environment. 38
4.2.2 Datasets 38
4.2.3 Evaluation 39
4.3 RESULTS AND DISCUSSION 41
4.3.1 2-Dimensional Embedding 44
5 APPLYING T-SNE TO REAL-WORLD APPLICATIONS WITH LARGE DATASETS. 48
5.1 SIMULATED WIDE-WARP ATSNE (SWW-ATSNE) 48
5.2 REAL-WORLD APPLICATIONS 48
5.2.1 Generative Adversarial Network 49
5.2.2 Deep Q-Network 49
5.2.3 Medium Neighborhood Preservation (MNP) 50
5.3 METHODS 51
5.3.1 Parameters. 52
5.3.2 Evaluation 52
5.4 RESULTS AND DISCUSSION 52
5.4.1 Overall comparison (RQ1, RQ2) 52
5.4.2 PCA Initialization and Global Structures Preservation (RQ3) 57
5.4.3 Real World Applications Interpretability (RQ4) 59

6 TECHNOLOGICAL AND BIBLIOGRAPHIC PRODUCTION 63
7 CONCLUSIONS AND FUTURE WORK 64
7.1 FUTURE WORK 65
REFERENCES 67
APPENDIX A – "REVISÃO BIBLIOGRÁFICA". 70
A.1 METODOLOGIA DE BUSCA POR TRABALHOS RELACIONADOS 70
A.2 AVALIAÇÃO DE TRABALHOS 70
A.2.1 Otimizações 71
A.2.2 Avaliação de resultados 74
A.2.3 Reprodutibilidade e Escalabilidade 75
A.2.4 Considerações finais 76
A.3 REFERÊNCIAS 77

1 INTRODUCTION

Recently, different subareas of Artificial Intelligence and Machine Learning are trying to solve problems containing large datasets which are difficult to process even in modern hardware. Applications such as Deep Convolutional Neural Networks or Deep Reinforcement Learning using this type of big data commonly create intermediate or final results that consists of sets of data points in a high dimensional space, which are impossible to interpret and visualize in its original form. To address this problem, several dimensionality reduction techniques can be used to approximate the structure of these high dimensional point sets by two-dimensional or three-dimensional representations that can easily be visualized in scatter plots. For the sake of simplicity, we will refer to the techniques that do not create the exact result as "approximate" techniques. This term should not be confused with the approximation algorithms for optimization problems.

The t-Distributed Stochastic Neighbor Embedding (t-SNE) (Van Der Maaten and Hinton, 2008) is a dimensionality reduction algorithm that was widely used in machine learning applications due to its efficiency in discovering natural clusters. For instance, t-SNE can be used to understand the results of Generative Adversarial Networks (GAN) models as illustrated in Figure 1.1. In the figure, it is possible to see the representation of artificial images generated with a GAN model. These representations are high-dimensional data points obtained from the activation of an intermediate layer of the neural network. With t-SNE, it is possible to convert this type of data into two-dimensional points, which allows the visualization of aspects such as the interpolation of images as shown in the figure. The t-SNE algorithm focuses on creating a low-dimensional embedding by preserving the local structures (nearest neighbors). Techniques that use this main idea, like the Uniform Manifold Approximation and Projection (UMAP) approach (McInnes et al., 2020), have gained significant attention recently. However, when projecting a large set of data points into a lower dimensional space to create an embedded representation from the original points, t-SNE presents quadratic computational time complexity, which is a limiting factor for this algorithm. Several strategies have been developed to overcome this problem with approximate algorithms and parallelizing them using Graphics Processing Units (GPU).

The high execution time to execute the t-SNE was differently studied in recent work (Van Der Maaten, 2014; Linderman et al., 2017) where the computational time complexity of t-SNE could be reduced to $O(N \log N)$ or $O(N)$. Still, even with the algorithmic improvements, many constant parameters could interfere in the scalability of these approximations. Further works were made to explore the potential of GPU hardware to improve the scalability of the mentioned algorithms using parallel computation methods (Chan et al., 2018, 2019; Pezzotti et al., 2019; Fu et al., 2019). The t-SNE-CUDA library[1] (Chan et al., 2018, 2019) utilizes the CUDA language to improve the performance of the approximate Barnes-Hut t-SNE (BH-tSNE) version (Van Der Maaten, 2014) and FIt-SNE (Linderman et al., 2017) in NVIDIA GPU hardware. The library can accomplish considerable execution speedup, achieving up to 700 times faster execution when compared to other well-known implementations such as the BH-tSNE CPU implementation available in the Scikit-Learn[2] library. Nevertheless, these GPU methods based in t-SNE can still take hours when executed in large datasets. Therefore, there are several

[1] https://github.com/CannyLab/tsne-cuda

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

challenges to improving these algorithms and reducing the execution time, which could be done by changing the algorithms used in each step or using efficient techniques that take advantage of GPUs architectures.

Different approaches inspired by t-SNE and BH-tSNE have also been proposed like Anchor-t-SNE (AtSNE) (Fu et al., 2019), t-SNE-CUDA (Chan et al., 2018, 2019), allowing the t-distributed Stochastic Neighbor Embedding strategy for millions of data points. However, there is a lack of studies on the impact of the different t-SNE approximations in applications with large datasets.

The AtSNE focuses on preserving the global structure and the stability of the resulted low-dimensional embedding. The experiments presented by Fu et al. (2019) have compared AtSNE with t-SNE-CUDA using the default parameters of the implementation, which does not allow the authors to run t-SNE-CUDA for large datasets. Therefore, it is possible to say that AtSNE was not fairly compared with t-SNE-CUDA since it is possible to choose better parameters for t-SNE-CUDA execution in the experiments described by Fu et al. (2019). Further, the AtSNE research has discussed the preservation of global structures using only qualitative and subjective analysis of the results. Other studies (Kobak and Linderman, 2019) have compared different types of strategies to improve the global structure preservation of t-SNE based methods using a criteria that can be measured.

One possible strategy to improve global quality preservation of t-SNE consists in choosing the Principal Component Analysis (PCA) to initialize t-SNE. According to Kobak and Linderman (2019), it could lead to equivalent results compared with UMAP, which is a technique also proposed to improve the preservation of global structures of the dimensionality reduction.

To the best of our knowledge, until now, no research evaluates the preservation of the global structures of t-SNE-CUDA considering different initialization strategies like PCA for large-scale datasets. One of the difficulties of measuring the global quality of dimensionality reduction is the potential high computational cost intrinsic to this measure, which could involve the computation of pairwise distances (Kobak and Linderman, 2019) requiring the execution of algorithms with quadratic computational time complexity.

Despite the usage of the t-SNE for different applications like reinforcement learning (Mnih et al., 2015), we cannot find any research investigating the importance of dimensionality reduction techniques for analyzing large datasets with high-dimensional points. Considering the difference between different dimensionality reduction methods like AtSNE and t-SNE-CUDA, the resulted low-dimensional points of these techniques may create different types of interpretation of the structure of the high-dimensional points input.

The main objective of this research is to study methods based in t-SNE, and investigate new methodologies to improve the execution time of this algorithm using GPU. This research also compares different t-SNE based techniques that take advantage of GPU and considering applying these techniques in real-world problems. We have investigated the improvement of the BH-tSNE algorithm implemented in t-SNE-CUDA, adapted to project data into two or three dimensions, as in Figure 1.2, while using different strategies proposed by Zola et al. (2014) to prevent inefficient memory access patterns One of the advantages to project data in three dimensions, rather than two, is the flexibility to represent points in space with more possibilities for visualization and for configurations, like the disposal of four equidistant points, which is not possible in two dimensions. We also extend these methods by proposing a new methodology to compute an approximate K-Nearest Neighbors graph (KNN graph), necessary in t-SNE-CUDA, that consists in computing the K nearest neighbors for all points in a given dataset. Furthermore, we also present two real-world artificial intelligence applications that could use t-SNE to interpret large datasets by visualizing it with dimensionality reduction.

The major contributions of this research are described as follows:

- We have adapted, implemented and investigated the two-dimensional BH-tSNE projection in t-SNE-CUDA to generate the embedding in three dimensions (3D).
- We have applied acceleration techniques to GPU BH-tSNE algorithm while preserving the quality of the projections.
- We propose a new dimensionality reduction method based in t-SNE for GPU called Simulated Wide-Warp t-SNE (SWW-tSNE) and have compared with others methods like the t-SNE-CUDA. It was possible to achieve an speedup of up to 900% in one of the most time-consuming steps of GPU BH-tSNE in 3 million point datasets, using an Implicit Tree data layout and Simulated Wide-Warp techniques.
- We propose a new approach to compute an approximate KNN graph in GPU, necessary in BH-tSNE, instead of the approach used in t-SNE-CUDA. With this approach, named Random Sample Forest KNN (RSFK), allied with the Simulated Wide-Warp technique, we have observed an up to 460% speedup in the general computation of BH-tSNE in GPU.
- Although in the current work our acceleration structures and SIMD parallelization techniques were used in a modern GPU setup, we have also verified (Zola et al., 2019) a potential for speeding up BH-tSNE in the context of multi-core processors. Likewise, the parallel SIMD KNN algorithm used in the current work could be effectively applied in a CPU vectorized implementation.
- We have introduced a new approach called Simulated Wide-Warp AtSNE (SWW-AtSNE), that combines the SWW-tSNE and AtSNE strategies.
- We have introduced a new metric to quantify the global structure preservation called Medium Neighborhood Preservation (MNP).
- We have compared PCA, UMAP, AtSNE, SWW-AtSNE, and SWW-tSNE using well-known practices to increase the global structure preservation of large datasets.
- We have analyzed and compared different dimensionality reduction techniques for data visualization in two different AI applications, namely Reinforcement Learning and Generative Adversarial Networks.

The rest of this work is organized as follows: Chapter 2 discusses the fundamentals necessary to understand the t-SNE, BH-tSNE, and t-SNE-CUDA. Chapter 3 presents previous research and contributions to the t-SNE algorithm and its implementation in GPU. The proposal of SWW-tSNE and RSFK will be introduced in Chapter 4, which will also present and analyse experiments to compare these methods with different dimensionality reduction techniques. Chapter 5 will introduce SWW-AtSNE and compare SWW-tSNE, SWW-AtSNE, and other methods using real-world applications that use dimensionality reduction to interpret data. The bibliographic and technological products will be mentioned in Chapter 6. Finally, Chapter 7 will discuss the conclusion conclusions of this thesis and the perspective of future works that could expand the scope of the research.

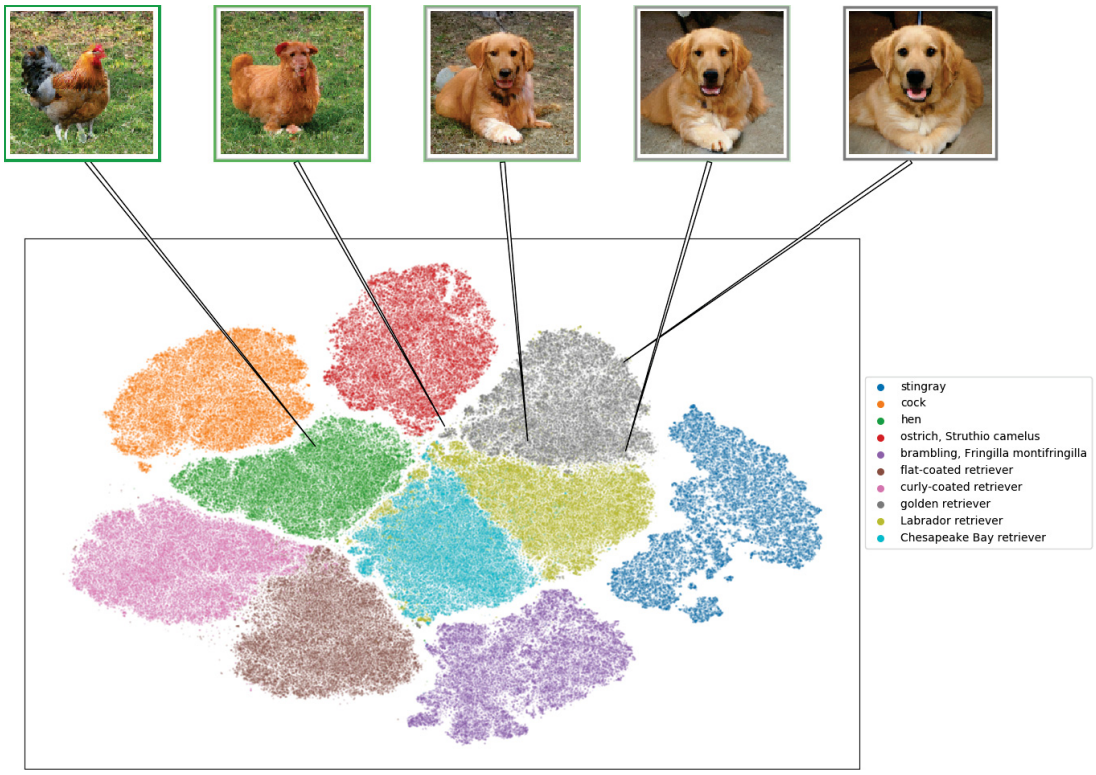
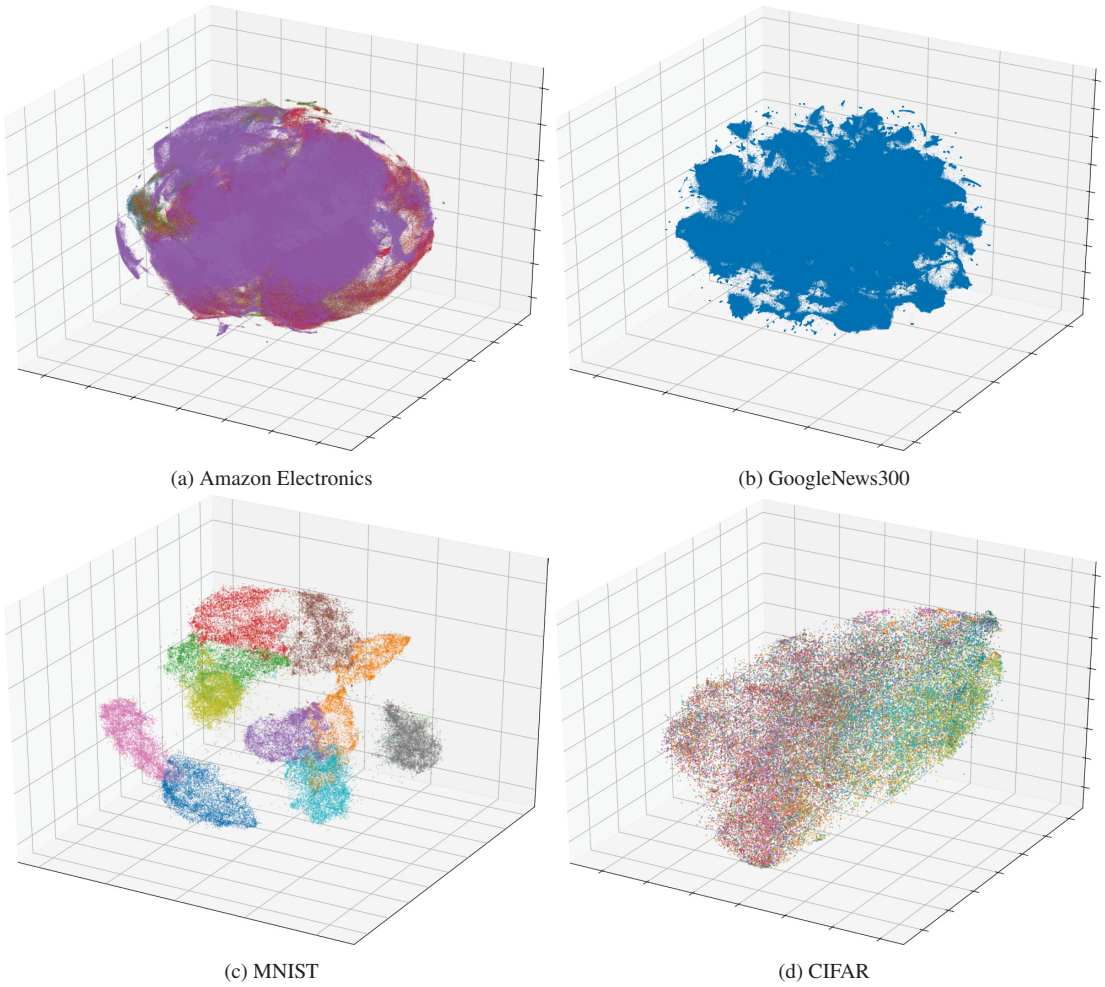


Figure 1.1: Visualization of a dataset consisting of 500005 points with 1536 dimensions, where each point represents an image generated with a Generative Adversarial Networks method. The figure illustrates the representation of these images using dimensionality reduction obtained by the execution of the Simulated Wide-Warp t-SNE technique using Principal Component Analysis as initialization. Each color represents a different class.



(a) Amazon Electronics

(b) GoogleNews300

(c) MNIST

(d) CIFAR

Figure 1.2: 3-dimensional embedding generated for different datasets. Each color represent a class for the original instance.

2 BACKGROUND

The t-SNE technique (Van Der Maaten and Hinton, 2008) is widely used for visualizing real-world datasets due to its efficiency in separating the data into clusters during the dimensionality reduction. The embedding created by t-SNE is done by modeling the dimensionality reduction as a problem where the objective is to increase the probability that if a pair of points x_i and x_j are close in the high-dimensional space (neighbors), their respective representations y_i and y_j in the low-dimensional space will also be close. Given a distance d_{ij} between x_i and x_j, the joint probability of any pair of high-dimensional points p_{ij} is defined as a gaussian distribution centered in each point, as showed in equations 2.1 and 2.2, where $\left\|x_i - x_j\right\|$ represents the euclidean distance between these two points. The standard deviation σ defines the gaussian distribution of each high-dimensional point, and it is computed choosing its optimal value by considering that each point contains an average number of neighbors. This estimated number of neighbors is called Perplexity, and as suggested by the author, should be significantly small (typically between 5 and 50). These joint probabilities are symmetrized using the Equation 2.3.

For the low-dimensional points joint probabilities q_{ij}, t-SNE uses a Student's t-distribution with one degree of freedom (Equation 2.4). The objective of the method is to minimize the difference between these two distributions by updating the low-dimensional points. To measure this difference, t-SNE uses the Kullback-Leibler divergence presented in Equation 2.5. With this difference, it is possible to estimate a cost function using the Equation 2.6, which will measure the overall impact of the Kullback-Leibler divergence considering the position of all points in the low-dimensional space. Finally, it is possible to obtain the movement required to "push" each point in order to minimize the Kullback-Leibler by computing the partial derivative for each point (Equation 2.7).

The original optimization algorithm proposed to implements the t-SNE technique was the Stochastic Gradient Descent (SGD) (Ruder, 2016), which allows the search for suitable local minima even for non-convex differentiable functions. Different theoretical properties derived from t-SNE have been discussed in previous works (Linderman and Steinerberger, 2019). In these studies, it is possible to observe that t-SNE can achieve exponential convergence to significant local minima in some controllable circumstances. However, since the stochasticity of the method and the high number of parameters, t-SNE could present different results when executed multiple times. Algorithm 1 describes the steps necessary to execute the t-SNE algorithm, where the step size η (or learning rate) and Momentum α parameters control the convergence of the SGD. First, the low-dimensional points are initialized, usually using a random gaussian distribution. The low-dimensional points will then be updated until the algorithm reaches a maximum number of steps or achieves pre-defined convergence criteria. The update step consists in computing the gradient $\frac{\partial C}{\partial \mathbf{y}_i}$ after the computation of the joint probabilities q_{ij}, which will be used to update the low-dimensional points considering the "intensity" of the update η and the momentum α that is related to the update of the previous iteration. It is important to mention the early exaggeration strategy proposed in the original t-SNE (Van Der Maaten and Hinton, 2008), where the pairwise similarities of the high-dimensional points are multiplied by a constant during a fixed initial number of iterations of the SGD, which helps the algorithm to achieve faster convergence.

$$d_{ij}^2 = \frac{\left\|\mathbf{x}_i - \mathbf{x}_j\right\|^2}{2\sigma_i^2} \tag{2.1}$$

$$p_{i|j} = \frac{\exp(-d_{ij}^2)}{\sum_{k \neq i} \exp(-d_{ik}^2)} \tag{2.2}$$

$$p_{ij} = \frac{p_{i|j} + p_{j|i}}{2N} \tag{2.3}$$

$$q_{ij} = \frac{\left(1 + \left\|y_i - y_j\right\|^2\right)^{-1}}{\sum_{k \neq l} \left(1 + \left\|y_k - y_l\right\|^2\right)^{-1}} \tag{2.4}$$

$$KL\left(P_i \| Q_i\right) = \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}} \tag{2.5}$$

$$C = \sum_i KL\left(P_i \| Q_i\right) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}} \tag{2.6}$$

$$\frac{\delta C}{\delta y_i} = 4 \sum_j \left(p_{ij} - q_{ij}\right)\left(y_i - y_j\right)\left(1 + \left\|y_i - y_j\right\|^2\right)^{-1} \tag{2.7}$$

Algorithm 1 t-SNE

Input: High-Dimensional points points $X \in \mathbb{R}^D$, Step size η, Momentum α, Low-Dimensional size D', Maximum iterations max_ite, Perplexity $Perpl$

Output: Low-Dimensional points $Y \in \mathbb{R}^{D'}$

1: Calculate p_{ij} for each pair of points in X using perplexity $Perpl$ (Equation 2.2)
2: $Y_0 \leftarrow initial\_random\_projection(\,)$
3: $Y_1 \leftarrow Y_0$
4: $t \leftarrow 1$
5: **repeat**
6: Calculate q_{ij} for each pair of points in Y_t (Equation 2.4)
7: Calculate the gradient $\frac{\partial C}{\partial \mathbf{y}_i}$ for each point $y_i \in Y_t$ (Equation 2.7)
8: $Y_{t+1} \leftarrow Y_t + \eta \frac{\delta C}{\delta Y} + \alpha_t \left(Y_t - Y_{t-1}\right)$
9: $t \leftarrow t + 1$
10: **until** Convergence or $t > max\_ite$
11: **return** Y_t

2.1 BARNES-HUT T-SNE

The standard t-SNE requires a computational time complexity of $O(N^2)$, which is not enough to allow the algorithm to escalate for million-points datasets. Van Der Maaten (2014) proposes the Barnes-Hut t-SNE (BH-tSNE) to surpass this limitation, which is an approximation of t-SNE that can be implemented as an algorithm with $O(N \log N)$ time complexity. The main idea of BH-tSNE is to rewrite the gradient of Equation 2.7 as the sum of two independent terms that can be individually approximated.

One of these terms is the attractive forces F_{attr}, defined in Equation 2.9, which will reflect the forces in the gradient that pull similar points together in the low-dimensional space. Note that in the expression of the sum of the attractive forces, pairs of distant points from each other will

be represented with a joint probability close to zero, leading to an insignificant contribution to the total sum. It is possible to approximate the result by choosing only K-neighborhood (nearest points) of each point, and then computing these joint probabilities. The p_{ij} value of points that are not neighbors can be considered 0 in the summation, which allows the computation of the approximated attractive forces in a linear time complexity controlled by the constant K. However, this method depends on the construction of a K-neighborhood of each point. This can be solved by an algorithm that computes the K-NN Graph of a cloud of data points. The author of BH-tSNE suggests the value of K as $\lfloor 3 \times Perplexity \rfloor$ and the vantage-point tree algorithm to construct an approximation of the K-NN Graph, which leads to good results in his experiments. However, it is possible to use other types of K-NN Graph algorithms to construct an approximation of the graph, like the Random Projection Forests discussed by Tang et al. (2016) during the proposal of LargeVis.

The second term of the rewritten gradient expression in the BH-tSNE is the repulsive forces F_{rep} (Equation 2.10) , which can be interpreted as the part of the gradient that "push" dissimilar points in order to minimize the Kullback-Leibler divergence. This equation can be interpreted as an N-Body simulation problem, where dissimilar (distant) data points have a weak interaction, and similar points have strong interaction. A widely used algorithm that allows a reliable approximation of N-body simulation problems is the Barnes-Hut algorithm (Barnes and Hut, 1986) with computational time complexity of $O(N \log N)$. This algorithm consists of constructing an M-ary tree that creates different partitions (quadrants or octants). Each partition contains a set of points and the pre-computed center of mass of all points contained in it, illustrated in Figure 2.1. Usually, the M-ary tree is a quadtree or an octree since these structures are enough to execute BH-tSNE for the dimensionality reduction to 2 or 3 dimensions. A traversal is performed in the generated tree for each point, and distant points can be approximated using the centroids (centers of mass) created during the tree construction.

The Algorithm 2 represents the steps that the Barnes-Hut algorithm requires to perform a traversal in the quadtree or octree. Each traversal can be controlled by a parameter $\theta \in [0, 1]$ that allows the user to modify the trade-off between computational time and the reliability quality. The value of θ is used to limit the traversal by checking the ratio between two values like shown in Equation 2.12. The first value is the distance of the point to the center of mass of the quadrants (or octants) related to each internal node in the tree. The second value is the radius of these quadrants. If the ratio is small enough compared to θ, then the interaction between the traversal point and the center of mass of this quadrant can be computed instead of the interaction between all leaf points inside the quadrant. This relation will be satisfied if the distance is great or the quadrant radius is small. If the distance between the traversal point and the center of mass of the quadrant is great, then the interaction will be small, and therefore can be approximated without harming the result. If the quadrant (or octant) radius is too small, it will contain points that are very close to each other. In this case, the interaction could also be approximated using only the distance to their center of mass.

The value of θ is typically 0.5, which allows the execution of the Barnes-Hut algorithm in $O(N \log N)$ time complexity (Barnes and Hut, 1986). Since the attractive forces and repulsive forces are the only terms necessary to rewrite the gradient cost in Equation 2.11, then using both approximations mentioned before to compute these forces will transform the computational time complexity of t-SNE to $O(N \log N)$. In this approximation, the fidelity of the K-NN Graph, the value of K, and θ are the principal parameters that allow the user of the algorithm to control the overall trade-off between execution time and quality of the approximation.

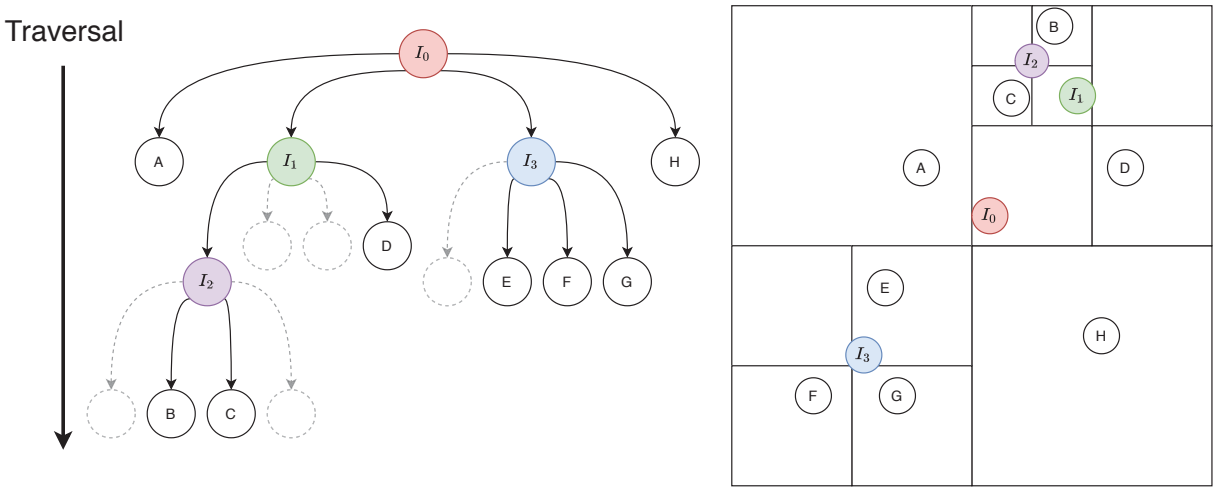


Figure 2.1: Example of a quadtree representation in the Barnes-Hut algorithm. Internal nodes of the trees represents the centers of mass and leafs nodes the input data points.

$$Z = \sum_{k \neq l} \left(1 + \|\mathbf{y}_k - \mathbf{y}_l\|^2\right)^{-1} \tag{2.8}$$

$$F_{attr} = \sum_{j \neq i} p_{ij} q_{ij} Z \left(\mathbf{y}_i - \mathbf{y}_j\right) \tag{2.9}$$

$$F_{rep} = -\sum_{j \neq i} q_{ij}^2 Z \left(\mathbf{y}_i - \mathbf{y}_j\right) \tag{2.10}$$

$$\frac{\partial C}{\partial \mathbf{y}_i} = 4 \left(F_{attr} + F_{rep}\right) \tag{2.11}$$

$$\frac{r_{cell}}{\|\mathbf{y}_i - \mathbf{y}_{cell}\|^2} < \theta \tag{2.12}$$

Algorithm 2 bh_traversal (*Recursive Barnes-Hut Tree traversal*)

Input: Body y_i, m-aryTree, Current node y_j, Threshold θ
Output: Forces vector with average forces applied into y_i

```
C ← children(y_j)
Forces ← 0⃗
for all c ∈ C where y_i ≠ c do
    r_cell ← radius(cell(c))
    y_cell ← centroid(cell(c))
    if Inequation (2.12) is not satisfied then
        Forces ← Forces + bh_traversal(y_i, m-aryTree, c, θ)      // Recursive call
    else
        // Approximates the interaction with distant cells
        Forces ← Forces + interaction(y_cell, y_i)
    end if
end for
return Forces
```

2.2 K-NEAREST NEIGHBORS GRAPH (KNN GRAPH)

The K-Nearest Neighbor Graph is a structure that can be computed from a set of data points. Denote X as the set of these points. The KNN Graph can be represented as a function G that takes any $x \in X$ as input and gives the K most similar objects of X ($|G(x)| = K$) that are different of x. $G(x)$ is usually mentioned as "neighborhood" of x. The similarity function can be defined as any function, but assume that the euclidean distance is used to measure this similarity (the closer the points are, the bigger is the similarity). In this case, for each $\overline{y} \notin G(x)$, $\max_{y \in G(x)}(\|x - y\|) \leq \|x - \overline{y}\|$. Therefore, there is no element out of $G(x)$ that is closer to x than any neighborhood element of x.

When this requirement is not satisfied, we say that the structure is an Approximate K-Nearest Neighbor Graph. The approximation quality will be related to the intersection of $G(x)$ and the real neighborhood of x computed with an exact method. The KNN Graph can be represented as a $N \times K$ matrix, where $N = |X|$. It will be interesting to keep another $N \times K$ matrix with the precomputed similarities in some contexts.

2.2.1 Approximate Nearest Neighbors Search

The FAISS library (Johnson et al., 2017) contains a method that computes an approximation of the K nearest neighbors of a query points set Q using a database of points P. This is sufficient to create the approximate KNN graph required by the BH-tSNE approximation by choosing the points in the HD as $Q = P$, thus performing an all-points KNN algorithm. The t-SNE-CUDA algorithm uses this method to compute the approximate KNN graph using GPU. The method consists in partitioning P in $nlist$ cells, and storing in an inverted file structure. For instance, the K-Means algorithm can be used by creating $nlist$ centroids and storing the associated points of each centroid as an inverted list. Then, the search for a query point can be approximated by selecting the $nprobe$ closest cells of the query point and then scanning the inverted list to compute the K closest points. The risk of this method lies in the fact that if the cell that contains a real neighbor of the query point is not chosen between the $nprobe$ cells, the exact neighbors will not be found. The described FAISS accuracy associated to a approximate KNN graph can be controlled by choosing the parameters $nprobe$ and $nlist$ (usually a multiple of $\sqrt{|P|}$).

The Random Projection Forest KNN consists of creating t trees according to Algorithm 3. This procedure ensures that different partitions of points will be created with no more than M points. Then, the algorithm searches the K closest points of each point, considering only the points that are in the same partition as neighbor candidates. The method used to split each partition in Algorithm 3 will lean towards allowing points that are close in the D-dimensional space remain in the same partition. The number t of partitions created must be diverse enough to contain the real neighbors of every point and consequently to allow controlling the accuracy of this method by increasing the number of trees created.

The described FAISS method performs generic searches using different sets of databases and queries. When this method is used for the approximate KNN graph computation, it could be a waste of computation since it is known that the queries and database points are the same. The operations made by the Random Projection Forest could help to achieve a better speedup in this case since we can compute the nearest neighbor candidates of each point during the tree construction.

To estimate an accuracy metric of an approximate KNN graph, we can compute and normalize the intersection of the estimated neighbors and the real neighbors similar to Equation 2.14. This metric is used by Tang et al. (2016) to compare the trade-off between computational

time and quality of the KNN graph created by the LargeVis algorithm. It is essential to mention that LargeVis also includes a post-processing step, after the construction of the graph, called neighbor exploring.

The neighbor exploring method consists in exploring the neighbors of neighbors for each point and compute the distances during the exploration, to search for new neighbors that are closer to the current K nearest neighbors. This process can be repeated iteratively and, as demonstrated by experiments (Tang et al., 2016), only needs a few number of iterations to achieve maximum accuracy. The scalability of this method depends on the size of K since the number of distances computed for each point is $O(K^2)$, which can leads to poor scalability when K is significantly high.

Algorithm 3 rp_tree (*Random Projection Tree*)

Input: A set of points P of D dimensions, A limit of points inside each leaf node M
Output: A list of subsets containing every point of P

if $|P| \leq M$ **then**
  return $\{P\}$
end if
Generate a random vector $\vec{r}$ in the D-dimensional space
$Proj \leftarrow$ Projection of each point of P onto $\vec{r}$
Select a break point $b \in [min(Proj), max(Proj)]$
$P1 \leftarrow \{p \in P$ if projection of p onto $\vec{r} \leq b\}$
$P2 \leftarrow \{p \in P$ if projection of p onto $\vec{r} > b\}$
return $rp\_tree(P1, M) + rp\_tree(P2, M)$

2.3 T-SNE-CUDA

Chan et al. (2018) proposed t-SNE-CUDA, and a subsequent improved version (Chan et al., 2019), which implements BH-tSNE algorithm using GPU and CUDA primitives. FIt-SNE (Linderman et al., 2017) was also implemented in GPU. In t-SNE-CUDA, the FAISS library (Johnson et al., 2017) is used to compute an approximate KNN of HD points with GPU parallelism in linear computational time complexity (Chan et al., 2019). As demonstrated by the author's experiments, the t-SNE-CUDA was bounded mainly by the approximate KNN graph computation and the attractive forces computation when the value of K is too high. The authors suggest using the multiple GPUs to compute the KNN graph in these cases, which is one option available in the FAISS library. When K is significantly small or multiple GPUs are used to compute the KNN graph, the computation of the repulsive forces also becomes one of the bottlenecks of t-SNE-CUDA.

The t-SNE-CUDA proposes two different alternatives to compute the repulsive forces: One based in the FIt-SNE algorithm, and the other based in the BH-tSNE. Even with the difference in the computational complexity of these methods, the different constants related to their implementations will also significantly impact the computational time. The computational time of these methods is $O(N)$ for FIt-SNE and $O(N \log N)$ for BH-tSNE. But, it is important to note that the computational time of both methods is significantly impacted by other implementation parameters (Chan et al., 2019). To the best of our knowledge, no technique was proposed to execute t-SNE with more than billions of data points as inputs. The input size is usually limited by the memory usage and computational time bottleneck of the KNN step. This can impact the interpretability of the computational complexity since the value of N has a practical limit.

Therefore, it is possible to perceive that the BH-tSNE could present faster execution than the FIt-SNE due to the constant time cost of the operations in their implementations, despite the difference in the computational complexity.

The original implementation[1] allows t-SNE-CUDA to create the projection only to a two-dimensional space. In this case, the computation of attractive forces at each SGD step uses cuBLAS library to apply sparse matrix multiplications, which was improved in (Chan et al., 2019) using GPU atomic operations.

As described in the original article, the Barnes-Hut steps are mainly composed by:

- Finding the bounding box of all points: The result of this step is used to construct initial cells of the Quadtree;
- Construction of the Quadtree;
- Nodes summarization: Computation of the radius, center of mass and number of points of each cell;
- Sorting of points in spatial order (Morton Order);
- Computation of the repulsive forces: Traverses the tree while evaluating forces for each body (Algorithm 2).

In current NVIDIA GPU architectures, there is a concept called *warp* that represents a set of 32 cores executing the same SIMD (*Single Instruction Multiple Data*) instruction at a given time. When cores of the same warp try follow different paths in the code, the GPU must necessarily serialize execution of some cores, partially wasting the potential parallel advantage of GPU SIMD hardware and leading to a problem called thread divergence. Sorting points in Morton order is an optional step, but it improves the chances that threads with close indices in the GPU agree in expanding the same cells (line 6 of Algorithm 2), and consequently reducing divergence.

2.4 ATSNE

The Anchor-t-SNE (AtSNE) (Fu et al., 2019) was proposed for being an algorithm well suited for implementations in GPU compared to t-SNE-CUDA. AtSNE is a t-SNE approach that uses the graph-based strategy similar to the LargeVis application (Tang et al., 2016), where the repulsive force is approximated in each iteration of the SGD using a sampling of non-neighbors of each data point.

The other novelty in the AtSNE approach is that it creates "anchors" obtained by a pre-clustering of the input points of t-SNE, where each anchor is related to a set of the input data points, and its position is the center of mass of these points. In the research, the authors discuss that the pre-clustering used by AtSNE does not need to be accurate. In the available code of the algorithm[2], the FAISS library is used to compute the K-NN, which also allows obtaining a reliable clustering of the input data points generated by a K-Means algorithm during the similarity search of the library.

These anchors of AtSNE are used to create an initial solution in the low-dimensional space by executing a dimensionality reduction of all anchors and then initializing the low-dimensional output points around the anchors using a normal distribution. In the t-SNE, the

[1] `https://github.com/CannyLab/tsne-cuda/tree/bh_tsne`.
[2] `https://github.com/ZJULearning/AtSNE`

initialization of the low-dimensional points is usually defined as a random normal distribution. Previous works have analyzed the potential influence of the initialization to improve the global quality of t-SNE and UMAP (Kobak and Linderman, 2019). Therefore, the usage of anchors in AtSNE to create an initialization could, by itself, allow the SGD to achieve a better global structure. Besides the initialization, the AtSNE also uses the anchor points A and its representation in the low-dimensional space B to rewrite the objective cost of t-SNE defined in Equation 2.6 as the Equation 2.13.

The cost function of AtSNE has two new terms compared to t-SNE: one term that increases with the Kullback-Leibler divergence of the similarities distributions $P(A)$ of the high-dimensional anchor points A and the distributions $P(B)$ of the low-dimensional anchor points B. The other term of the cost function will increase if a point is distant from its anchor during the stochastic optimization. In this way, AtSNE "forces" that the layout defined by the anchors must be preserved during the iterations of the SGD algorithm. The AtSNE optimization algorithm proposed by the authors consists of three steps in the SGD. First, the input data points are fixed, and then the low-dimensional anchor points are individually updated with one step of the SGD. The algorithm then fixes the anchor points and updates only the low-dimensional data points, which considers their anchors inside its K-neighborhood. The last step consists of updating the positions of the low-dimensional anchors using the updated locations of the low-dimensional points to compute the new mass centers.

$$C = \sum_i KL\left(P_i \| Q_i\right) + \sum_j KL\left(P(A_j) \| Q(B_j)\right) + \sum_j ||b_i - \frac{\sum_{y_k \in C_{b_i}} y_k}{|C_{b_i}|}||, b_i \in B \qquad (2.13)$$

2.5 THE QUALITY OF DIMENSIONALITY REDUCTION

Visualization quality can be subjective, considering the diversity of applications that requires interpreting high-dimensional data by visualizing it through dimensionality reduction. Commonly, researches inspired by t-SNE try to discuss the image of scatter plots obtained from applying their methods in a real-world dataset by a qualitative analysis. It is also common to use the dimensionality reduction algorithm to create a dataset embedding and obtain a quantitative measure of the quality of the methods to be analyzed. A measure to evaluate the quality of the dimensionality reduction is an issue not solved yet. Most of the researchers use qualitative analysis from the generated image on a low dimension. Besides, quantitative analysis was made in some studies by using the accuracy of KNN classification methods, here, the data need to be previously classified.

However, the KNN classification accuracy can be affected by outliers and real class intersection present in the data. Therefore, using the accuracy of the KNN to quantify the quality of the dimensionality reduction can be a low indicative analysis of neighborhood and structure preservation, besides depending on supervised datasets. One alternative solves the mentioned problems by measuring the real neighborhood preservation without using any label information (Lee et al., 2015). The average K-ary neighborhood preservation measures an average ratio by considering the preservation of the K closest points of each point for the low-dimensional representation compared to the high-dimensional input data points like described in Equation 2.14. In the equation, K represents the neighborhood size to be considered, N is the total number of points, with ν_i^K and n_i^K meaning the set of the K nearest neighbors of any point in the high and low-dimensional space respectively. The rescaled average K-ary neighborhood, or $R_{\mathrm{NX}}(K)$, is a modification of the previous metric, presented in Equation 2.15. With the $R_{\mathrm{NX}}(K)$ metric,

the quality measurement of random projections will tend to 0, which is not the case of the non-rescaled metric.

The evaluation of the neighborhood preservation indicates if the local structures were successfully preserved by the dimensionality technique analyzed. Usually, preserving these local structures is enough to create a reasonable representation of the data points for visualization analysis. However, this measurement does not consider the relationship between global structures and the relative position for distant points in the high-dimensional data points distribution. One of the objectives in the proposal of the AtSNE was to capture global information and preserve both local and global structures. However, the authors present only a qualitative analysis to interpret the global structure information preserved by the methods compared in their experiments.

As far as we are aware, there is a lack of researches that try to interpret the global structure preservation as a nonsubjective property of the dimensionality reduction results, especially for large datasets. In the study presented by Kobak and Linderman (2019), UMAP and t-SNE are compared by using two different metrics to evaluate the preservation of pairwise distances and the reproducibility of large-scale structures. However, the metrics of the mentioned research require the usage of large subsamples to be computed, which could be impractical for large datasets.

$$Q_{NX}(K) = \frac{1}{KN} \sum_{i=1}^{N} \left| \nu_i^K \cap n_i^K \right| \tag{2.14}$$

$$R_{\mathrm{NX}}(K) = \frac{(N-1)Q_{\mathrm{NX}}(K) - K}{N - 1 - K} \tag{2.15}$$

3 STATE OF THE ART

The t-SNE (Van Der Maaten and Hinton, 2008) method was based on the SNE (Stochastic neighbor embedding) technique (Hinton and Roweis, 2003). Both methods create a low-dimensional representation of each data point in the high-dimensional space by preserving the nearest neighbors of each point. This characteristic seems to create a successful data representation for visualization, specifically for supervised learning applications.

Other approaches based in linear projections such as the PCA (Principal component Analysis) (Hotelling, 1933) have been widely used in different areas to simplify the visualization of high-dimensional data points. However, these kinds of techniques fail to preserve non-linear structures in the data, which is not the case of t-SNE based techniques since its approach transform the data without linear projections.

Currently, Barnes-Hut t-SNE (Van Der Maaten, 2014), or BH-tSNE, is one of the most well-known implementations of t-SNE sustaining $O(N \log N)$ computational time complexity. This technique uses Barnes-Hut method (Barnes and Hut, 1986) to approximate a step in t-SNE called *Repulsive Forces Computation* which is, in the general case, the bottleneck in execution time. The Barnes-Hut algorithm is an approximate method controlled by a parameter θ, which specifies the trade-off between accuracy and time consumption.

Besides the approximation of the repulsive forces proposed in the BH-tSNE algorithm, the author also presents an approximation of a step called attractive forces, which depends on the construction of a k-Nearest Neighbors (KNN) graph. Initially, the BH-tSNE used a vantage-point tree data structure to solve the nearest neighbor's problem. However, it is possible to use other strategies to compute an approximate KNN graph. For instance, the LargeVis (Tang et al., 2016) project uses a method similar to t-SNE, but applies the Random Projection Forest KNN (Yan et al., 2019) algorithm to create the KNN graph.

Despite the contribution of the BH-tSNE to reduce the computation time to execute t-SNE, it still cannot scale for datasets with millions of data points and achieve reasonable time for real-world applications. Different works have introduced news approaches based in better approximations considering the trade-off between computational time and quality of the dimensionality reduction result. Some studies also focus on the scalability of t-SNE based techniques by improving its implementation with parallel primitives.

Recently, much work has been devoted to propose methodologies and techniques that efficiently implements the algorithms based on t-SNE with Graphics Processing Units (GPU) (Chan et al., 2018, 2019; Fu et al., 2019; Pezzotti et al., 2019). The utilization of this highly parallel, many-core, hardware enables the use of t-SNE with millions of data points. Chan, et al. (Chan et al., 2018, 2019) proposes the t-SNE-CUDA, which brings about BH-tSNE in GPU using a modified version of the Barnes-Hut algorithm that was based on the Burtscher and Pingali implementation (Burtscher and Pingali, 2011) and allows two-dimensional visualization of the entire ImageNet dataset (Deng et al., 2009). Further, t-SNE-CUDA uses FAISS (Johnson et al., 2017) library to compute a approximate KNN graph in the first step of the algorithm.

One interesting method that has gained attention in recent years is the UMAP technique (McInnes et al., 2020), which is inspired in t-SNE but is based in other theoretical concepts to preserve the topological and geometrical information of the data points. A recent study (Nolet et al., 2020) has been made to propose the UMAP using GPU primitives. This implementation is part of the cuML project, related to the RAPIDS (a suite of open-source software libraries). The main idea of the GPU implementation of UMAP in the mentioned work is to use the FAISS library

to compute the K-NN graph and use efficient structures to represent the data. In the original proposal of UMAP (McInnes et al., 2020), different discussions and observations were made considering its superiority in preserving the global quality during the dimensionality reduction. However, further investigations have shown a counter-proof by changing the initialization of t-SNE and comparing it with UMAP methodology (Kobak and Linderman, 2019). This comparison concludes that both techniques have similar efficiency in preserving global quality when the correct initialization of the low-dimensional points is chosen. In these studies, the global quality was compared using a subjective analysis of the resulting image or the pairwise distances of a subsample of the points. There are other options to measure global structure preservation, like analyzing the neighborhood preservation with different neighborhood sizes (Lee et al., 2013), which requires a high computational cost and is impracticable for large datasets.

Other implementations that focus on GPU's efficient usage were defined like Anchor t-SNE (AtSNE) (Fu et al., 2019). AtSNE is based in the LargeVis approach (Tang et al., 2016) where the K-NN graph is used to approximate part of the computation by negative sampling instead of using the Barnes-Hut algorithm. AtSNE also implements a new strategy that computes a high-dimensional skeleton layout of the points to improve the preservation of the global structures and preserve the stability of the results when the algorithm is executed with different random seeds.

Although the t-SNE-CUDA and AtSNE are significant contributions, these approaches are proposed and implemented specifically for NVIDIA hardware. Pezzotti et al. (2019) proposed a new approach based in the principles of General-purpose computing on graphics processing units (GPGPU) that avoids the challenge of improving the irregular memory access pattern in GPU of Barnes-Hut algorithm. The GPGPU approach could not surpass the t-SNE-CUDA implementation considering the execution time, but presents a competitive trade-off between quality and execution time in addition to the possibility of being implemented and executed in specific environments like web browsers via WebGL API.

Despite the existence of different works that approximate t-SNE with a linear complexity algorithms like GPGPU t-SNE, there is still a need to investigate Barnes-Hut t-SNE due to its competitive performance in computational time. Different approaches can be used to improve BH-tSNE. The Barnes-Hut method traditionally computes attractive forces in 3D or 2D particle systems. Techniques such as the Implicit Tree structure and Simulated Wide-Warp presented by Zola, et al. (Zola et al., 2014) mitigates the inefficient GPU memory access present in common Barnes-Hut implementation. Recently the Implicit Octree acceleration structure were successfully applied to BH with multi-core SIMD vectorization (Zola et al., 2019). The implicit tree consists in changing the tree data structure representation in the Barnes-Hut algorithm using an implicit representation instead of a sparse representation. Although the implicit representation lowers the memory footprint in the BH step, the bottleneck in memory consumption in t-SNE is located in the approximate KNN graph representation. However, the implicit tree structure allows an efficient execution of the operations in the Barnes-Hut algorithm, which significantly reduces the overall computational time. While some speedup could be acquired in a sequential version, it is mostly effective in the GPU parallel execution. The simulated wide warp is another technique that enables the simulation of virtual threads in GPU, and enhance throughput in GPU parallel/SIMD calculations. Combined with the implicit tree acceleration structure, SWW allows more regular access patterns to the memory hierarchy and speed up the algorithms.

Currently, there is a significant variety of methods and technologies available to compute the Nearest Neighbors using standard CPU architectures (Aumüller et al., 2019). However, there is a lack of options for those technologies in GPU. The FAISS library is a well known implementation used to create approximate KNN graphs in t-SNE-CUDA (Chan et al., 2018,

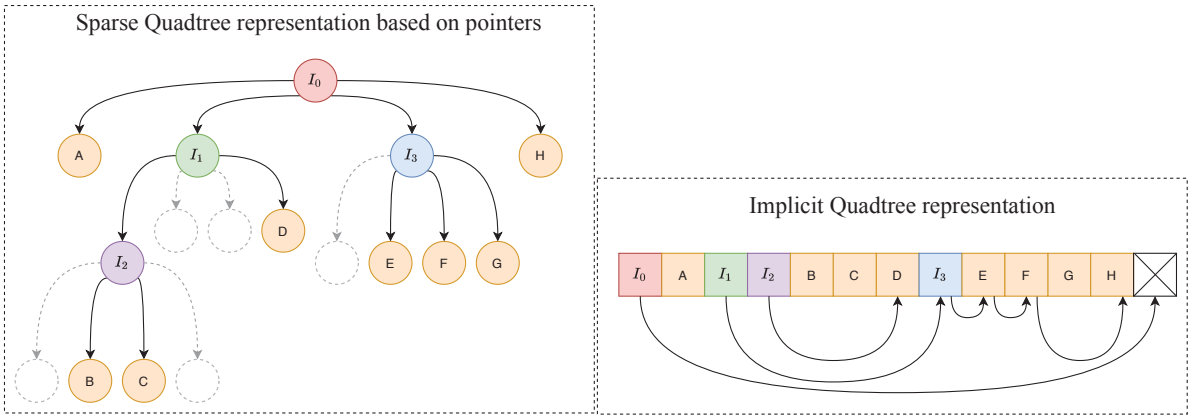
2019). The Random Projection Forest (RPF) algorithm for approximate nearest neighbor search has demonstrated competitive results considering the trade-off between accuracy and execution time (Aumüller et al., 2019). To the best of our knowledge, no specific research focuses on using the RPF algorithm to create KNN graphs (one of the required steps t-SNE approximations) using GPU primitives. RPF algorithm benefits from performing several NN searches and then combining the results, thus improving the accuracy of the overall search. LargeVis algorithm successfully uses the Annoy library (another RPF implementation) to compute a approximate KNN graph from high-dimensional points using multi-core CPUs, enabling the algorithm to achieve competitive results compared to the BH-tSNE implementation.

A broad examination of different works related to t-SNE was discussed in the document presented in the Master's qualification examination related to this thesis. The appendix A contains the chapter related to the methodology and results of the research, which allows examining the trends and gaps present in the State of The Art t-SNE.

4 IMPROVING THE SCALABILITY OF BARNES-HUT T-SNE WITH GPU

Despite the novelty of the BH-tSNE implementation in t-SNE-CUDA to scale for million-points datasets, there were still some limitations of the technique, mainly in the irregular GPU memory access, leading to a non-GPU friendly algorithm (Fu et al., 2019). This Chapter will present a new method named Simulated Wide-Warp t-SNE (SWW-tSNE) to overcome this limitation and improve the efficiency of the Barnes-Hut t-SNE implementation using two strategies (Zola et al., 2014) to improve the access pattern to the GPU memory. One of these strategies is the manipulation of the tree using an implicit structure, which avoids the usage of a sparse structure with pointer-based references (see Figure 4.1). Figure 4.1(a) shows that several pointers are allocated in the sparse representation of a quadtree but are useless since they are pointing to null nodes. Also, in a postorder traversal in the tree using a sparse representation, required in the Barnes-Hut algorithm, it will be necessary to use stacks to manipulate and control the order that each node is explored. However, the nodes can be put sequentially in a unique array as illustrated in Figure 4.1(b), which we refer to as the implicit representation. In the implicit representation, the nodes placed in the memory already consider the postorder required in the Barnes-Hut algorithm. The only pointers required in this tree are those that indicate where the traversal should jump if the Inequation 2.12 is satisfied. This precomputation of the order that each node will be consulted eliminates the need to use stacks in the traversal, enabling a better memory access pattern. The second strategy improves the algorithm used to traverse the tree using a method called Simulated Wide-Warp. In Figure 4.2, the main idea of the Simulated Wide-Warp is represented, which consists of improving the memory access pattern by creating virtual threads inside each GPU physical thread. A *warp* in the GPU is the concept that abstracts a set of cores that will follow the same instruction. The size of each *warp* is related to each GPU architecture, which is 32 for all NVIDIA architectures until now. The Simulated Wide-Warp virtually multiplies the size of each *warp* by a factor called *WarpWidth* through code implementation. The implementation consists in the execution of multiples virtual threads inside a unique thread, which allows the algorithm to share common memory data and reduce unnecessary loads in the memory. Using the Implicit Tree representation and the Simulated Wide-Warp technique in the SWW-tSNE, it is possible to obtain equivalent results compared to t-SNE-CUDA with the same reliability, but with a shorter execution time due to the efficient usage of the GPU memory.

The SWW-tSNE and RSFK algorithm will be presented in Section 4.1. Section 4.2 will introduce the methods used to compare SWW-tSNE with t-SNE-CUDA and compare RSFK with the FAISS algorithm. The results of these experiments will be discussed in section 4.3.



(a) Sparse representation (b) Implicit representation

Figure 4.1: Comparison between the representation of a quadtree using sparse and implicit and sparse structures.

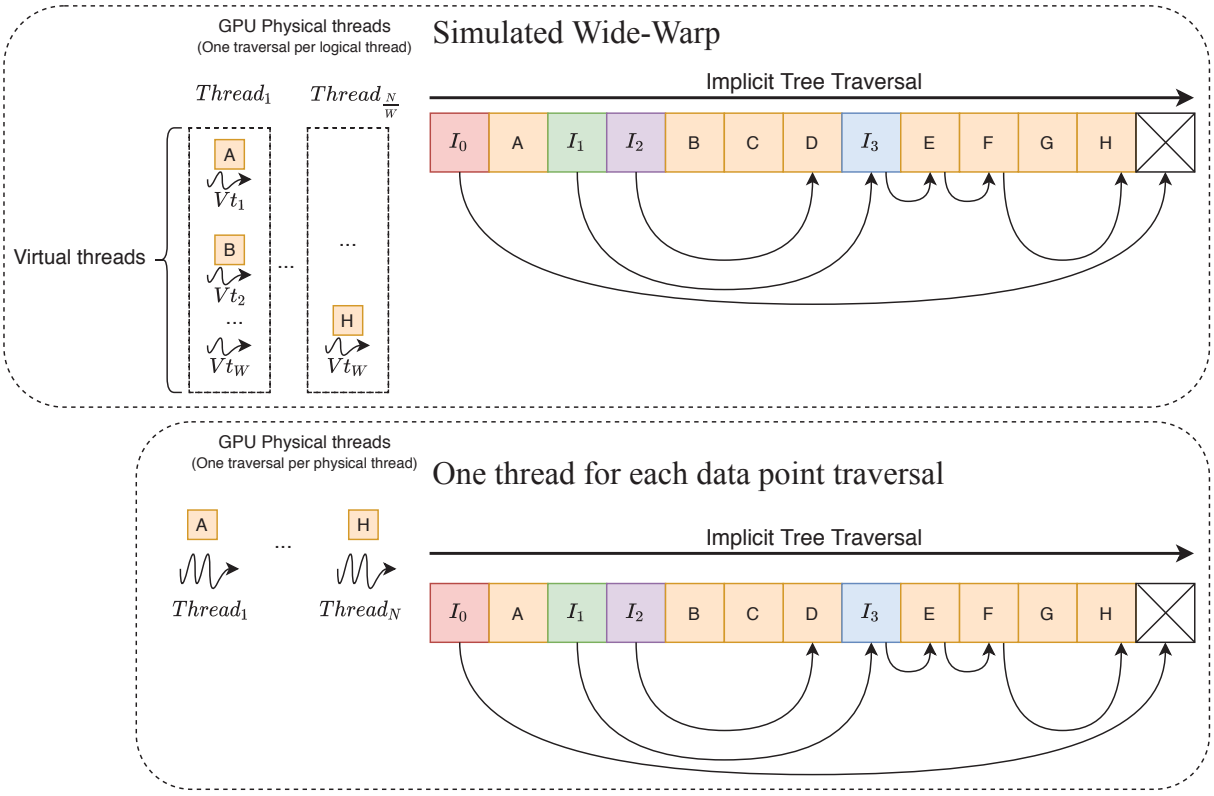


Figure 4.2: Illustration of the difference between a normal traverse in the implicit quadtree structure (bottom) and the same traversal using the Simulated Wide-Warp technique (top).

4.1 SIMULATED WIDE-WARP T-SNE (SWW-TSNE)

The t-SNE-CUDA Barnes-Hut code is based on the Barnes-Hut Simulation work of Burtscher and Pingali (Burtscher and Pingali, 2011) (BP-BH), with specific forces calculations modified to the t-SNE method and with dimensionality of N-Body simulation simplified to 2 dimensions. These implementations are iterative, instead of recursive as Algorithm 2, and are implemented

with a stack data structure in the traversal. The tree representation is *Sparse* and uses an array of integer pointers to child nodes in each non-leaf node.

Zola, et al. (Zola et al., 2014) presented a GPU implementation of the Barnes-Hut algorithm that surpasses BP-BH in computational time by using a different tree representation called *Implicit Tree* and proposed Simulated Wide-Warp (SWW), a technique that creates virtual threads simulating warps with more threads per core than what is physically supported by the GPU. These modifications allow BH algorithm to present better memory access patterns and also to acquire more accurate results with considerable speedup.

Another potential improvement in t-SNE-CUDA is related to the KNN graph construction, which is solved by the use of FAISS library. We believe that this choice does not take advantage of all features related to the needs of t-SNE. The t-SNE algorithm does not directly need the execution of a generic KNN search since the query points are also the database points. We propose an alternative algorithm to compute the KNN graph. This algorithm is based on the Random Projection Forest KNN that can be implemented in GPU and is described in Subsection 4.1.3.

4.1.1 Implicit Tree

In the Implicit Tree representation, leaf nodes and internal tree nodes are represented in a unique array, where internal nodes contain a pointer to the next internal node, assuming a pre-order traversal. Figure 4.1 illustrates the difference between Sparse and Implicit representations. An advantage of the Implicit representation consists in the fact that it needs to store only one pointer in each non-leaf node reducing the memory needed in comparison to the Sparse representation, where even null nodes need to be represented.

While saving space allowing larger trees and posing less bandwidth demand on the memory system, another benefit of the implicit representation manifests when traversing the tree in the BH algorithm: unlike the Sparse representation, there is no need to use a stack per thread when traversing the tree since the *skip-link* pointers in each internal node are the only information needed to walk through the tree. This allows the algorithm to access GPU memory more consistently, reducing register use, global memory accesses and execution time. However, it is essential to note that the KNN graph input has the size $\Theta(ND)$ and the output $\Theta(NK)$. Considering that N is the number of leaves in the quadtree or octree, then the number of internal nodes of these trees will not reach beyond N. Specifically, suppose that an octree was created, and it contains N_i internal nodes. In the sparse representation, each internal node will be represented by 12 values (usually 32 bits each): 3 values representing its position, 1 value the octant radius, and 8 value the pointers to its children. In the implicit representation, only one pointer is needed, which reduces the number of necessary values to $12N_i - 5N_i$. However, considering the total memory required when computing the KNN step in t-SNE, the saved memory by using the Implicit Tree structures will not be significant in these scenarios. Instead, it will provide a well-suited structure to improve the memory access pattern during the traversal operation.

4.1.2 Simulated Wide-Warp (SWW)

Modern GPU architectures allow a large number of threads but the actual number of active threads at a given time is proportional to the number of cores. The Simulated Wide-Warp technique consists in simulating extra virtual threads by exploring the large number of GPU registers (currently 65536 per GPU SM multiprocessor). This is done in a target algorithm by carefully coding specific loop unroll patterns that are not automatically captured by current compilers.

The simulation of virtual threads is done inside each physical thread. Virtual threads allow sharing thread local variables without additional real memory access. Basically, SWW is used to optimize forces calculation sections with minor changes to the implicit tree traversal steps in algorithm BH. The programmer must be careful to create specific variables needed by each virtual thread and not surpass the GPU register limit, otherwise leading to a problem called register spill where the extra local variables will be placed in global memory.

In SWW, the effective number of simulated threads corresponds to the amount of physical warp threads multiplied by a constant number (*WarpWidth*) of virtual threads implemented per physical thread. To lower thread divergence, the tree traversal in SWW-tSNE must ensure that all threads in a warp execute the same instruction. This can be ensured if at least one thread chooses to expand a node. In this case, all threads in the simulated warp must perform the same expansion, even if Equation (2.12) is not satisfied. With this guarantee, the algorithm can perform faster and return more accurate results. This is also ensured in t-SNE-CUDA but its efficiency in SWW-tSNE grows with the width of the virtual warp.

Since the coding of repulsive forces in t-SNE differs from the computation of gravitational forces in the traditional BH, the number of registers used per thread also tends to be different. As such, the impact of SWW to t-SNE algorithm needs to be verified to access potential speedup and quality of algorithm convergence for different values of WarpWidth when processing the standard datasets with diverse sizes.

4.1.3 GPU Random Sample Forest KNN

As t-SNE-CUDA uses the FAISS library to perform a query for each data point, in SWW-TSNE we took a different approach to harness parallelism to create approximate KNN graphs. We investigated the use of a modified Random Projection Forest algorithm using GPU. In this way, it is possible to directly create the KNN graph without performing the traditional K-Nearest Neighbor Search.

As discussed by Tang et al. (2016), it is possible to simplify the idea of Random Projection Forest KNN (Yan et al., 2019) by updating the criteria of splitting each partition of points described in Algorithm 3. This simplification, named here Random Sample Forest KNN (RSFK), removes the necessity of projecting each point in a random direction by selecting two arbitrary points. Then the algorithm computes the equidistant hyperplane between the two points and creates two new partitions.

In our proposal, we chose to use this minimalist method due to the simplicity of implementation. Further, there is a reduction of operations needed to construct each tree. The LargeVis implementation uses the Annoy system[1], which until the best of our knowledge does not allow the user to create the trees in parallel using multiples cores. We implement the RSFK approach using CUDA parallel primitives as described in Algorithm 5.

Algorithm 4 describes the simplification of the random projection tree by sampling two points to split the point set in two partitions separated by a hyperplane. A new depth level is built while there are points that are not related to valid leaves. The following steps construct each depth level: i) Verify the side of each point according to its "parent" hyperplane. ii) Create the new hyperplanes to split the current partitions. iii) Update the parent of each point. iv) Check the validity of each new hyperplane and find valid leaves.

The user specifies the minimum ($minBucketSize$) and maximum ($maxBucketSize$) number of points that must be in each tree leaf, which we also will refer to as a “bucket” of data points. Note that these parameters control the trade-off between the quality and execution time,

[1]https://github.com/spotify/annoy

since the more points are on the same leaf, the higher the number of distance calculations and neighboring candidates is computed. Also, an invalid hyperplane contains fewer points than the minimum parameter, and a valid leaf contains a total of points between the minimum and maximum parameters.

Another parameter that controls the trade-off between quality and computation time of the presented algorithm is the number of trees used to build the forest, as in Algorithm 5. Every tree that partitions the set of points is used to update the current neighborhood of each point by considering the points that lie in the same leaf as possible new neighbors. There are three choices that we adopted to perform efficient memory access in this step:

- Each GPU thread block evaluates and updates KNN graphs at each leaf.
- We use each Warp in GPU thread block to compute each pair of distances. This warp-centric approach is equivalent to evaluating the D dimensional distances with SIMD instructions, as the number of dimensions is high in big datasets. Also, this allows coalesced memory access to the higher dimensional points.
- The neighborhood of each point is not ordered by distance. Instead, we only keep an index of the farthest neighbor. The search and update in these sets are also performed in a warp-centric approach.

In the proposed approach, it is important to notice that we need to keep only the memory allocation of one tree, since the points are partitioned as each depth level is created, and after its creation, there is not any use for it. Considering the computational cost, the maximum of points inside each leaf controls a reduction of the computational time in the tree construction step and an increase in the neighborhood update.

4.2 METHODS

The original implementation of t-SNE-CUDA was modified to project the input data points of the algorithm into a tridimensional space, which we will call "Original version". We produced a new GPU BH-tSNE implementation that uses our RSFK algorithm for KNN graphs, performs the BH step using implicit trees and conducts the tree walking and forces calculations phase using SWW techniques, wich we will call *SWW-tSNE*. Then, different SWW-tSNE versions were created, increasing WarpWidth in each version, using the maximum number of threads per block and, consequently, the maximum number of registers available in the GPU, achieving up to WarpWidth=4 before the register spill problem occurs. It is essential to mention that in the experiments of this work we will consider only the optimization of the Tree Building and Traversal present in the Barnes-Hut algorithm. Also, note that the WarpWidth=1 version is equivalent to the Original, to the extent that it does not simulate larger than physical warps and the only difference is the Tree structure used by the algorithm.

The computation of the attractive forces in the Original version is still using a deprecated implementation with cuBLAS (Chan et al., 2018) that was improved in recent work (Chan et al., 2019) using GPU atomic operations, which was also implemented in the SWW-tSNE algorithm.

Also, we implemented the proposed RSFK algorithm using CUDA programming language. The experiments compare the trade-off between accuracy and computational time of RSFK, the IVFFLAT[2] index previously used in t-SNE-CUDA, and the FLATL2[3] index, a

[2]https://github.com/facebookresearch/faiss/wiki/Faster-search

[3]https://github.com/facebookresearch/faiss/wiki/Faiss-indexes

Algorithm 4 Random Sample Tree

Input: A set of points P of D dimensions, The maximum number of points inside each leaf node $maxBucketSize$, The minimum number of points inside each leaf node $minBucketSize$

Output: A partition of P where each subset contains at most $maxBucketSize$ and at least $minBucketSize$ points

Sample two points from P and create a hyperplane h_0 between them
for $p \in P$ in parallel **do**
  $parent[p] \leftarrow h_0$
end for
$ActivePoints \leftarrow P$
$ActiveNodes \leftarrow \{h_0\}$
$Leaves \leftarrow \{\}$
while $|ActiveNodes| > 0$ **do**
  for $p \in ActivePoints$ in parallel **do**
    Compute the side of p in the hyperplane $parent[p]$
  end for
  for $h \in ActiveNodes$ in parallel **do**
    Sample two points from the left side of h and create $h1$
    Sample two points from the right side of h and create $h2$
    $LeftChild[h] \leftarrow h1$
    $RightChild[h] \leftarrow h2$
    $ActiveNodes \leftarrow ActiveNodes - \{h\}$
    $ActiveNodes \leftarrow ActiveNodes + \{h1, h2\}$
    $Sibling[h1] \leftarrow h2$
    $Sibling[h2] \leftarrow h1$
  end for
  for $p \in ActivePoints$ in parallel **do**
    if p is in the left side in the hyperplane $parent[p]$ **then**
      $parent[p] \leftarrow LeftChild[parent[p]]$
    else
      $parent[p] \leftarrow RightChild[parent[p]]$
    end if
  end for
  for $p \in ActivePoints$ in parallel **do**
    $tp \leftarrow$ Number of active points whose parent is $parent[p]$
    if $tp \leq minBucketSize$ **then**
      $parent[p] \leftarrow Sibling[parent[p]]$
    else if $tp \leq maxBucketSize$ **then**
      $ActiveNodes \leftarrow ActiveNodes - \{parent[p]\}$
      $Leaves \leftarrow Leaves + \{parent[p]\}$
    end if
  end for
end while
return $Leaves$

Algorithm 5 Random Sample Forest KNN

Input: A set of points P of D dimensions, The maximum number of points inside each leaf node $maxBucketSize$, The minimum number of points inside each leaf node $minBucketSize$, The number of trees $ntree$, The number of neighbors K, Iterations used in Nearest Neighbors Exploring $NNEFactor$

Output: Updates the neighborhood of each point with new closest neighbors or maintains the same neighborhood

for $t \leftarrow 1$ to $ntree$ **do**
  // Create a random tree and a partition of the points with Random Sample Tree algorithm
    // For each partition created with the tree, assign it to a block of threads in GPU.
  $Leaves \leftarrow$ Algorithm 4 (P, $maxBucketSize$, $minBucketSize$)
  // For each partition created with the tree, assign it to a block of threads in GPU.
  for $l \in Leaves$ in parallel (GPU block) **do**
    // For each pair of points inside each leaf, assign it to a GPU warp, compute its distance and update the neighborhood of both points if necessary
    for each pair $(p1, p2) \mid parent[p1] = parent[p2] = l$, in parallel (GPU warp) **do**
      $d1 \leftarrow Distance(p1, p2)$
      $d2 \leftarrow$ Distance of the farthest neighbor of $p1$
      $d3 \leftarrow$ Distance of the farthest neighbor of $p2$
      if $d1 < d2$ **then**
        Insert $p2$ in $p1$ neighborhood and remove the farthest neighbor of $p1$
      end if
      if $d1 < d3$ **then**
        Insert $p1$ in $p2$ neighborhood and remove the farthest neighbor of $p2$
      end if
    end for
  end for
end for
// Execute the Neighbor Exploring with $NNEFactor$ iterations
for $nn \leftarrow 1$ to $NNEFactor$ **do**
  // Explore the neighborhood of each point for new neighbors and update its neighborhood if necessary
  for $p \in P$ in parallel **do**
    for $n \in neighborhood(p1)$ **do**
      for $p2 \in neighborhood(n)$ **do**
        $d1 \leftarrow Distance(p1, p2)$
        $d2 \leftarrow$ Distance of the farthest neighbor of $p1$
        if $d1 < d2$ **then**
          Insert $p2$ in $p1$ neighborhood and remove the farthest neighbor of $p1$
        end if
      end for
    end for
  end for
end for

brute-force approach to compute the KNN search using GPU also implemented in the FAISS library.

The source-code of the GPU Random Sample Forest KNN and SWW-tSNE will be publicly available.

4.2.1 Environment

All experiments reported in this work were performed in a 3.20GHz i5-4460 processor with 4 CPU cores, 16GB of processor RAM, and GPU NVIDIA GeForce RTX 2070 with 8GB of GPU RAM, running CUDA 10.1 tools. We used GNU GCC 4.8, and G++ 4.8 to compile each version of t-SNE to preserve compatibility with the original code. In the experiments that investigate graph construction computational time and quality, we use GNU GCC 7.5, and G++ 7.5 to compile RSFK and FAISS implementations.

4.2.2 Datasets

To observe the behavior of SWW-tSNE, datasets of different sizes and dimensions were used, presented in Table 4.1. The MNIST (LeCun and Cortes, 2010) and CIFAR-10 (Krizhevsky et al., 2009) datasets are widely used in supervised machine learning research, which consists of a set of 60000 and 50000 records that represent raw image data, both with ten different classes.

Dataset Lucid Inception represents data extracted from Lucid[4] library, which provides the intermediate layer activations of Convolution Neural Networks (CNN) models, by collecting activations of 100000 images from the ImageNet dataset in the Google Inception V1 CNN architecture (Szegedy et al., 2015).

A similar methodology proposed by Fu, et al. (Fu et al., 2019) was used to create the Amazon Electronics dataset. The FastText library (Joulin et al., 2016) was used to create a 100-dimensional text embedding of the 1689188 text reviews of electronic products from Amazon web store[5] (McAuley et al., 2015) in which every review also contains an integer value between 1 and 5 that represents the overall rate of the review.

The GoogleNews300 dataset[6] consist in the 100 dimensional word embedding of 3 million words created with the Word2Vec (Mikolov et al., 2013) model using the Google News text dataset.

Table 4.1: Dataset sizes used in experiments.

Dataset name	Total of points	Total of dimensions
CIFAR	50000	3072
MNIST	60000	784
Lucid Inception	100000	128
Amazon Electronics	1689188	100
GoogleNews300	3000000	300

[4] https://github.com/tensorflow/lucid
[5] http://jmcauley.ucsd.edu/data/amazon/
[6] https://code.google.com/archive/p/word2vec/

4.2.3 Evaluation

In the experiments of this study, we aim to investigate the efficiency of the proposed SWW-tSNE approach. In this sense, we analyze the impact of the modifications in relation to the quality and computational time. It is possible that the RSFK algorithm presents a different trade-off between quality and time when compared to IVFFLAT. Thus, we investigate the impact of the approximate KNN graph quality in the t-SNE quality and the trade-off by comparing directly the IVFFLAT and RSFK. Finally, we also compare the original implementation of Barnes-Hut t-SNE in t-SNE-CUDA with our proposed approach SWW-tSNE.

For all SWW-tSNE and t-SNE-CUDA experiments described in this chapter, we executed the method for 1000 iterations using the default parameters of t-SNE-CUDA. In KNN experiments different values of K were used (32, 64, and 128). During the t-SNE execution we chose to homogenize the value $K = 32$ in KNN computation of all datasets, considering that the GPU memory size limits the value of K to 32 in the approximate KNN graph computation of the GoogleNews300 dataset.

When executing SWW-tSNE in MNIST dataset, the approximate KNN graph was computed by considering 32 neighbors without the use of the technique *Nearest Neighbor Exploring*. For each run, a different number of trees was used in the RSFK algorithm (from 1 to 70). The corresponding t-SNE quality projection and approximate KNN graph accuracy was measured, every tree leaf was limited to contain from 33 ($minBucketSize$) up to 128 ($maxBucketSize$) points.

Parameter $minBucketSize$ is set automatically to $K + 1$ (33 in MNIST case), so that at each leaf, K neighbors can be found for all points in the leaf. Parameter $maxBucketSize$ can be configured, since it determines the amount of KNN pairs considered at a leaf and, hence, influencing accuracy.

The comparison between the approach using FAISS IVFFLAT approximation and our RSFK implementation to construct the approximate KNN graph was applied to the Amazon Electronics and GoogleNews300 dataset. The IVFFLAT was executed several times using $nlist = \sqrt{N}$, where N is the number of data points and different values of $nprobe$ (from 1 to 20). The RSFK algorithm was executed using different combinations of minimum and maximum number of points in each leaf, different numbers of iterations in the nearest neighbor exploring technique (0, 1, and 2), and a different number of trees (from 1 to 27 in Amazon Electronics dataset and from 1 to 896 in GoogleNews300 dataset). For every execution, we collect the respective approximate KNN graph accuracy and computational time.

In the Amazon Electronics dataset, different values of K were used (32, 64, and 128), but for GoogleNews300 we only could use 32 due to the GPU memory limitation. In the execution of the RSFK algorithm with $K = 128$ we tested a strategy where a leaf can contain less than 129 points. This strategy can lead to an invalid result where the neighborhood of some points can contain less than K elements. These cases can be solved by performing an additional tree at the end of the algorithm forcing leaves to contain between 129 and 258 points (independent of the initial parameters). This procedure ensures that each point will have at least K neighbor candidates.

We have compared the original t-SNE-CUDA version with SWW-tSNE considering the following methodology: For each dataset, the versions implemented where executed ten times to achieve more accurate analysis and three times in the GoogleNews300 dataset due to its size and limited resources. Further, we performed one execution of the original t-SNE-CUDA implementation with the Amazon Electronics and GoogleNews300 datasets by replacing the FAISS library by our RSFK implementation to compute the approximate KNN graph. With this latter experiment we can observe the effect of our RSFK method alone. We split SWW-tSNE

and the adapted t-SNE-CUDA in five main steps. We have collected the computational time of each step:

- KNN: Computation of the nearest neighbor of each data point in the high dimensional space. The result is used to precompute the p_{ij} values;
- Attractive Forces: Computation of (2.9) for each data point using the p_{ij} values;
- Tree Building: Step where all operations necessary to create the Octree are executed.
- Tree Conversion: Necessary step to convert the Sparse tree into an Implicit tree. This step is not needed in the Original version and can be removed in future SWW-tSNE implementations by using a direct massively parallel algorithm to construct the Implicit Tree, in this case, the construction of the sparse tree will also be unnecessary in the SWW version, further saving execution time.
- Tree Traversal: The computation of repulsive forces described in (2.10) using the Barnes-Hut algorithm by executing a Traversal in the corresponding Octree for each data point.

For the datasets with less than one million points, we execute SWW-tSNE with the FLATL2 implementation instead of the RSFK algorithm. In these datasets, the FLATL2 overcomes IVFLAT and RSFK in computational time and quality because the approximate methods may create unnecessary computational overhead. In the datasets, where the approximate KNN graph was computed with RSFK, we use the following parameters:

- Minimum Number of points inside each leaf ($minBucketSize$): 33
  $minBucketSize$ is set automatically to $K+1$, resulting 33 for experiments where $K = 32$.
- Maximum Number of points inside each leaf ($maxBucketSize$): 128
  This produces $buckets$ of size up to $4 * K$ thus enhancing accuracy as discussed before.
- Number of trees t was derived from the amount of work:
  $t = \lceil \frac{N*D}{\alpha * avgBucketSize} \rceil$, where N is the number of points, D is the number of dimensions, the average bucket size is $avgBucketSize = \frac{minBucketSize+maxBucketSize}{2}$.
  The associated constant $\alpha = 24913$ was empirically obtained in different tunning tests.
- Total of iterations in Neighbors Exploring technique: 1

The main focus of the experiments of this chapter, was to analyze and compare different methodologies of t-SNE with dimensionality reduction to 3 dimensions. However, we also perform a preliminary analysis of the SWW-tSNE to create 2-dimensional embeddings to demonstrate that all results discussed in this chapter can also be extended to 2 dimensions and compared with other methodologies present in state of the art, which will be further discussed in Chapter 5. To perform the experiment with dimensionality reduction to 2 dimensions, we have used the following methods: AtSNE[7] (Fu et al., 2019), BH-tSNE and FIt-SNE implementation of t-SNE-CUDA[8] (Chan et al., 2018, 2019), and our SWW-tSNE proposed method. We have used the GoogleNews300 dataset and the default parameters of each implementation except for the perplexity and the value of K for the approximate KNN graph construction. The methodology to create the KNN graph, the perplexity, and K values were the same of the experiments for three dimensions. After the execution of all methods, we have collected the execution time and computed the $R_{\mathrm{NX}}(32)$ metric to compare the quality of each resulted 2-dimensional embedding.

[7] https://github.com/ZJULearning/AtSNE/
[8] ttps://github.com/CannyLab/tsne-cuda/

4.3 RESULTS AND DISCUSSION

The impact of the approximate KNN graph accuracy in the t-SNE result is presented in Figure 4.3. The global structure of the data was already preserved with the low accuracy of the approximate KNN graph. Also, the impact of the increase in KNN graph accuracy in t-SNE quality decreases with higher values of approximate KNN graph accuracy. We believe that the reasons behind this property are related to the fact that the t-SNE can not entirely preserve the neighborhood of each point during the projection. Therefore, an increase in the approximate KNN graph quality does not necessarily yield an increase of t-SNE quality, which can also be affected by other parameters like perplexity, the number of neighbors considered, and the SGD parameters.

Figure 4.4 compares the trade-off between the quality and computational time of IVFFLAT and RSFK. It is possible to note that our RSFK approach surpasses the IVFFLAT approximation in different datasets using different values of K when building the approximate KNN graph. Subfigure 4.4(a) illustrates a scenario where both approximate methods (IVFFLAT and RSFK) demand more computational time and have inferior accuracy in comparison to the exact method, considering accuracies close to 0.9 or higher in the GoogleNews300 dataset. However, the use of RSFK is more suitable when the user does not require a high accuracy, which is the case inside the t-SNE.

The curves marked with "*NN exploring factor*" describe the use of RSFK with the Nearest Neighbor Exploring technique using different numbers of iteration. This method was useful in 32-NN graphs but harms the trade-off between computational time and quality for higher values of K. Subfigures 4.4(a), 4.4(b), and 4.4(c) illustrated this fact for the 64-NN graph. This behavior was already expected since the total of distances computed by neighbor exploring grows quadratic in K, and for this reason, we do not test this approach in the 128-NN graph experiments described in Figure 4.4(d).

An interesting approach to improve the use of RSFK algorithm in approximate KNN graphs with higher values of K is creating trees with leaves that may contain fewer points than K, as shown in Subfigure 4.4(d). However, this may lead to an invalid result where some leaves contain less than K neighbors. These invalid results can be prevented by creating a new tree with leaves that contains a total of points between K and $2K$. The creation of each tree with a reduced size of each leave implicates in a reduced number of distances computed in each tree, which allows the user to create several trees with a low contribution but faster, which improves the trade-off in the referenced experiment.

The usage of Implicit Tree structure and Simulated Wide-Warp strategies have demonstrated to be well suited to improve the scalability of SWW-tSNE without the necessity of rewriting or approximating the computations of t-SNE.

Table 4.4 shows the speedup in the Tree Traversal step in which it was possible to achieve an execution up to 9 times faster than the Original version of t-SNE-CUDA to perform the projection in three dimensions. The Simulated Wide-Warp technique was not wholly successful in the smaller datasets compared to the million size datasets, because when the WarpWidth parameter grows, the number of threads to process the computation of each point also increases. When the number of threads surpasses the real number of points to process, several threads will be idle. Therefore, the technique will not provide improvement of performance and possibly creating a computation overhead without benefits. This problem can be prevented by searching the ideal WarpWidth before the execution of t-SNE by using the information of the current GPU architecture and the number of points to be processed. It can be verified (Zola et al., 2014) that the usage of the Implicit Tree by itself allows the speedup of the algorithm even without using

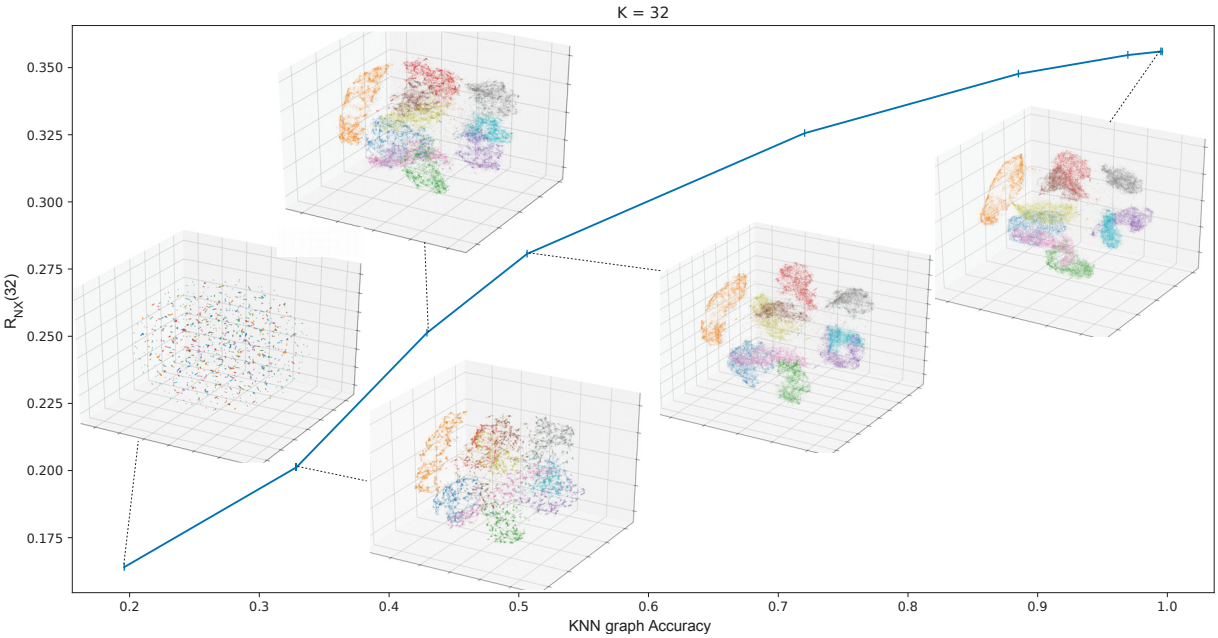


Figure 4.3: Correlation between the quality of approximate KNN graph and the respective t-SNE projection quality using the MNIST dataset for 1000 t-SNE iterations.

the Simulated Wide-Warp method. We have also observed this characteristic in our experiments by making WarpWidth=1.

Figure 1.2 shown in the introduction, illustrates the result obtained from the execution of t-SNE-CUDA on Amazon Electronics and GoogleNews300 datasets, and Figure 4.5 shows the time necessary for the execution of the principal steps of the algorithm. In the figure, it is possible to analyze the impact of the RSFK algorithm in the SWW-tSNE algorithm, which resulted in 223% speedup for the GoogleNews300 dataset experiment and 115% in the Amazon Electronics dataset case. Therefore, we point out that the KNN step was the bottleneck for the GoogleNews300 dataset. Tables 4.2 and 4.3 show the average computational time of each step. In scenarios where Barnes-Hut algorithm is improved, it is possible that the other steps (different from the Tree Traversal and forces calculations) become the bottleneck of execution time, like the computation of the attractive forces.

Table 4.5 presents the speedup of the SWW-tSNE by considering all steps of the t-SNE projection. We were able to achieve projections with at least the same quality as the Original version in up to 4 times faster. This acceleration reflects the impact of using the RSFK algorithm to compute an approximate KNN graph and the modification of the traversal in the octree by using the Implicit Tree representation and Simulated Wide-Warp.

The proposed modifications in SWW-tSNE were implemented in an environment wherein the parameter WarpWidth of the Simulated Wide-Warp technique was limited to create a maximum of four virtual threads for each physical thread before all registers available in the GPU were utilized. One of the most limiting factors to explore values greater than 4 is the use of local variables that grows linearly with the WarpWidth and dimensions of the lower-dimensional space. It is expected that when the same methodology is used to implement a 2-dimensional projection algorithm in SWW-tSNE, a higher number of virtual threads could be created, which would allow even greater speedups than the achieved for three dimensions.

To inspect the projection quality and convergence of each implementation, we analyze the results of each dataset. The convergence was interpreted as the average magnitude of forces

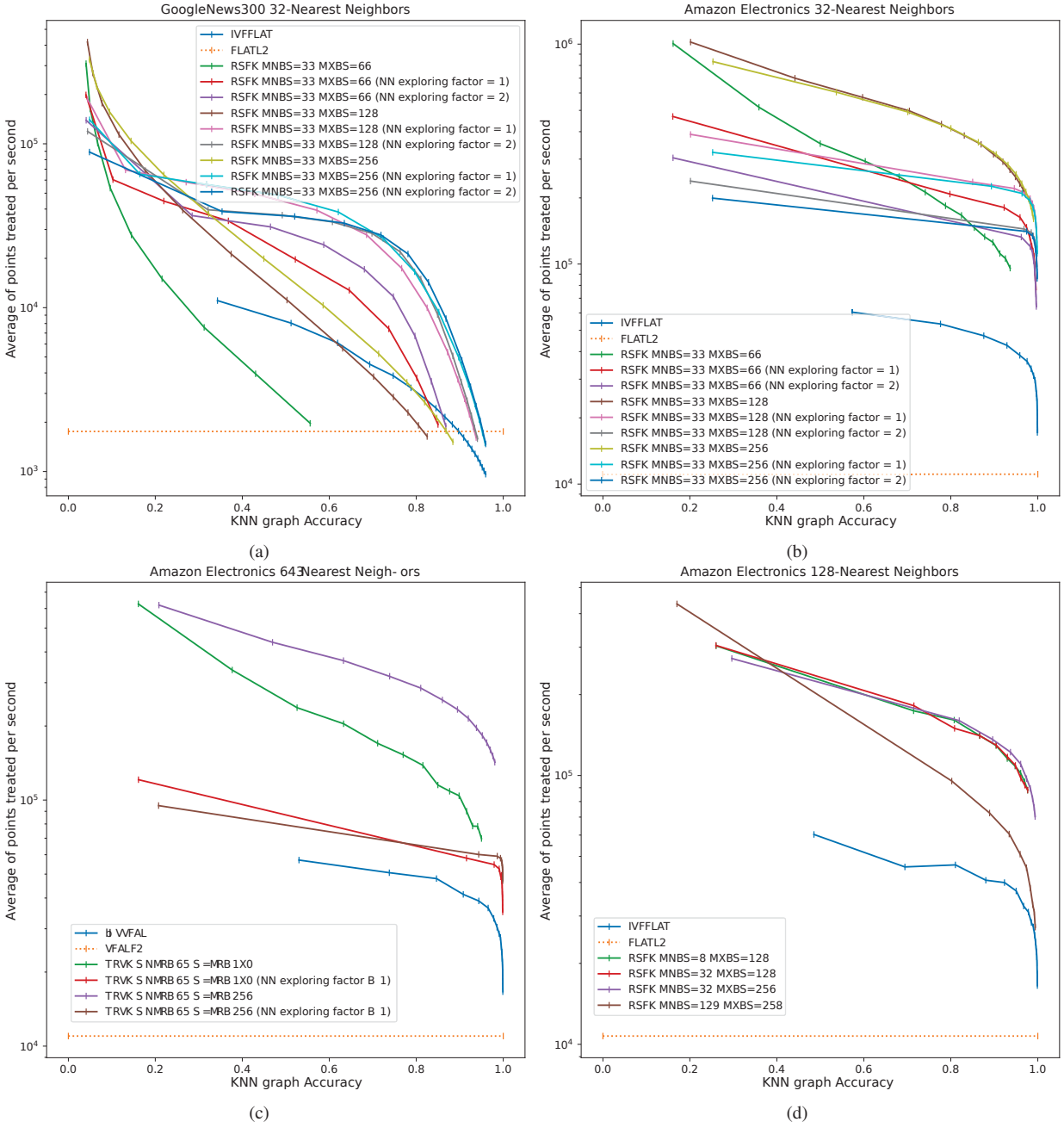


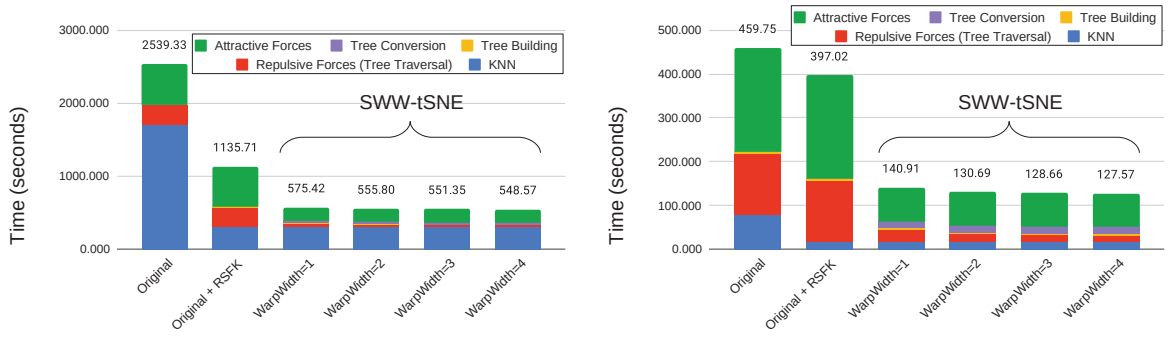
Figure 4.4: Comparison between IVFFLAT, FLATL2, and the proposed RSFK algorithm. The vertical axis represents the total of points divided by the time required to compute the approximate KNN graph (in log scale) and the horizontal axis respective accuracy. Each curve represents the execution of the algorithms with different configurations and by varying the parameters that control the trade-off between computational time and quality. The time of FLATL2 (exact result), which accuracy is 1.0, is used as the baseline and is dashed in the chart. The labels MNBS (*minBucketSize*) and MXBS (*maxBucketSize*) represent the minimum and maximum of points inside each leaf, respectively, in each configuration of the RSFK algorithm.

applied to each point during the iterations of the algorithm present in the original t-SNE-CUDA library, which is called “gradient norm”, and the quality as the $R_{\mathrm{NX}}(32)$ described in (2.15).

Table 4.6 contains the average and standard deviation of $R_{\mathrm{NX}}(32)$ in each dataset with different SWW-tSNE implementations. In the experiments, WarpWidth=4 seems to create a better result in two of the five datasets. In the three smallest datasets, the Kruskal-Wallis H test (Kruskal and Wallis, 1952) with a significance level equal to 0.05 presents a statistical difference between each SWW-tSNE version and the *Original*. This difference is explained by the fact that in these datasets, SWW-tSNE uses the exact KNN graph, and the "Original" approximation uses the IVFFLAT approximation. However, the Kruskal-Wallis H test indicates that there is not any statistical difference between different SWW-tSNE implementations in each dataset, which indicates that the WarpWidth parameter may not affect the quality of the t-SNE projection.

Also, this difference does not seem to impact in the convergence of the algorithm significantly, which is represented in Figure 4.6 that contains the result of one execution of t-SNE in MNIST dataset for each implementation and the gradient norm convergence in Figure 4.6(e). In Figure 4.6, it is possible to note that the number of iterations is more relevant than the difference between each version of SWW-tSNE implemented in this study.

These results are not expected since the Barnes-Hut must be more precise with higher values of WarpWidth (Zola et al., 2014). This characteristic can be explained by the fact that soft adjustments in the approximation of the repulsive forces do not necessarily impact positively in the algorithm convergence, since t-SNE tries to optimize a non-convex function and it is more susceptible to other parameters like the total number of iterations, learning rate, and projection initialization. Future work can explore these characteristics and create better approximations of Barnes-Hut t-SNE in GPU to improve its scalability without harming the quality of projection or the convergence of the algorithm.



(a) GoogleNews300

(b) Amazon Electronics

Figure 4.5: Execution time breakdown of different GPU t-SNE versions applied to Amazon Electronics dataset for 1000 t-SNE iterations. *Original* represents the standard t-SNE-CUDA, *Original+RSFK* represents the standard t-SNE-CUDA using our RSFK implementation instead of the FAISS library to compute the approximate KNN graph, and the other versions represent our implementations using Implicit Tree data structure and Simulated Wide-Warp with different WarpWidth sizes.

4.3.1 2-Dimensional Embedding

Table 4.7 presents the results obtained from the experiment with dimensionality reduction to 2 dimensions. It is clear that SWW-tSNE achieved a high competitive result compared to other techniques considering the local neighborhood preservation and, it had the lowest execution time, either for 2D or 3D embeddings. We could not execute AtSNE and the FIt-SNE implementation

Table 4.2: Computational time (seconds) for 1000 t-SNE iterations with the breakdown of each step, for different methods, to create 3-dimensional embeddings of the GoogleNews300 dataset. WarpWidth=n is abbreviated as WW_n.

Method	KNN	Repulsive Forces (Tree Traversal)	Tree Building	Tree Conversion	Attractive Forces	Total Time
Original	1702.734	268.808	8.319	-	559.465	2539.33
Original + RSFK	304.588	267.115	8.334	-	555.676	1135.71
SWW-tSNE WW_1	304.510	52.527	6.627	30.948	180.807	575.42
SWW-tSNE WW_2	306.613	32.212	6.473	29.329	181.170	555.80
SWW-tSNE WW_3	306.296	28.091	6.474	29.299	181.188	551.35
SWW-tSNE WW_4	304.192	27.395	6.474	29.308	181.203	548.57

Table 4.3: Computational time (seconds) for 1000 t-SNE iterations with the breakdown of each step, for different methods, to create 3-dimensional embeddings of the Amazon Electronics dataset. WarpWidth=n is abbreviated as WW_n.

Method	KNN	Repulsive Forces (Tree Traversal)	Tree Building	Tree Conversion	Attractive Forces	Total Time
Original	78.410	139.447	3.900	-	237.988	459.75
Original + RSFK	17.330	138.945	3.894	-	236.847	397.02
SWW-tSNE WW_1	17.462	27.801	3.156	15.651	76.839	140.91
SWW-tSNE WW_2	17.639	17.484	3.069	15.624	76.877	130.69
SWW-tSNE WW_3	17.682	15.293	3.072	15.769	76.844	128.66
SWW-tSNE WW_4	17.651	14.330	3.069	15.641	76.875	127.57

Table 4.4: Average speedup achieved and (standard deviation) in the **Barnes-Hut Tree Traversal (*Repulsive Forces Calculations*)** in the different versions of t-SNE-CUDA for 1000 t-SNE iterations. WarpWidth=n is abbreviated as WW_n.

Dataset name	Original	WW_1	WW_2	WW_3	WW_4
GoogleNews300	1	5.118 (0.034)	8.345 (0.035)	9.571 (0.116)	**9.813 (0.099)**
Amazon Electronics	1	5.017 (0.059)	7.977 (0.111)	9.120 (0.124)	**9.732 (0.085)**
Lucid Inception	1	1.961 (0.043)	2.558 (0.036)	**3.028 (0.058)**	2.474 (0.074)
MNIST	1	1.609 (0.023)	**2.337 (0.078)**	1.800 (0.075)	1.470 (0.065)
CIFAR	1	1.571 (0.026)	**2.202 (0.037)**	1.631 (0.033)	1.302 (0.030)

Table 4.5: Average speedup achieved and (standard deviation) of the **total time** execution in the different versions of t-SNE-CUDA for 1000 t-SNE iterations. WarpWidth=n is abbreviated as WW_n.

Dataset name	Original	WW_1	WW_2	WW_3	WW_4
GoogleNews300	1	4.413 (0.005)	4.569 (0.006)	4.606 (0.004)	**4.629 (0.007)**
Amazon Electronics	1	3.273 (0.007)	3.532 (0.008)	3.593 (0.006)	**3.620 (0.006)**
Lucid Inception	1	1.367 (0.015)	1.481 (0.012)	**1.547 (0.013)**	1.468 (0.016)
MNIST	1	2.149 (0.009)	**2.376 (0.020)**	2.221 (0.031)	2.087 (0.036)
CIFAR	1	2.823 (0.014)	**3.022 (0.015)**	2.847 (0.016)	2.695 (0.019)

Table 4.6: Average $R_{NX}(32)$ and (standard deviation) achieved for 1000 t-SNE iterations.

Dataset name	Original	WW_1	WW_2	WW_3	WW_4
GoogleNews300	0.0890346 (0.0000893)	**0.0913467 (0.0001298)**	0.0912798 (0.0001610)	0.0913150 (0.0001521)	0.0912839 (0.0001287)
Amazon Electronics	0.1325520 (0.0013960)	0.1328856 (0.0013855)	0.1328982 (0.0013888)	0.1329138 (0.0014039)	**0.1329228 (0.0013814)**
Lucid Inception	0.1969074 (0.0006490)	0.2007211 (0.0008368)	0.2007448 (0.0008405)	**0.2007813 (0.0008754)**	0.2007640 (0.0008370)
MNIST	0.3583563 (0.0010649)	0.3597579 (0.0009353)	**0.3597713 (0.0009455)**	0.3597516 (0.0009252)	0.3597661 (0.0009273)
CIFAR	0.1348499 (0.0009754)	0.1369669 (0.0008853)	0.1370094 (0.0008719)	0.1369815 (0.0009074)	**0.1370318 (0.0008832)**

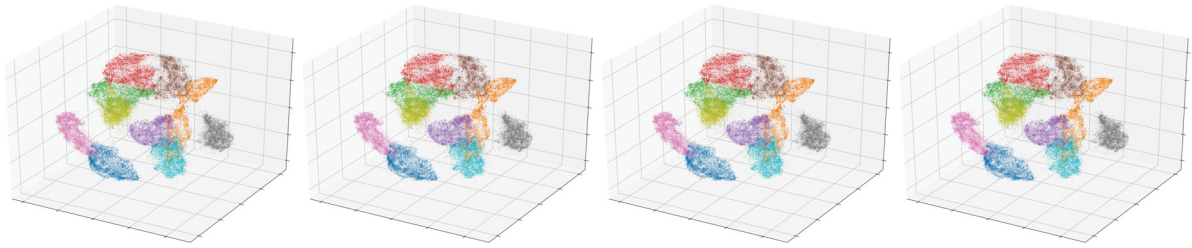
of t-SNE-CUDA with three dimensions since no implementation was available (no results in Table 4.7). However, it is possible to observe that there is a small slowdown when we consider the execution of t-SNE-CUDA and SWW-tSNE to create 3D instead of 2D embeddings. The 2D embedding execution time of t-SNE-CUDA is around 83% lower than the 3D embedding time. For the SWW-tSNE, it was around 91%, in the GoogleNews300 dataset, this indicates that the approximate KNN graph construction was one of the major bottlenecks. In SWW-tSNE, the use of the RSFK algorithm significantly reduces the computational time of the KNN graph computation compared to the original t-SNE-CUDA implementation, which was illustrated in Figure 4.5. Then, it is expected that when the time of the KNN step is reduced, the size of the low-dimensional space will be more relevant since it will impact other steps of the algorithm like those in the Barnes-Hut algorithm.

Considering the quality of the neighborhood preservation, we could observe that the execution with three dimensions achieved better preservation in each analyzed method than the execution with two dimensions. As we previously have discussed, this could be caused by the vaster possible configurations of the embeddings in the 3D space when compared to the 2-dimensional space. Therefore, the SGD algorithm may explore more configurations during the optimization search and converge to better local minimal in 3-dimensional space. In the results of Table 4.7, the local neighborhood preservation was remarkably different for the different methods. However, we did not investigate changing the default parameters of each technique, like the learning rate, since this analysis is beyond the scope of this study. This could explain the difference observed in the results. Therefore, further comparison and analysis of the quality of each method considering dimensionality reduction with two and three dimensions could be an interesting topic to be explored in future works.

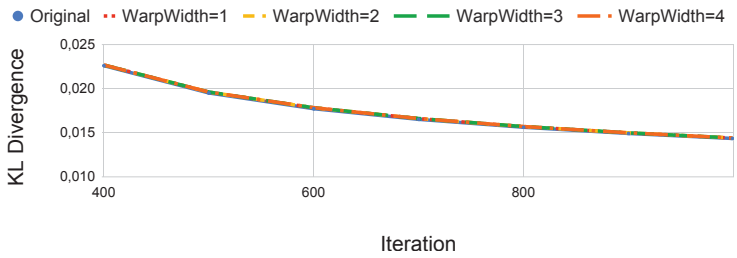
Table 4.7: Comparison of execution time and local neighborhood preservation of different methods to create 2 and 3-dimensional embeddings using the GoogleNews300 dataset for 1000 t-SNE iterations.

Method name	2D $R_{NX}(32)$	3D $R_{NX}(32)$	2D Execution time (seconds)	3D Execution time (seconds)	Slowdown 2D/3D
AtSNE	0.0340	-	9587.341	-	-
t-SNE-CUDA (FItSNE)	0.0559	-	1856.552	-	-
t-SNE-CUDA (BH-tSNE)[a]	0.0318	0.0890	2113.830	2539.326	0.832
SWW-tSNE	0.0600	0.0912	498.864	548.573	0.909

[a]The 3D version is our adapted implementation of the t-SNE-CUDA to project data into three dimensions.



(a) WarpWidth = 1 (b) WarpWidth = 2 (c) WarpWidth = 3 (d) WarpWidth = 4



(e) Convergence

Figure 4.6: Result of the 3-dimensional embedding and convergence generated for MNIST datasets using different versions and fixed random seed for 1000 t-SNE iterations.

5 APPLYING T-SNE TO REAL-WORLD APPLICATIONS WITH LARGE DATASETS

Chapter 4 presented a new method named SWW-tSNE and compared this proposal with other methods. This comparison was focused on applying dimensionality reduction to three dimensions without considering the input data context. This chapter will present a broad comparison of different dimensionality reduction techniques to create two-dimensional embeddings of large scale datasets obtained from real-world applications.

A new method named Simulated Wide-Warp AtSNE (SWW-AtSNE) will be proposed in Section 5.1. The comparison will focus in the importance of preserving local and global structures, and how this could interfere in the interpretation of the result of dimensionality reduction techniques. The Research Questions related to the importance of preserving local and global structures are presented in Section 5.2. This section will also introduce the context of two real-world applications used in the experiments performed in this chapter. Section 5.3 will present the methods used in the experiments of this chapter. The discussion of the results of these experiments will be reported in Section 5.4.

5.1 SIMULATED WIDE-WARP ATSNE (SWW-ATSNE)

In this research, we propose a modification of the SWW-tSNE that uses the main idea based in anchors of AtSNE, and we will refer to as Simulated Wide-Warp Anchor t-SNE (SWW-AtSNE). The SWW-AtSNE still uses the Barnes-Hut algorithm to approximate the result instead of the negative sampling strategy of AtSNE. The implementation of our algorithm does not depend on any structures related to the KNN graph construction, and therefore, could use any algorithm to create the cluster. Our approach suggests using a Random Sample Tree (RST), described by Algorithm 4 to create a partition of the input data points. The Random Sample Tree recursively samples two points and creates a hyperplane equidistant between these points. Each hyperplane will divide the set of points into two new partitions. The recursive division is executed as long as the algorithm can create two new partitions with more points than a parameter that defines the minimum of points that an anchor must be related to, which is controlled by the user. Using the Random Sample Tree, a pre-clustering of the input data points can be created with a linear time complexity algorithm and a computational time insignificant compared to the time required to execute all iterations of the SGD algorithm.

To the best of our knowledge, there is no theoretical proof that the AtSNE cost function (Equation 2.13) is differentiable and that the algorithm of the authors could necessarily converge to a local minimum. Therefore, we choose to mitigate this problem by simultaneously updating the points and the anchors by sharing the same normalization variables, which leads to a more stable algorithm since the SGD struggles with the optimization of non-convex complex functions.

5.2 REAL-WORLD APPLICATIONS

In Section 5.3, we will present our methodologies and the setup used in our experiments to answer the following Research Questions (RQ):

RQ1. Is the SWW-AtSNE proposal comparable to AtSNE and SWW-tSNE methods considering computational time and the quality of the resultant embedding?

RQ2. What is the relationship between the resulting cluster quality computed from different clustering methods (used to define the anchors) and the quality of the SWW-AtSNE?

RQ3. Is the usage of anchors in the t-SNE a strategy any better than other practices that could improve the global structure preservation of the resulted embedding, like PCA usage for the low-dimensional points initialization?

RQ4. How the improvement of the local and global structure representation impact the interpretation of the visualization?

To answer these questions, we have executed different dimensionality reduction algorithms using the MNIST and ImageNet datasets[1]. Moreover, we considered experimenting with the data obtained from two different artificial intelligence applications: Generative Adversarial Networks and Reinforcement Learning.

5.2.1 Generative Adversarial Network

A Generative Adversarial Network (GAN) model can be used to generate realistic images, mostly representing these images as high-dimensional data points. GAN architectures consist of training two different artificial neural networks: the generative and the discriminative networks. The generative network is trained to transform an instance from a noise domain into a domain containing realistic data instances. For example, suppose that the Generative model was trained with images of dogs. In that case, the model could transform a high-dimensional vector with random values into a real image of a dog that not necessarily exists in the training dataset. The discriminative network tries to identify if a given input, for instance, the generated dog's image, is a real image or an artificial image generated by the generative network. Usually, the noise input used in GAN models are high-dimensional vectors generated with a random uniform distribution, where different random seeds create different input. Each input will be transformed using convolutional neural networks into a different realistic artificial image. Various approaches use this idea to generate realistic artificial data from different domains, like text and images. The BigGAN (Brock et al., 2018) is a study that investigates GAN models specifically in the context of large-scale applications. In our experiments, we have used two of the available models of BigGAN[2][3] trained on the Imagenet dataset to generate 100000 images and their representations as vectors with 1536 dimensions for ten different classes. We have added five additional images to reproduce the example of the original research, where the generative model creates an interpolation between images of two different classes (see Figure 5.3).

5.2.2 Deep Q-Network

The last example of artificial intelligence application that we have considered in our experiments consists in the visualization of the training process of the Deep Q-Network (DQN) model (Mnih et al., 2015) to learn how to play the Phoenix Atari game available in the Gym library[4]. Using the Gym library, we can run any agent to control the player using the screen image and a reward quantification for the actions of the agent. In the library, we can run any number of episodes. An episode represents a complete simulation of the agent in the environment (game) since the beginning until the last frame where the player loses the game. The DQN is a break-point

[1] https://github.com/ZJULearning/AtSNE
[2] https://tfhub.dev/deepmind/biggan-deep-128/1
[3] https://tfhub.dev/vtab/cond-biggan/1
[4] https://gym.openai.com/envs/Phoenix-v0/

approach recently proposed in the reinforcement learning field, allowing computational agents to learn how to play some Atari games as good as humans. In the DQN model, the standard Q-Learning algorithm is used, where the agent learns to estimate how good it is to take any of the available actions for a specific state (Q-value). The Q-Values can be interpreted as the immediately expected reward that the agent playing the game will obtain if a given action was chosen at a particular state. In the policy that uses the trained model to play the game, the best Q-value is chosen to achieve states that give higher rewards, which are score-points in the game environment. The DQN model uses a Convolutional Neural Networks (CNN) to transform any image frame of the game into a high-dimensional vector with 512 dimensions representing the state of the agent. We have used the implementation available in the Keras-RL project[5] to train the DQN model for 1750000 episodes, and save the trained model in different stages of training (episodes 250000, 1000000 and 1750000). For each saved stage, we have executed the agent for 600 episodes. At every four frames, we have obtained the frame image, their representation as vectors with 512 dimensions, and the associated Q-values of the action that leads the agent to that state. The agent actions result reported in Table 5.1 shows that the model obtained from the last stage has an average reward smaller than the intermediate stage of the training. This is an impressive result considering that it is expected that a higher number of training episodes should result in better agents.

Table 5.1: Information of the different stages where the agent model were saved and their respective performance measured by the average reward for all steps from 600 episodes.

Number of training episodes	Generated Dataset Name	Average Reward and (standard deviation)
250000	Phoenix-V0-250K	11.79 (5.00)
1000000	Phoenix-V0-1M	29.68 (10.11)
1750000	Phoenix-V0-1M750K	27.85 (8.99)

5.2.3 Medium Neighborhood Preservation (MNP)

To measure the global preservation in dimensionality reduction for large datasets, we propose and use a new metric called Medium Neighborhood Preservation (MNP). Following the idea of the preservation of pairwise distances, we try to verify if the order of distances for non-neighbors points is preserved for all points. A sample s_X of size C must be sampled from the high-dimensional points by ensuring that $x_i \notin \nu_j^K : \forall x_i \in s_X; \forall j \in [1, \ldots, C]$, where K is the considered neighborhood size. Therefore, s_X is a set of points that are not neighbors, which gives a possible skeleton layout of the input data points. The MNP is computed using Equation 5.1 and considering R'_{NX} as the computation of Equation 2.15 but using the sample s_X and their representation s_Y in the low-dimensional space to compute the neighborhoods ν_i^K and n_i^K. The MNP can be computed multiple times by choosing different samples s_X and s_Y, which will give a more accurate average to measure the global information preserved by the dimensionality reduction result. As we will present in the Section 5.4, the MNP can successfully represent the global structure preservation for different dimensionality reduction techniques using low values of C regardless of the size of the input size. Therefore, MNP computation requires a constant time if the neighborhood of the points were already computed.

$$MNP(K) = R'_{\text{NX}}(\lfloor C/2 \rfloor) \tag{5.1}$$

[5] https://github.com/keras-rl/keras-rl

5.3 METHODS

Using the MNIST and ImageNet datasets, and the BigGAN and DQN applications mentioned before, we were able to prepare six different datasets presented in Table 5.2.

Table 5.2: Dataset useds in the experiments. Deep Neural Network is abbreviated as DNN.

Dataset Name	Number of data points	Number of dimensions	Dimensions description
MNIST	70000	784	Pixel values (gray images).
ImageNet	1275219	128	DNN Intermediate layer activation.
Phoenix-V0-250K	513575	512	DNN Intermediate layer activation.
Phoenix-V0-1M	983557	512	DNN Intermediate layer activation.
Phoenix-V0-1M750K	1043054	512	DNN Intermediate layer activation.
BigGAN	500005	1536	DNN Intermediate layer activation. The representation was created using the discriminative network related to the BigGAN model.

For these experiments, we have used a computer with a 3.20GHz i5-4460 CPU processor, 16GB of processor RAM memory, and an NVIDIA GeForce GPU RTX 2070 with 8GB of GPU RAM, running CUDA 10.1 toolkit. All datasets described in Table 5.2 were used to compare different methods of dimensionality reduction to 2 dimensions.

We have compared five different methods: SWW-tSNE, SWW-AtSNE, AtSNE, the GPU UMAP implementation available in the cuML project[6](version 0.14), and the CPU implementation of Principal Component analysis available in the Scikit-Learn library[7]. We could execute the UMAP method in the experiments using only the MNIST and ImageNet datasets due to our limited time and resources and the high computational cost of the used implementation.

All dimensionality reduction techniques were used to project data into 2 dimensions. Considering that the initialization of the low-dimensional points has potential influence in the global structure for t-SNE and similar techniques (Kobak and Linderman, 2019), we have configured the SWW-tSNE method to perform dimensionality reduction using two different strategies for pre-processing and initializing the data. One strategy consists in applying a dimensionality reduction of the input data points using the PCA with 16 components, which reduces the computational time and memory cost of the K-NN Graph construction used to create the pairwise similarities (Equation 2.3) of the high-dimensional data in t-SNE. This pre-processing is a discussed approach in previous researches to deal with datasets with many dimensions (Van Der Maaten, 2014). The other strategy uses PCA to initialize the low-dimensional embedding of t-SNE instead of a random normal distribution. Therefore, it initializes the resultant embedding of t-SNE with an excellent global quality (McInnes et al., 2020) but with poor neighborhood preservation since it is not the main objective of PCA.

The SWW-AtSNE was executed using two different clustering methods. The first clustering method is the Random Sample Tree (RST), a "weak" clustering that can be computed in negligible execution time compared to the time that t-SNE requires. The second clustering consists of applying kmcuda[8], a method with better quality result than RST, which implements the K-Means algorithm to use the GPU efficiently, but still requires a significant execution time compared to t-SNE. Further, we have computed the K-NN Graph approximation in SWW-tSNE and SWW-AtSNE algorithms with the Random Sample Forest KNN method .

[6] https://github.com/rapidsai/cuml

[7] https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html

[8] https://github.com/src-d/kmcuda

5.3.1 Parameters

We have used the following parameters to execute each technique described before:

- Pexplexity: 16
- Number of Neighbors (K): 48
- Total of iterations of SGD: 10000

For the SWW-tSNE and SWW-AtSNE approaches, we were able to adopt the suggestion of Linderman and Steinerberger (2019) to define a relation between the early exaggeration factor and the learning rate. Then, we have defined the $learning\ rate$ parameter as 1000 and the early exaggeration factor as $\frac{N}{10 \times learning\ rate}$, where N is the total number of data points of each dataset.

5.3.2 Evaluation

Each approach for dimensionality reduction was executed five different times using different random seeds. From the execution, we have collected the execution time and estimated the preservation of the local structures using $R_{\mathrm{NX}}(16)$ and the preservation of the global structure with an average of the 1000 different samples of MNP(16) and a sample size $C = 100$. The next section will also present the scatter plot obtained from the 2-dimensional embedding of the dimensionality reduction methods. For the BigGAN dataset, the position in the embedding of each class interpolation image was identified. Besides, we have analyzed the image frames with different associated Q-Values in the DQN datasets by comparing their respective positions in the embedding resulted from the different dimensionality reduction approaches.

5.4 RESULTS AND DISCUSSION

This section will discuss the results of our experiments and their relation to the research questions described in Section 5.3. The figures 1.1 and 5.3 illustrates the results of dimensionality reduction for the BigGan application. Figure 5.6 presents the visualization of different stages of an agent training with the DQN method using 2-dimensional embeddings obtained with SWW-tSNE. The measures of the quality of the result and execution time of each compared technique are presented in Figure 5.2.

5.4.1 Overall comparison (RQ1, RQ2)

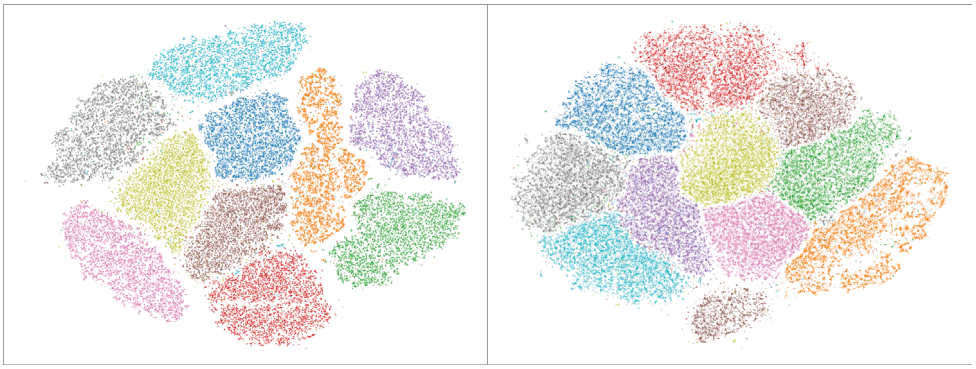
Figure 5.1 presents one result embedding of the MNIST dataset for each methodology compared in this research. The average computational time and quality (local and global preservation) of each method are presented in Figure 5.2 for different datasets. A comparison between the 2-dimensional embedding generated by t-SNE-CUDA and AtSNE is described by Fu et al. (2019). However, the comparison does not take into account the same number of nearest neighbors for both techniques to perform the K-NN graph construction, due to the limitation of executing t-SNE-CUDA with large datasets and bounded by memory and computational costs. In our experiments, since the number of neighbors is fixed for all techniques, we made a fairer comparison.

The execution time required for the cuML UMAP implementation for GPU was the largest between the compared techniques. This is an expected result since it is not based in any study that focuses on strategies for efficient approximations and implementation of UMAP in

GPU, which is not the case of AtSNE, t-SNE-CUDA, SWW-tSNE, and SWW-AtSNE. The global and local structures preservation does not justify the high computational time required for the used implementation of UMAP in GPU. However, it is essential to mention that further studies could improve UMAP to be implemented with efficient programming primitives for GPU and also be used with fine-tuning parameters, which is beyond the scope of this research.

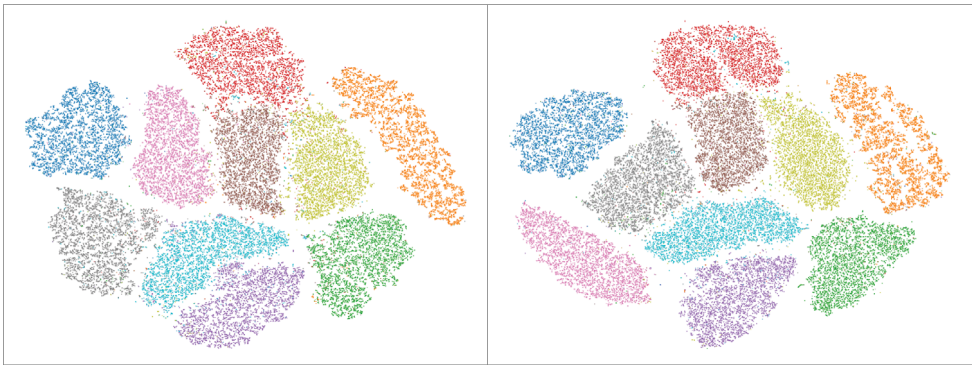
The AtSNE was able to preserve the quality for different runs, which can be observed in the low standard deviation in the Figure 5.2. However, the global structure preservation of AtSNE was not preserved better than SWW-tSNE or SWW-AtSNE except for one dataset (Subfigure 5.2(e)). Despite the inferior quality result, AtSNE could execute faster for the smaller datasets, but it is one of the most time-consuming methods for the largest datasets. This difference could be caused by the method used to approximate the repulsive forces in each technique, indicating that the negative sampling used in AtSNE without tunned parameters can disturb the convergence of the algorithm to achieve better local and global structures preservation. Further studies could try to investigate this characteristic with more experiments using different parameters.

Compared to SWW-tSNE, the SWW-AtSNE achieved better or similar global quality results without reducing the nearest neighbor preservation for all datasets. When we compare the methods to compute the clustering used to generate the anchors of SWW-AtSNE, it becomes evident in our experiments that the RST algorithm influences the algorithm to create results with better global preservation compared with the K-Means. The increase in the global preservation could be explained by the fact that the convergence of SGD in the SWW-AtSNE with RST could consider the similarities between distant points defined by t-SNE since these points could be related to the same anchor. This characteristic is possible since the cluster resulting from the RST is inferior (distant points could be clustered together) to K-Means. Even with GPU usage and the optimizations present in the kmcuda library, the computation of K-Means increases the computation time of SWW-AtSNE significantly compared with the usage of “weaker” clustering algorithms like the RST. The effort to create and use high-quality clusters could demand higher computational time and decrease the preservation of the global structures. Therefore, we cannot observe any reason for using high-quality clustering algorithms to generate the anchors of AtSNE and SWW-AtSNE.



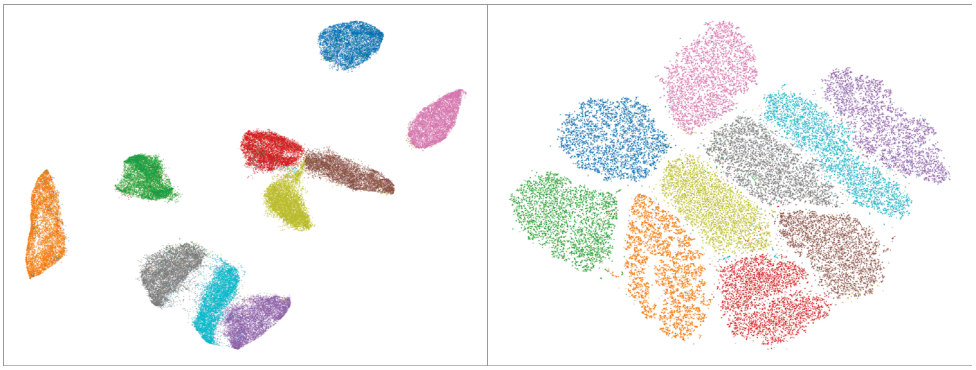
(a) SWW-AtSNE with K-Means.
$R_{\mathrm{NX}}(16)$=0.395. MNP(16)=0.083.

(b) AtSNE.
$R_{\mathrm{NX}}(16)$=0.308. MNP(16)=0.108.



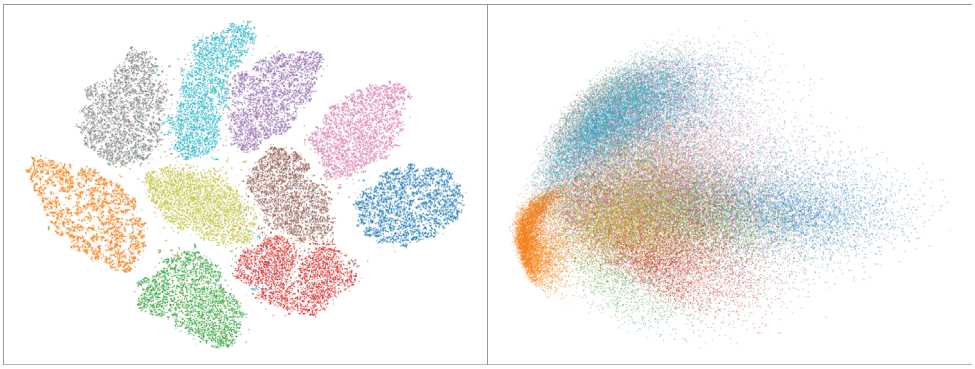
(c) PCA with 16 components + SWW-tSNE.
$R_{\mathrm{NX}}(16)$=0.312. MNP(16)=0.146.

(d) SWW-tSNE.
$R_{\mathrm{NX}}(16)$=0.395. MNP(16)=0.167.



(e) UMAP.
$R_{\mathrm{NX}}(16)$=0.165. MNP(16)=0.171.

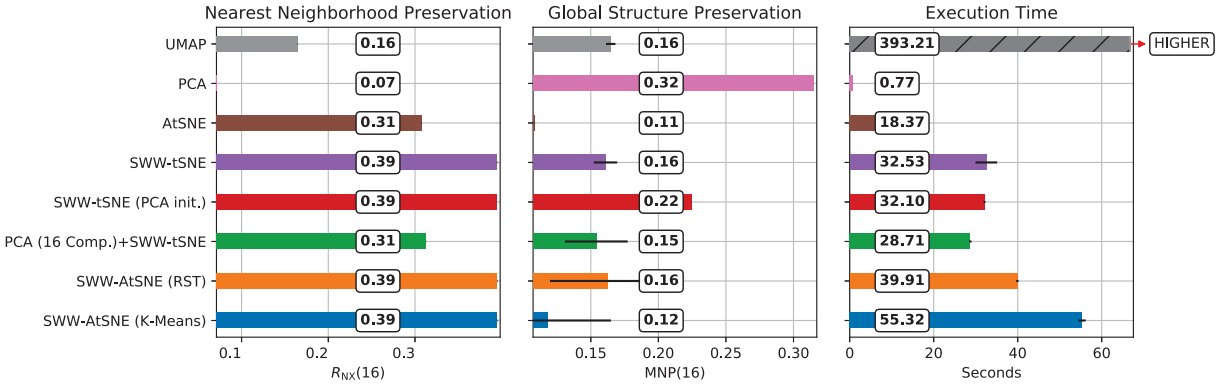
(f) SWW-AtSNE with Random Sample Tree.
$R_{\mathrm{NX}}(16)$=0.394. MNP(16)=0.199.



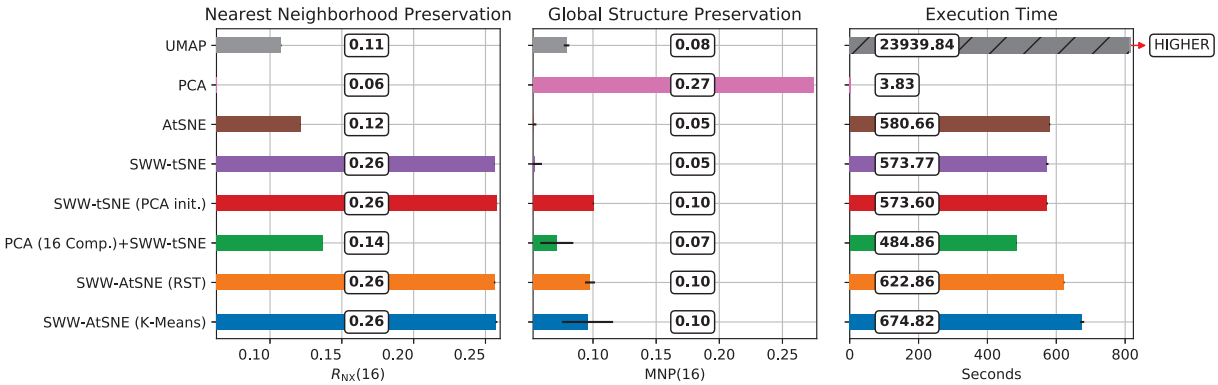
(g) SWW-tSNE with PCA Initialization.
$R_{NX}(16)$=0.394. MNP(16)=0.224.

(h) PCA.
$R_{NX}(16)$=0.071. MNP(16)=0.315.

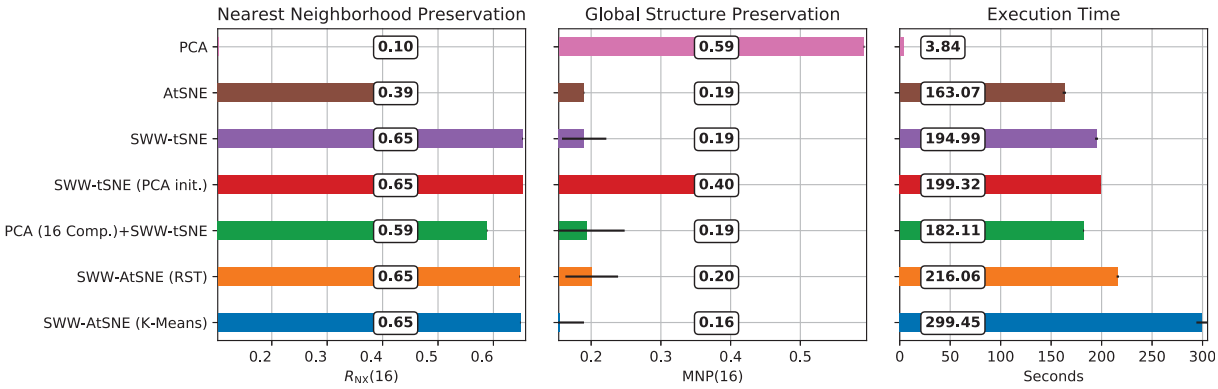
Figure 5.1: 2-Dimensional embedding result of the MNIST using different dimensionality reduction approaches. Note that plots (a) to (h) are in increasing order of Global Structure Preservation, as evaluated by our proposed MNP(16) metric.



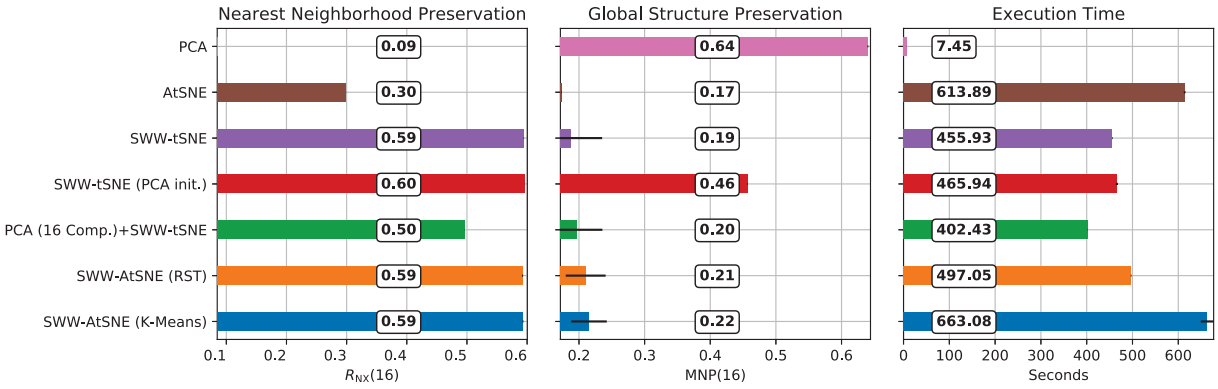
(a) MNIST. The axis scale of the computational time in the chart does not represent the execution time of UMAP, which is explicit in the bar.



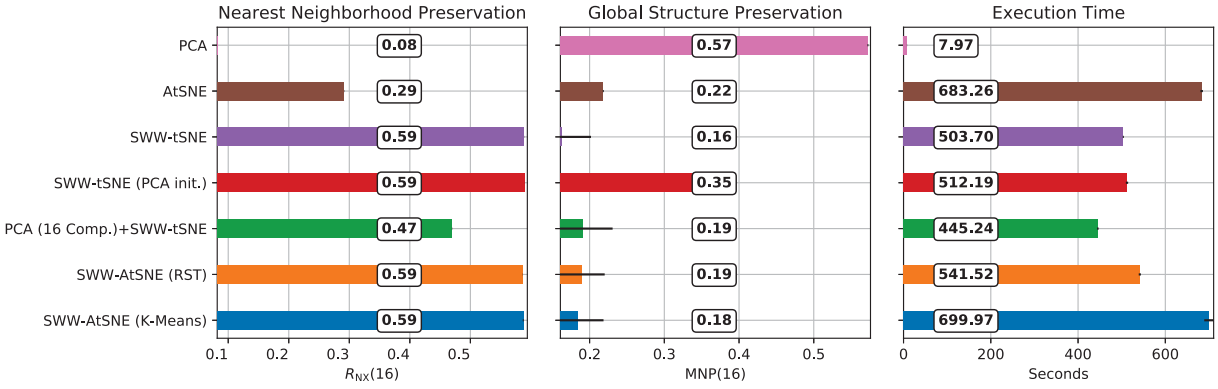
(b) ImageNet. The axis scale of the computational time in the chart does not represent the execution time of UMAP, which is explicit in the bar.



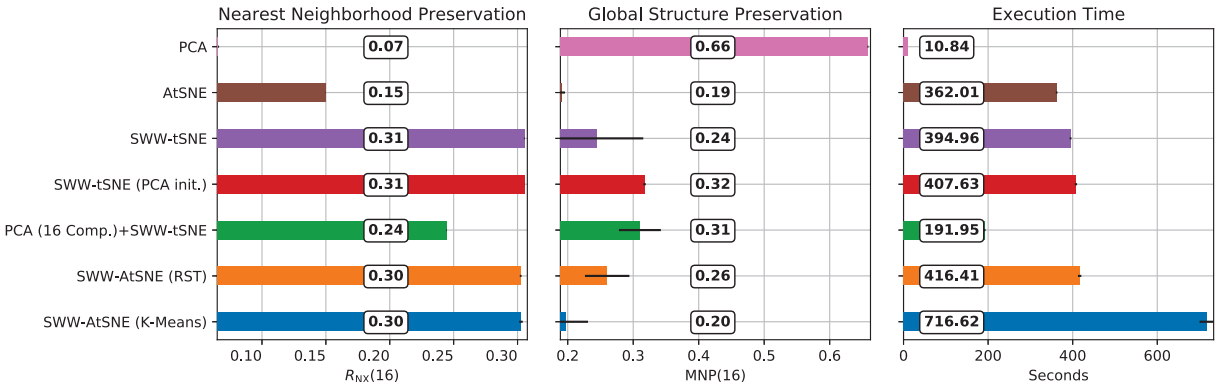
(c) Phoenix-V0-250K



(d) Phoenix-V0-1M



(e) Phoenix-V0-1M750K



(f) BigGAN

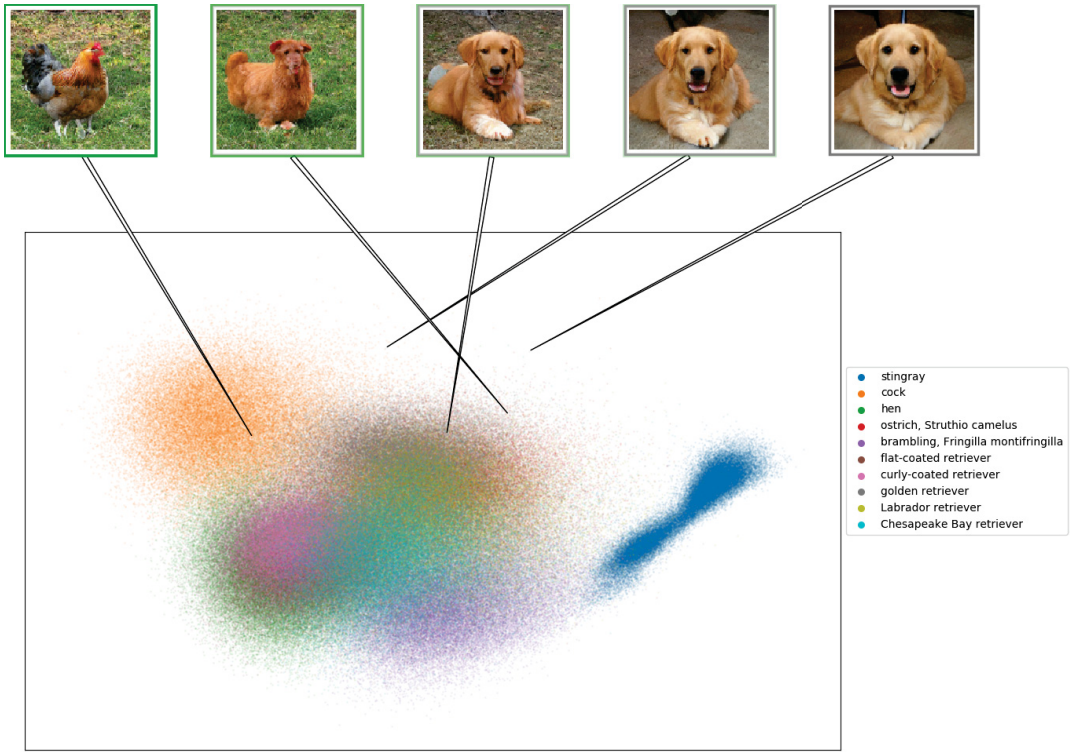
Figure 5.2: Comparative of different dimensionality reduction techniques for different datasets. The global structure preservation were measured using 1000 samples of the MNP(16) metric with sample size $C = 100$. The Nearest Neighborhood Preservation was measured using the $R_{NX}(16)$ metric.

5.4.2 PCA Initialization and Global Structures Preservation (RQ3)

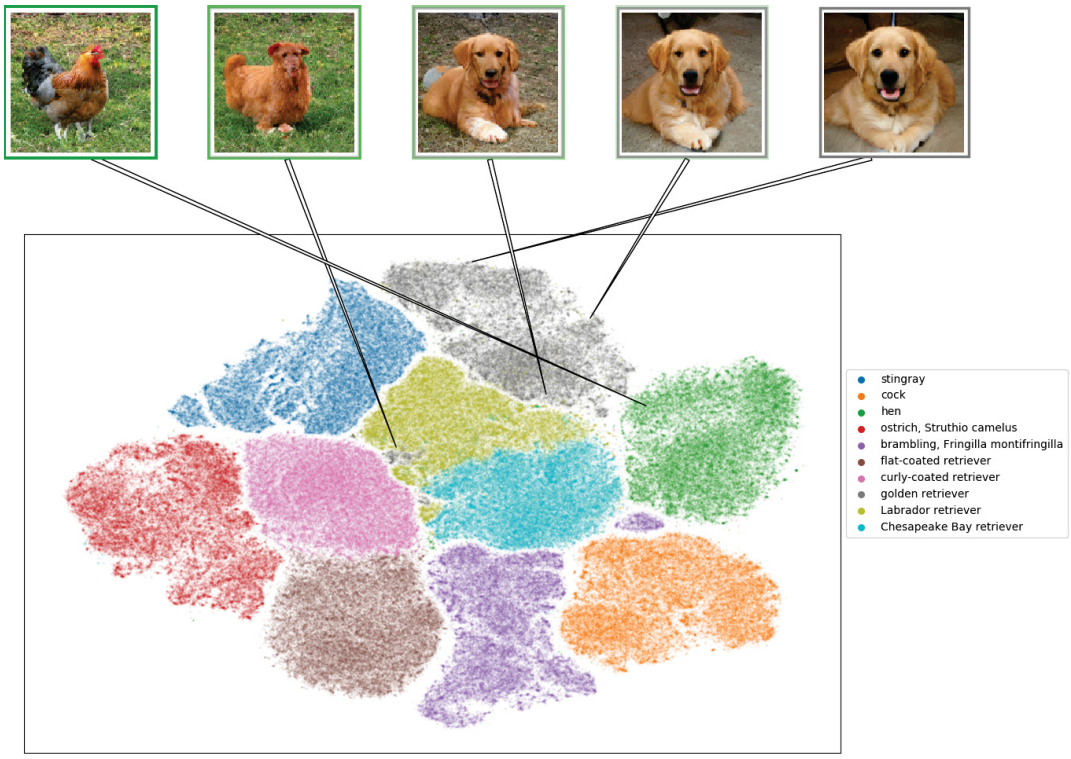
The best algorithm in our experiments to preserve global structures was the PCA, which could be executed in less than 11 seconds for all datasets. However, as expected, PCA could not preserve the nearest neighbors of each point since it is a technique that focuses exclusively on preserving the covariance of the data. The second best technique between the analyzed methods was the SWW-tSNE using the PCA initialization strategy. Using this strategy, we could significantly increase the global structure preservation, reduce the variability in the technique output, and maintain one of the best nearest neighborhood preservation for all datasets. Figure 5.3 presents an example using the BigGan dataset, where we want to preserve both local and global structures to observe a specific property of the data. In the example, the images generated by the generative network are represented by the discriminative network "perspective" as high-dimensional data points that were projected into two dimensions. Five of these images were specifically generated using a linear interpolation between two classes, which could be specified in the BigGan conditional model (Brock et al., 2018). If it is required to analyze the capacity of the discriminative network to represent this interpolation, a dimensionality reduction could be used to gain insight. In the figure, it is possible to see that only with a technique that reaches a good local and global preservation of the structures (Figure 1.1) could be useful to observe the interpolation in the 2-dimensional representation. This type of interpretation could be useful in interpreting the data in different AI applications. However, the users must be aware of the flaws of the methods to preserve the information of the high-dimensional data. Therefore, the dimensionality reduction techniques used for these applications should be widely chosen considering their capacities to preserve local and global structures.

Another approach to use PCA with t-SNE is to pre-process the data using PCA to reduce the dimensionality of the data using a small number of components. In Figure 5.2, we can see that this could significantly reduce the execution time of SWW-tSNE since it reduces the number of dimensions of the input for the K-NN Graph computation. However, it could reduce the local structure preservation, which was explicit in the ImageNet dataset (Subfigure 5.2(b)). This technique could be a well-suited if the user wants to execute t-SNE with high Perplexity, which will require a large number of K in the K-NN Graph, increasing the memory required and computation cost and can be mitigated by reducing the dimensionality of the input. Our experiments show that if this method is used, the user should be aware of the trade-off between

time and local structure preservation. Therefore, the user should widely choose the number of components to preserve as much information as possible.



(a) BigGan dataset visualization with PCA.



(b) BigGan dataset visualization with AtSNE.

Figure 5.3: Representation of images of the BigGAN dataset using dimensionality reduction obtained by the execution of PCA and AtSNE. Each color represents a different class.

5.4.3 Real World Applications Interpretability (RQ4)

Considering the usage of the dimensionality reduction to interpret the BigGan dataset, the representation illustrated in Subfigure 5.3(a), generated with PCA, has a high intersection between different classes of images. Moreover, the representation appears to contain only one cluster with some outliers, which is not the case. Unlike AtSNE (Subfigure 5.3(b)) and SWW-tSNE with PCA initialization (Figure 1.1), the image generated with PCA fails to preserve the local preservation. It consequently also fails to preserve real clusters present in the high-dimensional data points.

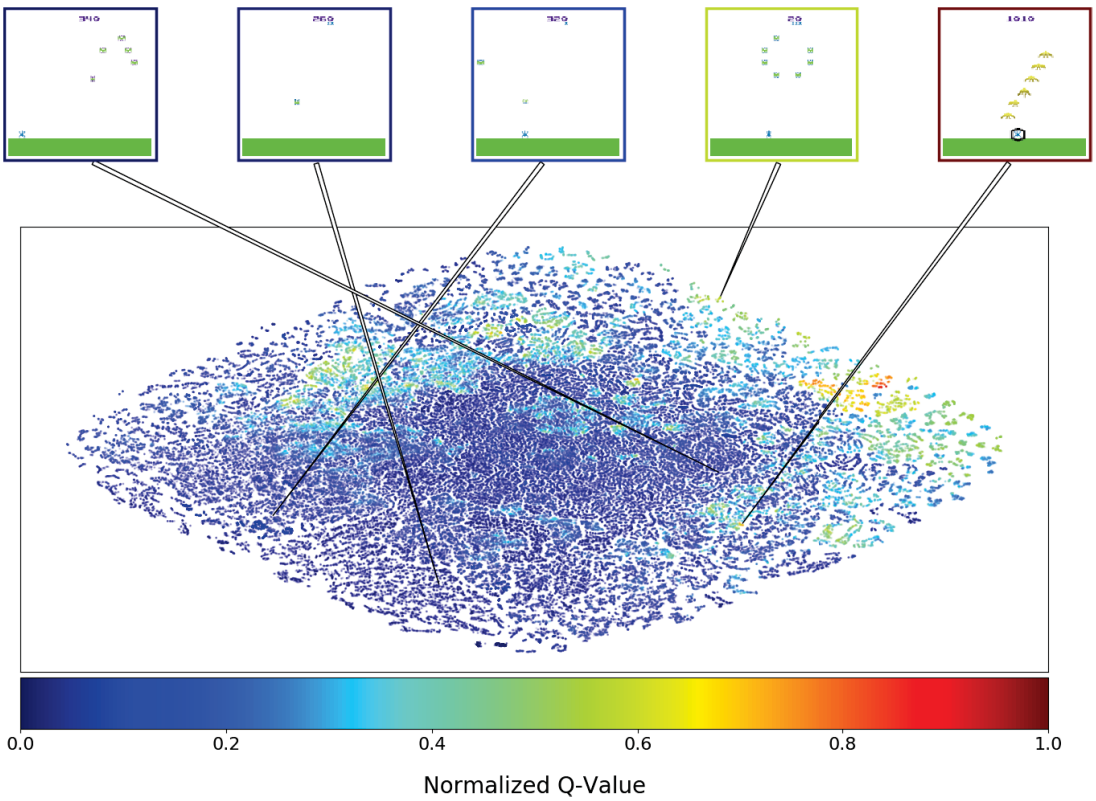
In the Subfigure 5.3(b) generated with AtSNE, it is clear to identify different clusters for each class. These clusters could be visualized even without the usage of the labels of the points. However, one of the image classes, related to the purple color, has been split in two different clusters. Another problem of AtSNE is that it could not represent the low-dimensional data points in a way that the linear interpolation between two images could be identified. Figure 5.2 indicates that SWW-tSNE with PCA initialization results have a higher local and global structure preservation compared with AtSNE. Furthermore, Figure 1.1 indicates that SWW-tSNE with PCA initialization does not suffer from the problems mentioned before for AtSNE to interpret the high-dimensional points of the BigGan dataset, and the linear interpolation is evident in the 2-dimensional embedding.

Figure 5.6 presents another application where data visualization through dimensionality reduction could be useful to interpret some characteristics of the high-dimensional data points. Each data point is associated with the DQN representation of a frame in the Phoenix game in the figure. The color of each point represents a Q-Value, which can be interpreted as an estimation for the agent of the expected reward for the action that leads to that specific state. We can see

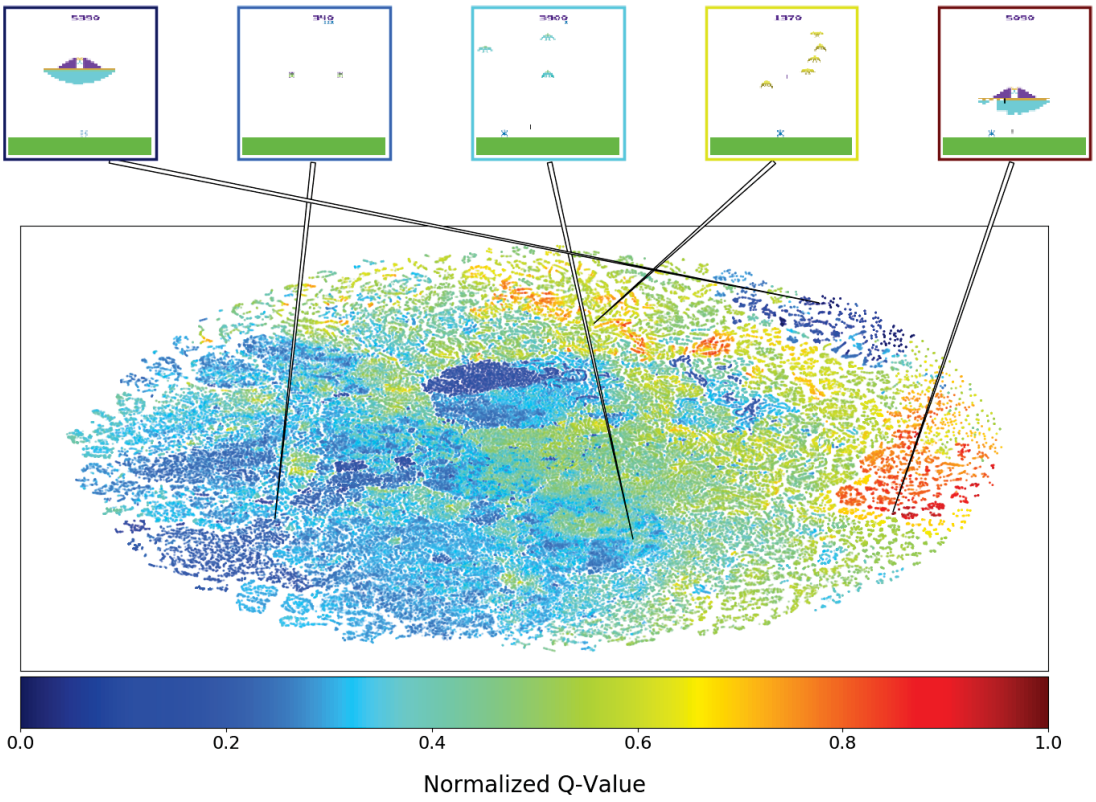
that in the earliest stage of the training, the Q-Values are relatively low (blue in the figure) for all executed episodes, compared with the other scenarios where the agent was trained with more episodes. This is expected since the agent could not reach advanced parts of the game at this stage. Therefore, it will associate high Q-Values for actions related to "easy" immediate rewards. For instance, the top-right frame in the Subfigure 5.4(a) has the best associated Q-Value and indicates a state where the player is invulnerable and able to destroy several enemies, which will guarantee the achievement of immediate rewards.

Another insight that t-SNE could give by analyzing the Figure 5.6 is a possible explanation for the result presented in Table 5.1. A possible explanation relies on the possibility that the method and parameters used to train the DQN model could not explore the environment enough to surpass one of the final challenges of the game, represented by the top-right frame in the Subfigure 5.5(a). The frames related to these final parts of the game probably would lead to the states close to the end of the game where the player loses, which have low Q-Values and affect the agent model considering the Q-Learning algorithm. Therefore, during the training stage, the agent could update the model to give lower Q-values for these advanced parts of the gameplay.

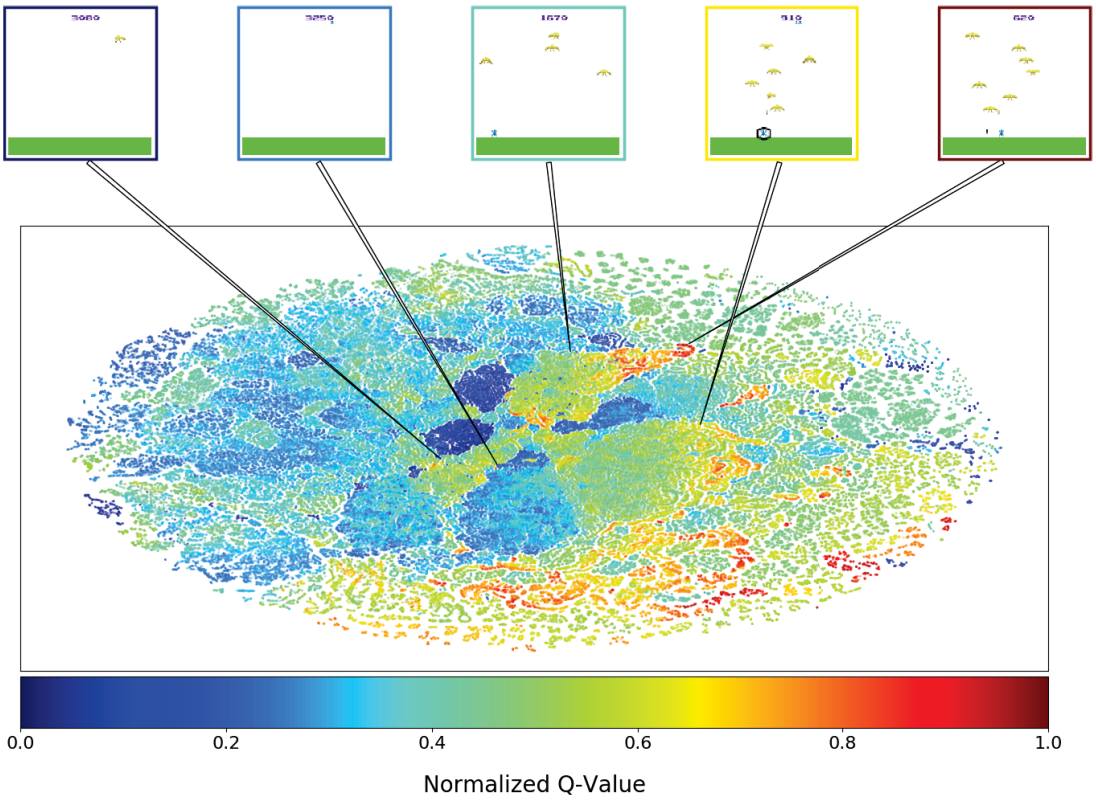
With the 2-dimensional embedding, we can see that the agent represented several data points with high Q-value and are similar to the frame with the higher Q-value of Subfigure 5.5(a), represented by several clusters preserved by the global structure preservation of the SWW-tSNE method using PCA initialization. However, these structures cannot be identified in the 2-dimensional embedding of the Subfigure 5.6(a), which reinforces the possibility that the agent deteriorates the Q-values of advanced parts of the game. If it is correct, this could support the propositions of new strategies to use the DQN model with different policies for the exploration-exploitation trade-off.



(a) Phoenix-V0-250K visualization with SWW-tSNE using PCA initialization.



(a) Phoenix-V0-1M visualization with SWW-tSNE using PCA initialization.



(a) Phoenix-V0-1M750K visualization with SWW-tSNE using PCA initialization.

Figure 5.6: Phoenix-V0 visualization with SWW-tSNE using PCA initialization. The points color represent the normalized Q-Value related to the action that leads the agent to that state. The frames at top of each image represent samples of states with different Q-Values. The color of each frame were inverted in order to obtain better contrast in the image.

6 TECHNOLOGICAL AND BIBLIOGRAPHIC PRODUCTION

The development of this research allows the production of four bibliographic works and two technological projects. The bibliographical production are:

- Meyer, B. H., Pozo, A. T. R., and Zola, W. M. N. (2020). Improving barnes-hut t-sne scalability in gpu with efficient memory access strategies. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8
- Meyer, B. H., Pozo, A. T. R., and Zola, W. M. N. (in press 2021). Improving Barnes-Hut t-SNE algorithm in modern GPU architectures with Random Forest KNN and Simulated Wide-Warp. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*
- Meyer, B. H., Pozo, A. T. R. and Zola, W. M. N. (under review 2021). t-SNE based techniques comparison for dealing with large scale datasets. Expert Systems with Applications
- Meyer, B. H., Pozo, A. T. R. and Zola, W. M. N. (under review 2021) Algoritmo RSFK para busca de similaridade em GPU. In Anais da XXI Escola Regional de Alto Desempenho da Região Sul. SBC, 2021

The technological production consists of implementing two libraries that can be used in C++ and Python programming languages. The first library is implementing the SWW-tSNE and SWW-AtSNE, which was based in the t-SNE-CUDA project available in GitHub [1]. Several modifications were made to improve the project, including the possibility of using other KNN Graph algorithms rather than the usage of the FAISS library. The second library is the implementation of RSFK, which allows the users to compute the KNN Graph and partitioning the input data with one or more Random Sample Tree.

The SWW-tSNE and SWW-AtSNE will be publically available at `https://github.com/BrunoMeyer/sww-tsne` after the remaining publications and when all code dependencies were revised. The RSFK library is already available at `https://github.com/BrunoMeyer/gpu-rsfk`.

The public repositories will contain example codes to execute the experiments described in this research, which will improve the reproducibility of the methods and facilitate the development of future works.

[1] `https://github.com/CannyLab/tsne-cuda/`

7 CONCLUSIONS AND FUTURE WORK

Despite t-SNE-CUDA efficiency and competitiveness among state-of-the-art t-SNE implementations in GPUs, to the best of our knowledge, there are no previous work verifying the impact of Implicit Tree representation and SWW techniques in GPU t-SNE. Further, we are unaware of any work that investigates GPU implementations of the Random Projection Forest with strategies like those proposed in the LargeVis algorithm. LargeVis is a well-known CPU implementation that uses concepts like t-SNE to perform dimensionality reduction.

This research has investigated the advantages of these techniques considering the potential speedup in the Barnes-Hut and approximate KNN graph construction steps of t-SNE. This investigation led to the proposal of SWW-tSNE, which allows up to 460% faster execution of dimensionality reduction than the t-SNE-CUDA.

One of the most computationally expensive steps of t-SNE-CUDA is the calculation of repulsive forces while traversing the tree in the modified Barnes-Hut phase, which is a memory bound step. Our current work further contributes to mitigate the memory accesses and reduce execution time. Our experiments demonstrate that strategies such as the usage of the Implicit Trees and Simulated Wide-Warp can speedup the tree traversal and repulsive forces calculations step in up to 980% in 3 million point datasets. The t-SNE-CUDA algorithm uses the FAISS library to compute an approximate KNN graph using GPU, needed to execute the t-SNE method. We have proposed a simplification of the Random Projection Forest, the RSFK, that is similar to the methods used in LargeVis algorithm. The performance of our RSFK method was analyzed in GPU manycores. The experiments presented in this study indicate a fair competition with the IVFFLAT method presented in FAISS library when we compared the trade-off between time and quality of these algorithms to build the approximate KNN graph.

The experiments related in Chapter 4 and 5 confirms that SWW-tSNE could obtain better or equivalents results compared with other dimensionality reduction techniques like t-SNE-CUDA, but with faster execution. These results were performed considering the dimensionality reduction to three and two dimensions.

This research also compares different approaches for dimensionality reduction applied to data visualization tasks considering different AI applications with large datasets. Acknowledging the difficulties of processing large datasets, we have focused in techniques proposed to achieve higher scalability using GPU hardware. We have measured the capacities of each technique to preserve local and global structures of high-dimensional points when these are projected in two dimensions. To take advantage of the computational time of SWW-tSNE and the strategy to preserve the global structures of AtSNE, we have proposed the SWW-AtSNE, which combines and preserve both of these characteristics.

In our experiments, the GPU implementation used for UMAP had the worst execution time between the compared techniques. Therefore, it requires further studies to be compared with other dimensionality reduction techniques for large datasets like the AtSNE, t-SNE-CUDA, SWW-tSNE, and SWW-AtSNE. We also cannot observe the superiority of UMAP to preserve local or global structures compared to other methods. However, it is important to mention that we have not explored approaches for tunning any parameters of the technique, which could be investigated in future works.

AtSNE and SWW-AtSNE mostly differ by the method used to compute the repulsive forces and the technical details of implementation since both algorithms derive from different t-SNE implementations. AtSNE could achieve a reasonable computational time for the smaller

datasets. However, SWW-AtSNE executed faster for the largest datasets and achieved better or similar preservation (local and global structures) for all datasets. This is evidence that the negative sampling approximation used by AtSNE is worse than the Barnes-Hut approximation. Another possible explanation is that the original AtSNE implementation depends on the right choice of parameters to perform dimensionality reduction without losing the information of local and global structures.

In our experiments, we compared two methods to compute the clustering to construct the anchors in SWW-AtSNE. It was evident that the usage of high-quality methods like K-Means could be a waste of computation since we could not perceive any evidence that relates the clustering quality with the dimensionality reduction quality. On the contrary, we have noted evidence that it is desirable to create clusters where distant points are clustered together and related to the same anchor. This possibility explains the result observed in our experiments where SWW-AtSNE with Random Sample Tree clustering achieves higher global structure preservation than the same technique with K-Means.

The AtSNE technique was proposed to improve the global quality of t-SNE. We cannot observe any evidence that this approach is better than simple practices like the initialization of the low-dimensional embedding with the PCA technique. However, the usage of anchors in the AtSNE and in our proposed SWW-AtSNE method slightly improves the preservation of the global structures compared to the standard SWW-tSNE approach.

Considering the two real-world applications analyzed in this research, we have presented possible uses of t-SNE to interpret large datasets and explain the importance of preserving local and global structures when dimensionality reduction is used to visualize high-dimensional points. In the application of Generative Adversarial Networks, we have extended the example of BigGAN, where two classes are interpolated with different images by analyzing if the representation of the discriminative network corresponds to the expected interpolation. Through dimensionality reduction, the interpolation was clear only in the embedding of SWW-tSNE with PCA initialization, which had the second-best global structures preservation and one of the best nearest neighborhood preservation for the BigGan dataset. Therefore, the users of dimensionality reduction techniques for visualization tasks should be aware of the pitfalls that these techniques can provide. When the information of the high-dimensional points is not preserved, it could lead to misinterpretations of the real data.

7.1 FUTURE WORK

It was clear in this research that the scalability of t-SNE could be improved with new approximated algorithms and with efficient strategies to implement of these algorithms in GPU. The Simulated Wide-Warp (SWW) was widely studied in this work. It will be interesting to further study the effectiveness of SWW applied to other steps of GPU implementations of t-SNE.

Additionally, the FAISS library used in the original t-SNE-CUDA allows the use of multiple GPUs to construct the approximate KNN graph. This feature is not trivial to add in the proposed RSFK algorithm used in SWW-tSNE and was not discussed in this research. Future works can approach this scenario by improving our methods and comparing it with the IVFFLAT and other resources present in FAISS library.

We have observed that t-SNE using PCA initialization achieves better preservation of global structures compared to AtSNE. However, we cannot see any reason why future studies do not try to elaborate methods to merge PCA initialization and the anchor strategy to achieve results with global preservation even higher than the observed in this research.

Different studies can elaborate methodologies to improve the UMAP implementation, which has good results considering local and global structures preservation. However, the available GPU implementation of UMAP is still inferior to other GPU implementations for dimensionality reduction like the SWW-tSNE.

Another possibility to expand the scope of this research is to investigate the usage of the methods cited in this research for data visualization in other artificial intelligence applications.

REFERENCES

Aumüller, M., Bernhardsson, E., and Faithfull, A. (2019). ANN-benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 10609 LNCS:34–49.

Barnes, J. and Hut, P. (1986). A hierarchical O(N log N) force-calculation algorithm. *Nature*, 324(6096):446–449.

Brock, A., Donahue, J., and Simonyan, K. (2018). Large scale gan training for high fidelity natural image synthesis. *arXiv preprint arXiv:1809.11096*.

Burtscher, M. and Pingali, K. (2011). An efficient cuda implementation of the tree-based barnes hut n-body algorithm. In *GPU computing Gems Emerald edition*, pages 75–92. Elsevier.

Chan, D. M., Rao, R., Huang, F., and Canny, J. F. (2018). t-SNE-CUDA: GPU-Accelerated t-SNE and its applications to modern data. *Proceedings - 2018 30th International Symposium on Computer Architecture and High Performance Computing, SBAC-PAD 2018*, pages 330–338.

Chan, D. M., Rao, R., Huang, F., and Canny, J. F. (2019). GPU accelerated t-distributed stochastic neighbor embedding. *Journal of Parallel and Distributed Computing*, 131:1–13.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). ImageNet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee.

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

Hinton, G. and Roweis, S. (2003). Stochastic neighbor embedding. *Advances in Neural Information Processing Systems*.

Hotelling, H. (1933). Analysis of a complex of statistical variables into principal components. *Journal of educational psychology*, 24(6):417.

Johnson, J., Douze, M., and Jégou, H. (2017). Billion-scale similarity search with GPUs. *arXiv preprint arXiv:1702.08734*.

Joulin, A., Grave, E., Bojanowski, P., and Mikolov, T. (2016). Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*.

Kobak, D. and Linderman, G. C. (2019). Umap does not preserve global structure any better than t-sne when using the same initialization. *bioRxiv*.

Krizhevsky, A., Hinton, G., et al. (2009). Learning multiple layers of features from tiny images. Technical report, Citeseer.

Kruskal, W. H. and Wallis, W. A. (1952). Use of ranks in one-criterion variance analysis. *Journal of the American statistical Association*, 47(260):583–621.

LeCun, Y. and Cortes, C. (2010). MNIST handwritten digit database.

Lee, J. A., Peluffo-Ordóñez, D. H., and Verleysen, M. (2015). Multi-scale similarities in stochastic neighbour embedding: Reducing dimensionality while preserving both local and global structure. *Neurocomputing*, 169:246–261.

Lee, J. A., Renard, E., Bernard, G., Dupont, P., and Verleysen, M. (2013). Type 1 and 2 mixtures of Kullback-Leibler divergences as cost functions in dimensionality reduction based on similarity preservation. *Neurocomputing*, 112:92–108.

Linderman, G. C., Rachh, M., Hoskins, J. G., Steinerberger, S., and Kluger, Y. (2017). Efficient algorithms for t-distributed stochastic neighborhood embedding. *ArXiv*, abs/1712.09005.

Linderman, G. C. and Steinerberger, S. (2019). Clustering with t-sne, provably. *SIAM Journal on Mathematics of Data Science*, 1(2):313–332.

McAuley, J., Targett, C., Shi, Q., and Van Den Hengel, A. (2015). Image-based recommendations on styles and substitutes. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 43–52. ACM.

McInnes, L., Healy, J., and Melville, J. (2020). Umap: Uniform manifold approximation and projection for dimension reduction.

Meyer, B. H., Pozo, A. T. R., and Zola, W. M. N. (2020). Improving barnes-hut t-sne scalability in gpu with efficient memory access strategies. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8.

Meyer, B. H., Pozo, A. T. R., and Zola, W. M. N. (in press 2021). Improving Barnes-Hut t-SNE algorithm in modern GPU architectures with Random Forest KNN and Simulated Wide-Warp. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533.

Nolet, C. J., Lafargue, V., Raff, E., Nanditale, T., Oates, T., Zedlewski, J., and Patterson, J. (2020). Bringing umap closer to the speed of light with gpu acceleration. *arXiv preprint arXiv:2008.00325*.

Pezzotti, N., Thijssen, J., Mordvintsev, A., Thomas, H., Lew, B. V., Lelieveldt, B. P. F., Eisemann, E., and Vilanova, A. (2019). GPGPU linear complexity t-SNE optimization. *IEEE Transactions on Visualization and Computer Graphics*, 2626(c).

Ruder, S. (2016). An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9.

Tang, J., Liu, J., Zhang, M., and Mei, Q. (2016). Visualizing large-scale and high-dimensional data. *25th International World Wide Web Conference, WWW 2016*, pages 287–297.

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

Van Der Maaten, L. and Hinton, G. (2008). Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605.

Yan, D., Wang, Y., Wang, J., Wang, H., and Li, Z. (2019). K-nearest Neighbors Search by Random Projection Forests. *IEEE Transactions on Big Data*, 7790.

Zola, W., Delgado, A., and Blanco, R. (2019). Caminhamento paralelo barnes-hut com vetorização avx2. In *Anais do XX Simpósio em Sistemas Computacionais de Alto Desempenho*, pages 454–461, Porto Alegre, RS, Brasil. SBC.

Zola, W. M. N., Bona, L. C. E., and Silva, F. (2014). Fast GPU parallel n-body tree traversal with Simulated Wide-Warp. *Proceedings of the International Conference on Parallel and Distributed Systems - ICPADS*, 2015-April:718–725.

APPENDIX A – "REVISÃO BIBLIOGRÁFICA"

A proposta foi embasada por quatro principais trabalhos apresentados na Tabela A.1. Dois desses trabalhos apresentam as técnicas SNE e t-SNE (Hinton and Roweis, 2003; Van Der Maaten and Hinton, 2008) enquanto os outros apresentam metodologias para reduzir o tempo de execução da técnica utilizando aproximações, estruturas de dados eficientes (Van Der Maaten, 2014) e paradigmas de programação paralela em GPU (Chan et al., 2018).

Neste Capítulo serão apresentados trabalhos relacionados ao tema desta proposta. Na Seção A.1 é apresentada a metodologia utilizada para encontrar trabalhos relacionados e posteriormente na Seção A.2 são discutidas as características observadas nos trabalhos encontrados.

A.1 METODOLOGIA DE BUSCA POR TRABALHOS RELACIONADOS

Para encontrar os trabalhos relacionados ao tema desta proposta, foi estabelecida uma metodologia ilustrada na Figura A.1 para auxiliar na busca de artigos científicos que abordam a técnica t-SNE e suas variações. Como primeira etapa, uma *String* de Busca foi gerada para encontrar trabalhos presentes nas bibliotecas digitais IEE, ScienceDirect e Scopus de forma que os resultados da pesquisa incluíssem os artigos de controle apresentados na Tabela A.1 e limitando os trabalhos encontrados para a área da computação.

A *String* de Busca encontrada é representada pela expressão "((t-sne) **OR** (t-distributed **AND** stochastic **AND** neighbor **AND** embedding) **OR** (stochastic **AND** neighbor **AND** embedding))", que apresenta as palavras e termos presentes nos resumos e palavras-chave dos trabalhos científicos presentes nas bibliotecas digitais. Posteriormente à definição da *String* de Busca, a procura pelos trabalhos relacionados nas três bibliotecas foi realizada e foram encontrados 693 trabalhos, dos quais 188 são duplicações e assim totalizando 505 trabalhos diferentes.

Em seguida foram eliminados os trabalhos cujos títulos ou palavras-chave indicavam apenas aplicações da técnica abordada, restando 99 documentos não classificados. Com isso, espera-se que as contribuições não significativas para o escopo do presente estudo fossem descartadas. De forma semelhante, os resumos dos artigos não classificados foram utilizados para identificar os trabalhos que contribuíram para a análise ou melhora de desempenho computacional do t-SNE (ou técnicas semelhantes), o que resultou em 28 documentos não classificados. Como última etapa, os artigos não acessíveis ou desconsiderados após a leitura completa foram descartados seguindo os critérios mencionados na Seção A.2, restando 22 trabalhos relacionados ao tema desta proposta. Após a finalização da metodologia descrita, um artigo adicional foi identificado e adicionado ao processo de revisão devido à sua relevância para o presente estudo, que não foi encontrado na execução da metodologia por ser um trabalho relativamente recente e ainda não estar indexado nas bibliotecas digitais utilizadas.

A.2 AVALIAÇÃO DE TRABALHOS

Após a leitura dos trabalhos relacionados foram identificados os tipos de otimizações da técnica estudada e principais contribuições de cada trabalho. Dessa forma foi possível identificar tendências e possíveis contribuições que podem ser feitas, além de identificar o estado da arte do t-SNE. A Tabela A.3 apresenta diferentes características presentes nos trabalhos analisados,

Título do trabalho	Descrição
Stochastic neighbor embedding (Hinton and Roweis, 2003)	Proposta inicial do SNE. Aplica a técnica na base de dados MNIST e discute sobre a convergência, vantagens e desvantagens do algoritmo.
Visualizing Data using t-SNE (Van Der Maaten and Hinton, 2008)	Proposta da técnica t-SNE. Expande os conceitos já apresentados no SNE e outras variações. Resolve principalmente o *crowding problem*, problema presente em várias outras técnicas além do SNE.
Accelerating t-SNE using tree-based algorithms (Van Der Maaten, 2014)	Primeira implementação do BH t-SNE. Aproximação que reduz o custo para $O(N \log N)$.
T-SNE-CUDA: GPU-Accelerated T-SNE and its Applications to Modern Data (Chan et al., 2018)	Primeiro trabalho a implementar e apresentar a implementação do t-SNE em GPU. Utiliza a biblioteca FAISS para calcular o KNN aproximado.

Table A.1: Artigos de controle utilizados na metodologia para buscar trabalhos relacionados

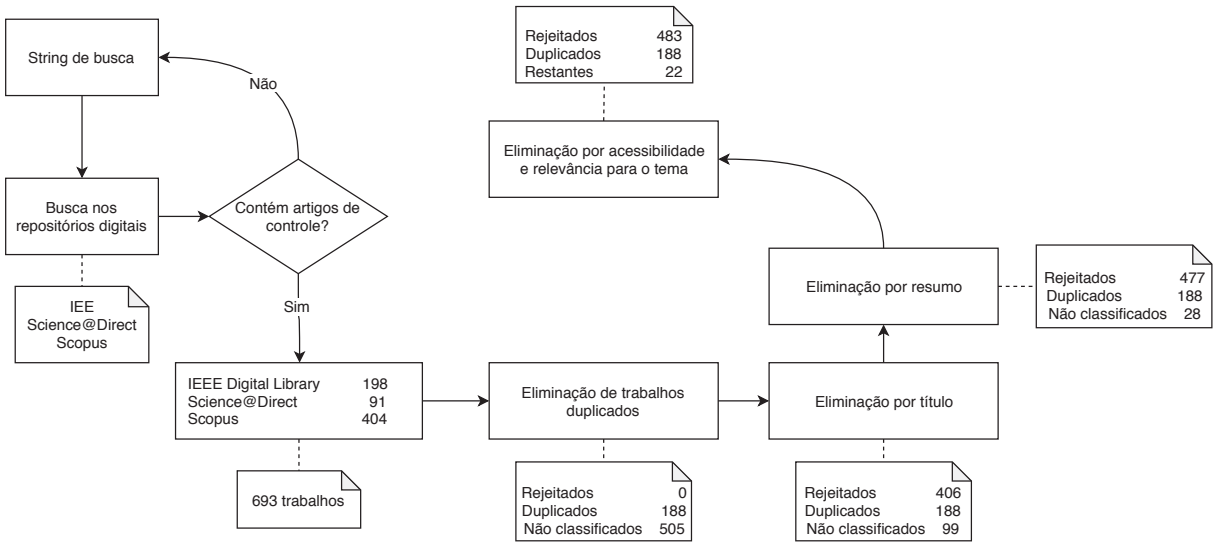


Figure A.1: Metodologia utilizada para buscar trabalhos relacionados.

descritas na Tabela A.2, agrupadas nas categorias “Avaliação de qualidade”, “Inicialização”, “Tipo de otimização” e “Reprodutibilidade”. Além disso, as aplicações de redução de dimensionalidade foram analisadas de forma que a escalabilidade fosse levada em conta, considerando as maiores bases de dados identificadas em cada trabalho (apresentado na Tabela A.4). Durante a revisão dos trabalhos selecionados, foi observado que alguns desses trabalhos não são completamente relacionados ao tema desta proposta, porém, será discutido brevemente as contribuições para a escalabilidade do t-SNE de cada abordagem identificada.

A.2.1 Otimizações

Após a proposta do SNE (Hinton and Roweis, 2003) e do t-SNE (Van Der Maaten and Hinton, 2008), outros trabalhos apresentaram soluções para problemas do algoritmo, como versões paramétricas da técnica, alterações no processo de convergência do algoritmo e aproximações para reduzir o custo computacional. Van Der Maaten (2009) apresentou uma versão paramétrica do t-SNE para transformar novos dados após a execução do algoritmo utilizando redes neurais profundas com o modelo de redes *autoencoders*. Gisbrecht et al. (2012) também utilizaram um

Table A.2: Categorias e descrições das abreviaturas utilizadas na Tabela A.3.

Categoria	Sigla	Descrição
Avaliação de qualidade: Utilizada para avaliar a qualidade das projeções (resultados) do uso da redução de dimensionalidade.	M1	Visualização qualitativa da projeção
	M2	*Precisão*
	M3	*Recall*
	M4	*Trustworthiness*
	M5	Convergência (KL-Divergence)
	M6	KNN
	M7	Tempo de execução do algoritmo
	M8	*K-ary neighbourhood preservation*
Inicialização: Técnica utilizada para inicializar a solução ou pré-processar os dados.	I1	**PCA**: O algoritmo PCA foi utilizado anteriormente à execução do algoritmo
	I2	**Aleatória**: O trabalho inicializa os pontos projetados com valores derivados de alguma distribuição aleatória
Tipo de otimização: Contribuição que o trabalho oferece para a otimização do t-SNE ou variantes.	O1	Alteração da função que altera o custo (cálculo do gradiente) da técnica
	O2	**CPU**: O trabalho discute características sobre a implementação do algoritmo considerando aspectos de CPU
	O3	**GPU**: O trabalho discute características sobre a implementação do algoritmo considerando aspectos de GPU
	O4	**Aproximação**: O trabalho apresenta técnicas que possibilitam aproximar o cálculo do gradiente
Reprodutibilidade: Características relacionadas à facilidade da reprodutibilidade do trabalho analisado.	R1	O código-fonte da implementação foi disponibilizado e é de fácil acesso
	R2	A principal base de dados utilizada no trabalho está disponível e é de fácil acesso
Outros	**OT**	

Table A.3: Características dos trabalhos selecionados. O símbolo ✓ representa a confirmação da característica em um trabalho, ✗ representa a não presença e valores em branco indicam características não identificadas.

	Avaliação de qualidade									Inicialização			Tipo de otimização				Reprodutibilidade	
	M1	M2	M3	M4	M5	M6	M7	M8	**OT**	I1	I2	**OT**	O1	O2	O3	O4	R1	R2
(Hinton and Roweis, 2003)	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✓	✓
(Nam et al., 2004)	✓	✗	✗	✗	✓	✗	✓	✗	✗	✗	✓	✗	✓	✗	✗	✗	✗	✓
(Van Der Maaten and Hinton, 2008)	✓	✗	✗	✗	✗	✓	✗	✗	✗	✓	✓	✗	✗	✗	✗	✗	✓	✓
(Van Der Maaten, 2009)	✓	✗	✗	✓	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓
(Yang et al., 2010)	✓	✗	✗	✗	✓	✗		✗	✗	✗	✓	✗	✓	✗	✗	✗	✗	✓
(Xie et al., 2011)	✓	✗	✗	✗	✗	✓	✗	✗	✓				✓	✗	✗	✗	✗	✗
(Vladymyrov e Carreira-Perpiñán, 2012)	✓	✗	✗	✗	✓	✗		✗	✗	✗	✓	✓	✓	✗	✗	✗	✗	✓
(Gisbrecht et al., 2012)	✓	✗	✗	✗	✗	✗	✗	✓	✗				✗	✗	✗	✗	✗	✓
(Kim et al., 2014)	✓	✓	✗	✗	✗	✓	✗	✗	✓	✗	✓	✗	✓	✗	✗	✗	✗	✓
(Vladymyrov e Carreira-Perpiñán, 2014)	✓	✗	✗	✗	✓	✗	✓	✗	✗				✓	✗	✗	✓	✗	✗
(Van Der Maaten, 2014)	✓	✗	✗	✗	✗	✓	✓	✗	✗	✓	✓	✗	✓	✗	✗	✓	✓	✓
(Parviainen, 2015)	✓	✗	✗	✗	✗	✗	✗	✓	✓				✗	✗	✗	✓	✗	✓
(Lee et al., 2015)	✓	✗	✗	✗	✗	✗	✓	✓	✓	✓	✗	✓	✓	✗	✗	✗	✗	✓
(Tang et al., 2016)	✓	✗	✗	✗	✗	✓	✓	✗	✗				✓	✗	✗	✓	✓	✓
(Pezzotti et al., 2016)	✓	✓	✓	✗	✗	✗	✓	✗	✗			✓	✗	✗	✗	✓	✗	
(Shen et al., 2017)	✓	✗	✗	✗	✗	✗	✗	✓	✗				✓	✗	✗	✗	✗	✗
(Pezzotti et al., 2017)	✓	✓	✗	✗	✗	✗	✓	✗	✓	✗	✓	✓	✗	✗	✗	✓	✓	✓
(Chan et al., 2018)	✓	✗	✗	✗	✗	✗	✓	✗	✗	✗	✓	✗	✗	✗	✓	✓	✓	✓
(Passalis e Tefas, 2018)	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✓	✓	✗	✓	✓
(Jo et al., 2018)	✓	✗	✗	✗	✓	✗	✓	✗	✗	✗	✓	✓	✗	✗	✗	✓	✓	✓
(Pezzotti et al., 2019)	✓	✓	✓	✗	✓	✗	✓	✗	✗	✗	✓	✗	✓	✗	✓	✓	✓	✓
(Chan et al., 2019)	✓	✗	✗	✗	✗	✗	✗	✓	✓	✗	✓	✗	✗	✗	✓	✓	✓	✓
(Fu et al., 2019)	✓	✗	✗	✗	✗	✓	✓	✗	✗	✗	✓	✓	✓	✗	✓	✓	✓	✓

modelo paramétrico utilizando poucos dados para treinar o modelo e aplica-lo a uma quantidade de dados relativamente maior de forma que a qualidade da projeção fosse significativamente preservada.

Nam et al. (2004) apresentaram uma alteração do SNE que adiciona "ruídos" durante as iterações do algoritmo, o que aparenta melhorar a convergência em diferentes cenários. Yang et al. (2010) alteraram o t-SNE de forma que a técnica chamada "atualizações multiplicativas" é utilizada para evitar a busca exaustiva dos parâmetros do gradiente descedente estocástico. Xie et al. (2011) criaram uma variação do t-SNE chamada *Multiview stochastic neighbor embedding* (m-SNE) de forma que a função a ser otimizada passa a ser convexa, o que simplifica a convergência do algoritmo, além de utilizar o método de Nesterov para acelerar o gradiente descendente estocástico. Shen et al. (2017) também utilizam o método de Nesterov para melhorar a convergência do algoritmo em uma variação do t-SNE chamada mm-tSNE.

Vladymyrov e Carreira-Perpiñán (2012) analisaram a relação do SNE, t-SNE e técnicas semelhantes com grafos laplacianos de métodos espectrais para redução de dimensionalidade. Kim et al. (2014) propuseram uma versão supervisionada do t-SNE que utiliza informações sobre os rótulos dos dados de entrada do algoritmo, superando diferentes técnicas em várias métricas de qualidade de projeção. Lee et al. (2015) apresentaram uma técnica que considera a relação de pontos distantes durante a projeção chamada *multi-scale-similarities*, na qual a média de diversas funções gaussianas é utilizada de forma que informações sobre a estrutura global dos pontos seja preservada e a convergência do algoritmo não leva a mínimos locais ruins, porém a complexidade computacional é maior que o t-SNE e consequentemente dificulta seu uso em bases de dados de larga escala.

As propostas originais do SNE e t-SNE possuem custo computacional assintótico $O(N^2)$, e pode ser simplificado com aproximações que preservam a qualidade da projeção. Van Der Maaten (2014) apresenta uma das principais contribuições para o t-SNE que consiste em abstrair o cálculo das forças de repulsão em um problema de simulação de corpos utilizando o algoritmo de Barnes-Hut, o que também é feito no trabalho de Vladymyrov e Carreira-Perpiñán (2014) de forma que outro algoritmo é utilizado para a simulação de corpos. Parviainen (2015) apresenta uma metodologia para resolver o problema de simulação de corpos, aplicado ao t-SNE, que não é baseada em árvores e utiliza grafos que conectam os corpos, porém não compara o tempo de execução apesar de relatar resultados significativos em relação à qualidade das projeções e apresenta um custo assintótico melhor que o BH t-SNE no pior caso. Pezzotti et al. (2019) apresentaram uma abordagem que não utiliza a simulação de corpos para minimização da função de custo do t-SNE utilizando campos que representam direções e magnitudes das forças de atração e repulsão que devem ser aplicadas em cada região do espaço dos pontos projetados. Além disso, o trabalho discute vários aspectos que permitem implementar o algoritmo de forma eficiente em GPU.

A.2.2 Avaliação de resultados

Dentre os trabalhos selecionados, os tipos de avaliação de qualidade mais comuns são a medida do tempo de execução e a visualização dos resultados projetados para duas dimensões. Devido à especificidade de alguns trabalhos, algumas avaliações particulares foram utilizadas.

Em relação à qualidade das estruturas globais das projeções, foi observado que apenas dois trabalhos discutiram aspectos sobre essa característica. O trabalho de Lee et al. (2015) foi o primeiro, dentre os analisados, que discute sobre a qualidade global, e utiliza a versão normalizada do *K-ary neighbourhood preservation* para avaliar os resultados da técnica proposta. Fu et al. (2019) discutem sobre a qualidade global dos resultados que sua técnica oferece, uma

vez que um "esqueleto", gerado pelo algoritmo de clusterização *K-Means*, é utilizado até o final da execução do algoritmo, porém utiliza apenas o KNN para avaliar os resultados.

É interessante perceber que em alguns dos trabalhos mais recentes (Chan et al., 2019; Fu et al., 2019), diversas características sobre o uso de recursos computacionais são discutidos. Chan et al. (2018) discutem sobre o problema da escalabilidade da etapa de cálculo das forças de atração, que depende do algoritmo KNN (em GPU) e apresentam uma alternativa para o problema que consiste em utilizar múltiplas GPU. Fu et al. (2019) apresentam o consumo de memória de diferentes variações e implementações eficientes do t-SNE, e discutem sobre a inviabilidade da execução de algumas dessas implementações, como o tsne-cuda, em alguns cenários.

A.2.3 Reprodutibilidade e Escalabilidade

É possível observar na Tabela A.4 um crescimento no tamanho da maior base de dados ao longo dos anos após a proposta inicial da técnica, característica que é possível devido à evolução e acessibilidade de computadores com arquiteturas eficientes e também pelo avanço de técnicas que implementam e aproximam o t-SNE. É importante mencionar que alguns dos trabalhos utilizaram várias bases de dados com variações de dimensões e número de pontos, e por simplificação a tabela apresenta a maior base de dados considerando o número de pontos como fator principal.

Para resolver o problema da alta dimensionalidade dos dados de entrada, Van Der Maaten and Hinton (2008), Van Der Maaten (2014) e Lee et al. (2015) utilizaram a técnica PCA para reduzir a dimensionalidade de bases de dados anteriormente ao uso do t-SNE. Outros trabalhos utilizaram versões limitadas ou aproximadas do KNN para calcular as similaridades dos pontos no início do algoritmo (Chan et al., 2018, 2019; Fu et al., 2019). Chan et al. (2018) e Chan et al. (2019) utilizaram a biblioteca FAISS para implementar o KNN aproximado em GPU e Fu et al. (2019) utilizam a técnica *Inverted Files* (IVF) para aproximar o KNN e reduzir o uso de memória além de aumentar a eficiência da paralelização em GPU. Tang et al. (2016) apresentaram o método LargeVis, que possibilita calcular um grafo de vizinhos mais próximos aproximados e usa-lo para reduzir a dimensionalidade dos dados de maneira similar ao t-SNE, de forma que o cálculo da similaridade dos pontos é computada de forma eficiente e precisa.

Van Der Maaten (2014) propõe o BH t-SNE, que foi a primeira implementação disponibilizada do t-SNE com complexidade $O(N \log N)$. Vladymyrov e Carreira-Perpiñán (2014) também propuseram uma técnica baseada em simulação de corpos cuja complexidade computacional é linear, porém não utilizam nenhuma métrica que indique a qualidade da projeção além da análise da convergência do algoritmo. Devido à disponibilização do código-fonte e fácil acessibilidade às bases de dados utilizadas por Van Der Maaten (2014), o BH t-SNE foi popularizado, o que facilitou sua aplicação e estudo em diversos outros trabalhos que tentam otimizar a técnica.

Passalis e Tefas (2018) apresentaram o *framework* chamado *PySEF* que permite executar o t-SNE de forma que o usuário especifique a função de similaridade, e implementaram a técnica utilizando a ferramenta *PyTorch* que é conhecida pela sua eficiência computacional e simples adaptação para utilizar CPU ou GPU. O trabalho de Chan et al. (2018) foi o primeiro a apresentar uma implementação (tsne-cuda) do BH t-SNE em GPU, o que foi continuado em seu próximo trabalho (Chan et al., 2019) que realiza otimizações e também implementa em GPU o FIt-SNE (FFT-accelerated *Interpolation-based t-SNE*), que resolve o problema de simulação de corpos utilizando interpolações transformadas rápidas de Fourier.

Pezzotti et al. (2019) sugeriram uma implementação alternativa do t-SNE em GPU que não depende da linguagem CUDA, e portanto não necessita de produtos da NVIDIA. Essa implementação demonstra ter um maior custo de tempo computacional e resultados de qualidade de projeção melhores comparado ao tsne-cuda .

Fu et al. (2019) apresentaram uma abordagem implementada para GPU que utiliza o algoritmo *K-Means*, de forma que uma ancoragem é feita para o "esqueleto" da estrutura global dos pontos da dimensão original e reduzida. Dessa forma, o algoritmo preserva a estrutura global e permite uma execução significativamente mais rápida devido à aproximação que pode ser feita quando o "esqueleto" é utilizado para "mover" vários pontos durante a convergência do algoritmo.

Table A.4: Tamanho da maior base de dados utilizada para aplicar o t-SNE dentre os trabalhos selecionados. Trabalhos que não permitiram a identificação (ou não eram aplicáveis) estão representados por "?".

Trabalho	Número de dimensões	Número de pontos
(Hinton and Roweis, 2003)	256	3000
(Nam et al., 2004)	560	530
(Van Der Maaten and Hinton, 2008)	784	6000
(Van Der Maaten, 2009)	8100	40121
(Yang et al., 2010)	?	?
(Xie et al., 2011)	8000	29780
(Vladymyrov e Carreira-Perpiñán, 2012)	784	20000
(Gisbrecht et al., 2012)	784	10000
(Kim et al., 2014)	?	?
(Vladymyrov e Carreira-Perpiñán, 2014)	784	1020000
(Van Der Maaten, 2014)	273	1105455
(Parviainen, 2015)	3072	5000
(Lee et al., 2015)	784	3000
(Tang et al., 2016)	100	3997963
(Pezzotti et al., 2016)	39	1000000
(Shen et al., 2017)	?	?
(Pezzotti et al., 2017)	39	1000000
(Chan et al., 2018)	4096	1200000
(Passalis e Tefas, 2018)	784	60000
(Jo et al., 2018)	784	60000
(Pezzotti et al., 2019)	300	3000000
(Chan et al., 2019)	50	10000000
(Fu et al., 2019)	96	19531329

A.2.4 Considerações finais

Este Capítulo abordou trabalhos relacionados ao tema de interesse desta proposta. Percebe-se um interesse comum dentre os trabalhos recentes para reduzir o tempo de execução do t-SNE e técnicas similares de forma que o algoritmo possa ser aplicado a bases de dados maiores.

Essas otimizações computacionais normalmente consistem em utilizar aproximações e paralelizações das técnicas. Chan et al. (2019) apresentaram, até o presente momento, a implementação mais rápida do BH t-SNE utilizando GPU.

Além disso, percebe-se uma inconsistência da padronização das métricas utilizadas para avaliar os resultados das projeções realizadas nos trabalhos relacionados. Em específico, poucos trabalhos discutem e apresentam metodologias para avaliar a estrutura global dos dados originais após a redução de dimensionalidade.

A.3 REFERÊNCIAS

Chan, D. M., Rao, R., Huang, F. e Canny, J. F. (2018). t-SNE-CUDA: GPU-Accelerated t-SNE and its applications to modern data. *Proceedings - 2018 30th International Symposium on Computer Architecture and High Performance Computing, SBAC-PAD 2018*, páginas 330–338.

Chan, D. M., Rao, R., Huang, F. e Canny, J. F. (2019). GPU accelerated t-distributed stochastic neighbor embedding. *Journal of Parallel and Distributed Computing*, 131:1–13.

Fu, C., Zhang, Y., Cai, D. e Ren, X. (2019). AtSNE: Efficient and robust visualization on GPU through hierarchical optimization. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 25:176–186.

Gisbrecht, A., Mokbel, B. e Hammer, B. (2012). Linear basis-function t-SNE for fast nonlinear dimensionality reduction. *Proceedings of the International Joint Conference on Neural Networks*, páginas 1–8.

Hinton, G. e Roweis, S. (2003). Stochastic neighbor embedding. *Advances in Neural Information Processing Systems*.

Jo, J., Seo, J. e Fekete, J. D. (2018). PANENE: A Progressive Algorithm for Indexing and Querying Approximate k-Nearest Neighbors. *IEEE Transactions on Visualization and Computer Graphics*, páginas 1–14.

Kim, H., Choo, J., Reddy, C. K. e Park, H. (2014). Doubly supervised embedding based on class labels and intrinsic clusters for high-dimensional data visualization. *Neurocomputing*, 150(PB):570–582.

Lee, J. A., Peluffo-Ordóñez, D. H. e Verleysen, M. (2015). Multi-scale similarities in stochastic neighbour embedding: Reducing dimensionality while preserving both local and global structure. *Neurocomputing*, 169:246–261.

Nam, K., Je, H. e Choi, S. (2004). Fast Stochastic Neighbor Embedding :. *IEEE International Joint Conference on Neural Networks*, 1:123–128.

Parviainen, E. (2015). A graph-based N-body approximation with application to stochastic neighbor embedding. *Neural Networks*, 75:1–11.

Passalis, N. e Tefas, A. (2018). PySEF: A python library for similarity-based dimensionality reduction. *Knowledge-Based Systems*, 152:186–187.

Pezzotti, N., Höllt, T., Lelieveldt, B., Eisemann, E. e Vilanova, A. (2016). Hierarchical Stochastic Neighbor Embedding. *Computer Graphics Forum*, 35(3):21–30.

Pezzotti, N., Lelieveldt, B. P., Van Der Maaten, L., Höllt, T., Eisemann, E. e Vilanova, A. (2017). Approximated and user steerable tSNE for progressive visual analytics. *IEEE Transactions on Visualization and Computer Graphics*, 23(7):1739–1752.

Pezzotti, N., Thijssen, J., Mordvintsev, A., Thomas, H., Lew, B. V., Lelieveldt, B. P. F., Eisemann, E. e Vilanova, A. (2019). GPGPU linear complexity t-SNE optimization. *IEEE Transactions on Visualization and Computer Graphics*, 2626(c).

Shen, X., Zhu, X., Jiang, X., He, T. e Hu, X. (2017). Visualization of disease relationships by multiple maps t-SNE regularization based on Nesterov accelerated gradient. *Proceedings - 2017 IEEE International Conference on Bioinformatics and Biomedicine, BIBM 2017*, 2017-Janua(2):604–607.

Tang, J., Liu, J., Zhang, M. e Mei, Q. (2016). Visualizing large-scale and high-dimensional data. *25th International World Wide Web Conference, WWW 2016*, páginas 287–297.

Van Der Maaten, L. (2009). Learning a parametric embedding by preserving local structure. *Journal of Machine Learning Research*, 5:384–391.

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

Van Der Maaten, L. e Hinton, G. (2008). Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605.

Vladymyrov, M. e Carreira-Perpiñán, M. (2014). Linear-time training of nonlinear low-dimensional embeddings. *Journal of Machine Learning Research*, 33:968–977.

Vladymyrov, M. e Carreira-Perpiñán, M. Á. (2012). Partial-Hessian strategies for fast learning of nonlinear embeddings. *Proceedings of the 29th International Conference on Machine Learning, ICML 2012*, 1:345–352.

Xie, B., Mu, Y., Tao, D. e Huang, K. (2011). M-SNE: Multiview stochastic neighbor embedding. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 41(4):1088–1096.

Yang, Z., Wang, C. e Oja, E. (2010). Multiplicative updates for t-SNE. *Proceedings of the 2010 IEEE International Workshop on Machine Learning for Signal Processing, MLSP 2010*, (Mlsp):19–23.