A High Performance Graph Embedding Platform

Yan Xie
Futurewei Technologies
Santa Clara, California, USA
yan.xie@futurewei.com

Zhaoxi Zhang
Futurewei Technologies
Santa Clara, California, USA
zhaoxi.zhang@futurewei.com

Yinglong Xia
Futurewei Technologies
Santa Clara, California, USA
yinglong.xia@futurewei.com

ABSTRACT

In recent years, graph neural networks (GNN) gains increasing attention in both academia and industry. By incorporating both the features and structural information of entities, GNN not only leverages the power of deep learning, but also preserves the advantages of graph analysis, such as being intuitive for modeling and analysis. However, due to additional complexity for neural training, GNN suffers more from performance challenges than traditional network analysis does, especially when the underlying graph scales up. In this paper, we focus on random walk based graph embedding for representation learning, a fundamental branch of GNN, to develop a high performance platform for industrial use. The platform is built on top of a graph engine, which supports several highly efficient operators on data manipulation in graph embedding, such as neighborhood walking and negative sampling. A dynamic scheduler is utilized to assign operators between the graph engine and a deep learning framework. Experimental results on representative datasets show that our platform is significantly more efficient than the baseline methods; while still can derive graph embeddings of high quality.

CCS CONCEPTS

• **Computing methodologies** → Neural networks; • **Information systems** → Database management system engines; • **Applied computing** → Enterprise architecture frameworks.

KEYWORDS

graph neural network, graph embedding, computing platform

ACM Reference Format:
Yan Xie, Zhaoxi Zhang, and Yinglong Xia. 2019. A High Performance Graph Embedding Platform. In *proceedings of 2019 ACM SIGKDD Conference (DLG workshop).* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Graph, as a natural data representation to entities and their interconnections, is nowadays ubiquitous in a wide range of real-world applications. Graph Neural Network (GNN) explores graph using deep learning. Among various GNN methods, graph embedding is to learn a representation of graph vertices into a low-dimensional vector space, and the representation can preserve both the vertex feature information and the topological information. DeepWalk [8] and Node2Vec [4] are two representative examples of random walk based graph embedding algorithms.

Motivated by many real use cases, we seek an efficient platform for large scale graph embedding. After a thorough study on state-of-the-art, we observed that the performance bottleneck is largely caused by the graph data access/manipulation, in contrast to traiditoinal deep leanring where the training computation dominates the execution time; besides, we found that almost every step of a random walk based graph embedding is highly parallelizable, making it promising to implement a high performance framework. Thus, we present an integrated platform for end-to-end graph embedding, which includes 1) a high performance distributed graph engine that provides optimial access to graph data; 2) a group of graph operators e.g. sampling built on top of our graph engine; and 3) a model training layer that learns the representation.

The rest of the paper is organized as follows: in Section 2, some background and related work is addressed. We investigate the problems and present our platform in Section 3, followed by an extensive empirical study in Section 4. Finally, the work is concluded in Section 5.

2 BACKGROUND AND RELATED WORK

Algorithm 1: Random-walk graph embedding

```
Input: a graph $G = (V, E)$, number of walks $r$, walk length $l$
Output: vertex embedding $\mathbf{h}$
for $iter \leftarrow 1$ to $r$ do
    for each vertex $v \in V$ do
        $walks$ += random_walk_sampler($G, v, l$);
    end
end
$dl\_input \leftarrow$ preprocess($walks$);
$\mathbf{h} \leftarrow$ training($dl\_input$) ;
```

Graph embedding has been well studied in literature, which essentially represents each vertex of the input graph as a vector in a low dimensional space; while preserving the similarity implied by vertex features as well as the topology. In general, graph embedding techniques can be divided into the following categories [11]: 1) matrix factorization [9][12] that obtains the embedding by factorizing the matrix used to represent graph property; 2) random walk based graph embedding [8][4][2] that performs random walk on the graph and then feeds the processed information to a neural network training model like Word2Vec [6] ; 3) graph deep learning

KDD '19 (DLG workshop), August 2019, Anchorage, Alaska, USA
© 2019 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

that is based on either auto-encoder [10] or graph convolutional network with unsupervised learning [5].

Random walk based graph embedding starts off by conducting random walk on graph, using various walking strategies [8][4][2]. Using the generated walking paths, the technique generates vertex pairs for the training step. The last step is to learn the representation using neural network models e.g. Word2Vec [6]. A general framework of random walk based graph embedding algorithms is shown in Algorithm 1.

3 FRAMEWORK

3.1 Observations

Random-walk-based graph embedding methodologies often consists of two parts, namely the random walk sampler and the embedding learning. We conducted extensive study on open source implementations of those algorithms, and below are some key observations. More details are available in the experiment section.

- Both biased and unbiased random walk samplers suffer from serious performance deterioration because of poorly designed storage structure. A reasonable and efficient graph storage structure is highly preferable, especially when the graph size is at a large-scale.
- When comparing the execution time of random walk sampler against embedding learning, random walk sampler dominates the execution time of the whole procedure. This is to say, in a large-scale system, it is critical to improve the efficiency of random walk sampler.
- To generate random walk paths, the sampler origins from one vertex and starts walking based on the graph structure. There is no conflicting resource requirement if multiple samplers starting from different vertices walk on the same graph simultaneously. Therefore, random walk sampler is highly parallelizable.

3.2 Our platform

To tackle the issues stated in the above section, we devise a novel GNN platform illustrated in Figure 1, with emphasis on both data manipulation and model training for GNN.

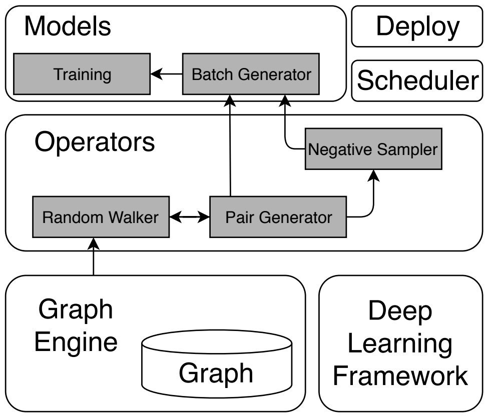


Figure 1: System Architecture and Workflow

Essential engines: The proposed system is built on top of two essential engines, i.e., a graph engine and a neural network framework such as Tensorflow. In contrast to many existing GNN frameworks/platforms, a comprehensive *graph engine* is utilized as a core fundamental component in our design, in charge of both GNN data import and the support of data-oriented operators, which is motivated by the first observation described in Section 3.1. Note that there is a *graph database* in the graph engine, which implements the data management based on the *property graph model* (PGM). The PGM consists of both the graph topology (structural information) and the attributes associated with each vertex and edge, a.k.a. properties. The other fundamental component is a *deep learning framework*, such as TensorFlow, to support the operators for training models, such as those training a Word2Vec model using the loss for cross entropy.

Operator management: All the *operators* involved in our GNN computation, no matter for data processing or model training, are managed at the operator layer shown in Figure 1. A few operators are illustrated at the layer in the figure, although more are available for use. Note that the operators are actually implemented by calling the APIs of the two essential platforms. For instance, the operator of *random walker* enables local random walk from any designated roots in a graph. The operator calls the graph engine to obtain the neighbors of a vertex and determines according to user configuration which neighbors to explore. An operator can invoke another operators, as long as they are orchestrated by users through the scheduler module. A group of orchestrated operators form up a model.

Model management: This layer consists of pre-built and user-defined models through the orchestration of the operators. Two relevant modules, the *deployer* and the *scheduler*, are introduced. The former deploys a model to infrastructure with the graph engines and deep learning framework installed; the latter allocate operators invoked by a model and improve the workload balance through dynamic task scheduling.

In our platform, a sample **workflow** of random walk based graph embedding is shown in Figure 1. After the graph is digested and processed into the graph database, with the support of graph engine, the operator in the operator layer can work on the graph. Specifically for two graph embedding methods DeepWalk and Node2Vec, a random walker sampler first starts to transform the graph into a list of paths based on certain transition probabilities, i.e.,

$$P(v_i = v|v_{i-1} = u) = \begin{cases} p_{u,v}, & \text{if } (u,v) \text{ in E} \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

where u is the current vertex, v is the next vertex in the path, and $p_{u,v}$ denotes the transition probability from u to v. DeepWalk and Node2Vec use uniform and biased transition probabilities, respectively. Compared to original implementation with no support from graph engine, our random walker sampler utilizes the graph engine service to achieve extremely higher efficiency during the sampling process. To further accelerate the process, walking paths are fed into pair generation operator to do the preprocess before the information is finally ready for the training part.

Both DeepWalk and Node2Vec use the Word2Vec [6] model to train graph embedding for each vertex. There has been some study on how to efficiently train the Word2Vec model for large vocabularies [7]. Moreover, some work is devoted to the acceleration on GPU for the training part [1], so the optimization of the Word2Vec model training is beyond the scope of this paper.

4 EMPIRICAL STUDY

In this section, we evaluate the performance of our platform in along with a comparison with open source implementations of DeepWalk [8] and Node2Vec [4]. Section 4.1 summarizes different datasets in the test and experiment infrastructure. Section 4.2 gives details of the experiment results from various perspectives.

4.1 Datasets and Infrastructure

The experiment is conducted on various datasets of different sizes. Table 1 lists all the datasets used in our experiments.

Table 1: Summary of the datasets

Dataset	# Vertices	# Total edges	# Classes
Blogcatalog	10,312	333,983	39
Gowalla	196,591	950,327	N/A
Amazon	334,863	925,872	N/A
Youtube	1,134,890	2,987,264	N/A
Reddit	232,965	11,606,919	N/A

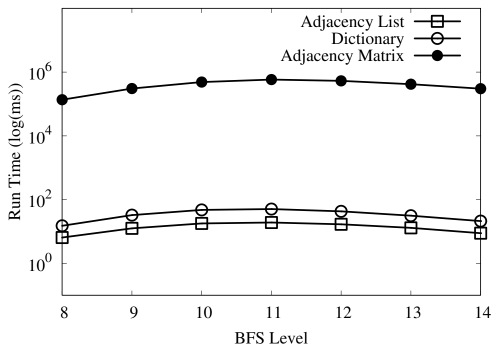
Please note that the dataset *Youtube* and *Reddit* are only used on our proposed platform while all other datasets are tested in both our platform and the baselines. Because of the large size of *Youtube* and *Reddit* datasets, neither baseline can finish in a reasonable amount of time.

All experiments are conducted on a 44-cores linux server with Intel(R) Xeon(R) CPU E5-2699 v4 @ 2.20GHz and 1.5T main memory. Details about parameter setting in the test will be given when presenting the experimental results.

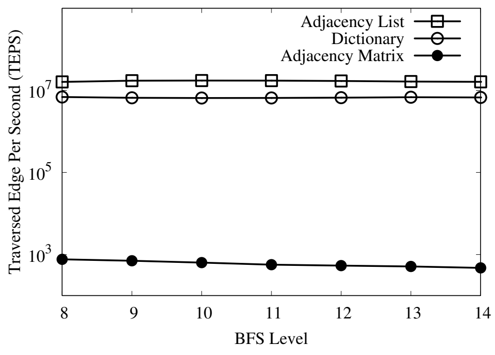
4.2 Results

In the a computation platform, it is critical to have the storage and data structure reasonably designed. We first study how different structures have impact on the efficiency. Specifically, we use different data structures to store the graph in main memory and test how the efficiency varies for breadth first search, as random walk sampler is in general BFS, DFS or a combination of Both. As shown in Figure 2 showing result for the dataset *amazon*, adjacency list gains the highest efficiency in terms of both running time and traversed edge per second (TEPS). It is orders of magnitude faster when compared with adjacency matrix, and almost 2 times faster than the structure that stores edges of each vertex as a dictionary. This observation indicates that when graph is stored with some structure similar with adjacency list, the computation cost is much lower. This structure is what we use in our graph engine. Due to space limit, results on other datasets and for depth first search (DFS) are omitted. They show very similar results with Figure 2.

Another observation shown during our study on random walk based graph embedding is that in random walk based graph embedding, it often takes significantly longer for random walk sampling part to finish rather than embedding training part, especially for more advanced and complicated algorithm like Node2Vec. Figure 3 illustrates the running time comparison between random walk sampling and embedding training in Node2Vec. In the test, we set number of walks as 40 while the length of each walk is also 40. As we can see, in all three datasets, the execution time of random walk sampling dominates. It can occupies as much as 96.5% of the whole execution time for graph with relatively high average node degree like *Blogcatalog*. Therefore, improving the efficiency of random walk sampling can greatly help the overall performance of graph embedding platform.



(a) Run Time



(b) TEPS

Figure 2: Efficiency Comparison on Different Structures

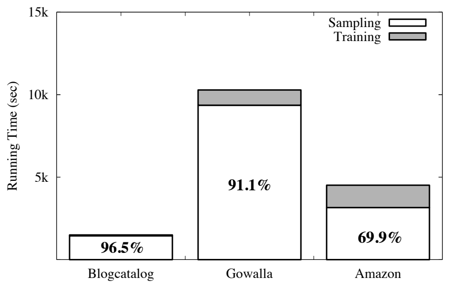
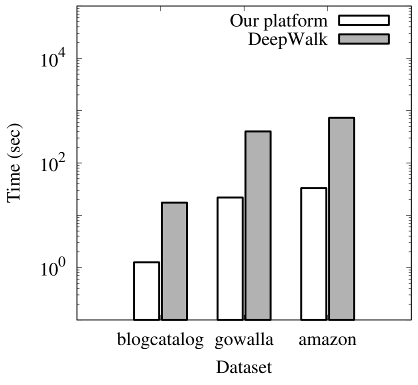
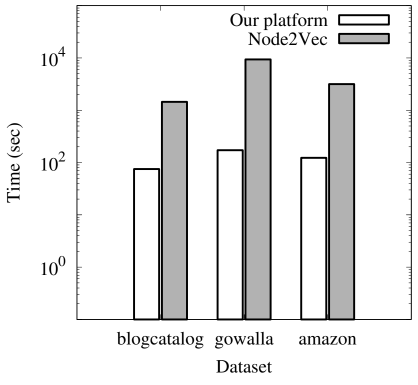


Figure 3: Sampling v.s. Training

Figure 4a and Figure 4b evaluate the sampling time of our graph embedding platform compared with the open source implementations of DeepWalk and Node2Vec, respectively. Our proposed planform uses the in-house powerful graph engine service with random walk operator and preprocess operator. Therefore, it shows superior performance improvement over original DeepWalk and Node2Vec. For example, for the dataset *Gowalla*, DeepWalk uses roughly 401 seconds to do random walk sampling. On our high performance graph embedding platform, it only take 21.87 seconds, which results in a 20X speedup. The improvement is further amplified in Node2vec with biased random walk. To complete the biased random walking sampling in Node2Vec for the dataset *Gowalla*, the original implementation uses 9,359 seconds, while our platform needs only 171.92 seconds, indicating a more than 50X speedup.



(a) Sampling Time



(b) Sampling Time

Figure 4: Efficiency Comparison

While the performance in terms of efficiency has been greatly improved in our platform, we still want to preserve the quality of the embedding as before. Here, we further evaluate the quality of the embedding in the context of node classification. To be specific, we use the dataset *Blogcatalog* that has the label information to learn its embedding on each vertex, then feed the embedding into a downstream logistic regression model. The quality of the embedding can be measured as the node classification accuracy. The result is plotted in Figure 5 with x-axis indicating the portion of labeled data used. As it shows, in both DeepWalk and Node2Vec, the quality of the embedding from our platform is comparable with the original implementations in all settings.

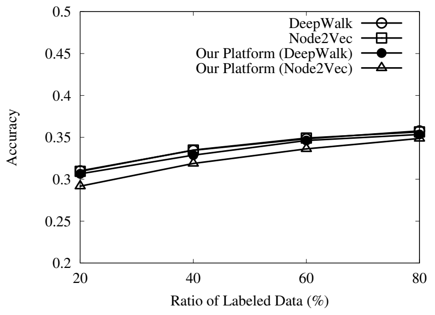
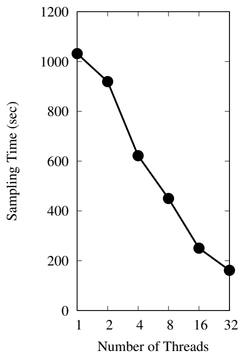
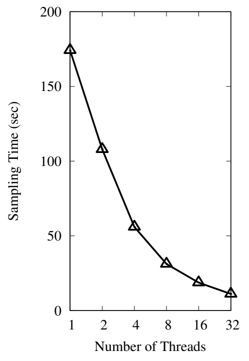


Figure 5: Quality of Graph Embedding in Node Classification

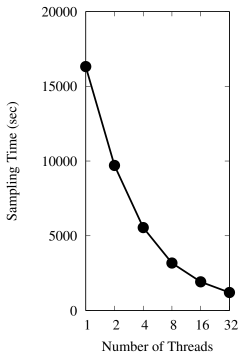


(a) Node2Vec

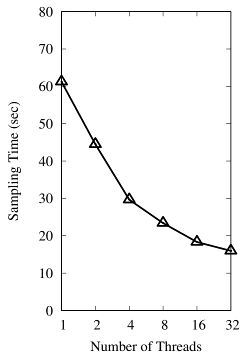


(b) DeepWalk

Figure 6: Efficiency on Multi-thread (*Youtube*)



(a) Node2Vec



(b) DeepWalk

Figure 7: Efficiency on Multi-thread (*Reddit*)

The result of the multi-thread test of our high performance graph embedding platform on dataset *Youtube* and *Reddit* are shown in

Figure 6 and 7, respectively. We vary the number of threads from 1 to 32, and then record the corresponding random walk sampling time. The datasets are not tested in either original DeepWalk or original Node2Vec as the datasets are of large size and cannot be done in a reasonable amount of time. As demonstrated in Figure 6 and 7, both DeepWalk and Node2Vec in our platform are highly parallelizable with respect to the number of threads.

5 CONCLUSION

In this work, we propose an integrated graph neural network framework with the focus on random walk based graph embedding. In our framework, we take advantage of our internal graph engine service to provide graph storage and fast access in a distributed fashion. To better assist graph embedding, some common operators such as biased and unbiased random walk based sampling are implemented on top of the graph engine. After sampling, a graph structure can be converted to regular vectorized input suitable for word embedding training. Therefore, a model training layer based on general deep learning library such as Tensorflow can kick in and output the embedding of the vertices in the graph.

Note that the overall structure of our framework is generic and suitable for other graph neural network methods as well. In the future, we will integrate other graph neural network methods such as graph convolutional networks (GCN) into our platform. Thus our platform can be a unified and powerful graph neural network platform for more industrial and practical large-scale applications.

REFERENCES

[1] John F. Canny, Huasha Zhao, Bobby Jaros, Ye Chen, and Jiangchang Mao. 2015. Machine learning at the limit. In *2015 IEEE International Conference on Big Data, Big Data 2015, Santa Clara, CA, USA, October 29 - November 1, 2015*. IEEE Computer Society, 233–242. https://doi.org/10.1109/BigData.2015.7363760

[2] Yuxiao Dong, Nitesh V. Chawla, and Ananthram Swami. 2017. metapath2vec: Scalable Representation Learning for Heterogeneous Networks. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Halifax, NS, Canada, August 13 - 17, 2017*. ACM, 135–144. https://doi.org/10.1145/3097983.3098036

[3] Palash Goyal and Emilio Ferrara. 2018. Graph embedding techniques, applications, and performance: A survey. *Knowl.-Based Syst.* 151 (2018), 78–94. https://doi.org/10.1016/j.knosys.2018.03.022

[4] 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 (KDD '16)*. ACM, New York, NY, USA, 855–864. https://doi.org/10.1145/2939672.2939754

[5] William L. Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (Eds.). 1025–1035. http://papers.nips.cc/paper/6703-inductive-representation-learning-on-large-graphs

[6] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. In *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.). http://arxiv.org/abs/1301.3781

[7] Erik Ordentlich, Lee Yang, Andy Feng, Peter Cnudde, Mihajlo Grbovic, Nemanja Djuric, Vladan Radosavljevic, and Gavin Owens. 2016. Network-Efficient Distributed Word2Vec Training System for Large Vocabularies. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management (CIKM '16)*. ACM, New York, NY, USA, 1139–1148. https://doi.org/10.1145/2983323.2983361

[8] 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 (KDD '14)*. ACM, New York, NY, USA, 701–710. https://doi.org/10.1145/2623330.2623732

[9] Xiaobo Shen, Shirui Pan, Weiwei Liu, Yew-Soon Ong, and Quan-Sen Sun. 2018. Discrete Network Embedding. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden.*, Jérôme Lang (Ed.). ijcai.org, 3549–3555. https://doi.org/10.24963/ijcai.2018/493

[10] Daixin Wang, Peng Cui, and Wenwu Zhu. 2016. Structural Deep Network Embedding. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, Balaji Krishnapuram, Mohak Shah, Alexander J. Smola, Charu C. Aggarwal, Dou Shen, and Rajeev Rastogi (Eds.). ACM, 1225–1234. https://doi.org/10.1145/2939672.2939753

[11] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. 2019. A Comprehensive Survey on Graph Neural Networks. *CoRR* abs/1901.00596 (2019). arXiv:1901.00596 http://arxiv.org/abs/1901.00596

[12] Hong Yang, Shirui Pan, Peng Zhang, Ling Chen, Defu Lian, and Chengqi Zhang. 2018. Binarized attributed network embedding. In *IEEE International Conference on Data Mining, ICDM 2018, Singapore, November 17-20, 2018*. IEEE Computer Society, 1476–1481. https://doi.org/10.1109/ICDM.2018.8626170