Bioinformatics, 37(19), 2021, 3377–3379
doi: 10.1093/bioinformatics/btab202
Advance Access Publication Date: 24 March 2021
Applications Note

OXFORD

Systems biology

PecanPy: a fast, efficient and parallelized Python implementation of *node2vec*

Renming Liu[1] and Arjun Krishnan [1,2,*]

[1]Department of Computational Mathematics, Science and Engineering, Michigan State University, East Lansing, MI 48824, USA and [2]Department of Biochemistry and Molecular Biology, Michigan State University, East Lansing, MI 48824, USA

*To whom correspondence should be addressed.

Associate Editor: Pier Luigi Martelli

Received on August 23, 2020; revised on March 7, 2021; editorial decision on March 15, 2021; accepted on March 23, 2021

Abstract

Summary: Learning low-dimensional representations (embeddings) of nodes in large graphs is key to applying machine learning on massive biological networks. *Node2vec* is the most widely used method for node embedding. However, its original Python and C++ implementations scale poorly with network density, failing for dense biological networks with hundreds of millions of edges. We have developed PecanPy, a new Python implementation of *node2vec* that uses cache-optimized compact graph data structures and precomputing/parallelization to result in fast, high-quality node embeddings for biological networks of all sizes and densities.

Availabilityand implementation: PecanPy software is freely available at https://github.com/krishnanlab/PecanPy.

Contact: arjun@msu.edu

Supplementary information: Supplementary data are available at *Bioinformatics* online.

1 Introduction

Large-scale molecular networks are powerful models that capture interactions between biomolecules (genes, proteins, metabolites) on a genome scale (McGillivray *et al.*, 2018) and provide a basis for predicting novel associations between individual genes/proteins and various cellular functions, phenotypic traits and complex diseases (Liu *et al.*, 2020; Sharan *et al.*, 2007). An area of research that has gained rapid adoption in network science across disciplines is learning low-dimensional numerical representations, or 'embeddings', of nodes in a network for easily leveraging machine-learning (ML) algorithms to analyze large networks (Cai *et al.*, 2018; Goyal and Ferrara, 2018; Hamilton *et al.*, 2018). Since each node's embedding vector concisely captures its network connectivity, node embeddings can be conveniently used as feature vectors in any ML algorithm to learn/predict node properties or links (Hamilton *et al.*, 2018). One of the earlier node embedding methods that continues to show good performance in various node classification tasks, especially on biological networks (Liu *et al.*, 2020; Nelson et al., 2019; Yue *et al.*, 2019), is a random-walk based approach called *node2vec* (Grover and Leskovec, 2016). Recent studies on the task of network-based gene classification have shown that *node2vec* achieves the best performance among the state-of-the-art embedding methods for gene classification (Yue *et al.*, 2019), and that using embedding generated from *node2vec* achieves prediction performance comparable to the state-of-the-art label propagation methods (Liu *et al.*, 2020).

However, despite its popularity, the original *node2vec* software implementations [written in Python (https://github.com/aditya-grover/node2vec), and in C++ (https://github.com/snap-stanford/snap/tree/master/examples/node2vec)] have significant bottlenecks in seamlessly using *node2vec* on all current biological networks. First, due to inefficient memory usage and data structure, they do not scale to large and dense networks produced by integrating several data sources on a genome-scale (17–26k nodes and 3–300mi edges) (Greene *et al.*, 2015; Szklarczyk *et al.*, 2015). Next, the pleasingly parallel precomputations of calculating transition probabilities and generating random walks are not parallelized in the original software. Resolving these issues will make it possible to embed large and dense biological networks, and even large biological knowledge graphs. Finally, the original implementations only support integer-type node identifiers (IDs), making it inconvenient to work with molecular networks typically available in databases where nodes may have non-integer IDs.

Recent work presented in preprints (Zhou *et al.*, 2018) and unpublished code repositories have proposed other implementations of *node2vec* (Supplementary Table S1). However, they either do not provide publicly available software or do not present a full analysis of their implementation, including a benchmark that ensures the quality of the resulting embeddings. Here, we present PecanPy, an efficient Python implementation of *node2vec* that is parallelized, memory efficient and accelerated using Numba with a cache-optimized data structure (Supplementary Fig. S1). We have extensively benchmarked our software using networks from the original study and multiple additional large biological networks to demonstrate both the computational performance and the quality of the node embeddings. In the rest of this paper, we first summarize the

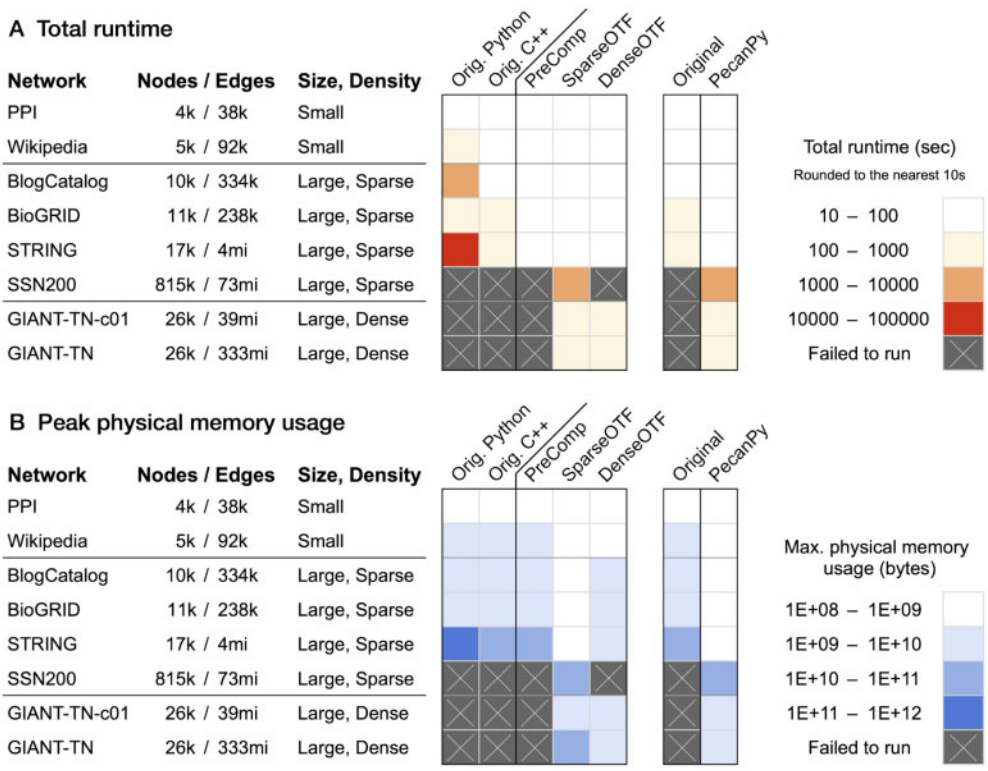

Fig. 1. Summary of runtime and memory of PecanPy and the original implementations of node2vec using multiple cores. The eight networks of varying sizes and densities are along the rows. The software implementations are along the columns. The first heatmap (on the left) shows the performance of the original Python and C++ software along with the three modes of PecanPy (PreComp, SparseOTF and DenseOTF). The adjacent 2-column heatmap (on the right) summarizes the performance of the original (best of Python and C++ versions) and PecanPy (best of PreComp, SparseOTF and DenseOTF) implementations. Lighter colors correspond to lower runtime in (**A**) and lower peak physical memory usage in (**B**). Crossed gray indicates that the particular implementation (column) failed to run for a particular network (row)

optimization and the performance of PecanPy and then go into the details of our implementation.

2 Implementation notes

The *node2vec* program consists of four stages: loading, preprocessing, walking and training (detailed description of the *node2vec* software is in Supplementary Notes). Comprehensive evaluation of the four stages (Supplementary Figs S2–S6) shows that training only takes 1.2% (median) of the total runtime for the original Python implementation, in contrast to 95.1% for the original C++ implementation. The raw training runtimes indicate that training the skip-gram using the gensim Python package is consistently an order of magnitude faster than the original C++ implementation. We chose to focus on Python because it is currently the most widely used high-level language in machine learning, making it convenient to use and to develop further as part of the community. Further, the Numba compiler can be used to achieve C++ level performance. We reimplemented *node2vec* in Python and optimized the first three inefficient stages of the algorithm by: (i) implementing computationally and memory-optimized graph data structures with efficient loading strategies; (ii) providing an option to bypass the need to store all transition probabilities, leading to a significant reduction in memory usage; and (iii) parallelizing the processes of transition probability computation and walk generation. We have also made string-type node IDs acceptable.

We present these improvements as PecanPy, a new software for parallelized, efficient and accelerated *node2vec* in Python (Supplementary Fig. S1). PecanPy operates in three different modes—*PreComp*, *SparseOTF* and *DenseOTF*—each optimized for networks of different sizes and densities (Supplementary Table S2). *PreComp* precomputes and stores all second order transition probabilities as in the original implementations, and hence is more suitable for small and sparse networks. On the other hand, *SparseOTF* and *DenseOTF* both compute second order transition probabilities On-The-Fly during walk generation without saving them. *SparseOTF* (like *PreComp*), with its use of a compact sparse row matrix representation, is better-suited for networks that are large and sparse. *DenseOTF*, which uses the full adjacency matrix as the underlying graph data structure, is well-suited for dense networks. The different modes and optimizations are detailed in the Supplementary Notes.

3 Benchmarks

We comprehensively benchmarked PecanPy and the original Python and C++ implementations of *node2vec* on a collection of eight networks including three networks from the original *node2vec* paper (Grover and Leskovec, 2016) and five large biological networks that together span a wide range of sizes (∼4k to 800k nodes and ∼38k—333mi edges) and densities (0.02–100%; Supplementary Table S3; Fig. 1, Supplementary Figs S7–S9; see Supplementary Notes on the choice of implementations) (Greene *et al.*, 2015; Law *et al.*, 2020; Stark *et al.*, 2006; Szklarczyk *et al.*, 2015). All software testing was done using 28 core Intel Xeon CPU E5-2680 v4 @2.4 GHz.

First, across the board, PecanPy (in one of its three modes) is substantially faster than the original implementations. In fact, there are three large networks (SSN200, GIANT-TN-c01, GIANT-TN) that run successfully only using PecanPy's *OTF* implementations. Other implementations failed to run the GIANT-TN network due to memory limitations that arise from storing second-order transition probabilities. The original software failed to run SSN200 because they do not support non-integer-type node IDs; this is supported in PecanPy. *DenseOTF* failed for SSN200 since its dense-network

design requires more than 5TB of memory to create a double precision dense matrix of size 800k. However, it considerably improves memory usage and speed for large dense networks like GIANT-TN, due to the more efficient network loading scheme, i.e. reading a numpy array file instead of a text (edge list) file (Supplementary Fig. S4). For relatively small and sparse networks (e.g. BioGRID, BlogCatalog), using PreComp invariably results in faster walk generation, thus achieving an overall shorter runtime (Supplementary Fig. S4). These results underscore the importance of the three modes of PecanPy. Similarly, in terms of memory usage, one of the modes of the PecanPy reduces the maximum resident size by up to two orders of magnitude compared to the original implementations. All these trends are magnified on a single core (Supplementary Figs S5, S7 and S9). Another implementation nodevectors that handled at least one dense network better than the original implementations still overall performed worse than PecanPy (Supplementary Fig. S10).

Finally, to ensure the quality of node embeddings generated by our new implementations, we evaluated their use as feature vectors in node classification tasks using datasets from the original paper (BlogCatalog, PPI, Wikipedia; see *Methods*). As shown in Supplementary Figure S11, our implementations achieve the same performance as the original Python implementation (complete Wilcoxon statistics and description can be found in Supplementary Table S5 and Supplementary Notes). On the other hand, the original C++ implementation is significantly worse than the original Python implementation for PPI (Wilcoxon P-value = 1.98e-3), while being significantly better for Wikipedia (Wilcoxon P-value = 2.41e-4). These differences are likely due to the different skip-gram implementations in the C++ and the Python implementations. The performance of nodevectors feature vectors is worse than those from all other implementations. The code for reproducing all the benchmarking results presented here are available at https://github.com/krishnanlab/PecanPy_benchmarks.

4 Conclusion

We have developed an efficient *node2vec* Python software—PecanPy—with significant improvement in both memory utilization and speed. Extensive benchmarking demonstrates that PecanPy efficiently generates quality node embeddings for networks at multiple scales including large (>800k nodes) and dense (fully connected network of 26k nodes) networks that the original implementations failed to execute. PecanPy is freely available at https://github.com/krishnanlab/PecanPy, can be easily installed via the pip package-management system (https://pypi.org/project/pecanpy/), and has been confirmed to work on a variety of networks, weighted or unweighted, with a wide range of sizes and densities. Therefore, it can find broad utility beyond biology.

Acknowledgements

The authors thank Christopher A. Mancuso, Anna Yannakopoulos and the rest of the Krishnan Lab for valuable discussions and feedback on the manuscript. They were grateful to Charles T. Hoyt for making the software pip installable and for an extensive code review.

Funding

This work was primarily supported by US National Institutes of Health (NIH) grants [R35 GM128765 to A.K.] and supported in part by MSU start-up funds to A.K.

Conflict of Interest: none declared.

References

Cai,H. *et al.* (2018) A comprehensive survey of graph embedding: problems, techniques and applications. *IEEE Trans. Knowl. Data Eng.*, **30**, 1616–1637.

Goyal,P. and Ferrara,E., (2018) Graph embedding techniques, applications, and performance: a survey. *Knowledge Based Syst.*, **151**, 78–94.

Greene,C.S. *et al.* (2015) Understanding multicellular function and disease with human tissue-specific networks. *Nat. Genet.*, **47**, 569–576.

Grover,A. and Leskovec,J. (2016) node2vec: scalable feature learning for networks. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pp. 855–864. Association for Computing Machinery, New York, NY, USA.

Hamilton,W.L. *et al.* (2018) Representation Learning on Graphs: Methods and Applications. *arXiv:1709.05584 [cs]*.

Law,J.N. *et al.* (2020) Accurate and efficient gene function prediction using a multi-bacterial network. *Bioinformatics*.

Liu,R. *et al.* (2020) Supervised-learning is an accurate method for network-based gene classification. *Bioinformatics*, **36**, 3457–3465.

McGillivray,P. *et al.* (2018) Network analysis as a grand unifier in biomedical data science. *Annu. Rev. Biomed. Data Sci.*, **1**, 153–180.

Nelson,W. *et al.* (2019) To embed or not: network embedding as a paradigm in computational biology. *Front. Genet.*, **10**, 381.

Sharan,R. *et al.* (2007) Network-based prediction of protein function. *Mol. Syst. Biol.*, **3**, 88.

Stark,C. *et al.* (2006) Biogrid: a general repository for interaction datasets. *Nucleic Acids Res.*, **34**, D535–D539.

Szklarczyk,D. *et al.* (2015) STRING v10: protein–protein interaction networks, integrated over the tree of life. *Nucleic Acids Res.*, **43**, D447–D452.

Yue,X. *et al.* (2019) Graph embedding on biomedical networks: methods, applications and evaluations. *Bioinformatics*, **36**, 1241–1251.

Zhou,D. *et al.* (2018) Efficient Graph Computation for Node2Vec. *arXiv: 1805.00280 [cs]*. arXiv: 1805.00280.