Little Ball of Fur: A Python Library for Graph Sampling

Benedek Rozemberczki
The University of Edinburgh
Edinburgh, United Kingdom
benedek.rozemberczki@ed.ac.uk

Oliver Kiss
Central European University
Budapest, Hungary
kiss_oliver@phd.ceu.edu

Rik Sarkar
The University of Edinburgh
Edinburgh, United Kingdom
rsarkar@inf.ed.ac.uk

ABSTRACT

Sampling graphs is an important task in data mining. In this paper, we describe *Little Ball of Fur* a Python library that includes more than twenty graph sampling algorithms. Our goal is to make node, edge, and exploration-based network sampling techniques accessible to a large number of professionals, researchers, and students in a single streamlined framework. We created this framework with a focus on a coherent application public interface which has a convenient design, generic input data requirements, and reasonable baseline settings of algorithms. Here we overview these design foundations of the framework in detail with illustrative code snippets. We show the practical usability of the library by estimating various global statistics of social networks and web graphs. Experiments demonstrate that *Little Ball of Fur* can speed up node and whole graph embedding techniques considerably with mildly deteriorating the predictive value of distilled features.

KEYWORDS

network sampling, graph sampling, subsampling, network analytics, graph mining

ACM Reference Format:
Benedek Rozemberczki, Oliver Kiss, and Rik Sarkar. 2020. Little Ball of Fur: A Python Library for Graph Sampling. In *Proceedings of 16th International Workshop on Mining and Learning with Graphs (San Diego '20)*. ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Modern graph datasets such as social networks and web graphs are large and can be mined to extract detailed insights. However, the large size of the datasets pose fundamental computational challenges on graphs [9, 16]. Exploratory data analysis and computation of basic descriptive statistics can be time consuming on real world graphs. More advanced graph mining techniques such as node and edge classification or clustering can be completely intractable on full size datasets such as web graphs.

One of the fundamental techniques to deal with large datasets is sampling. On simple datasets such as point clouds, sampling preserves most of the distributional features of the data and forms the basis of machine learning [38]. However, graphs represent complex interrelations, so that naive sampling can destroy the salient features that constitute the value of the graph data. Graph sampling algorithms therefore need to be sensitive to the various features that are relevant to the downstream tasks. Such features include statistics such as diameter, clustering coefficient [6], transitivity or degree distribution. In more complex situations, graphs are used for community detection, classification, edge prediction etc [13]. A sampling algorithm should be representative with respect to such downstream learning tasks.

Various graph sampling procedures have been proposed with different objectives [14]. The implementation of the graph sampling technique and choice of its parameters used for the subgraph extraction can affect its utility for the task in question. A toolbox of well understood graph sampling techniques can make it easier for researchers and practitioners to easily perform graph sampling, and have consistent reproducible sampling across projects. Our goal is to make a large number of graph sampling techniques available to a large audience.

Present work. We release *Little Ball of Fur* – an open-source Python library for graph sampling. This is the first publicly available and documented Python graph sampling library. The general design of our framework is centered around an end-user friendly application public interface which allows for fast paced development and interfacing with other graph mining frameworks.

We achieve this by applying a few core software design principles consistently. Sampling techniques are implemented as individual classes and have pre-parametrized constructors with sensible default settings, which include the number of sampled vertices or edges, a random seed value and hyperparameters of the sampling method itself. Algorithmic details of the sampling procedures are implemented as private methods in order to shield the end-user from the inner mechanics of the algorithm. These concealed workings of samplers rely on the standard Python library and Numpy [44]. Practically, sampling techniques only provide a single shared public method (sample) which returns the sampled graph. Sampling procedures use NetworkX [12] graphs as the input and the output adheres to the same widely used generic format.

We demonstrate the practical applicability of our framework on various social networks and web graphs (e.g. Facebook, LastFM, Deezer, Wikipedia). We show that our package allows the precise estimation of macro level statistics such as average degree, transitivity, and degree correlation. We provide evidence that the use of sampling routines from *Little Ball of Fur* can reduce the runtime of node and whole graph embedding algorithms. Using these embeddings as input features for node and graph classification tasks we establish that the embeddings learned on the subsampled graphs extract high quality features.

Our contributions. Specifically, the main contributions of our work can be summarized as:

San Diego '20, 08-24, 2020, San Diego, CA

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

(1) We present *Little Ball of Fur* a Python graph sampling library which includes various node, edge and exploration based subgraph sampling techniques.
(2) We use code snippets to discuss the design principles which we applied when we developed our package. We examine the presence of standard hyperparameter settings, inner sampling mechanics which are implemented as private methods, the unified application public interface and the use of open-source scientific data structures.
(3) We demonstrate on various real world social network and webgraph datasets how sampling a subgraph with *Little Ball of Fur* affects the estimation accuracy of graph-level macro statistics. We present real world graph mining case studies where using our sampling framework speeds up graph embedding.
(4) We provide a detailed documentation of our sampling package with a tutorial and case studies with code snippets.

The rest of this paper has the following structure. In Section 2 we overview the relevant literature about graph sampling. This discussion covers node, edge, and exploration sampling techniques, and the possible applications of sampling from graphs. The design principles which we followed when *Little Ball of Fur* was developed are discussed in Section 3 with samples of illustrative Python code. The subsampling techniques provided by our framework are evaluated in Section 4. We present results about network statistic estimation performance, machine learning case studies about node and graph classification. The paper concludes with Section 5 where we discuss our main findings and point out directions for future works. We open sourced the software package and it can be downloaded from https://github.com/benedekrozemberczki/littleballoffur; the Python package can be installed via the *Python Package Index*. A comprehensive documentation can be accessed at https://little-ball-of-fur.readthedocs.io/en/latest/ with a step-by-step tutorial.

2 RELATED WORK

In this section we briefly overview the types of graph subsampling techniques included in *Little Ball of Fur* and the node and graph level representation learning algorithms used for the experimental evaluation of the framework.

2.1 Graph sampling techniques

Graph subsampling procedures have three main groups – node, edge, and exploration-based techniques. We give a brief overview of these techniques in this section.

2.1.1 Node sampling. Techniques which sample nodes select a set of representative vertices and extract the induced subgraph among the chosen vertices. Nodes can be sampled uniformly without replacement (RN) [39], proportional to the degree centrality of nodes (RDN) [1] or according to the pre-calculated PageRank score of the vertices (PRN) [19]. All of these methods assume that the set of vertices and edges in the graph is known ex-ante.

2.1.2 Edge sampling. The simplest link sampling algorithm retains a randomly selected subset of edges by sampling those uniformly without replacement (RE) while another approach is to randomly select nodes and an edge that belongs to the chosen node (RNE) [17]. These techniques can be hybridized by alternating between node-edge sampling and random edge selection with a parametrized probability (HRNE) [17].

By randomly selecting a set of retained edges one implicitly samples nodes. Because of this the random edge selection can be followed up by an induction step (TIES) [2] in which the additional edges between chosen nodes are all added. This step can be a partial induction (PIES) [2] if the edges were sampled in a streaming fashion and only edges with already sampled nodes are selected in the induction step.

2.1.3 Exploration based sampling. Node and edge sampling techniques do not extract representative subsamples of a graph by exploring the neighbourhoods of seed nodes. In comparison exploration based sampling techniques probe the neighborhood of a seed node or a set of seed vertices.

A group of exploration based sampling techniques uses search strategies on the graph to extract a subsample. The simplest search based strategies include classical traversal methods such as breadth first search (BFS) and depth first search (DFS) [5]. Snow ball sampling (SB) [10] is a restricted version of BFS where at maximum a fixed k number of neighbors is traversed. Forest fire (FF) sampling [20] is a parametrized stochastic version of SB sampling where the constraint on the maximum number of traversed neighbours only holds in expectation. A local greedy search based technique is community structure expansion sampling [25] (CSE) which starting with a seeding node adds new nodes to the sampled set which can reach the largest number of unknown nodes. Another simpler search based sampling technique is the random node-neighbor (RNN) [19] algorithm which randomly selects a set of seed nodes, takes the neighbors in a single hop and induces the edges of the resulting vertex set. Searching for shortest paths (SP) [31] between randomly sampled nodes can be used for selecting sequences of nodes and edges to induce a subsampled graph.

A large family of exploration based graph sampling strategies is based on random walks (RW) [8]. These techniques initiate a random walker on a seed node which traverses the graph and induces a subgraph which is used as the sample. Random walk based sampling has numerous shortcomings and a large number of sampling methods tries to correct for specific limitations.

One of the major limitations is that random walks are inherently biased towards visiting high degree nodes in the graph [14], Metropolis-Hastings random walk (MHRW) [15, 40] and its rejection constrained variant (RC-MHRW) [23] address this bias by making the walker prone to visit lower degree nodes.

Another major shortcoming of random walk based sampling is that the walker might get stuck in the closely knit community of the seed node. There are multiple ways to overcome this. The first one is the use of non-backtracking random walks (NBTRW) [18] which removes the tottering behaviour of random walks. The second one is circulating the neighbors of every node with a vertex specific queue (CNRW) [47]. A third strategy involves teleports - the random walker jumps (RWJ) [32] with a fixed probability to a random node from the current vertex. A fourth approach is to make the walker biased towards weak links by creating a common neighbor aware random walk sampler (CNARW) [24] which is biased towards neighbors with low neighborhood overlap. A fifth

strategy is using multiple random walkers simultaneously which form a so called frontier of random walkers (FRW) [32]. These techniques can be combined with each other in a modular way to overcome the limitations of random walk based sampling.

There are other possible modifications to traditional random walks which we implemented in *Little Ball of Fur*. One example is random walk with restart (RWR) [19], which is similar to RWJ sampling, but the teleport always ends with the seed node or loop erased random walks (LERW) [45] which can sample spanning trees from a source graph uniformly.

2.2 Node and whole graph embedding

Our experimental evaluation includes node and graph classification for which we use features extracted with neighbourhood preserving node embeddings and whole graph embedding techniques.

2.2.1 Neighbourhood preserving node embedding. Given a graph $G = (V, E)$ neighbourhood preserving node embedding techniques [3, 11, 27, 29, 30, 36, 41] learn a mapping $f : V \rightarrow \mathbb{R}^d$ which maps the nodes $v \in V$ into a d dimensional Euclidean space. In this embedding space a pre-determined notion of proximity of nodes is approximately preserved by learning the mapping. The vector representations created by the embedding procedure can be used as input features for node classifiers.

2.2.2 Whole graph embedding and statistical fingerprinting. Starting with a set of graphs $\mathcal{G} = (G_1, \ldots, G_n)$ whole graph embedding and statistical fingerprinting procedures [4, 26, 37, 42, 43] learn a mapping $h : \mathcal{G} \rightarrow \mathbb{R}^d$ which maps the graphs $G \in \mathcal{G}$ to a d dimensional Euclidean space. In this space those graphs which have similar structural patterns are close to each other. The vector representations distilled by these whole graph embedding techniques are useful inputs for graph classification algorithms.

3 DESIGN PRINCIPLES

We overview the core design principles that we applied when we designed *Little Ball of Fur*. Each design principle is discussed with illustrative examples of Python code which we explain in detail.

3.1 Encapsulated sampler hyperparameters, random seeding, and parameter inspection

Graph sampling methods in *Little Ball of Fur* are implemented as individual classes which all inherit from the *Sampler* class. A *Sampler* object is created by using the constructor which has default out-of-the-box hyperparameter settings. These default settings are available in the documentation and can be customized by re-parametrizing the *Sampler* constructor. The hyperparameters of the sampling techniques are stored as *public attributes* of the *Sampler* instance which allows for inspection of the hyperparameter settings by the user. Each graph sampling procedure has a seed parameter – this value is used to set a random seed for the standard Python and NumPy random number generators. This way the subsample extracted from a specific graph with a fixed seed is always going to be the same.

The code snippet in Figure 1 illustrates the encapsulated hyperparameter and inspection features of the framework. We start the script by importing a simple random walk sampler from the package (line 1). We initialize the first random walk sampler instance without changing the default hyperparameter settings (line 3). As the seed and hyperparameters are exposed we can print the seed parameter which is a public attribute of the sampler (line 4) and we can see the default value of the seed. We create a new instance with a parametrized constructor which sets a new seed (line 6) which modifies the value of the publicly available random seed (line 7).

```
1 from littleballoffur import RandomWalkSampler
2
3 sampler = RandomWalkSampler()
4 print(sampler.seed)
5
6 sampler = RandomWalkSampler(seed=41)
7 print(sampler.seed)
```

Figure 1: Re-parametrizing and initializing the constructor of a random walk sampler by changing the random seed.

3.2 Achieving API consistency and non proliferation of classes

The graph sampling procedures included in *Little Ball of Fur* are implemented with a consistent application public interface. As we discussed the parametrized constructor is used to create the sampler instance and the samplers all have a single available *public method*. The subsample of the graph is extract by the use of the *sample* method which takes the source graph and calls the private methods of the sampling algorithm.

We limited the number of classes and methods in *Little Ball of Fur* with a straightforward design strategy. First, the graph sampling procedures do not rely on custom data structures to represent the input and output graphs. Second, inheritance from the *Sampler* ensures that private methods which check the input format requirements do not have to be re-implemented on a case-by-case basis for each sampling procedure.

```
1 import networkx as nx
2 from littleballoffur import RandomWalkSampler
3
4 graph = nx.watts_strogatz_graph(1000, 10, 0)
5
6 sampler = RandomWalkSampler()
7 sampled_graph = sampler.sample(graph)
8
9 print(nx.transitivity(sampled_graph))
```

Figure 2: Using a random walk sampler on a Watts-Strogatz graph without changing the default sampler settings.

In Figure 2, first we import *NetworkX* and the random walk sampler from *Little Ball of Fur* (lines 1-2). Using these libraries we create a Watts-Strogatz graph (line 4) and a random walk sampler with the default hyperparameter settings of the sampling procedure (line 6). We sample a subgraph with the public *sample* method of the

random walk sampler (line 7) and print the transitivity calculated for the sampled subgraph (line 8).

```
1 import networkx as nx
2 from littleballoffur import ForestFireSampler
3
4 graph = nx.watts_strogatz_graph(1000, 10, 0)
5
6 sampler = ForestFireSampler()
7 sampled_graph = sampler.sample(graph)
8
9 print(nx.transitivity(sampled_graph))
```

Figure 3: Using a forest fire sampler on a Watts-Strogatz graph without changing the default sampler settings.

The piece of code presented in Figure 2 can be altered seamlessly to perform Forest Fire sampling by modifying the sampler import (line 2) and changing the constructor (line 7) – these modifications result in the example in Figure 3.

These illustrative sampling pipelines presented in Figures 2 and 3 demonstrate the advantage of maintaining API consistency for the samplers. Changing the graph sampling technique that we used only required minimal modifications to the code. First, we replaced the import of the sampling technique from the *Little Ball of Fur* framework. Second, we used the constructor of the newly chosen sampling technique to create a sampler instance. Finally, we were able to use the shared *sample* method and the same pipeline to calculate the transitivity of the sampled graph.

3.3 Standardized dataset ingestion and limitations

The shared public *sample* method of the node, edge and exploration based sampling algorithms takes a *NetworkX* graph as input and the returned subsample is also a *NetworkX* graph. The subsampling does not change the indexing of the vertices.

The rich ecosystem of graph subsampling methods and the consistent API required that *Little Ball of Fur* was designed with a limited scope and we made restrictive assumptions about the input data used for sampling. Specifically, we assume that vertices are indexed numerically, the first index is 0 and indexing is consecutive. We assume that the graph that is passed to the sample method is undirected and unweighted (edges have unit weights). In addition, we assume that the graph forms a single strongly connected component and orphaned nodes are not present. Heterogeneous, multiplex, multipartite and attributed graphs are not handled by the 1.0 release of the sampling framework.

The sampler classes all inherit private methods that check whether the input NetworkX graph violates the listed assumptions. These are called within the *sample* method before the sampling process itself starts. When any of the assumptions is violated an error message is displayed for the end-user about the wrong input and the sampling is halted.

4 EXPERIMENTAL EVALUATION

To evaluate the sampling algorithms implemented in *Little Ball of Fur* we perform a number of experiments on real world social networks and webgraphs. Details about these datasets are discussed in Subsection 4.1. We show how randomized spanning tree sampling can be used to speed up node embedding in Subsection 4.2 without reducing predictive performance. Our ablation study about graph classification in Subsection 4.3 demonstrates how connected graph subsampling can accelerate the application of whole graph embedding techniques. We present results about estimating graph level descriptive statistics in Subsection 4.4.

4.1 Datasets

We use real world social network and webgraph datasets to compare the performance of sampling procedures and test their utility for speeding up classification tasks.

4.1.1 Node level datasets. The datasets used for graph statistic estimation and node classification are all available on SNAP [21], and descriptive statistics can be found in Table 1.

- **Facebook Page-Page** [33] is a webgraph of verified official Facebook pages. Nodes are pages representing politicians, governmental organizations, television shows and companies while the edges are links between the pages. The related task is multinomial node classification for the 4 page categories.
- **Wikipedia Crocodiles** [33] is a webgraph of Wikipedia pages about crocodiles. Nodes are the pages and edges are mutual links between the pages. The potential task is binary node classification
- **LastFM Asia** [37] is a social network of LastFM (English music streaming service) users who are located in Asian countries. Nodes are users and links are mutual follower relationships. The task on this dataset is multinomial node classification – one has to predict the location of the users.
- **Deezer Hungary** [34] is a social network of Hungarian Deezer (French music streaming service) users. Nodes are users located in Hungary and edges are friendships. The relevant task is multi-label multinomial node classification - one has to list the music genres liked by the users.

Table 1: Statistics of social networks used for comparing sampling and node classification algorithms.

	Facebook Page-Page	Wikipedia Crocodiles	LastFM Asia	Deezer Hungary
Nodes	22,470	11,631	7,624	47,538
Density	0.0007	0.0025	0.0010	0.0002
Transitivity	0.2323	0.0261	0.1786	0.0929
Diameter	15	11	15	12
Labels	4	2	18	84

4.1.2 Graph level datasets. Our classification study on subsampled sets of graphs utilized forum threads and small sized social networks of developers. The respective descriptive statistics of these datasets are in Table 2.

- **Reddit Threads 10K** [35] is a random subsample of 10 thousand graphs from the original Reddit threads datasets.

Threads can be discussion and non-discussion based and the task is the binary classification of them according to these two categories.

- **GitHub StarGazers** [35] is a set of small sized social networks. Each social network is a community of developers who starred a specific machine learning or web development package on Github. The task is to predict the type of the repository based on the community graph.

Table 2: Descriptive statistics and size of the graph datasets for graph subsampling and whole graph classification.

		Nodes		Density		Diameter	
Dataset	Graphs	Min	Max	Min	Max	Min	Max
Reddit Threads 10K	10,000	11	97	0.021	0.291	2	22
GitHub StarGazers	12,725	10	957	0.003	0.561	2	18

4.2 Node classification with randomly sampled spanning tree embeddings of networks

Node embedding vectors [29, 30] are useful compact descriptors of vertex neighbourhoods when it comes to solving classification problems. In traditional classification scenarios the whole graph is used to learn the node embedding vectors. In this experiment we study a situation where the embedding vectors are learned from a randomly sampled spanning tree of the original graph. We compare the predictive value of node embeddings learned on the whole graph with ones learned from spanning trees extracted with randomized BFS [17], DFS [17] and LERW [45]. The main advantage of randomized spanning trees is that storing the whole graph requires $O(|E|)$ memory when we learn the node embedding. In contrast storing a sampled spanning tree only requires $O(|V|)$ space.

4.2.1 Experimental settings. The experimental pipeline which we used for node classification has four stages.

(1) *Graph sampling.* The BFS, DFS and LERW sample based embeddings start with the extraction of a random spanning tree with *Little Ball of Fur*. This sample is fed to the embedding procedure.
(2) *Upstream model.* The sampled graph is fed to the unsupervised upstream models DeepWalk [29] and Walklets [30] which learn the neighbourhood preserving node embedding vectors. We used the Karate Club [35] implementation of these models with the default hyperparameter settings.
(3) *Downstream model.* We fed the node embedding vectors as input features for a logistic regression classifier – we used the scikit-learn implementation [28] with the default hyperparameter settings. The downstream models were trained with a varying ratio of nodes.
(4) *Evaluation.* We report average AUC values on the test set calculated from a 100 seeded sampling, embedding and downstream model training runs.

4.2.2 Experimental results. We report the predictive performance for the Facebook Page-Page and LastFM Asia graphs respectively on Figures 4 and 5. First, we see that the features extracted from the BFS, DFS and LERW sampled spanning trees are less valuable

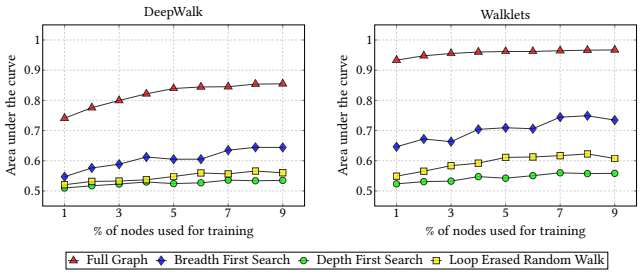


Figure 4: Node classification performance on the Facebook Page-Page graph [33] evaluated by average AUC scores on the test set calculated from a 100 seeded experimental runs.

for node classification based on the predictive performance on these two social networks. In plain words node embeddings of randomly sampled spanning trees produce inferior features. Second, the marginal gains of additional training data are smaller when the embedding is learned from a subsampled graph. Third, DFS sampled node embedding features have a considerably lower quality compared to the BFS and LERW sampled node embedding features. Finally, the Walklets based higher order proximity preserving embeddings have a superior predictive performance compared to the DeepWalk based ones even when the graph being embedded is a randomly sampled spanning tree of the source graph.

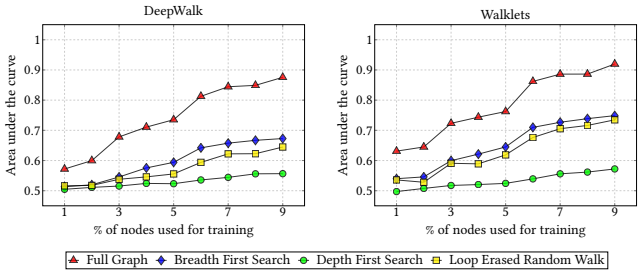


Figure 5: Node classification performance on the LastFM Asia graph [37] evaluated by average AUC scores on the test set calculated from a 100 seeded experimental runs.

4.3 An ablation study of graph classification

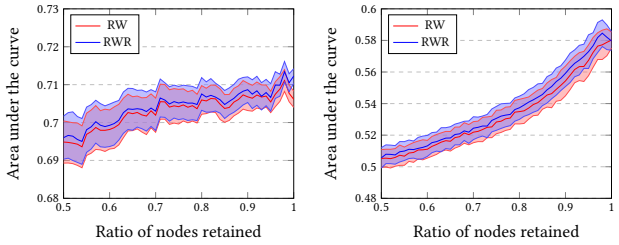
Graph classification procedures use the whole graph embedding vectors as input to discriminate between graphs based on structural patterns. Using subsamples of the graphs and extracting structural patterns from those can speed up this classification process. We will investigate how exploration based sampling techniques perform when they are used to obtain the samples used for the embedding.

4.3.1 Experimental settings. The data processing which we used for the evaluation of graph classification performance has four stages.

(1) *Graph sampling.* We sample both datasets using the RW [8] and RWR [19] methods implemented in *Little Ball of Fur* 100 times for each retainment rate with different random seeds.

Using these algorithms ensures that the graphs' connectivity patterns are unchanged.

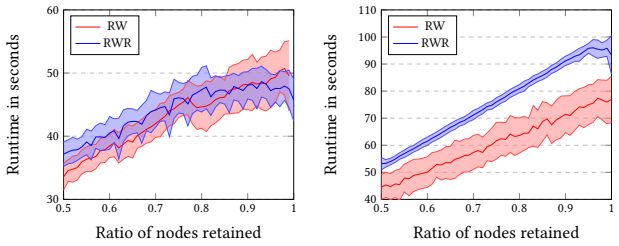
(2) *Upstream model.* Following the sampling, all of the samples are embedded using the Graph2Vec [26] algorithm. This procedure uses the presence of subtrees as structural patterns.
(3) *Downstream model.* With the embedding vectors as covariates, we estimate a logistic regression for each dataset and retainment rate. We rely on the scikit-learn implementation [28] of the classifier with the default hyperparameter settings. We use 80% of the graphs to train the classifier.
(4) *Evaluation.* The classification performance is evaluated using the AUC based on the remaining 20% of graphs which form the test set.



(a) Reddit Threads **(b) Github Stargazers**

Figure 6: Graph classification performance on the Reddit Threads and GitHub Stargazers graph datasets [35] evaluated by average AUC scores on the test set calculated from 100 seeded experimental runs. We also report standard deviations around the mean performance.

4.3.2 Experimental results. We report mean AUC values along with a standard deviation band in Figure 6 for the Reddit Threads and Github Stargazers datasets with the Random Walk and Random Walk with Restart sampling methods. Lower retainment rate is associated with a lower classification performance, as it can be expected. The more ragged, step function-like pattern observed in case of the Reddit threads dataset is likely to be due to the interplay of structural pattern downsampling and the generally smaller graphs in the dataset.



(a) Reddit Threads **(b) Github Stargazers**

Figure 7: Graph embedding runtime on the Reddit Threads and GitHub Stargazers graph datasets [35] calculated from 100 experimental runs. We also report standard deviations around the mean performance.

We report the mean runtime of the graph embedding process with a band of standard deviations in Figure 7. As we decrease the retainment rate, a significant decrease in runtime is prevalent. There is a clear trade-off between runtime and predictive performance. It is, however, worth noting that while the runtime associated with the 50% retainment rate is, in most cases close to half of the runtime using the whole graphs, the loss in classification power in case of the Reddit Threads dataset is less significant.

4.4 Estimating descriptive statistics

A traditional task for the evaluation of graph sampling algorithms is the estimation of graph level descriptive statistics. The graph level descriptive statistic is calculated for the sampled graph and it is compared to the ground truth value which is calculated based on the whole set of nodes and edges. A well performing sampling procedure is ought to give a precise estimate for the graph level quantity of interest with a reasonably small subsample of the graph.

4.4.1 Experimental settings. The pipeline used for estimating the graph level descriptive statistics had two stages.

(1) *Graph sampling.* Node and exploration based sampling procedures sample 50% of vertices, while the edge sampling techniques select 50% of the edges to extract a subgraph.
(2) *Descriptive statistic estimation.* We calculated the average of the clustering coefficient (transitivity), average node degree and the degree correlation for the sampled graphs. We did 10 seeded experimental runs to get an estimate of the statistics and calculated the standard error around these averages.

4.4.2 Experimental results. The ground truth and estimated descriptive statistics are enclosed in Table 3 for all of the node level datasets. Blocks of rows correspond to node, edge, random walk based and non random based exploration sampling algorithms. In each block of methods bold numbers denote the best performing sampling technique (closest to the ground truth) in a given category for a specific estimated descriptive statistic and dataset.

We can make a few generic observations about the quality of descriptive statistic estimates. First, there is not a clearly superior sampling technique. This holds generally and within all of the main categories of considered algorithms. Specifically, there is not a superior node, edge or expansion based sampling procedure. Second, the induction based edge sampling techniques (TIES and PIES) give a good estimate of the statistics, but the induction step includes more than 50% of the edges. Because of this, the obtained good estimation performance is somewhat misleading as the majority of edges is still retained after the induction step. Third, edge sampling algorithms sometimes fail to estimate the direction of the degree correlation properly. Finally, the random walk based techniques generally tend to overestimate the average degree. This is not surprising considering that these are biased towards high degree nodes.

5 CONCLUSION AND FUTURE DIRECTIONS

In this paper we described *Little Ball of Fur* – an open-source Python graph sampling framework built on the widely used scientific computing libraries NetworkX [12] and NumPy [44]. In detail it provides techniques for node, edge, and exploration based graph sampling.

Table 3: Ground truth and estimated descriptive statistics of the web graphs and social networks. We calculated average statistics from 10 seeded experimental runs and included the standard errors below the mean. We included the ground truth values based on the whole graph (first block) with estimates obtained with node (second block), edge (third block) and exploration (fourth and fifth blocks) sampling algorithms. Bold numbers denote for each category the best estimate for a given dataset.

	Facebook Page-Page			Wikipedia Crocodiles			LastFM Asia			Deezer Hungary		
	Clustering Coefficient	Average Degree	Degree Correlation	Clustering Coefficient	Average Degree	Degree Correlation	Clustering Coefficient	Average Degree	Degree Correlation	Clustering Coefficient	Average Degree	Degree Correlation
Truth	0.232	15.205	0.085	0.026	29.365	−0.277	0.179	7.294	0.017	0.093	9.377	0.207
RN [39]	**0.229** 0.002	7.531 0.060	**0.070** 0.003	**0.028** 0.001	14.293 0.388	**-0.284** 0.008	**0.177** 0.004	3.642 0.032	**0.020** 0.010	**0.092** 0.001	4.699 0.010	0.190 0.002
DRN [1]	0.261 0.001	21.514 0.021	0.080 0.001	0.038 0.001	40.750 0.054	−0.324 0.001	0.231 0.001	9.531 0.020	0.045 0.001	0.102 0.001	**8.551** 0.007	**0.211** 0.001
PRN [19]	0.270 0.001	**16.228** 0.032	0.136 0.001	0.049 0.001	**32.370** 0.074	−0.290 0.001	0.236 0.001	**8.209** 0.022	0.064 0.002	0.098 0.001	7.251 0.007	0.231 0.001
RE [17]	0.116 0.001	8.470 0.004	**0.084** 0.001	0.013 0.001	15.462 0.009	**-0.277** 0.001	0.090 0.001	4.422 0.009	**0.019** 0.002	0.046 0.001	5.018 0.001	0.183 0.001
RNE [17]	0.092 0.001	7.602 0.001	−0.075 0.001	0.007 0.001	14.682 0.001	−0.231 0.001	0.046 0.001	3.674 0.001	−0.108 0.001	0.042 0.001	4.701 0.001	0.056 0.001
HRNE [17]	0.081 0.001	7.194 0.002	−0.005 0.001	0.007 0.001	13.550 0.005	−0.234 0.001	0.046 0.001	3.562 0.003	−0.070 0.002	0.039 0.001	4.501 0.001	0.095 0.001
TIES [2]	0.235 0.001	16.564 0.007	0.083 0.001	**0.026** 0.001	30.720 0.016	−0.278 0.001	0.190 0.001	8.218 0.010	0.027 0.001	**0.094** 0.001	**9.748** 0.001	0.204 0.001
PIES [2]	**0.231** 0.001	**15.357** 0.008	0.087 0.001	**0.026** 0.001	**29.142** 0.015	−0.283 0.001	**0.186** 0.001	**7.247** 0.010	0.037 0.001	0.086 0.001	8.501 0.003	**0.209** 0.001
RW [8]	0.255 0.001	22.535 0.022	0.073 0.001	0.036 0.001	41.648 0.196	−0.325 0.001	0.224 0.001	9.878 0.021	0.039 0.003	0.104 0.001	9.221 0.008	0.218 0.001
RWR [19]	0.253 0.003	20.282 0.293	0.092 0.007	0.043 0.003	38.967 0.549	−0.313 0.006	0.222 0.003	9.078 0.087	0.036 0.004	0.098 0.001	9.122 0.082	**0.213** 0.003
RWJ [32]	0.271 0.001	18.615 0.036	0.123 0.001	0.047 0.001	34.475 0.074	−0.297 0.001	0.233 0.001	9.012 0.032	0.067 0.003	0.102 0.001	8.351 0.016	0.233 0.002
MHRW [15, 40]	0.280 0.002	**17.903** 0.113	0.134 0.003	0.119 0.002	**29.914** 0.223	−0.146 0.004	0.232 0.001	**8.854** 0.041	0.102 0.007	0.106 0.001	7.761 0.023	0.242 0.003
RC-MHRW [23]	0.266 0.001	21.070 0.064	0.106 0.002	0.072 0.001	35.758 0.159	−0.254 0.002	0.232 0.001	9.594 0.031	0.078 0.003	0.103 0.001	**8.553** 0.021	0.237 0.002
FRW [32]	0.063 0.001	5.745 0.059	0.069 0.004	0.004 0.001	5.813 0.058	**-0.280** 0.003	0.084 0.001	4.485 0.032	**0.018** 0.008	0.033 0.001	3.243 0.005	0.091 0.002
CNRW [47]	0.255 0.001	22.590 0.038	0.072 0.001	0.037 0.001	41.645 0.104	−0.324 0.001	0.223 0.001	9.924 0.018	0.036 0.002	0.104 0.001	9.254 0.017	0.218 0.001
CNARW [24]	**0.239** 0.001	21.117 0.033	**0.082** 0.001	**0.026** 0.001	41.064 0.101	−0.348 0.001	**0.220** 0.001	9.508 0.032	0.052 0.002	**0.094** 0.001	9.140 0.013	0.218 0.001
NBT-RW [18]	0.257 0.001	22.353 0.048	0.076 0.001	0.038 0.001	41.264 0.144	−0.322 0.001	0.226 0.001	9.818 0.027	0.049 0.002	0.104 0.001	9.106 0.010	0.230 0.001
SB [10]	**0.238** 0.002	20.671 0.223	0.069 0.004	0.057 0.004	37.278 0.576	−0.292 0.009	0.207 0.002	9.131 0.121	−0.008 0.005	**0.093** 0.001	**9.913** 0.103	0.122 0.003
FF [20]	**0.238** 0.001	19.219 0.089	**0.079** 0.002	0.074 0.002	**33.190** 0.262	−0.227 0.006	0.204 0.001	9.034 0.025	0.051 0.001	0.096 0.001	10.120 0.013	0.197 0.001
CSE [25]	0.229 0.002	**13.116** 0.046	0.070 0.003	**0.026** 0.001	20.314 0.345	**-0.290** 0.006	**0.191** 0.003	6.544 0.055	0.006 **0.006**	0.089 0.002	6.554 0.001	0.165 0.001
SP [31]	0.221 0.001	12.842 0.062	0.106 0.002	0.037 0.001	23.451 0.125	−0.292 0.001	0.203 0.001	**7.258** 0.032	0.043 0.002	0.079 0.001	8.176 0.007	**0.209** 0.001

We reviewed the general conventions which we used for implementing graph sampling algorithms in *Little Ball of Fur*. The framework offers a limited number of public methods, ingests and outputs data in a widely used graph format, and embodies preset default hyperparameters. We presented practical implications of these desgin features with illustrative examples of Python code snippets. Using various social networks and web graphs we had shown that using sampled graphs extracted with *Little Ball of Fur* one can approximate ground truth graph level statistics such as transitivity and the degree correlation coefficient. We also found evidence that sampling subgraphs with our framework can accelerate node and graph classification without extremely reducing the predictive performance.

As we have emphasized *Little Ball of Fur* assumes that the inputted graph is undirected and unweighted. In the future we envision to relax these assumptions about the input. Furthermore, we aim to extend our framework by including multiplex [7], attributed, and heterogeneous [22, 46] graph sampling algorithms with new releases of the framework.

ACKNOWLEDGEMENTS

Benedek Rozemberczki was supported by the Centre for Doctoral Training in Data Science, funded by EPSRC (grant EP/L016427/1).

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] Shaosheng Cao, Wei Lu, and Qiongkai Xu. 2015. Grarep: Learning graph representations with global structural information. In *Proceedings of the 24th ACM international on conference on information and knowledge management*. ACM, 891–900.
[4] Hong Chen and Hisashi Koga. 2019. GL2vec: Graph Embedding Enriched by Line Graphs with Edge Features. In *International Conference on Neural Information Processing*. Springer, 3–14.
[5] Christian Doerr and Norbert Blenn. 2013. Metric convergence in social network sampling. In *Proceedings of the 5th ACM workshop on HotPlanet*. 45–50.
[6] David Easley, Jon Kleinberg, et al. 2010. *Networks, crowds, and markets*. Vol. 8. Cambridge university press Cambridge.
[7] Minas Gjoka, Carter T Butts, Maciej Kurant, and Athina Markopoulou. 2011. Multigraph sampling of online social networks. *IEEE Journal on Selected Areas in Communications* 29, 9 (2011), 1893–1905.
[8] Minas Gjoka, Maciej Kurant, Carter T Butts, and Athina Markopoulou. 2010. Walking in facebook: A case study of unbiased sampling of osns. In *2010 Proceedings IEEE Infocom*. Ieee, 1–9.
[9] Joseph E Gonzalez, Yucheng Low, Haijie Gu, Danny Bickson, and Carlos Guestrin. 2012. Powergraph: Distributed graph-parallel computation on natural graphs. In *Presented as part of the 10th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 12)*. 17–30.
[10] Leo A Goodman. 1961. Snowball sampling. *The annals of mathematical statistics* (1961), 148–170.
[11] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on*

Knowledge discovery and data mining. 855–864.
[12] 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).
[13] William L Hamilton, Rex Ying, and Jure Leskovec. 2017. Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584* (2017).
[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] U Kang, Charalampos E Tsourakakis, and Christos Faloutsos. 2009. Pegasus: A peta-scale graph mining system implementation and observations. In *2009 Ninth IEEE international conference on data mining*. IEEE, 229–238.
[17] 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.
[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] 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.
[20] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. 2005. Graphs over time: densification laws, shrinking diameters and possible explanations. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*. 177–187.
[21] Jure Leskovec and Andrej Krevl. 2014. SNAP Datasets: Stanford Large Network Dataset Collection. http://snap.stanford.edu/data.
[22] Jhao-Yin Li and Mi-Yen Yeh. 2011. On sampling type distribution from heterogeneous social networks. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 111–122.
[23] Rong-Hua Li, Jeffrey Xu Yu, Lu Qin, Rui Mao, and Tan Jin. 2015. On random walk based graph sampling. In *2015 IEEE 31st International Conference on Data Engineering*. IEEE, 927–938.
[24] 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.
[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] Annamalai Narayanan, Mahinthan Chandramohan, Rajasekar Venkatesan, Lihui Chen, and Yang Liu. 2017. graph2vec: Learning distributed representations of graphs. (2017).
[27] Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. 2016. Asymmetric transitivity preserving graph embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. 1105–1114.
[28] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. 2011. Scikit-learn: Machine learning in Python. *the Journal of machine Learning research* 12 (2011), 2825–2830.
[29] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 701–710.
[30] Bryan Perozzi, Vivek Kulkarni, Haochen Chen, and Steven Skiena. 2017. Don't Walk, Skip!: online learning of multi-scale network embeddings. In *Proceedings of the 2017 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2017*. ACM, 258–265.
[31] Alireza Rezvanian and Mohammad Reza Meybodi. 2015. Sampling social networks using shortest paths. *Physica A: Statistical Mechanics and its Applications* 424 (2015), 254–268.
[32] 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.
[33] Benedek Rozemberczki, Carl Allen, and Rik Sarkar. 2019. Multi-scale Attributed Node Embedding. arXiv:cs.LG/1909.13021
[34] Benedek Rozemberczki, Ryan Davies, Rik Sarkar, and Charles Sutton. 2019. GEMSEC: Graph Embedding with Self Clustering. In *Proceedings of the 2019 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2019*. ACM, 65–72.
[35] Benedek Rozemberczki, Oliver Kiss, and Rik Sarkar. 2020. An API Oriented Open-source Python Framework for Unsupervised Learning on Graphs. arXiv:cs.LG/2003.04819
[36] Benedek Rozemberczki and Rik Sarkar. 2018. Fast Sequence-Based Embedding with Diffusion Graphs. In *International Workshop on Complex Networks*. Springer, 99–107.
[37] Benedek Rozemberczki and Rik Sarkar. 2020. Characteristic Functions on Graphs: Birds of a Feather, from Statistical Descriptors to Parametric Models. arXiv:cs.LG/2005.07959
[38] Shai Shalev-Shwartz and Shai Ben-David. 2014. *Understanding machine learning: From theory to algorithms*. Cambridge university press.
[39] 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.
[40] Daniel Stutzbach, Reza Rejaie, Nick Duffield, Subhabrata Sen, and Walter Willinger. 2008. On unbiased sampling for unstructured peer-to-peer networks. *IEEE/ACM Transactions on Networking* 17, 2 (2008), 377–390.
[41] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. Line: Large-scale information network embedding. In *Proceedings of the 24th international conference on world wide web*. 1067–1077.
[42] Anton Tsitsulin, Davide Mottin, Panagiotis Karras, Alexander Bronstein, and Emmanuel Müller. 2018. Netlsd: hearing the shape of a graph. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2347–2356.
[43] Saurabh Verma and Zhi-Li Zhang. 2017. Hunt for the unique, stable, sparse and fast feature learning on graphs. In *Advances in Neural Information Processing Systems*. 88–98.
[44] Stéfan van der Walt, S Chris Colbert, and Gael Varoquaux. 2011. The NumPy array: a structure for efficient numerical computation. *Computing in Science & Engineering* 13, 2 (2011), 22–30.
[45] David Bruce Wilson. 1996. Generating random spanning trees more quickly than the cover time. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*. 296–303.
[46] Cheng-Lun Yang, Perng-Hwa Kung, Chun-An Chen, and Shou-De Lin. 2013. Semantically sampling in heterogeneous social networks. In *Proceedings of the 22nd International Conference on World Wide Web*. 181–182.
[47] Zhuojie Zhou, Nan Zhang, and Gautam Das. 2015. Leveraging History for Faster Sampling of Online Social Networks. *Proceedings of the VLDB Endowment* 8, 10 (2015).