A Generic Approach to Scale Graph Embedding Methods

Jiongqian Liang
The Ohio State University
liang.albert@outlook.com

Saket Gurukar
The Ohio State University
gurukar.1@osu.edu

Srinivasan Parthasarathy
The Ohio State University
srini@cse.ohio-state.edu

ABSTRACT

Recently there has been a surge of interest in designing graph embedding methods. Few, if any, can scale to a large-sized graph with millions of nodes due to both computational complexity and memory requirements. In this paper, we relax this limitation by introducing the MultI-Level Embedding (MILE) framework – a generic methodology allowing contemporary graph embedding methods to scale to large graphs. MILE repeatedly coarsens the graph into smaller ones using a hybrid matching technique to maintain the backbone structure of the graph. It then applies existing embedding methods on the coarsest graph and refines the embeddings to the original graph through a novel graph convolution neural network that it learns. The proposed MILE framework is agnostic to the underlying graph embedding techniques and can be applied to many existing graph embedding methods without modifying them. We employ our framework on several popular graph embedding techniques and conduct embedding for real-world graphs. Experimental results on three large-scale datasets demonstrate that MILE significantly boosts the speed (order of magnitude) of graph embedding while also often generating embeddings of better quality for the task of node classification.

CCS CONCEPTS

• **Computing methodologies** → *Neural networks*; *Machine learning*; • **Information systems** → *Data mining*;

KEYWORDS

Network Embedding, Graph Convolutional Networks

1 INTRODUCTION

In recent years, *graph embedding* has attracted much interest due to its broad applicability for tasks such as vertex classification [7] and full network visualization [9]. However, such methods rarely scale to large datasets (e.g., graphs with over 1 million nodes) since they are computationally expensive and often memory intensive. For example, random-walk-based embedding techniques, such as DeepWalk [7] and Node2Vec [2], require a large amount of CPU time to generate a sufficient number of walks and train the embedding model. As another example, embedding methods based on matrix factorization, including GraRep [1] and NetMF [8], requires constructing an enormous objective matrix (usually much denser than adjacency matrix) on which matrix factorization is performed. Even a medium-size graph with 100K nodes can easily require hundreds of GB of memory using those methods. On the other hand, many graph datasets in the real world tend to be large-scale with millions or even billions of nodes. To the best of our knowledge, none of the existing efforts examines how to scale up graph embedding in a **generic** way. We make the first attempt to close this gap. There are very few efforts, focusing on the scalability of network embedding [3, 10]. Such efforts are specific to a particular embedding strategy and do not offer a generic strategy to scale other embedding techniques. Incidentally, the earlier efforts at scalability are actually orthogonal to our strategy and can potentially be employed along with our efforts to afford even greater speedup. In this paper, we focus on following questions:

(1) Can we scale up the existing embedding techniques in an agnostic manner so that they can be directly applied to larger datasets?

(2) Can the quality of such embedding methods be strengthened by incorporating the holistic view of the graph?

To tackle these problems, we propose a $\underline{\text{M}}$ult$\underline{\text{I}}$-$\underline{\text{L}}$evel $\underline{\text{E}}$mbedding (MILE) framework for graph embedding. Its overview is shown in Figure 1. Our approach relies on a three-step process: **first**, we repeatedly coarsen the original graph into smaller ones by employing a hybrid matching strategy; **second**, we compute the embeddings on the coarsest graph using an existing embedding mechanism and **third**, we propose a novel refinement model based on learning a graph convolution network to refine the embeddings from the coarsest graph to the original graph – learning a graph convolution network allows us to compute a refinement procedure that levers the dependencies inherent to the graph structure and the embedding method of choice.

To summarize, we find that:
i) MILE is generalizable. Our MILE framework is agnostic to the underlying graph embedding techniques and treats them as black boxes. **ii) MILE is scalable.** We show that the proposed framework can *significantly improve the scalability of the embedding methods* (**up to 30-fold**), by reducing the running time and the memory consumption. **iii) MILE generates high-quality embeddings.** In many cases, we find that the quality of embeddings improves by levering MILE (in some cases is in excess of 10%). **iv) MILE's ability to learn a data- and embedding- sensitive refinement procedure is key to its effectiveness.** Other design choices such as the hybrid coarsening strategy also enable MILE to produce quality embeddings in a scalable fashion.

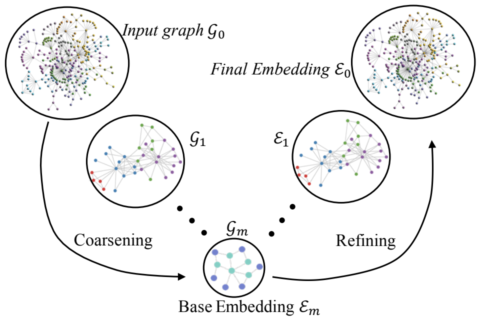
2 METHODOLOGY

MILE framework [1] consists of three key phases: graph coarsening, base embedding, and embeddings refining.

This work was done when the first author was a student at The Ohio State University. The author now works at Google.

KDD'18 Deep Learning Day, August 2018, London, UK
© 2018 Copyright held by the owner/author(s).
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

[1]For more details, please refer [6]



Input graph $\mathcal{G}_0$
Final Embedding $\mathcal{E}_0$
$\mathcal{G}_1$
$\mathcal{E}_1$
$\mathcal{G}_m$
Coarsening
Refining
Base Embedding $\mathcal{E}_m$

Figure 1: An overview of the multi-level embedding framework.

2.1 Graph Coarsening

In this phase, the input graph $\mathcal{G}$ (or $\mathcal{G}_0$) is repeatedly coarsened into a series smaller graphs $\mathcal{G}_1, \mathcal{G}_2, ..., \mathcal{G}_m$ such that $|V_0| > |V_1| > ... > |V_m|$. In order to coarsen a graph from $\mathcal{G}_i$ to $\mathcal{G}_{i+1}$, multiple nodes in $\mathcal{G}_i$ are collapsed to form super-nodes in $\mathcal{G}_{i+1}$, and the edges incident on a super-node are the union of the edges on the original nodes in $\mathcal{G}_i$ [4]. Here the set of nodes forming a super-node is called a *matching*. We propose a hybrid matching technique containing two matching strategies that can efficiently coarsen the graph while retaining the global structure.

2.1.1 Structural Equivalence Matching (SEM). Given two vertices u and v in an unweighted graph $\mathcal{G}$, we call they are *structurally equivalent* if they share the same neighborhoods.

2.1.2 Normalized Heavy Edge Matching (NHEM). Heavy edge matching is a popular matching method for graph coarsening [4]. For an unmatched node u in $\mathcal{G}_i$, its heavy edge matching is a pair of vertices (u, v) such that the weight of the edge between u and v is the largest. In this paper, we propose to normalize the edge weights when applying heavy edge matching using the formula as follows

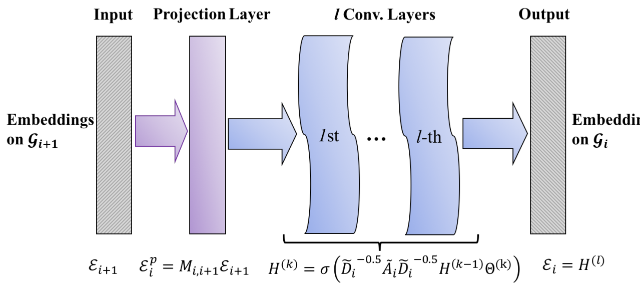
$$W_i(u,v) = \frac{A_i(u,v)}{\sqrt{D_i(u,u) \cdot D_i(v,v)}}. \tag{1}$$

Here, the weight of an edge is normalized by the degree of the two vertices on which the edge is incident. Intuitively, it penalizes the weights of edges connected with high-degree nodes. As we will show in Sec. 2.3, this normalization is tightly connected with the graph convolution kernel.

2.1.3 A Hybrid Matching Method. We use a hybrid of two matching methods above for graph coarsening. To construct $\mathcal{G}_{i+1}$ from $\mathcal{G}_i$, we first find out all the structural equivalence matching (SEM) $\mathcal{M}_1$, where $\mathcal{G}_i$ is treated as an unweighted graph. This is followed by the searching of the normalized heavy edge matching (NHEM) $\mathcal{M}_2$ on $\mathcal{G}_i$. Nodes in each matching are then collapsed into a super-node in $\mathcal{G}_{i+1}$. Note that some nodes might not be matched at all and they will be directly copied to $\mathcal{G}_{i+1}$.

Formally, we build the adjacency matrix A_{i+1} of $\mathcal{G}_{i+1}$ through matrix operations. To this end, we define the *matching matrix* storing the matching information from graph $\mathcal{G}_i$ to $\mathcal{G}_{i+1}$ as a binary matrix $M_{i,i+1} \in \{0,1\}^{|V_i| \times |V_{i+1}|}$. The r-th row and c-th column of $M_{i,i+1}$ is set to 1 if node r in $\mathcal{G}_i$ will be collapsed to super-node c in $\mathcal{G}_{i+1}$, and is set to 0 if otherwise. Each column of $M_{i,i+1}$ represents a matching with the 1s representing the nodes in it. Each unmatched vertex appears as an individual column in $M_{i,i+1}$ with merely one entry set to 1. Following this formulation, we construct the adjacency matrix of $\mathcal{G}_{i+1}$ by using

$$A_{i+1} = M_{i,i+1}^T A_i M_{i,i+1}. \tag{2}$$



Input
Projection Layer
l Conv. Layers
Output
Embeddings on $\mathcal{G}_{i+1}$
lst
...
l-th
Embeddings on $\mathcal{G}_i$
$\mathcal{E}_{i+1}$
$\mathcal{E}_i^p = M_{i,i+1}\mathcal{E}_{i+1}$
$H^{(k)} = \sigma\left(\widetilde{D}_i^{-0.5}\tilde{A}_i\widetilde{D}_i^{-0.5}H^{(k-1)}\Theta^{(k)}\right)$
$\mathcal{E}_i = H^{(l)}$

Figure 2: Architecture of the embeddings refinement model. The input layer is the embeddings $\mathcal{E}_{i+1}$ of the coarsened graph $\mathcal{G}_{i+1}$. The projection layer computes the projected embeddings $\mathcal{E}_i^p$ based on the matching matrix $M_{i,i+1}$ using Eq. 4. Following this, the projected embeddings go through l graph convolution layers and output the refined embeddings $\mathcal{E}_i$ of graph $\mathcal{G}_i$ at the end. Note the model parameters $\Theta^{(k)}$ ($k = 1...l$) are shared among all the refinement steps ($\mathcal{G}_{i+1}$ to $\mathcal{G}_i$, where $i = m-1...0$).

2.2 Base Embedding on Coarsened Graph

The size of the graph reduces drastically after each iteration of coarsening, halving the size of the graph in the best case. We coarsen the graph for m iterations and apply the graph embedding method $f(\cdot)$ on the coarsest graph $\mathcal{G}_m$. Denoting the embeddings on $\mathcal{G}_m$ as $\mathcal{E}_m$, we have

$$\mathcal{E}_m = f(\mathcal{G}_m). \tag{3}$$

Since our framework is agnostic to the adopted graph embedding method, we can use any graph embedding algorithm for base embedding.

2.3 Embeddings Refinement

The final phase of MILE is the embeddings refinement phase. Given a series of coarsened graph $\mathcal{G}_0, \mathcal{G}_1, \mathcal{G}_2, ..., \mathcal{G}_m$, their corresponding matching matrix $M_{0,1}, M_{1,2}, ..., M_{m-1,m}$, and the node embeddings $\mathcal{E}_m$ on $\mathcal{G}_m$, we seek to develop an approach to derive the node embeddings of $\mathcal{G}_0$ from $\mathcal{G}_m$. we propose to use a graph-based neural network model to perform embeddings refinement.

2.3.1 Graph Convolution Network for Embeddings Refinement. We project embeddings from coarse-grained graph $\mathcal{G}_{i+1}$ to the fine-grained graph $\mathcal{G}_i$ using

$$\mathcal{E}_i^p = M_{i,i+1}\mathcal{E}_{i+1} \tag{4}$$

where we call $\mathcal{E}_i^p$ as *projected embeddings*. The proposed graph-based neural network model derives the embedding $\mathcal{E}_i$ on graph $\mathcal{G}_i$ using $\mathcal{E}_i = \mathcal{R}(\mathcal{E}_i^p, A_i)$.

We define our embedding refinement model $\mathcal{R}$ as a l-layer graph convolution model [5]

$$\mathcal{E}_i = \mathcal{R}\left(\mathcal{E}_i^p, A_i\right) \equiv H^{(l)}\left(\mathcal{E}_i^p, A_i\right). \tag{5}$$

where

$$H^{(k)}(X,A) = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(k-1)}(X,A)\Theta^{(k)}\right) \tag{6}$$

where $\sigma(\cdot)$ is an activation function, $\Theta^{(k)}$ is a layer-specific trainable weight matrix, $H^{(0)}(X, A) = X$, $\tilde{A} = A + \lambda D$, $\tilde{D}(i,i) = \sum_j \tilde{A}(i,j)$, $\Theta \in \mathbb{R}^{d \times d}$, $\lambda \in [0,1]$ is a hyper-parameter for controlling the weight of self-loop and D is a diagonal matrix with entries $D(i,i) = \sum_j A(i,j)$. The architecture of the refinement model is shown in Figure 2.

The intuition behind this refinement model is to integrate the structural information of the current graph $\mathcal{G}_i$ into the projected embedding $\mathcal{E}_i^p$ by repeatedly performing the spectral graph convolution. To some extent, each layer of graph convolution network in Eq. 6 can be regarded as one iteration of embedding propagation in the graph following the re-normalized adjacency matrix $\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}$. This re-normalized matrix is also well aligned with the way we conduct normalized heavy edge matching in Eq. 1, where we apply the same way of re-normalization on the adjacency matrix for edge matching. However, we point out that the graph convolution model goes beyond just simple propagation in that the activation function is applied for each iteration of propagation and each dimension of the embedding interacts with other dimensions controlled by the weight matrix $\Theta^{(k)}$.

2.3.2 Refinement Model Learning. The learning of the refinement model is essentially learning $\Theta^{(k)}$ for each $k \in [1, l]$ according to Eq. 6. Here we study how to design the learning task and construct the loss function. Since the graph convolution model $H^{(l)}(\cdot)$ aims to predict the embeddings $\mathcal{E}_i$ on graph $\mathcal{G}_i$, we can directly run a base embedding on $\mathcal{G}_i$ to generate the "ground-truth" embeddings and use the difference between these embeddings and the predicted ones as the loss function for training. We propose to learn $\Theta^{(k)}$ on the coarsest graph and **reuse** them across all the levels for refinement. The loss function for model learning is as follows:

$$L = \frac{1}{|V_m|} \left\| \mathcal{E}_m - H^{(l)}(\mathcal{E}_m, A_m) \right\|^2. \tag{7}$$

With the above loss function, we adopt gradient descent with back-propagation to learn the parameters $\Theta^{(k)}$, $k \in [1, l]$. In the subsequent refinement steps, we apply the same set of parameters $\Theta^{(k)}$ to infer the refined embeddings. We point out that the training of the refinement model is rather efficient as it is done on the coarsest graph, which is usually much smaller than the original graph. The embeddings refinement process involves merely sparse matrix multiplications using Eq. 5 and is relatively affordable compared to conducting embedding on the original graph. With these different components, we summarize the whole algorithm of our MILE framework in Algorithm 1.

Time Complexity: It is non-trivial to derive the exact time complexity of MILE as it is dependent on the graph structure, the chosen base embedding method, and the convergence rate of the GCN model training. Here, we provide a rough estimation of the time complexity. For simplicity, we assume the number of vertices and the number of edges are reduced by factor α and β respectively at each step of coarsening ($\alpha > 1.0$ and $\beta > 1.0$), i.e., $V_i = \frac{1}{\alpha} V_{i-1}$ and $E_i = \frac{1}{\beta} E_{i-1}$. (we found α and β in range [1.5, 2.0], empirically). With m levels of coarsening, the coarsening complexity is approximately $O((1 - 1/\beta^{\mathrm{m}})/(1 - 1/\beta) \times E))$ and since $1/\beta^{\mathrm{m}}$ is small, the complexity reduces to $O(\frac{\beta}{\beta - 1} \times E)$. For the base embedding phase, if the embedding algorithm has time complexity of $T(V, E)$, the complexity of the base embedding phase is $T(\frac{V}{\alpha^m}, \frac{E}{\beta^m})$. For the refinement phase, the time complexity can be divided into two parts, i.e. the GCN model training and the embedding inference applying the GCN model. The former has the complexity similar to the original GCN $O(k_1 * E_m)$ [5], where k_1 is a small constant related to embedding dimensionality and the number of training epoches. $T(\frac{V}{\alpha^m}, \frac{E}{\beta^m}) + O(k * E)$.

Algorithm 1 Multi-Level Algorithm for Graph Embedding

Input: A input graph $\mathcal{G}_0 = (V_0, E_0)$, # coarsening levels m, and a base embedding method $f(\cdot)$.
Output: Graph embeddings $\mathcal{E}_0$ on $\mathcal{G}_0$.

```
1: Coarsen $\mathcal{G}_0$ into $\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_m$ using proposed hybrid matching method.
2: Perform base embedding on the coarsest graph $\mathcal{G}_m$ (See Eq. 3).
3: Learn the weights $\Theta^{(k)}$ using the loss function in Eq. 7.
4: for $i = (m-1) \ldots 0$ do
5:     Compute the projected embeddings $\mathcal{E}_i^p$ on $\mathcal{G}_i$.
6:     Use Eq. 6 and Eq. 5 to compute refined embeddings $\mathcal{E}_i$.
7: Return graph embeddings $\mathcal{E}_0$ on $\mathcal{G}_0$.
```

3 EXPERIMENTS AND ANALYSIS

We evaluate the proposed MILE framework on three datasets mainly PPI, Youtube and Yelp Dataset. The baselines are DeepWalk [7], Line [9] and NetMF [8].

Dataset	# Nodes	# Edges	# Classes
PPI	10,312	333,983	39
YouTube	1,134,890	2,987,624	47
Yelp	8,938,630	39,821,123	22

Table 1: Dataset Information

Baseline-specific Settings: Baseline **DeepWalk**: length of random walks as 80, number of walks per node as 10, and context windows size as 10. Baseline **NETMF**: Window size to 10 and the rank h to 1024, and we lever the approximate version.

MILE-specific Settings: For the graph convolution network model, the self-loop weight λ is set to 0.05, the number of hidden layers l is 2, and tanh($\cdot$) is used as the activation function, the learning rate is set to 0.001 and the number of training epochs is 200. The Adam Optimizer is used for model training.

Evaluation Metrics: To evaluate the quality of the embeddings, we perform multi-label node classification [2, 7]. After the graph embeddings are learned for nodes (label is not used for this part), we run a 10-fold cross validation using the embeddings as features and report the average Micro-F1 and average Macro-F1.

3.1 MILE Framework Performance

We first evaluate the performance of our MILE framework when applied to different graph embedding methods. For PPI and Youtube dataset, we show the results of MILE under two settings of coarsening levels m in Table 2 and on our largest dataset Yelp we show results with varying number of coarsening levels and respective running times in Figure 3. Note that, none of the three graph embedding methods can successfully conduct graph embedding on Yelp within 60 hours on a modern machine with 28 cores and 128 GB RAM (NETMF run out of memory).

We make the following observations:

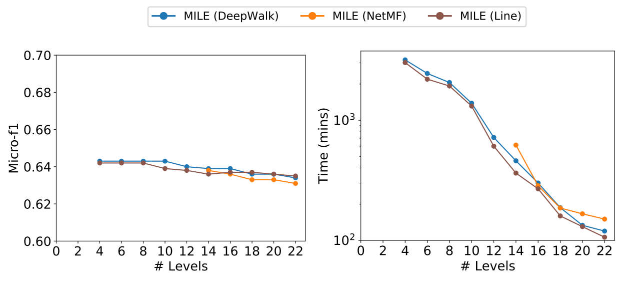
Method	Micro-F1	Macro-F1	Time (mins)
DeepWalk	23.0	18.6	2.42
MILE (DeepWalk, $m = 1$)	25.6(11.3%↑)	20.4(9.7%↑)	1.22(2.0×)
MILE (DeepWalk, $m = 2$)	25.5(10.9%↑)	20.7(11.3%↑)	0.67(3.6×)
Line	25.0	19.5	2.27
MILE (Line, $m = 1$)	25.8 (3.2%↑)	19.8 (1.5%↑)	1.22 (1.9×)
MILE (Line, $m = 2$)	24.7 (-1.2%↓)	19.0 (-2.6%↓)	0.68 (3.3×)
NetMF	24.6	20.1	0.65
MILE (NetMF, $m = 1$)	26.9(9.3%↑)	21.6(7.5%↑)	0.27(2.5×)
MILE (NetMF, $m = 2$)	26.7(8.5%↑)	21.1(5.0%↑)	0.17(3.9×)

(a) PPI Dataset

Method	Micro-F1	Macro-F1	Time (mins)
DeepWalk	45.2	34.7	604.83
MILE (DeepWalk, $m = 6$)	46.1(2.0%↑)	38.5(11.0%↑)	55.20(11.0×)
MILE (DeepWalk, $m = 8$)	44.3(-2.0%↓)	35.3(1.7%↑)	37.35(16.2×)
Line	46.0	35.0	583.37
MILE (Line, $m = 6$)	46.2 (0.4%↑)	36.2 (3.4%↑)	53.97 (10.81×)
MILE (Line, $m = 8$)	44.4 (-3.5%↓)	35.7 (2.0%↑)	33.41 (17.46×)
NetMF	N/A	N/A	> 574.75
MILE (NetMF, $m = 6$)	40.9	27.8	35.22(>16.3×)
MILE (NetMF, $m = 8$)	39.2	25.5	19.22(>29.9×)

(b) YouTube Dataset

Table 2: Performance of MILE compared to the original embedding methods. DeepWalk, Line and NetMF denotes the original method. We set the number of coarsening levels m to 1 and 2 for PPI, while choosing 6 and 8 for YouTube (due to its larger scale). The Micro-F1 and Macro-F1 are in $\underline{10^{-2}}$ scale while the column Time shows the running time in minutes. The numbers within the parenthesis by the reported Micro-F1 and Macro-F1 scores are the *relative percentage* of change compared to the original method, e.g., MILE (DeepWalk, $m = 1$) vs. DeepWalk. "↑" and "↓" respectively indicate improvement and decline. Numbers along with "×" is the speedup compared to the original method. "N/A" indicates the method runs out of 128 GB memory and we show the amount of running time spent when it happens.



(a) Micro-F1 **(b) Running Time**

Figure 3: Running MILE on Yelp dataset.

- **MILE is scalable**. MILE greatly boosts the speed of the embedding methods. On PPI and YouTube, we are able to achieve speedup ranging from 1.9× to 29.9× while often improving qualitative performance. On YouTube where the coarsening level is 6 and 8, we observe more than 11× speedup for DeepWalk, more than 10× speedup for Line and more than 16× speedup for NetMF. On youtube, original NetMF runs out of memory within 9.5 hours while MILE (NetMF) only takes around 35 minutes ($m = 6$) or 20 minutes ($m = 8$). On Yelp, MILE reduces the running time of DeepWalk from 53 hours (coarsening level 4) to 2 hours (coarsening level 22) while reducing the Micro-F1 score just by 1% (from 0.643 to 0.634).
- **MILE improves quality**. For the smaller coarsening levels across all the datasets and methods, MILE-enhanced embeddings often offer a qualitative improvement over the original embedding method. Even with the higher number of coarsening level ($m = 2$ for PPI; $m = 8$ for YouTube), MILE in addition to being much faster can still improve, qualitatively, over the original methods on all datasets, e.g., MILE(NetMF, $m = 2$) $\gg$ NETMF on PPI. We observe speedup on MILE(Line) with $m = 2$ for PPI; $m = 8$ for YouTube too, however this speedup comes up with an additional cost to quality of embeddings. On Yelp, we observe that MILE significantly reduces the running time while the Micro-F1 score remains almost unchanged. There is no change in the Micro-F1 score from coarsening level 4 to 10, where the running time is improved by a factor of two. We conjecture the observed improvement on quality is because the embeddings begin to rely on a more holistic view of the graph.
- **MILE supports multiple embedding strategies.** For DeepWalk, we observe consistent improvements in scalability (up to 11-fold on the YouTube dataset) as well as quality using MILE with a single level of coarsening (or $m = 6$ for YouTube). However, when the coarsening level is increased, the additional speedup afforded (up to 17-fold) comes at a mixed cost to quality. This observation is consistent for Line. For NetMF, we observe that the method runs out of memory for YouTube and Yelp dataset on a modern machine with 28 cores and 128 GB RAM, however MILE (NetMF) can still generate embeddings in 20 mins for YouTube ($m = 6$) and less than 3 hrs for Yelp ($m = 22$).

4 CONCLUSION

In this work, we propose a novel multi-level embedding (MILE) framework to scale up graph embedding techniques, without modifying them. Our framework incorporates existing embedding techniques as black boxes, and significantly improves the scalability of extant methods by reducing both the running time and memory consumption. Additionally, MILE also provides a lift in the quality of node embeddings in most of the cases. A fundamental contribution of MILE is its ability to learn a refinement strategy that depends on both the underlying graph properties and the embedding method in use. In the future, we plan to generalize our framework for information-rich graphs, such as heterogeneous information networks and attributed graphs.

REFERENCES

[1] S. Cao, W. Lu, and Q. Xu. Grarep: Learning graph representations with global structural information. In *CIKM'15*.
[2] A. Grover and J. Leskovec. Node2vec: Scalable feature learning for networks. In *KDD'16*.
[3] X. Huang, J. Li, and X. Hu. Accelerated attributed network embedding. In *SDM'17*.
[4] G. Karypis and V. Kumar. Multilevelk-way partitioning scheme for irregular graphs. In *JPDC'98*.
[5] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In *ICLR'17*.
[6] J. Liang, S. Gurukar, and S. Parthasarathy. Mile: A multi-level framework for scalable graph embedding. In *arXiv preprint arXiv:1802.09612*, 2018.
[7] B. Perozzi, R. Al-Rfou, and S. Skiena. Deepwalk: Online learning of social representations. In *KDD'14*.
[8] J. Qiu, Y. Dong, H. Ma, J. Li, K. Wang, and J. Tang. Network embedding as matrix factorization: Unifyingdeepwalk, line, pte, and node2vec. In *WSDM'18*.
[9] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei. Line: Large-scale information network embedding. In *WWW'15*.
[10] C. Yang, M. Sun, Z. Liu, and C. Tu. Fast network embedding enhancement via high order proximity approximation. In *IJCAI'17*.