Go-Network: a graph sampling library written in Go

Jože M. Rožanec
Jožef Stefan Institute
Ljubljana, Slovenia
joze.rozanec@ijs.si

Matias Rožanec
Facultad de Ingeniería, Universidad de Buenos Aires
Buenos Aires, Argentina
mrozanec@fi.uba.ar

ABSTRACT

Go-Network is a Go language package for network generation and sampling. The core package provides basic data structures representing undirected graphs. Go-Network currently supports only integer values on graph nodes and edges. The library implements (a) data loading utilities supporting frequent graph formats, (b) algorithms for synthetic graph generation (e.g., Erdős-Rényi graphs), and thirty implementations of graph sampling algorithms. Among the many benefits the library inherits from Go (designed as a replacement for C++) are the compilation and execution speed (compiles directly to machine code) and its great support for concurrency while being memory savvy. These factors make the library a powerful tool for scientific purposes. We briefly describe the existing functionality, compare it against another graph sampling library (*Little Ball of Fur*), describe our design decisions, and draw attention to future work. Go-Network is publicly available and can be imported from https://github.com/graph-massivizer/go-network.

CCS CONCEPTS

• **Software and its engineering → Software libraries and repositories**; • **Computing methodologies → Artificial intelligence**.

KEYWORDS

Artificial Intelligence, Graph Sampling Algorithm, Scalable Graph Processing

ACM Reference Format:
Jože M. Rožanec and Matias Rožanec. 2024. Go-Network: a graph sampling library written in Go. In *Companion of the 15th ACM/SPEC International Conference on Performance Engineering (ICPE '24 Companion), May 7–11, 2024, London, United Kingdom.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3629527.3652903

1 INTRODUCTION

Our environment is full of systems for which it is challenging to derive collective behavior based on the knowledge of its components. Such systems are known under the term *complex systems*. They can be modeled as networks to capture the interactions between the system's components of relevance to analyze the actual behavior or make predictions. Networks may require modeling billions of nodes and their relationships [3]. Nevertheless, when graphs get very large, working with them becomes challenging. Among techniques that reduce their size while preserving properties of relevance, we find graph sampling and graph summarization [14, 24]. While graph sampling considers a subset of nodes and edges from the original graph, graph summarization reduces the graph to a smaller data structure that maintains the relevant properties. With the increasing relevance of graph neural networks, graph sampling has regained new relevance: sampling methods have become an indispensable strategy to speed up their training [6, 23, 34].

When applying graph sampling techniques, attention must be devoted to whether particular sampling techniques preserve relevant aspects and information of the graph relevant to downstream tasks. While some research was devoted to understanding how graph sampling techniques affect a specific graph, the variety of tasks and aspects to be considered make it an open and relevant research question [20, 23, 29, 39].

Motivation. Multiple libraries have been developed for network analysis and operation. NetworkX [13] has long been a reference library for graph processing, implementing a wide range of generators and graph processing algorithms. In the same line, the Stanford Network Analysis Project (SNAP) library [21] has been developed to provide efficient implementations for graph processing at scale and collecting relevant network datasets. Among distributed processing implementations, we find the GraphX module, which was implemented on the top of Apache Spark [11]. Nevertheless, in contrast to the abovementioned libraries, it provides a narrow selection of implementations of graph processing algorithms. More recently, standards have been developed for graph frameworks to ensure standard building blocks for expressing graph algorithms in the language of linear algebra (e.g., GraphBLAS [4]), and several implementations were developed for them (e.g., SuiteSparse [7] or GraphBLAST [37]). In addition, multiple libraries have been developed for deep learning on graphs (e.g., Deep Graph Library [36] and PyTorch Geometric [8]. Nevertheless, when it comes to graph sampling, while particular sampling algorithms have been released (e.g., GraphSAINT [40]), few libraries gather graph sampling algorithms in a single package. One such library is *Little Ball of Fur* [32], which provides a Python interface and implementations for over twenty sampling techniques. We propose Go-Network, a library for efficient network generation and sampling, to address this void in the Go language.

Contribution. We present Go-Network, an open-source graph generation and sampling library implemented in Go. We describe the library design and implementation, highlighting particular goals and choices.

Outline. The paper has four sections. Section 2 briefly describes our choice for the Go language. Section 3 reviews graph sampling methods. Section 4 describes the library implementation, design choices, and the implemented graph sampling algorithms. Finally, Section 5 concludes the paper and outlines future research.

2 GO LANGUAGE FOR MACHINE LEARNING DEVELOPMENT

The Go programming language was designed and introduced by Google in 2009 as a statically typed and compiled programming language, prioritizing simplicity, safety, and concurrency. While faster than, e.g., Python, it is currently an overlooked language for the development of machine learning libraries, mainly due to its lack of native support for CUDA and the limited amount of specialized libraries focused on statistics, calculus, and matrix manipulation - key to efficient machine learning implementations. Nevertheless, its simplicity and ease of implementing concurrency make it an attractive option for developing multi-threaded implementations requiring overly complex code, e.g., if developed in C++. We expect that, with time, the current shortcomings of the Go language ecosystem will be overcome, making it the go-to option for machine learning library development.

3 RELATED WORK

This section briefly describes graph sampling algorithms, focusing on the subset considered in two graph sampling libraries: *Little Ball of Fur* [32] and Go-Network.

3.1 Graph sampling

We consider graph sampling methods to be divided into categories based on two aspects: (a) the node/link selection criteria and (b) the operation applied to the graph upon node/link selection. The node/link selection criteria are usually divided into node-, link-, exploration-based, and hybrid methods. On the other hand, three operations can be applied to the graph: node/link preservation, contraction, or deletion. Following this taxonomy, we briefly summarize thirty graph sampling methods (see Table 1), and present them in detail in the following subsections.

Much effort has been invested into characterizing the graph sampling methods to understand what properties from the source graph are preserved in the graph sample [19, 20, 35, 38]. In particular, Krishnamurthy et al. [16] have shown that deletion or contraction methods allow resampling graphs to about 70% of their original size while keeping some original graph properties (e.g., the power-law distribution is respected). The author highlighted that among the benefits of resampling are simulation speed-ups: the authors estimated that reducing a graph to up to 70% of the original size can lead to simulation speed-ups of 11x or 37x for O(n^2) or O(n^3) simulations. For a detailed overview of sampling techniques and their properties, we defer the reader to the surveys by Hu et al. [14] Qi [28], and Liu et al. [23].

3.1.1 Node/link contraction. Sampling by node/link contraction involves iteratively merging edges in a large graph, reducing its size while preserving key properties. This process continues until a representative sample is obtained, enabling efficient analysis and exploration of the original graph's structure and characteristics. We depict the procedure in Fig. 1.

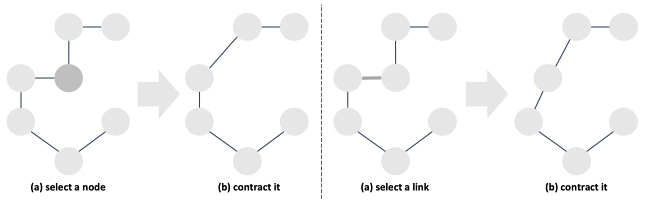

Figure 1: Graph sampling by contraction works by incrementally contracting (merging) nodes/links from a graph. The Figure depicts two types of contractions node-based (on the left) and link-based (on the right). The selected node/link is colored in dark gray.

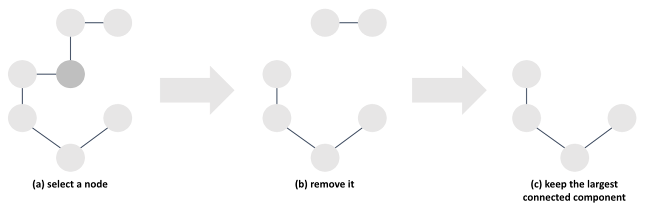

Figure 2: Graph sampling by deletion works by incrementally removing nodes/links from a graph. The Figure depicts a case of node-based deletion sampling. The selected node is colored in dark gray.

3.1.2 Node/link deletion. Sampling by node/link deletion is meant to gradually remove nodes/links from the graph until the desired graph size is achieved. This technique was introduced by Krishnamurthy et al. [16], who considered such sampling should follow three steps: (i) select nodes/links to be removed (only a small percentage (3%-5%) of nodes/links) and delete them, (ii) compute the connected components, preserving the largest and deleting the rest, and (iii) restart the procedure until achieving the desired graph size. We consider (ii) to be done to ensure the resulting graph mirrors a property observed in real-world graphs: that all of their elements are linked [3]. We depict the procedure in Fig. 2.

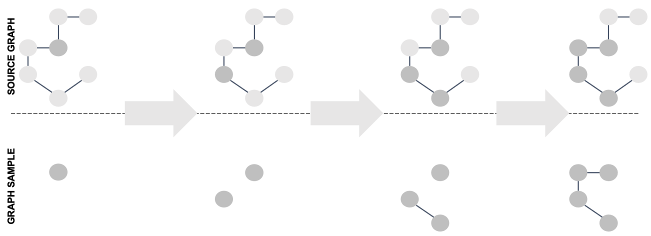

Figure 3: The Figure depicts a case of node-based preservation sampling. The selected nodes are colored in dark gray.

3.1.3 Node/link preservation. While the deletion methods select nodes and edges to delete them from the graph, preservation methods do the opposite: they retain them. Graph sampling by preservation works by incrementally adding nodes/links from the source graph to a new (initially empty) one. While the deletion-based methods follow a procedure that ensures the resulting graph has a single connected component, the preservation methods cannot provide such guarantees. We depict the procedure in Fig. 3.

Node/link	Selection criteria	Operation applied to node/link selection		
		Contraction	Deletion	Preservation
Node	**Breadth First Search**			
	Circulated Neighbors Random Walk			[41]
	Common Neighbor Aware Random Walk			[22]
	Community Structure Expansion			[25]
	Depth First Search			
	Diffusion			[33]
	Diffussion Tree			[33]
	Forest Fire			[20]
	Frontier			[30]
	Inclusive Random Neighbour			[26]
	Loop Erased Random Walk			[17]
	Metropolis Hastings Random Walk			[15]
	Non-Back Tracking Random Walk			[18]
	PageRank			[20]
	Random		[16]	[35]
	Random Degree		[16]	[1]
	Random Neighbour			[5]
	Random Walk			[20]
	Random Walk With Jump			[20]
	Random Walk With Restart			[20]
	Shortest Path			
	Snowball			[12]
	SpikyBall			[31]
Link	**Hybrid of RL and RNL**		[16]	
	Inclusive Random Node/Link			[26]
	Random Link	[16]	[16]	
	Random Link With Induction			[2]
	Random Link With Partial Induction			[2]
	Random Node/Link	[16]	[16]	
	Random Walk			[20]

Table 1: The table lists various graph sampling methods, referencing the scientific works in which they were introduced.

3.1.4 *Node/link selection strategies.*

Multiple strategies have been devised to perform node/link selection. In this section, we introduce some of them. In Table 1, we reference the scientific works in which they were introduced, considering their intersection with the operations applied upon node/link selection. Below, we briefly introduce each of them:

- **Breadth-First Search**: starting at a random node, it performs breadth-first search, including all of the nodes/links traversed until achieving the desired size;
- **Circulated Neighbors Random Walk**: simulates a random walker where the nodes of a neighborhood are randomly shuffled to ensure the walker can escape from closely knit communities;
- **Common Neighbor Aware Random Walk**: simulates a random walker that has a preference for neighbors with a lower number of common neighbors;
- **Community Structure Expansion**: given a random node from the graph, it chooses a node already connected to existing sampled nodes, always generating a connected graph;
- **Depth First Search**: starting at a random node, it performs depth-first search, including all of the nodes/links traversed until achieving the desired size;
- **Diffusion**: simulates a diffusion process, sampling nodes/links affected by the process;
- **Diffussion Tree**: is initiated by selecting a random node and expanding via random walks to neighboring nodes. It aims to efficiently capture local neighborhood structures while preserving connectivity, yielding a representative subgraph for analysis or further sampling;
- **Forest Fire**: simulates the fire spread through a forest, where each node represents a tree and edges represent potential paths of fire propagation. Starting from a randomly chosen node, it iteratively spreads fire to neighboring nodes based on a predefined probability parameter, typically resulting in a graph structure characterized by clusters and long-range connections resembling a forest fire propagation pattern;
- **Frontier**: iteratively expands a frontier set of nodes that consists of nodes adjacent to the current subgraph. This method efficiently explores the graph structure, allowing for the generation of representative subgraphs for various graph analysis tasks such as clustering, community detection, or pattern mining;
- **Inclusive Random Neighbour**: similar to random neighbor sampling, it includes the random node and the sampled neighbor into the sampled graph;
- **Loop Erased Random Walk**: is a stochastic algorithm used to generate random spanning trees on graphs. It operates by performing a random walk on the graph, erasing loops encountered during the walk, and ultimately constructing a spanning tree of the graph based on the path traversed without forming cycles;
- **Metropolis Hastings Random Walk**: is a Markov Chain Monte Carlo method for sampling graphs based on a target distribution. It iteratively explores the space of possible graphs by proposing changes to the current graph state and accepting or rejecting these changes based on a probability criterion derived from the Metropolis-Hastings algorithm, ensuring convergence to the desired distribution;
- **Non-Back Tracking Random Walk**: generates random walks on a graph where the walker does not backtrack to its previous node at the next step, ensuring a path without repetition. This technique is often employed in graph analysis and sampling to explore the structure of the graph efficiently while avoiding redundant traversal;
- **Non-Back Tracking Random Walk**: samples selecting nodes with a probability proportional to their PageRank scores, preserving the structural properties essential for PageRank calculations;
- **Random**: randomly selects a node/link from the graph;
- **Random Degree**: randomly selects a node from the graph with a probability proportional to their degree, ensuring that the resulting graph maintains similar connectivity patterns to the original one;
- **Random Node**: randomly selects a node from the graph and then switches it for a randomly sampled neighbor;
- **Random Walk**: simulates a random walk through its nodes and edges. It involves starting from a random node, then iteratively moving to neighboring nodes according to a stochastic process, resulting in a sequence of visited nodes that represent a sample from the graph's structure;
- **Random Walk With Jump**: similar to the Random Walk, but for each step, a decision is made with a probability of c=0.15 whether to continue the random walk or to jump to some random node within the graph;
- **Random Walk With Restart**: similar to the Random Walk, but for each step, a decision is made with a probability of c=0.15 whether to continue the random walk or to the starting node;
- **Random Walk With Restart**: starts by selecting two random nodes to compute the shortest path between them later;

- **Snowball**: iteratively samples nodes based on their connectivity to previously sampled nodes, expanding the sampling radius in a snowball-like manner. It starts with a small set of seed nodes and gradually adds nodes connected to the sampled nodes, typically used for capturing local neighborhoods in large graphs efficiently;
- **SpikyBall**: constructs graphs by placing nodes on the surface of a high-dimensional sphere and connecting them based on geometric rules, resulting in graphs with a spiky appearance.

4 LIBRARY DESIGN AND IMPLEMENTATION

4.1 Implemented methods

In Table 2, we provide a matrix describing the implemented graph sampling methods, grouping them based on whether the sampled element is a node or link, and two dimensions: (a) graph node/link selection criteria and (b) the operation applied on the graph upon the selected node/link.

Node/link	Selection criteria	Operation applied to node/link selection		
		Contraction	Deletion	Preservation
Node	Breadth First Search			BoF
	Circulated Neighbors Random Walk			BoF
	Common Neighbor Aware Random Walk			BoF
	Community Structure Expansion			BoF
	Depth First Search			BoF
	Diffusion			BoF
	Difussion Tree			BoF
	Forest Fire			BoF
	Frontier			BoF
	Inclusive Random Neighbour	GoN	GoN	GoN
	Loop Erased Random Walk			BoF
	Metropolis Hastings Random Walk			BoF
	Non-Back Tracking Random Walk			BoF
	PageRank			BoF
	Random	GoN	GoN	BoF, GoN
	Random Degree	GoN	GoN	BoF, GoN
	Random Neighbour	GoN	GoN	BoF, GoN
	Random Walk	GoN	GoN	BoF, GoN
	Random Walk With Jump	GoN	GoN	BoF, GoN
	Random Walk With Restart	GoN	GoN	BoF, GoN
	Shortest Path			BoF
	Snowball			BoF
	SpikyBall			BoF
Link	Hybrid of RL and RNL	GoN	GoN	BoF, GoN
	Inclusive Random Node/Link			
	Random Link	GoN	GoN	BoF, GoN
	Random Link With Induction			BoF
	Random Link With Partial Induction			BoF
	Random Node/Link	GoN	GoN	BoF, GoN
	Random Walk			

Table 2: The table lists a variety of graph sampling methods, indicating which ones are supported by Go-Network (GoN) and which ones by the *Little Ball of Fur* library.

4.2 Main modules

Overview. Go-Network is written in the Go language. The library declares a core graph model package and defines a graph interface. The current release only supports undirected graphs. The graph nodes and edges are modeled as integer values. Three kinds of utilities have been made available so far: (a) data loading and persistence, (b) graph generation, and (c) graph sampling.

Data loading and persistence. Various utilities are provided to load and persist graphs encoded in various formats. We find edge lists and adjacency lists among the supported formats.

Graph sampling. Thirty graph sampling algorithms were implemented. To ensure extensibility regarding graph sampling algorithm implementations, a Visitor pattern [10] was used. Doing so allows new methods to be seamlessly included without changing existing graph interfaces.

4.3 Design choices

To support a wide range of graph sampling algorithms while keeping a stable graph interface, we implemented a Visitor pattern [27] invoked on a graph. In particular, the sampling strategy is implemented as a Visitor and calls upon a graph, which provides itself to the Visitor to perform the sampling and then return the sampled graph. Given certain sampling algorithms follow the same structure (e.g., deletion sampling strategies perform incremental deletions and select the biggest connected component), such structure was implemented as a Template pattern [9], ensuring only the relevant sampling strategy is provided, reusing the rest of the code from the base struct. In Go-Network, graph sampling is not meant to be destructive. Therefore, a deep copy of the original graph is created before a sampling operation starts with deletion or contraction sampling strategies. Each time sampling is invoked on a graph, a new graph instance will be returned. This choice was made to ensure (i) immutability, given immutable data is implicitly concurrent-safe, and each concurrent process may operate on the same data without modifying it, and (ii) once a graph is loaded, multiple graph sampling algorithms can be applied simultaneously without altering it while obtaining the corresponding sampled graphs. This design decision may be reviewed in the future based on our benchmarking experience performed on massive graphs.

5 CONCLUSION AND FUTURE WORK

In this paper, we have introduced Go-Network and compared it against another graph sampling library: *Little Ball of Fur*. Go-Network does not cover the wide range of node/link selection strategies *Little Ball of Fur* yet provides. Nevertheless, Go-Network can already boast more sampling strategies, providing implementations that span graph sampling by contraction, deletion, and preservation, while *Little Ball of Fur* provides only implementations considering graph preservation. We consider supporting graph contraction and deletion as key to scalable graph sampling while ensuring the graph nodes remain connected [16]. Future work will focus on three areas: (a) enrich data loading and graph generation capabilities, (b) implement additional graph sampling techniques while adding support for multi-threaded and distributed execution, and (c) benchmark the graph sampling algorithms while comparing them against other implementations.

ACKNOWLEDGMENTS

Graph-Massivizer receives funding from the Horizon Europe research and innovation program of the European Union. Its grant management number is 101093202:https://graph-massivizer.eu/.

REFERENCES

[1] Lada A Adamic, Rajan M Lukose, Amit R Puniyani, and Bernardo A Huberman. 2001. Search in power-law networks. *Physical review E* 64, 4 (2001), 046135.

[2] Nesreen K Ahmed, Jennifer Neville, and Ramana Kompella. 2013. Network sampling: From static to streaming graphs. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 8, 2 (2013), 1–56.

[3] Albert-László Barabási and Márton Pósfai. 2016. *Network science.* Cambridge University Press, Cambridge. http://barabasi.com/networksciencebook/

[4] Benjamin Brock, Aydın Buluç, Timothy Mattson, Scott McMillan, and José Moreira. 2019. The graphblas c api specification. *GraphBLAS. org, Tech. Rep* (2019).

[5] Reuven Cohen, Shlomo Havlin, and Daniel Ben-Avraham. 2003. Efficient immunization strategies for computer networks and populations. *Physical review letters* 91, 24 (2003), 247901.

[6] Weilin Cong, Rana Forsati, Mahmut Kandemir, and Mehrdad Mahdavi. 2020. Minimal variance sampling with provable guarantees for fast training of graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining.* 1393–1403.

[7] Timothy A Davis. 2019. Algorithm 1000: SuiteSparse: GraphBLAS: Graph algorithms in the language of sparse linear algebra. *ACM Transactions on Mathematical Software (TOMS)* 45, 4 (2019), 1–25.

[8] Matthias Fey and Jan Eric Lenssen. 2019. Fast graph representation learning with PyTorch Geometric. *arXiv preprint arXiv:1903.02428* (2019).

[9] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. 1993. Design patterns: Abstraction and reuse of object-oriented design. In *ECOOP'93—Object-Oriented Programming: 7th European Conference Kaiserslautern, Germany, July 26–30, 1993 Proceedings 7.* Springer, 406–431.

[10] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. 1995. *Design patterns: elements of reusable object-oriented software.* Pearson Deutschland GmbH.

[11] Joseph E Gonzalez, Reynold S Xin, Ankur Dave, Daniel Crankshaw, Michael J Franklin, and Ion Stoica. 2014. {GraphX}: Graph processing in a distributed dataflow framework. In *11th USENIX symposium on operating systems design and implementation (OSDI 14).* 599–613.

[12] Leo A Goodman. 1961. Snowball sampling. *The annals of mathematical statistics* (1961), 148–170.

[13] Aric Hagberg, Pieter Swart, and Daniel S Chult. 2008. *Exploring network structure, dynamics, and function using NetworkX.* Technical Report. Los Alamos National Lab.(LANL), Los Alamos, NM (United States).

[14] Pili Hu and Wing Cheong Lau. 2013. A survey and taxonomy of graph sampling. *arXiv preprint arXiv:1308.5865* (2013).

[15] Christian Hübler, Hans-Peter Kriegel, Karsten Borgwardt, and Zoubin Ghahramani. 2008. Metropolis algorithms for representative subgraph sampling. In *2008 Eighth IEEE International Conference on Data Mining.* IEEE, 283–292.

[16] Vaishnavi Krishnamurthy, Michalis Faloutsos, Marek Chrobak, Li Lao, J-H Cui, and Allon G Percus. 2005. Reducing large internet topologies for faster simulations. In *International Conference on Research in Networking.* Springer, 328–341.

[17] Gregory F Lawler. 1999. Loop-erased random walk. *Perplexing Problems in Probability: Festschrift in Honor of Harry Kesten* (1999), 197–217.

[18] Chul-Ho Lee, Xin Xu, and Do Young Eun. 2012. Beyond random walk and metropolis-hastings samplers: why you should not backtrack for unbiased graph sampling. *ACM SIGMETRICS Performance evaluation review* 40, 1 (2012), 319–330.

[19] Sang Hoon Lee et al. 2006. Statistical properties of sampled networks. *Physical review E* 73, 1 (2006), 016102.

[20] Jure Leskovec and Christos Faloutsos. 2006. Sampling from large graphs. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining.* 631–636.

[21] Jure Leskovec and Rok Sosič. 2016. Snap: A general-purpose network analysis and graph-mining library. *ACM Transactions on Intelligent Systems and Technology (TIST)* 8, 1 (2016), 1–20.

[22] Yongkun Li, Zhiyong Wu, Shuai Lin, Hong Xie, Min Lv, Yinlong Xu, and John CS Lui. 2019. Walking with perception: Efficient random walk sampling via common neighbor awareness. In *2019 IEEE 35th International Conference on Data Engineering (ICDE).* IEEE, 962–973.

[23] Xin Liu, Mingyu Yan, Lei Deng, Guoqi Li, Xiaochun Ye, and Dongrui Fan. 2021. Sampling methods for efficient training of graph convolutional networks: A survey. *IEEE/CAA Journal of Automatica Sinica* 9, 2 (2021), 205–234.

[24] Yike Liu, Tara Safavi, Abhilash Dighe, and Danai Koutra. 2018. Graph summarization methods and applications: A survey. *ACM computing surveys (CSUR)* 51, 3 (2018), 1–34.

[25] Arun S Maiya and Tanya Y Berger-Wolf. 2010. Sampling community structure. In *Proceedings of the 19th international conference on World wide web.* 701–710.

[26] Yitzchak Novick and Amotz Bar-Noy. 2023. Inclusive random sampling in graphs and networks. *Applied Network Science* 8, 1 (2023), 56.

[27] Jens Palsberg and C Barry Jay. 1998. The essence of the visitor pattern. In *Proceedings. The Twenty-Second Annual International Computer Software and Applications Conference (Compsac'98)(Cat. No. 98CB 36241).* IEEE, 9–15.

[28] Xiao Qi. 2022. A Review: Random Walk in Graph Sampling. *arXiv preprint arXiv:2209.13103* (2022).

[29] Amir H Rasti, Mojtaba Torkjazi, Reza Rejaie, D Stutzbach, N Duffield, and W Willinger. 2008. Evaluating sampling techniques for large dynamic graphs. *Univ. Oregon, Tech. Rep. CIS-TR-08* 1 (2008).

[30] Bruno Ribeiro and Don Towsley. 2010. Estimating and sampling graphs with multidimensional random walks. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement.* 390–403.

[31] Benjamin Ricaud, Nicolas Aspert, and Volodymyr Miz. 2020. Spikyball sampling: Exploring large networks via an inhomogeneous filtered diffusion. *Algorithms* 13, 11 (2020), 275.

[32] Benedek Rozemberczki, Oliver Kiss, and Rik Sarkar. 2020. Little ball of fur: a python library for graph sampling. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management.* 3133–3140.

[33] Benedek Rozemberczki and Rik Sarkar. 2018. Fast sequence-based embedding with diffusion graphs. In *Complex Networks IX: Proceedings of the 9th Conference on Complex Networks CompleNet 2018 9.* Springer, 99–107.

[34] Marco Serafini and Hui Guan. 2021. Scalable graph neural network training: The case for sampling. *ACM SIGOPS Operating Systems Review* 55, 1 (2021), 68–76.

[35] Michael PH Stumpf, Carsten Wiuf, and Robert M May. 2005. Subnets of scale-free networks are not scale-free: sampling properties of networks. *Proceedings of the National Academy of Sciences* 102, 12 (2005), 4221–4224.

[36] Minjie Yu Wang. 2019. Deep graph library: Towards efficient and scalable deep learning on graphs. In *ICLR workshop on representation learning on graphs and manifolds.*

[37] Carl Yang, Aydın Buluç, and John D Owens. 2022. GraphBLAST: A high-performance linear algebra-based graph framework on the GPU. *ACM Transactions on Mathematical Software (TOMS)* 48, 1 (2022), 1–51.

[38] Sooyeon Yoon et al. 2007. Statistical properties of sampled networks by random walks. *Physical Review E* 75, 4 (2007), 046114.

[39] Muhammad Irfan Yousuf, Izza Anwer, and Raheel Anwar. 2023. Empirical characterization of graph sampling algorithms. *Social Network Analysis and Mining* 13, 1 (2023), 66.

[40] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. 2019. Graphsaint: Graph sampling based inductive learning method. *arXiv preprint arXiv:1907.04931* (2019).

[41] Zhuojie Zhou, Nan Zhang, and Gautam Das. 2015. Leveraging history for faster sampling of online social networks. *arXiv preprint arXiv:1505.00079* (2015).