BH-tsNET, FIt-tsNET, L-tsNET: Fast tsNET Algorithms for Large Graph Drawing

Amyra Meidiana
The University of Sydney, Australia

Seok-Hee Hong
The University of Sydney, Australia

Kwan-Liu Ma
University of California at Davis, CA, USA

Abstract

The `tsNET` algorithm adapts the popular dimensional reduction method `t-SNE` for graph drawing to compute high-quality drawings, preserving the neighborhood and clustering structure. However, its $O(nm)$ runtime results in poor scalability for large graphs. In this poster, we present three fast algorithms for reducing the time complexity of `tsNET` to $O(n \log n)$ time and $O(n)$ time, by integrating new fast methods for computation of high-dimensional probabilities and entropy computation with fast `t-SNE` algorithms for computation of KL divergence gradient. Specifically, we present two $O(n \log n)$-time algorithms `BH-tsNET` and `FIt-tsNET`, incorporating partial BFS-based high-dimensional probability computation and a new quadtree-based entropy computation with fast `t-SNE` algorithms, and $O(n)$-time algorithm `L-tsNET`, introducing a new fast interpolation-based entropy computation. Extensive experiments using benchmark data sets confirm that `BH-tsNET`, `FIt-tsNET`, and `L-tsNET` outperform `tsNET`, achieving 93.5%, 96%, and 98.6% faster runtime, respectively, while computing similar quality drawings in terms of quality metrics (neighborhood preservation, stress, shape-based metrics, and edge crossing) and visual comparison.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases tsNET, t-SNE, Large Graph Drawing

Digital Object Identifier 10.4230/LIPIcs.GD.2025.54

Category Poster Abstract

1 Introduction

The `tsNET` [3] algorithm for graph drawing utilizes the popular dimension reduction method `t-SNE` *(t-Stochastic Neighborhood Embedding)* [11], which aims to preserve the *neighborhood* of data points in a low-dimensional projection.

`t-SNE` models the distances of data points in the high- and low-dimensional spaces as probability distributions. A low-dimensional projection is then computed by minimizing the *KL (Kullback-Leibler) divergence* [4] between the two probability distributions, typically using gradient descent, by moving the points in the projection according to the gradient of the KL divergence. The runtime of `t-SNE` is $O(n^2)$, due to the computation of pairwise distances between all pairs of data points.

`tsNET` uses the *graph theoretic distance* (i.e., shortest path) between vertices in a graph $G = (V, E)$ as the high-dimensional distance, which takes $O(nm)$ time, where $n = |V|$ and $m = |E|$. Specifically, `tsNET` adds the following two more terms to the cost function, on top of the KL divergence term used by `t-SNE`: an $O(n)$-time *compression* term to accelerate untangling the drawing, and an $O(n^2)$-time *entropy* term for reducing clutter.

`tsNET` computes high-quality graph drawings which preserve the clustering and neighborhood structure of vertices. However, it is not scalable to large graphs due to the $O(nm)$ runtime for all-pair shortest path computation and $O(n^2)$ runtime for the cost function.

33rd International Symposium on Graph Drawing and Network Visualization (GD 2025).
Editors: Vida Dujmović and Fabrizio Montecchiani; Article No. 54; pp. 54:1–54:5
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Recently, fast `t-SNE` algorithms have been introduced. For example, `BH-SNE` *(Barnes-Hut SNE)* [10] utilizes kNN (k-Nearest Neighbors) and quadtree to reduce the computation of high-dimensional probabilities and KL divergence gradient to $O(n \log n)$ time. More recently, `FIt-SNE` *(FFT-accelerated Interpolation-based t-SNE)* [5] utilizes $O(n \log n)$-time approximate kNN to compute the high-dimensional probabilities, and a FFT (Fast Fourier Transform) [8]-accelerated interpolation to further improve the runtime of the KL divergence gradient computation to $O(n)$ time.

However, algorithms reducing the overall runtime of `tsNET` have not been investigated yet. To reduce the runtime of `tsNET`, there are three $O(n^2)$-time components that need to be reduced: computation of all pair shortest paths, computation of KL divergence gradient, and entropy computation. While it was mentioned as a possible future work to use fast `t-SNE` algorithms to improve the runtime of the KL divergence term [3], methods to reduce the runtime of the *entropy* term were not considered yet.

In this poster, we present three new fast `tsNET` algorithms for drawing large graphs, integrating new fast methods for all-pair shortest path computation and entropy computation, with fast `t-SNE` algorithms. Specifically, our fast algorithms use a new $O(n)$-time partial BFS method for all-pair shortest path computation.

1. The first $O(n \log n)$-time `BH-tsNET` algorithm uses a new $O(n \log n)$-time quadtree-based entropy computation, integrated with $O(n \log n)$-time quadtree-based KL divergence computation of `BH-SNE`.
2. Faster $O(n \log n)$-time `FIt-tsNET` algorithm integrates the $O(n \log n)$-time quadtree-based entropy computation with $O(n)$-time interpolation-based KL divergence computation of `FIt-SNE`.
3. The fastest $O(n)$-time `L-tsNET` algorithm uses a new $O(n)$-time FFT-accelerated interpolation-based entropy computation, integrating $O(n)$-time interpolation-based KL divergence computation of `FIt-SNE`.

Table 1 summarizes the time complexity analysis of our fast algorithms. For the technical details, see the full version of this poster [7].

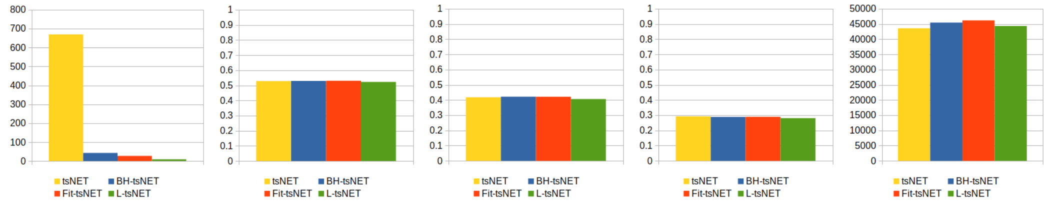
Table 1 Time complexity analysis of fast `tsNET` algorithms.

Algorithm	Shortest Path	KL Divergence	Entropy	Overall
`tsNET`	$O(nm)$	$O(n^2)$	$O(n^2)$	$O(nm)$
`BH-tsNET`	$\boldsymbol{O(n)}$	$O(n \log n)$	$\boldsymbol{O(n \log n)}$	$\boldsymbol{O(n \log n)}$
`FIt-tsNET`	$\boldsymbol{O(n)}$	$O(n)$	$\boldsymbol{O(n \log n)}$	$\boldsymbol{O(n \log n)}$
`L-tsNET`	$\boldsymbol{O(n)}$	$O(n)$	$\boldsymbol{O(n)}$	$\boldsymbol{O(n)}$

2 Comparison Experiments

We implement `BH-tsNET`, `FIt-tsNET`, and `L-tsNET` in Python using the `t-SNE` from the scikit-learn library [9], and in C++ based on `BH-SNE` [10] and `FIt-SNE` [5]. We use benchmark graphs used in the `tsNET` evaluation [3] as well as standard test suites commonly used in graph drawing studies: real-world scale-free graphs, biological networks, and mesh graphs. For the details, see the full version of this poster [7].

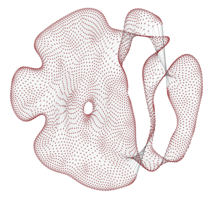
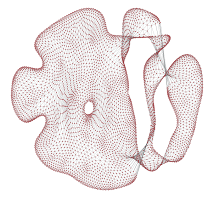
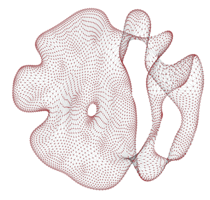
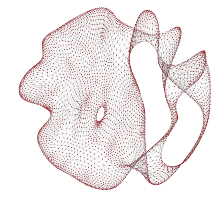
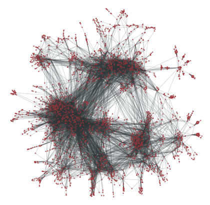
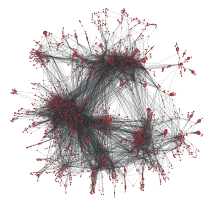
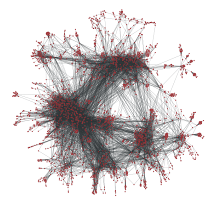
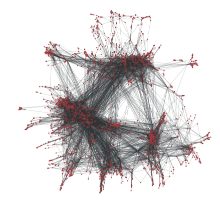
Extensive evaluation demonstrates that `BH-tsNET`, `FIt-tsNET`, and `L-tsNET` obtain significant runtime improvements over `tsNET`, on average 93.5%, 96%, and 98.6% respectively. See Figure 1(a).



(a) Runtime **(b)** Neigh. preserv. **(c)** Stress **(d)** Shape-based **(e)** Edge cross.

Figure 1 Average runtime and quality metrics: `BH-tsNET`, `FIt-tsNET`, and `L-tsNET` all run significantly faster than `tsNET`, with highly similar quality metrics.

Table 2 `BH-tsNET`, `FIt-tsNET` and `L-tsNET` compute the same quality drawings as `tsNET`.

tsNET	BH-tsNET	FIt-tsNET	L-tsNET
3elt			
			
oflights			
			
GION_7			
			

Surprisingly, our algorithms compute almost the same quality drawings as `tsNET`, in terms of quality metrics (neighborhood preservation [3], stress [1], edge crossings, shape-based metrics [2]), see Figures 1(b)-(e). Visual comparisons also confirm that our algorithms produce almost the same drawings as `tsNET`, see Table 2.

Moreover, we compare our algorithms to `DRGraph`, another linear-time `t-SNE`-based graph drawing algorithm with a different optimization function. Experiments show that our algorithms perform especially well over `DRGraph` in visualizing graphs with a regular structure, such as mesh graphs. For the details, see the full version of this poster [7].

Recently, we also presented fast GUMAP algorithms, designed explicitly for graph drawing based on UMAP, which reduce the time complexity from $O(nm)$ to $O(n^2 \log n)$ and $O(n)$, utilizing spectral sparsification and edge sampling. For the details, see [6].

References

1 Giuseppe Di Battista, Peter Eades, Roberto Tamassia, and Ioannis G Tollis. *Graph drawing: algorithms for the visualization of graphs.* Pearson, 1998.

2 Peter Eades, Seok-Hee Hong, An Nguyen, and Karsten Klein. Shape-based quality metrics for large graph visualization. *JGAA*, 21(1):29–53, 2017. doi:10.7155/jgaa.00405.

3 Johannes F Kruiger, Paulo E Rauber, Rafael Messias Martins, Andreas Kerren, Stephen Kobourov, and Alexandru C Telea. Graph layouts by t-SNE. In *CGF*, volume 36(3), pages 283–294, 2017. doi:10.1111/cgf.13187.

4 Solomon Kullback. *Information theory and statistics.* Dover Publications Inc., 1997.

5 George C Linderman, Manas Rachh, Jeremy G Hoskins, Stefan Steinerberger, and Yuval Kluger. Fast interpolation-based t-SNE for improved visualization of single-cell rna-seq data. *Nature methods*, 16(3):243–245, 2019.

6 Amyra Meidiana and Seok-Hee Hong. SS-GUMAP, SL-GUMAP, SSSL-GUMAP: fast UMAP algorithms for large graph drawing. *CoRR*, abs/2509.19703, 2025. doi:10.48550/ARXIV.2509.19703.

7 Amyra Meidiana, Seok-Hee Hong, and Kwan-Liu Ma. BH-tsNET, FIt-tsNET, L-tsNET: fast tsNET algorithms for large graph drawing. *CoRR*, abs/2509.19785, 2025. doi:10.48550/ARXIV.2509.19785.

8 Henri J Nussbaumer. The fast fourier transform. In *Fast Fourier Transform and Convolution Algorithms*, pages 80–111. Springer, 1981.

9 Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *JMLR*, 12:2825–2830, 2011. doi:10.5555/1953048.2078195.

10 Laurens Van Der Maaten. Accelerating t-SNE using tree-based algorithms. *JMLR*, 15(1):3221–3245, 2014. doi:10.5555/2627435.2697068.

11 Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *HMLR*, 9(11), 2008.

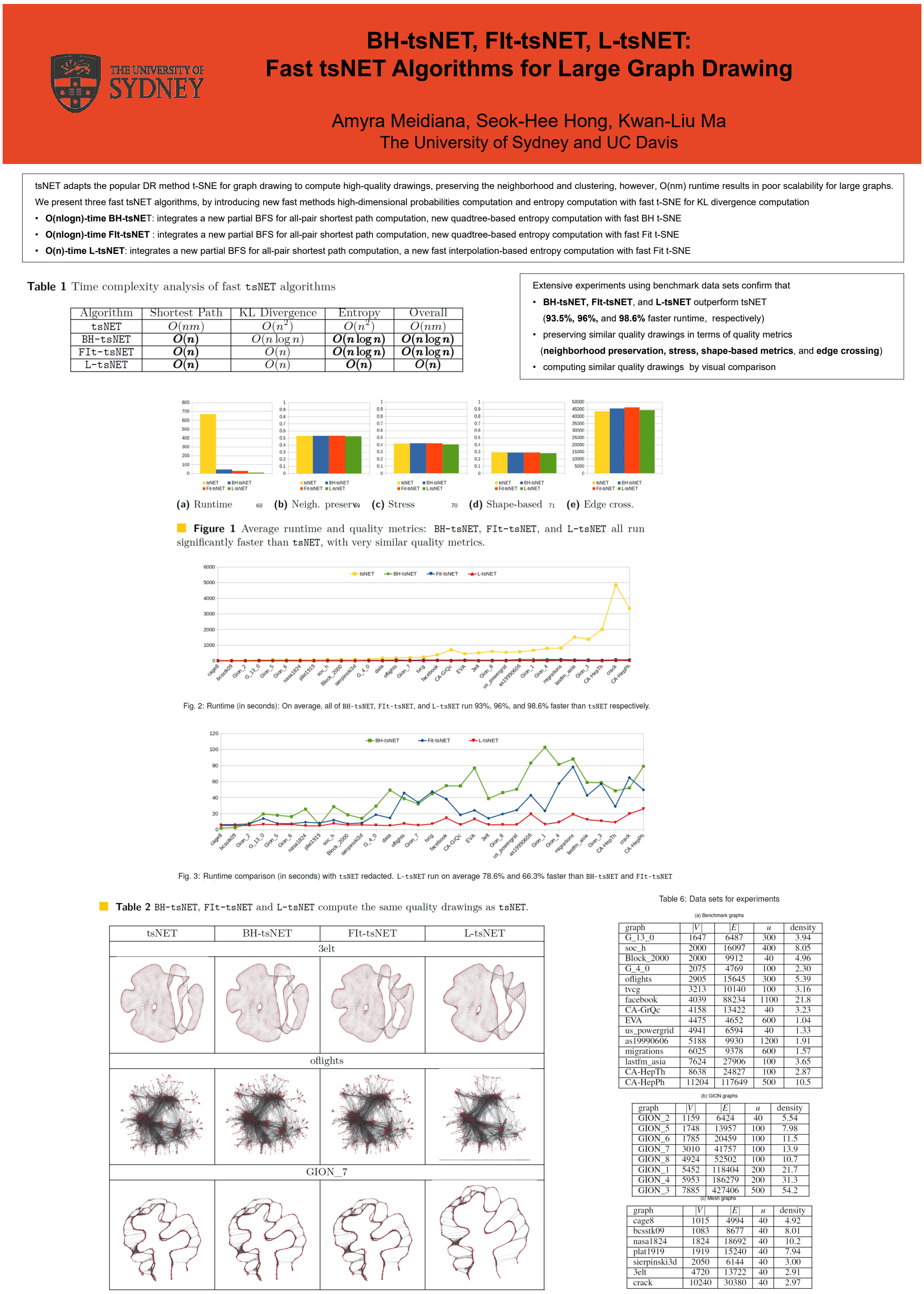


Figure 2 GD 2025 Poster.