Swift Neighbor Embedding of Sparse Stochastic Graphs

Nikos Pitsianis[†‡], Dimitris Floros[†], Alexandros-Stavros Iliopoulos[‡], and Xiaobai Sun[‡]

[†]Department of Electrical and Computer Engineering, Aristotle University of Thessaloniki, Thessaloniki, 54124, Greece
[‡]Department of Computer Science, Duke University, Durham, NC 27708, USA

July 23, 2019

Contents

Summary

SG-t-SNE-Π is a high-performance software for swift embedding of a large, sparse, stochastic graph/network into a d-dimensional space ($d = 1, 2, 3$) on a shared-memory computer, especially on personal laptop and desktop computers. Graphs/networks are an important type of relational data, arising ubiquitously in real-world applications and various research fields. Such data include biological networks, social networks, communication networks, food webs, word co-occurrence networks, etc; see [3, 8] for more real-world networks. Graph embedding maps each vertex of the graph to a d-dimensional feature vector. Graph embedding into a d-dimensional space with $d = 1, 2, 3$ is frequently used in data-based scientific studies for visual inspection of data, interpretation of network-based analysis results, interactive inquiries, and hypothesis generation.

The software SG-t-SNE-Π and its underlying algorithm are built upon precursor algorithms and software for stochastic neighbor embedding of high-dimensional data, namely the original Stochastic Neighbor Embedding (SNE) algorithm by Hinton and Roweis [2], the algorithm for t-distributed Stochastic Neighbor Embedding (t-SNE) by van der Maaten and Hinton [7], and their variants [4, 6].[1][2] The t-SNE algorithm has successfully assisted scientific discoveries, as reported in numerous articles in Nature and Science magazines. However, previous t-SNE algorithms and software are limited in two aspects: (i) The algorithms require that the data points be in a metric space and the associated graph (internally generated) be regular with a constant degree. In many real-world networks, the vertices do not readily reside in a metric space, and their degrees vary greatly, far from constant. (ii) The software is limited in practical use either to small graphs/networks or to embedding into $d < 3$ dimensions. We remove both limitations. SG-t-SNE-Π admits arbitrary large, sparse, stochastic graphs/networks. It is demonstrated in [5] for novel, autonomous embedding of large, real-world stochastic networks. SG-t-SNE-Π also enables fast three-dimensional (3D) graph embedding, which preserves and reveals more or even critical structural information as shown in [5], on modern laptop and desktop computers.

SG-t-SNE-Π is implemented in C++. It takes as input a stochastic graph and outputs d-dimensional coordinate vectors, one for each vertex. We provide two additional interfaces. The first is to support the conventional t-SNE, with its typical interface and wrappers [6], which converts data points in a metric space to a stochastic k-nearest neighbor graph. The second is a MATLAB interface. SG-t-SNE-Π is used to obtain all numerical experiments in the research article [5] and the accompanying supplementary material.[3]

[1] https://github.com/lvdmaaten/bhtsne

[2] https://github.com/KlugerLab/FIt-SNE

[3] http://t-sne-pi.cs.duke.edu

1 Overview

We introduce SG-t-SNE-Π, a high-performance software for swift embedding of a large, sparse, stochastic graph $\mathcal{G} = (V, E, \mathbf{P}_c)$ into a d-dimensional space ($d = 1, 2, 3$) on a shared-memory computer. The algorithm SG-t-SNE and the software t-SNE-Π were first described in Reference [5]. The algorithm is built upon precursors for embedding a k-nearest neighbor (kNN) graph, which is distance-based and regular with constant degree k. In practice, the precursor algorithms are also limited up to 2D embedding or suffer from overly long latency in 3D embedding. SG-t-SNE removes the algorithmic restrictions and enables d-dimensional embedding of arbitrary stochastic graphs, including, but not restricted to, kNN graphs. SG-t-SNE-Π expedites the computation with high-performance functions and materializes 3D embedding in shorter time than 2D embedding with any precursor algorithm on modern laptop/desktop computers.

1.1 Precursor algorithms

The original t-SNE [7] has given rise to several variants. Two of the variants, t-SNE-BH [6] and FIt-SNE [4], are distinctive and representative in their approximation approaches to reducing algorithmic complexity. They are, however, limited to kNN graph embedding. Specifically, at the user interface, a set of $n = |V|$ data points, $\mathcal{X} = \{\mathbf{x}_j\}_{j=1}^n$, is provided in terms of their feature vectors $\mathbf{x}_j$ in an L-dimensional vector space equipped with a metric/distance function. The input parameters include d for the embedding dimension, k for the number of near-neighbors, and u for the perplexity. A t-SNE algorithm maps the data points $\mathcal{X}$ to data points $\mathcal{Y} = \{\mathbf{y}_j\}_{j=1}^n$ in a d-dimensional space.

There are two basic algorithmic stages in a conventional t-SNE algorithm. In the preprocessing stage, the kNN graph is generated from the feature vectors $\{\mathbf{x}_j\}$ according to the metric function and input parameter k. Each data point is associated with a graph vertex. Next, the kNN graph is cast into a stochastic one, $\mathbf{P}_c = [p_{j|i}]$, and symmetrized to $\mathbf{P} = [p_{ij}]$,

$$p_{j|i}(\sigma_i) = \frac{a_{ij} \exp\left(-d_{ij}^2/2\sigma_i^2\right)}{\sum_l a_{il} \exp(-d_{il}^2/2\sigma_i^2)}, \quad \mathbf{P} = (\mathbf{P}_c + \mathbf{P}_c^{\mathrm{T}})/2. \tag{1}$$

where $\mathbf{A} = [a_{ij}]$ is the binary-valued adjacency matrix of the kNN graph, with zero diagonal elements (i.e., the graph has no self-loops), and d_{ij} is the distance between $\mathbf{x}_i$ and $\mathbf{x}_j$. The Gaussian parameters σ_i are determined by the point-wise equations related to the same perplexity value u,

$$-\sum_j a_{ij} p_{j|i}(\sigma_i) \log(p_{j|i}(\sigma_i)) = \log(u). \tag{2}$$

The next stage is to determine and locate the embedding coordinates $\mathcal{Y}^* = \{\mathbf{y}_j^*\}_{j=1}^n$ by minimizing the Kullback-Leibler divergence

$$\mathcal{Y}^* = \arg\min_{\mathcal{Y}} \mathrm{KL}(\mathbf{P} \| \mathbf{Q}(\mathcal{Y})), \tag{3}$$

where matrix $\mathbf{Q}(\mathcal{Y}) = [q_{ij}]$ is made of the ensemble $\mathcal{Y}$ regulated by the Student t-distribution,

$$q_{ij} = \frac{1}{Z} \frac{1}{1 + \|\mathbf{y}_i - \mathbf{y}_j\|^2}, \qquad Z = \sum_{k \neq \ell} \frac{1}{1 + \|\mathbf{y}_k - \mathbf{y}_\ell\|^2}. \tag{4}$$

In other words, the objective of (3) is to find the optimal stochastic matching between $\mathbf{P}$ and $\mathbf{Q}$ defined, respectively, over the feature vector set $\mathcal{X}$ and the embedding coordinate set $\mathcal{Y}$. The optimal matching is obtained numerically by applying the gradient descent method. A main difference among the precursor algorithms lies in how the gradient of the objective function is computed.

1.2 Approximation of the gradient

The computation per iteration step is dominated by the calculation of the gradient. Van der Maaten reformulated the gradient into two terms [6]:

$$\frac{\partial(\mathrm{KL})}{\partial \mathbf{y}_i} = \frac{4}{Z} \underbrace{\sum_{i \neq j} p_{ij} q_{ij} (\mathbf{y}_i - \mathbf{y}_j)}_{\text{attraction term}} - \frac{4}{Z} \underbrace{\sum_{i \neq j} q_{ij}^2 (\mathbf{y}_i - \mathbf{y}_j)}_{\text{repulsion term}}, \tag{5}$$

The attractive interaction term can be cast as the sum of $2d$ matrix-vector products with the sparse matrix $\mathbf{PQ} = [p_{ij} q_{ij}]$. The vectors are composed of the embedding coordinates, one in each dimension. The repulsive interaction term can be cast as the sum of $2d$ matrix-vector products with the dense matrix $\mathbf{QQ} = [q_{ij} q_{ij}]$. For clarity, we simply refer to the two terms as the $\mathbf{PQ}$ term and the $\mathbf{QQ}$ term, respectively.

The $\mathbf{QQ}$ (repulsion) term is in fact a broad-support, dense convolution with the Student t-distribution kernel on non-equispaced, scattered data points. As the matrix is dense, a naive method for calculating the term takes $O(n^2)$ arithmetic operations. The quadratic complexity limits the practical use of t-SNE to small graphs. Two types of existing approaches reduce the quadratic complexity to $O(n \log n)$, they are typified by t-SNE-BH and FIt-SNE. The algorithm t-SNE-BH, introduced by van der Maaten [6], is based on the Barnes-Hut algorithm. The broad-support convolution is factored into $O(\log n)$ convolutions of narrow support, at multiple spatial levels, each narrowly supported algorithm takes $O(n)$ operations. FIt-SNE, presented by Linderman et al. [4], may be viewed as based on non-uniform fast Fourier transforms. The execution time of each approximate algorithm becomes dominated by the $\mathbf{PQ}$ (attraction) term computation. The execution time also faces a steep rise from 2D to 3D embedding.

1.3 SG-t-SNE-Π

With the algorithm SG-t-SNE we extend the use of t-SNE to any sparse stochastic graph $\mathcal{G} = (V, E, \mathbf{P}_\mathrm{s})$. The key input is the stochastic matrix $\mathbf{P}_\mathrm{s} = [p_{j|i}]$, $\sum_j p_{j|i} = 1$, associated with the graph, where $p_{j|i}$ is not restricted to the form of (1). We introduce a parametrized, non-linear rescaling mechanism to explore the graph sparsity. We determine rescaling parameters γ_i by

$$\sum_j a_{ij}\, \phi\Big(p_{j|i}^{\gamma_i}\Big) = \lambda, \tag{6}$$

where $\lambda > 0$ is an input parameter and ϕ is a monotonically increasing function. We set $\phi(x) = x$ in the present version of SG-t-SNE-Π. Unlike (2), the rescaling mechanism (6)

imposes no constraint on the graph, its solution exists unconditionally. For the conventional t-SNE as a special case, we set $\lambda = 1$ by default. One may still make use of and exploit the benefit of rescaling ($\lambda \neq 1$).

With the implementation SG-t-SNE-Π, we accelerate the entire gradient calculation of SG-t-SNE and enable practical 3D embedding of large sparse graphs on modern desktop and laptop computers. We accelerate the computation of both **QQ** and **PQ** terms by utilizing the matrix structures and the memory architecture in tandem.

1.3.1 Accelerated accumulation of attractive interactions

The matrix **PQ** in the attractive interaction term of (5) has the same sparsity pattern as matrix **P**, regardless of iterative changes in **Q**. Sparsity patterns are generally irregular. Matrix-vector products with irregular sparse matrix invoke irregular memory accesses and incur non-equal, prolonged access latencies on hierarchical memories. We moderate memory accesses by permuting the rows and columns of matrix **P** such that rows and columns with similar nonzero patterns are placed closer together. The permuted matrix becomes block-sparse with denser blocks, resulting in better data locality in memory reads and writes.

The permuted matrix **P** is stored in the Compressed Sparse Blocks (CSB) storage format [1]. We utilize the CSB routines for accessing the matrix and calculating the matrix-vector products with the sparse matrix **PQ**. The elements of the **PQ** matrix are formed on the fly during the calculation of the attractive interaction term.

1.3.2 Accelerated accumulation of repulsive interactions

We factor the convolution in the repulsive interaction term of (5) into three consecutive convolutional operations. We introduce an internal equispaced grid within the spatial domain of the embedding points at each iteration, similar to the approach used in FIt-SNE [4]. The three convolutional operations are:

`S2G`: Local translation of the scattered (embedding) points to their neighboring grid points.[4]

`G2G`: Convolution across the grid with the same t-distribution kernel function, which is symmetric, of broad support, and aperiodic.

`G2S`: Local translation of the gridded data to the scattered points.

The `G2S` operation is a gridded interpolation and `S2G` is its transpose; the arithmetic complexity is $O(w^d n)$, where w is the interpolation window size per side. Convolution on the grid takes $O(n_g \log(n_g))$ arithmetic operations, where n_g is the number of grid points, i.e., the grid size. The grid size is determined by the range of the embedding points at each iteration, with respect to the error tolerance set by default or specified by the user. In the current implementation, the local interpolation method employed by SG-t-SNE-Π is accurate up to cubic polynomials in d separable variables ($d = 1, 2, 3$).

Although arithmetic complexity is substantially reduced in comparison to the quadratic complexity of the direct way, the factored operations suffer either from memory access latency

[4]The local interpolation employed by SG-t-SNE-Π is different from that used in FIt-SNE. The interpolation used in the latter is piecewise over non-overlapping grid cells.

or memory capacity issues, which were not recognized or resolved in existing t-SNE software. The scattered translation incurs high memory access latency. The aperiodic convolution on the grid suffers from excessive use of memory when the grid is periodically extended in all sides at once by zero padding. The exponential memory growth with d limits the embedding dimension or the graph size.

We resolve these memory latency and capacity issues in SG-t-SNE-Π. Prior to `S2G`, we relocate the scattered data points to the grid bins. This binning process has two immediate benefits. It improves data locality in the subsequent interpolation. It also establishes a data partition for parallel, multi-threaded execution of the scattered interpolation. We omit the parallelization details. For `G2G`, we implement aperiodic convolution by operator splitting, without using extra memory.

1.3.3 Rapid intra-term and inter-term data relocations

In sparse or structured matrix computation of $O(n \log(n))$ arithmetic complexity, the execution time is dominated by memory accesses. We have described in sections 1.3.1 and 1.3.2 how we use intra-term permutations to improve data locality and reduce memory access latency in computing the attraction and the repulsion terms of (5). In addition, we permute and relocate in memory the embedding data points between the two terms, at every iteration step. The inter-term data relocation is carried out at multiple layers, exploiting block-wise memory hierarchy. The data permutation overhead is well paid-off by the much shortened time for arithmetic calculation with the permuted data. We use Π in the software name SG-t-SNE-Π to signify the importance and the role of the permutations in accelerating t-SNE algorithms, including the conventional one, and enabling 3D embeddings.

1.4 Supplementary material

Supplementary material and performance plots are at `http://t-sne-pi.cs.duke.edu`.

2 References

[1] A. Buluç, J. T. Fineman, M. Frigo, J. R. Gilbert, and C. E. Leiserson. Parallel Sparse Matrix-Vector and Matrix-Transpose-Vector Multiplication Using Compressed Sparse Blocks. In *Proceedings of the 21st Annual Symposium on Parallelism in Algorithms and Architectures*, pages 233–244, 2009.

[2] G. E. Hinton and S. T. Roweis. Stochastic Neighbor Embedding. In *Advances in Neural Information Processing Systems*, volume 15, pages 857–864, 2003.

[3] I. A. Kovács, K. Luck, K. Spirohn, Y. Wang, C. Pollis, S. Schlabach, W. Bian, D.-K. Kim, N. Kishore, T. Hao, M. A. Calderwood, M. Vidal, and A.-L. Barabási. Network-Based Prediction of Protein Interactions. *Nature Communications*, 10(1):1240, 2019.

[4] G. C. Linderman, M. Rachh, J. G. Hoskins, S. Steinerberger, and Y. Kluger. Fast Interpolation-Based t-SNE for Improved Visualization of Single-Cell RNA-Seq Data. *Nature Methods*, 16(3):243–245, 2019.

[5] N. Pitsianis, A.-S. Iliopoulos, D. Floros, and X. Sun. Spaceland Embedding of Sparse Stochastic Graphs. In *IEEE High Performance Extreme Computing Conference*, 2019. (To appear).

[6] L. van der Maaten. Accelerating T-SNE Using Tree-Based Algorithms. *Journal of Machine Learning Research*, 15(Oct):3221–3245, 2014.

[7] L. van der Maaten and G. Hinton. Visualizing Data Using T-SNE. *Journal of Machine Learning Research*, 9(Nov):2579–2605, 2008.

[8] J. Yang and J. Leskovec. Defining and Evaluating Network Communities Based on Ground-Truth. *Knowledge and Information Systems*, 42(1):181–213, 2015.

3 Getting started

3.1 System environment

SG-t-SNE-Π is developed for shared-memory computers with multi-threading, running Linux or macOS operating system. The source code is (to be) compiled by a C++ compiler supporting Cilk. The current release is tested with the GNU `g++` compiler 7.4.0 and the Intel `icpc` compiler 19.0.4.233.

3.2 Prerequisites

SG-t-SNE-Π uses the following open-source libraries:

- `FFTW3` 3.3.8
- `METIS` 5.1.0
- `FLANN` 1.9.1
- `Intel TBB` 2019
- `Doxygen` 1.8.14

On Ubuntu:

```
sudo apt-get install libtbb-dev libflann-dev libmetis-dev libfftw3-dev doxygen
```

On macOS:

```
sudo port install flann tbb metis fftw-3 doxygen
```

3.3 Installation

3.3.1 Basic instructions

To generate the SG-t-SNE-Π library, test and demo programs:

```
./configure
make all
```

To specify the C++ compiler:

```
./configure CXX=<compiler-executable>
```

To test whether the installation is successful:

```
bin/test_modules
```

To generate the documentation:

```
make documentation
```

3.3.2 Support of the conventional t-SNE

SG-t-SNE-Π supports the conventional t-SNE algorithm, through a set of preprocessing functions. Issue

```
make tsnepi
```

to generate the `bin/tsnepi` binary, which is fully compatible with the existing wrappers provided by van der Maaten [6].

3.3.3 MATLAB interface

To compile the SG-t-SNE-Π MATLAB wrappers, use the `--enable-matlab` option in the `configure` command. The default MATLAB installation path is `/opt/local/matlab`; otherwise, set `MATLABROOT`:

```
./configure --enable-matlab MATLABROOT=<matlab-path>
```

3.4 Usage demo

We provide two data sets of modest size for demonstrating stochastic graph embedding with SG-t-SNE-Π:

```
tar -xvzf data/mobius-graph.tar.gz
bin/demo_stochastic_matrix mobius-graph.mtx
```

```
tar -xvzf data/pbmc-graph.tar.gz
bin/demo_stochastic_matrix pbmc-graph.mtx
```

The MNIST data set can be tested using existing wrappers provided by van der Maaten [6].

4 License and community guidelines

The SG-t-SNE-Π library is licensed under the GNU general public license v3.0. To contribute to SG-t-SNE-Π or report any problem, follow our contribution guidelines and code of conduct.

5 Contributors

Design and development:

Nikos Pitsianis[†‡], Dimitris Floros[†], Alexandros-Stavros Iliopoulos[‡], Xiaobai Sun[‡]

[†] Department of Electrical and Computer Engineering, Aristotle University of Thessaloniki, Thessaloniki 54124, Greece

[‡] Department of Computer Science, Duke University, Durham, NC 27708, USA