GRAPHZOOM: A MULTI-LEVEL SPECTRAL APPROACH FOR ACCURATE AND SCALABLE GRAPH EMBEDDING

Anonymous authors
Paper under double-blind review

ABSTRACT

Graph embedding techniques have been increasingly deployed in a multitude of different applications that involve learning on non-Euclidean data. However, existing graph embedding models either fail to incorporate node attribute information during training or suffer from node attribute noise, which compromises the accuracy. Moreover, very few of them scale to large graphs due to their high computational complexity and memory usage. In this paper we propose GraphZoom, a multi-level framework for improving both accuracy and scalability of unsupervised graph embedding algorithms. GraphZoom first performs graph fusion to generate a new graph that effectively encodes the topology of the original graph and the node attribute information. This fused graph is then repeatedly coarsened into a much smaller graph by merging nodes with high spectral similarities. GraphZoom allows any existing embedding methods to be applied to the coarsened graph, before it progressively refine the embeddings obtained at the coarsest level to increasingly finer graphs. We have evaluated our approach on a number of popular graph datasets for both transductive and inductive tasks. Our experiments show that GraphZoom increases the classification accuracy and significantly reduces the run time compared to state-of-the-art unsupervised embedding methods.

1 INTRODUCTION

Recent years have seen a surge of interest in graph embedding, which aims to encode nodes, edges, or (sub)graphs into low dimensional vectors that maximally preserve graph structural information. Graph embedding techniques have shown promising results for various applications such as vertex classification, link prediction, and community detection (Zhou et al., 2018); (Cai et al., 2018); (Goyal & Ferrara, 2018). However, current graph embedding methods have several drawbacks. On the one hand, random-walk based embedding algorithms, such as DeepWalk (Perozzi et al., 2014) and node2vec (Grover & Leskovec, 2016), attempt to embed a graph based on its topology without incorporating node attribute information, which limits their embedding power. Later, graph convolutional networks (GCN) are developed with the basic notion that node embeddings should be smooth over the graph (Kipf & Welling, 2016). While GCN leverages both topology and node attribute information for simplified graph convolution in each layer, it may suffer from high frequency noise in the initial node features, which compromises the embedding quality (Maehara, 2019). On the other hand, few embedding algorithms can scale well to large graphs with millions of nodes due to their high computation and storage cost (Zhang et al., 2018a). For example, graph neural networks (GNNs) such as GraphSAGE (Hamilton et al., 2017) collectively aggregate feature information from the neighborhood. When stacking multiple GNN layers, the final embedding vector of a node involves the computation of a large number of intermediate embeddings from its neighbors. This will not only drastically increase the number of computations among nodes but also lead to high memory usage for storing the intermediate results.

In literature, increasing the accuracy and improving the scalability of graph embedding methods are largely viewed as two orthogonal problems. Hence most research efforts are devoted to addressing only one of the problems. For instance, Chen et al. (2018) and Fu et al. (2019) proposed multi-level methods to obtain high-quality embeddings by training unsupervised models at every level; but their techniques do not improve scalability due to the additional training overhead. Liang et al. (2018) developed a heuristic algorithm to coarsen the graph by merging nodes with similar local structures. They use GCN to refine the embedding results on the coarsened graphs, which not only is time-consuming to train but may also degrade accuracy when multiple GCN layers are stacked together.

More recently, Akbas & Aktas (2019) proposed a similar strategy to coarsen the graph, where certain properties of the graph structure are preserved. However, this work lacks proper refinement methods to improve the embedding quality.

In this paper we propose GraphZoom, a multi-level spectral approach to enhancing the quality and scalability of unsupervised graph embedding methods. Specifically, GraphZoom consists of four kernels: (1) graph fusion, (2) spectral graph coarsening, (3) graph embedding, and (4) embedding refinement. More concretely, graph fusion first converts the node feature matrix into a feature graph and then fuses it with the original topology graph. The fused graph provides richer information to the ensuing graph embedding step to achieve a higher accuracy. Spectral graph coarsening produces a series of successively coarsened graphs by merging nodes based on their spectral similarities. We show that our coarsening algorithm can efficiently and effectively retain the first few eigenvectors of the graph Laplacian matrix, which is critical for preserving the key graph structures. During the graph embedding step, any of the existing unsupervised graph embedding techniques can be applied to obtain node embeddings for the graph at the coarsest level. [1] Embedding refinement is then employed to refine the embeddings back to the original graph by applying a proper graph filter to ensure embeddings are smoothed over the graph.

We validate the proposed GraphZoom framework on three transductive benchmarks: Cora, Citeseer and Pubmed citation networks as well as two inductive dataset: PPI and Reddit for vertex classification task. We further test on friendster dataset which contains 8 million nodes and 400 million edges to show the scalability of GraphZoom. Our experiments show that GraphZoom can improve the classification accuracy over all baseline embedding methods for both transductive and inductive tasks. Our main technical contributions are summarized as follows:

- **GraphZoom generates high-quality embeddings.** We propose novel algorithms to encode graph structures and node attribute information in a fused graph and exploit graph filtering during refinement to remove high frequency noise. This results in an increase of the embedding accuracy over the prior arts by up to 19.4%.
- **GraphZoom improves scalability.** Our approach can significantly reduce the embedding run time by effectively coarsening the graph without losing the key spectral properties. Experiments show that GraphZoom can accelerate the entire embedding process by up to 40.8x while producing a similar or better accuracy than state-of-the-art techniques.
- **GraphZoom is highly composable.** Our framework is agnostic to underlying graph embedding techniques. Any of the existing unsupervised embedding methods, either transductive or inductive, can be incorporated by GraphZoom in a plug-and-play manner.

2 Related Work

GraphZoom draws inspiration from multi-level graph embedding and graph filtering to boost the performance and speed of unsupervised embedding methods.

Multi-level graph embedding. Multi-level graph embedding attempts to coarsen the graph in a series of levels where graph embedding techniques can be applied on those coarsened graphs with decreasing size. Chen et al. (2018); Lin et al. (2019) coarsen the graph into several levels and then perform embedding on the hierarchy of graphs from the coarsest to the original one. Fu et al. (2019) adopt a similar idea by hierarchical sampling original graph into multi-level graphs whose embedding vectors are concatenated to obtain the final node embeddings of the original graph. Both of these works only focus on improving embedding quality without improving the scalability. Later, Zhang et al. (2018b); Akbas & Aktas (2019) attempt to improve graph embedding scalability by only embedding on the coarsest graph. However, their approaches lack proper refinement methods to generate high-quality embeddings of the original graph. Liang et al. (2018) propose MILE, which only trains the coarsest graph to obtain coarse embeddings, and leverages GCN as embeddings refinement method to improve embedding quality. Nevertheless, MILE requires to train a GCN model which is very time consuming for large graphs and cannot support inductive embedding models due to the transductive property of GCN.

Graph filtering. Graph filters are direct analogs of classical filters in signal processing field, but intended for signals defined on graphs. Shuman et al. (2013) defined graph filters in both vertex and

[1] In this work, we do not attempt to preserve node label information in the coarsened graph. Nonetheless, we believe that our approach can be extended to support supervised embedding models such as GAT (Gulcehre et al., 2019) and PPNP (Klicpera et al., 2019), as briefly discussed in Section 5.

spectral domains, and applies graph filter in image denoising and reconstruction tasks. Recently, Maehara (2019) showed the fundamental link between graph embedding and filtering by proving that GCN model implicitly exploits graph filter to remove high frequency noise from the node feature matrix; a filter neural network (gfNN) is then proposed to derive a stronger graph filter to improve the embedding results. Li et al. (2019) further derived two generalized graph filters and apply them on graph embedding models to improve their embedding quality for various classification tasks.

3 GraphZoom Framework

Figure 1 shows the proposed GraphZoom framework which consists of four key phases: Phase (1) is graph fusion, which constructs a weighted graph that fuses the information of both the graph topology and node attributes; In Phase (2), a spectral graph coarsening process is applied to form a hierarchy of coarsened fused graphs with decreasing size; In Phase (3), any of the prior graph embedding methods can be applied to the fused graph at the coarsest level; In Phase (4), the embedding vectors obtained at the coarsest level are mapped onto a finer graph using the mapping operators determined during the coarsening phase. This is followed by a refinement (smoothing) procedure; by iteratively applying Phase (4) to increasingly finer graphs, the embedding vectors for the original graph can be eventually obtained. In the rest of this section, we describe each of these four phases in more detail.

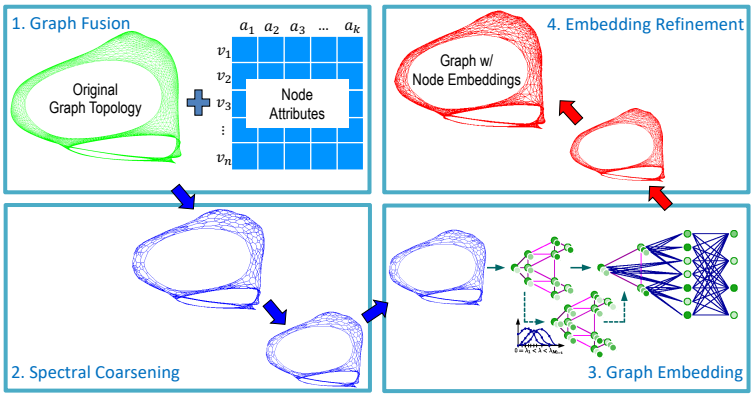


Figure 1: Overview of the GraphZoom framework.

3.1 Phase 1: Graph Fusion

Graph fusion aims to construct a weighted graph that has the same number of nodes as the original graph but potentially different set of edges (weights) that encapsulate the original graph topology as well as node attribute information. Specifically, given an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with N nodes, its adjacency matrix $A_{topo} \in \mathbb{R}^{N \times N}$ and its node attribute (feature) matrix $X \in \mathbb{R}^{N \times K}$, where K corresponds to the dimension of node attribute vector, graph fusion can be interpreted as a function $f(\cdot)$ that outputs a weighted graph $\mathcal{G}_{fusion} = (\mathcal{V}, \mathcal{E}_{fusion})$ represented by its adjacency matrix $A_{fusion} \in \mathbb{R}^{N \times N}$, namely, $A_{fusion} = f(A_{topo}, X)$.

Graph fusion first converts the initial attribute matrix X into a weighted node attribute graph $\mathcal{G}_{feat} = (\mathcal{V}, \mathcal{E}_{feat})$ by generating a k-nearest-neighbor (kNN) graph based on the l^2-$norm$ distance between the attribute vectors of each node pair. Note that a straightforward implementation requires comparing all possible node pairs and then selecting top-k nearest neighbors. However, such a naïve approach has a worst-case time complexity of $O(N^2)$, which certainly does not scale to large graphs. To allow constructing the attribute graph in linear time, we leverage our $O(|\mathcal{E}|)$ complexity spectral graph coarsening scheme described with details in Section 3.2. More specifically, our approach starts with coarsening the original graph $\mathcal{G}$ to obtain a substantially reduced graph that has much fewer nodes. Note that such a procedure is very similar to spectral graph clustering, which aims to group nodes into clusters of high conductance (Peng et al., 2015). Once such node clusters are formed through spectral coarsening, selecting the top-k nearest neighbors within each cluster can be accomplished in $O(M^2)$, where M is the averaged node count within the same cluster. Since we have roughly N/M clusters, the total run time for constructing the ap-

proximate kNN graph becomes $O(MN)$. When a proper coarsening ratio ($M \ll N$) is chosen, say $M = 50$, the overall run time complexity will become almost linear. For each edge in the attribute graph, we assign its weight $w_{i,j}$ according to the cosine similarity of two nodes' attribute vectors: $w_{i,j} = (X_{i,:} \cdot X_{j,:})/(\|X_{i,:}\|\|X_{j,:}\|)$, where $X_{i,:}$ and $X_{j,:}$ are the attribute vectors of node i and j. Finally, we can construct the fused graph by combining the topological graph and the attribute graph: $A_{fusion} = A_{topo} + \beta A_{feat}$, where β allows us to balance the graph topological and node attribute information in the fusion process. The fused graph will enable the underlying graph embedding model to utilize both graph topological and node attribute information, and thus can be fed into any downstream graph embedding procedures to further improve embedding quality.

3.2 Phase 2: Spectral Coarsening

Graph coarsening via global spectral embedding. To reduce the size of the original graph while preserving important spectral properties (e.g., the first few eigenvalues and eigenvectors of the graph Laplacian matrix [2]), a straightforward way is to first embed the graph into a k-dimensional space using the first k eigenvectors of the graph Laplacian matrix, which is also known as the spectral graph embedding technique (Belkin & Niyogi, 2003; Peng et al., 2015). Next, the graph nodes that are close to each other in the low-dimensional embedding space can be aggregated to form the coarse-level nodes and subsequently the reduced graph. However, it will be very costly to calculate the eigenvectors of the original graph Laplacian, especially for very large graphs.

Graph coarsening via local spectral embedding. In this work, we leverage an efficient yet effective local spectral embedding scheme to identify node clusters based on emerging graph signal processing techniques (Shuman et al., 2013). There are obvious analogies between the traditional signal processing (Fourier analysis) and graph signal processing: (1) The signals at different time points in classical Fourier analysis correspond to the signals at different nodes in an undirected graph; (2) The more slowly oscillating functions in time domain correspond to the graph Laplacian eigenvectors associated with lower eigenvalues or the more slowly varying (smoother) components across the graph. Instead of directly using the first few eigenvectors of the original graph Laplacian, we apply the simple smoothing (low-pass graph filtering) function to k random vectors to obtain smoothed vectors for k-dimensional graph embedding, which can be achieved in linear time.

Consider a random vector (graph signal) x that can be expressed with a linear combination of eigenvectors $\boldsymbol{u}$ of the graph Laplacian. Low-pass graph filters can be adopted to quickly filter out the "high-frequency" components of the random graph signal or the eigenvectors corresponding to high eigenvalues of the graph Laplacian. By applying the smoothing function on $\boldsymbol{x}$, a smoothed vector $\tilde{\boldsymbol{x}}$ can be obtained, which can be considered as a linear combination of the first few eigenvectors:

$$\boldsymbol{x} = \Sigma_{i=1}^{N} \alpha_i \boldsymbol{u}_i \quad \xRightarrow{\text{smoothing}} \quad \tilde{\boldsymbol{x}} = \Sigma_{i=1}^{n} \tilde{\alpha}_i \boldsymbol{u}_i \ , \ \ n \ll N \tag{1}$$

More specifically, we apply a few (e.g. five to ten) Gauss-Seidel iterations for solving the linear system of equations $L_G x^{(i)} = 0$ to a set of t initial random vectors $T = (x^{(1)}, \ldots, x^{(t)})$ that are orthogonal to the all-one vector $\mathbf{1}$ satisfying $\mathbf{1}^\top x^{(i)} = 0$, and L_G is the Laplacian matrix of graph $\mathcal{G}$ or $\mathcal{G}_{fusion}$. Based on the smoothed vectors in T, each node is embedded into a t-dimensional space such that nodes p and q are considered spectrally similar if their low-dimensional embedding vectors $x_p \in \mathbb{R}^t$ and $x_q \in \mathbb{R}^t$ are highly correlated. Here the node distance is measured by the spectral node affinity $a_{p,q}$ for neighboring nodes p and q (Livne & Brandt, 2012; Chen & Safro, 2011):

$$a_{p,q} = \frac{(\|(\boldsymbol{T}_{p,:}, \boldsymbol{T}_{q,:})\|)^2}{(\boldsymbol{T}_{p,:}, \boldsymbol{T}_{p,:})(\boldsymbol{T}_{q,:}, \boldsymbol{T}_{q,:})}, \quad (\boldsymbol{T}_{p,:}, \boldsymbol{T}_{q,:}) = \Sigma_{k=1}^{t} (\boldsymbol{x}_p^{(k)} \cdot \boldsymbol{x}_q^{(k)}), \tag{2}$$

Once the node aggregation schemes are determined, the graph mapping operators on each level $(H_0^1, H_1^2, \cdots, H_{m-1}^m)$ can be obtained and leveraged for constructing a series of spectrally-reduced graphs $\mathcal{G}_1, \mathcal{G}_2, \cdots, \mathcal{G}_m$, where $\mathcal{G}_0$ ($\mathcal{G}_{fusion}$) is the original fused graph, and $\mathcal{V}_0 = N > \mathcal{V}_1 > \cdots > \mathcal{V}_m$. For example, the coarser graph Laplacian $L_{G_{i+1}}$ can be computed by

$$\boldsymbol{L}_{G_{i+1}} = \boldsymbol{H}_i^{i+1} \boldsymbol{L}_{G_i} \boldsymbol{H}_{i+1}^i, \ \ \boldsymbol{H}_{i+1}^i = (\boldsymbol{H}_i^{i+1})^T \tag{3}$$

We emphasize that the aggregation scheme based on the above spectral node affinity calculations will have a (linear) complexity of $O(|\mathcal{E}_{fusion}|)$ and thus allow preserving the spectral (global or structural) properties of the original graph in a highly efficient and effective way. As suggested in (Zhao & Feng, 2019; Loukas, 2019), a spectral sparsification procedure can be applied to effectively control densities of coarse level graphs. In this work, a similarity-aware spectral sparsification tool "GRASS" (Feng, 2018) has been adopted for achieving a desired graph sparsity at the coarsest level.

[2] Laplacian matrix L is defined as $L = D - A$, where D is degree matrix and A is adjacency matrix.

3.3 PHASE 3: GRAPH EMBEDDING

Embedding the Coarsest Graph. Once the coarsest graph $\mathcal{G}_m$ is constructed, node embeddings E_m on $\mathcal{G}_m$ can be obtained by $E_m = l(\mathcal{G}_m)$, where $l(\cdot)$ can be any unsupervised embedding methods.

3.4 PHASE 4: EMBEDDING REFINEMENT

Once the base node embedding results are available, we can easily project the node embeddings from graph $\mathcal{G}_{i+1}$ to the fine-grained graph $\mathcal{G}_i$ with the corresponding projection operator H_{i+1}^i:

$$\hat{\boldsymbol{E}}_i = \boldsymbol{H}_{i+1}^i \boldsymbol{E}_{i+1} \tag{4}$$

Due to the property of the projection operator, embedding of the node in coarse-grained graph will be directly copied to the nodes of the same aggregation set in the fine-grained graph. In this case, spectrally-similar nodes in the fine-grained graph will have the same embedding results if they are aggregated into a single node during the coarsening phase.

To further improve the quality of the mapped embeddings, we apply a local refinement process motivated by Tikhonov regularization to smooth the node embeddings over the graph by minimizing the following objective:

$$\min_{\boldsymbol{E}_i} \{ \left\| \boldsymbol{E}_i - \hat{\boldsymbol{E}}_i \right\|_2^2 + Tr(\boldsymbol{E}_i{}^{\mathsf{T}} \boldsymbol{L}_i \boldsymbol{E}_i) \}, \tag{5}$$

where L_i and E_i are the normalized Laplacian matrix and mapped embedding matrix of the graph at the i-th coarsening level, respectively. The refined embedding matrix $\tilde{\boldsymbol{E}}_i$ is obtained by solving Eq. (5), whose first term enforces the refined embeddings to agree with mapped embeddings while the second term employs Laplacian smoothing to smooth $\tilde{\boldsymbol{E}}_i$ over the graph. By taking the derivative of the objective function in Eq. (5) and setting it to zero, we have:

$$\boldsymbol{E}_i = (\boldsymbol{I} + \boldsymbol{L}_i)^{-1} \hat{\boldsymbol{E}}_i, \tag{6}$$

where $\boldsymbol{I}$ is the identity matrix. However, obtaining refined embeddings in this way is very time consuming since it involves matrix inversion whose time complexity is $O(N^3)$. Instead, we exploit a more efficient graph filter to smooth the embeddings. Let the term $(I + L)^{-1}$ denoted by $h(L)$, then its corresponding graph filter in spectral domain is $h(\lambda) = (1 + \lambda)^{-1}$. To avoid the inversion term, we approximate $h(\lambda)$ by its first-order Taylor expansion, namely, $\tilde{h}(\lambda) = 1 - \lambda$. We then generalize $\tilde{h}(\lambda)$ to $\tilde{h}_k(\lambda) = (1 - \lambda)^k$, where k controls the power of graph filter. After transforming $\tilde{h}_k(\lambda)$ into spatial domain, we have: $\tilde{h}_k(L) = (I - L)^k = (D^{-\frac{1}{2}} A D^{-\frac{1}{2}})^k$, where A is the adjacency matrix and D is the degree matrix. It can be proved that adding a proper self-loop for every node in the graph can enable $\tilde{h}_k(L)$ to more effectively filter out high-frequency noise components (Maehara, 2019) (more details are available in Appendix G). Thus, we modify the adjacency matrix as $\tilde{A} = A + \sigma I$, where σ is a small value to ensure every node has its own self-loop. Finally, the low-pass graph filter can be utilized to smooth the mapped embedding matrix, as shown in (7).

$$\boldsymbol{E}_i = (\tilde{\boldsymbol{D}}_i{}^{-\frac{1}{2}} \tilde{\boldsymbol{A}}_i \tilde{\boldsymbol{D}}_i{}^{-\frac{1}{2}})^k \hat{\boldsymbol{E}}_i = (\tilde{\boldsymbol{D}}_i{}^{-\frac{1}{2}} \tilde{\boldsymbol{A}}_i \tilde{\boldsymbol{D}}_i{}^{-\frac{1}{2}})^k \boldsymbol{H}_{i+1}^i \boldsymbol{E}_{i+1} \tag{7}$$

We iteratively apply Eq. (7) to obtain the embeddings of the original graph (i.e., E_0). Note that our refinement stage does not involve training and can be simply considered as several (sparse) matrix multiplications, which can be computed efficiently.

4 EXPERIMENTS

We have performed comparative evaluation of GraphZoom framework against several existing state-of-the-art unsupervised graph embedding techniques and multi-level embedding frameworks on five standard graph-based dataset (transductive as well as inductive). In addition, we evaluate the scalability of GraphZoom on Friendster dataset that contains 8 million nodes and 400 million edges. Finally, we further analyze GraphZoom kernels separately to show their effectiveness.

4.1 EXPERIMENTAL SETUP

Datasets. The statistics of datasets used in our experiments are demonstrated in Table 1. We use Cora, Citeseer, Pubmed, Friendster for transductive task and PPI, Reddit for inductive task. We choose the same training and testing size used in Kipf & Welling (2016); Hamilton et al. (2017).

Table 1: Statistics of datasets used in our experiments

Dataset	Type	Task	Nodes	Edges	Classes	Features
Cora	Citation network	Transductive	2,708	5,429	7	1,433
Citeseer	Citation network	Transductive	3,327	4,732	6	3,703
Pubmed	Citation network	Transductive	19,717	44,338	3	500
PPI	Molecular network	Inductive	14,755	222,055	121	50
Reddit	Social network	Inductive	232,965	57,307,946	210	5,414
Friendster	Social network	Transductive	7,944,949	446,673,688	5,000	N/A

Transductive baseline models. Many existing graph embedding techniques are essentially transductive learning methods, which require all nodes in the graph be present during training, and their embedding models have to be retrained whenever a new node is added. We compare GraphZoom with transductive models DeepWalk, node2vec, and Deep Graph Infomax (DGI) (Velikovi et al., 2019) [3] that have shown the state-of-the-art unsupervised embedding results on the datasets used in our experiments. We further compare GraphZoom with two multi-level frameworks: HARP (Chen et al., 2018) and MILE (Liang et al., 2018), which have shown improvement upon DeepWalk and node2vec in either embedding quality or scalability.

Inductive baseline models. Inductive graph embedding models can be trained without seeing the whole graph structure and their trained models can be applied on new nodes added to graph. To show GraphZoom can also enhance inductive learning, we compare it against GraphSAGE (Hamilton et al., 2017) using four different aggregation functions.

More details of datasets and baselines are available in Appendix A and B. We optimize hyper-parameters of DeepWalk, node2vec, DGI (Velikovi et al., 2019) and GraphSAGE on original datasets as embedding baseline, and then we choose the same hyper-parameters to embed coarsened graph in HARP, MILE and our GraphZoom framework. We run all the experiments on a machine running Linux with an Intel Xeon Gold 6242 CPU (32 cores, 2.40GHz) and 384 GB of RAM.

4.2 Performance and Scalability of GraphZoom

Table 2: Summary of results in terms of mean classification accuracy and CPU time for transductive task, on Cora, Citeseer and Pubmed datasets — DW, N2V, and GZoom denote DeepWalk, node2vec, and GraphZoom, respectively; l means the graph coarsening level.

	Cora		Citeseer		Pubmed	
Method	Accuracy(%)	Time(secs)	Accuracy(%)	Time(secs)	Accuracy(%)	Time(mins)
DeepWalk	71.4	97.8	47.0	120.0	69.9	14.1
HARP(DW)	71.3	296.7(0.3×)	43.2	272.4(0.4×)	70.6	33.9(0.4×)
MILE(DW, l=1)	71.9	68.7(1.4×)	46.5	53.7(2.2×)	69.6	7.0(2.0×)
MILE(DW, l=2)	71.3	30.9(3.2×)	47.3	22.5(5.3×)	66.7	4.4(2.3×)
MILE(DW, l=3)	70.6	15.9(6.1×)	47.1	9.9(12.1×)	64.5	2.5(5.8×)
GZoom(DW, l=1)	76.9	39.6(2.5×)	49.7	19.6(2.1×)	75.3	4.0(3.6×)
GZoom(DW, l=2)	**77.3**	15.6(6.3×)	**50.8**	6.7(6.0×)	75.9	1.7(8.3×)
GZoom(DW, l=3)	75.1	**2.4(40.8×)**	49.5	**1.3(30.8×)**	**77.2**	**0.6(23.5×)**
node2vec	71.5	119.7	45.8	126.9	71.3	15.6
HARP(N2V)	72.3	171.0(0.7×)	44.8	174.3(0.7×)	70.1	46.1(0.3×)
MILE(N2V, l=1)	72.1	57.3(2.1×)	46.1	60.9(2.1×)	70.8	7.3(2.1×)
MILE(N2V, l=2)	71.8	30.0(4.0×)	45.7	28.8(4.4×)	67.3	4.3(3.6×)
MILE(N2V, l=3)	68.5	16.5(7.2×)	45.2	15.6(8.1×)	61.8	1.8(8.0×)
GZoom(N2V, l=1)	**77.3**	43.5(2.8×)	**54.7**	38.1(3.3×)	77.0	3.0(5.2×)
GZoom(N2V, l=2)	77.0	13.5(8.9×)	51.7	15.3(8.3×)	**77.8**	1.5(10.4×)
GZoom(N2V, l=3)	75.3	**3.0(39.9×)**	50.7	**4.5(28.2×)**	77.4	**0.4(39.0×)**
DGI(Random-Init)	69.3	-	**61.9**	-	69.6	-
DGI	82.3	89.74	**71.8**	94.63	76.8	23.68
GZoom(DGI, l=1)	**83.9**	27.35(3.3×)	71.1	29.31(3.2×)	77.1	9.61(2.5×)
GZoom(DGI, l=2)	83.8	15.19(5.9×)	70.8	17.91(5.3×)	**77.6**	4.36(5.4×)
GZoom(DGI, l=3)	83.5	**8.03(11.2×)**	70.7	**9.60(9.8×)**	76.9	**2.11(11.2×)**

[3] Since the authors of DGI only release transductive version (implementation of DGI can be found here: https://github.com/PetarV-/DGI), we compare GraphZoom with it for transductive task.

Since HARP and MILE only support transductive learning, we compare them with GraphZoom for transductive task with DeepWalk, node2vec, and DGI (Velikovi et al., 2019) as embedding kernels. For inductive task, we compare GraphZoom with GraphSAGE using four different aggregation functions. Results of both transductive learning task and inductive learning task are summarized in Tables 2 and 3, respectively.

We report the mean classification accuracy for transductive task and micro-averaged F1 score for inductive task as well as CPU time after 10 runs for all the baselines and GraphZoom. We measure the CPU time for graph embedding as the total run time of DeepWalk, node2vec, DGI, and GraphSAGE. We use the sum of CPU time for graph coarsening, graph embedding, and embedding refinement as total run time of HARP and MILE. Similarly, we sum up the CPU time for graph fusion, graph coarsening, graph embedding, and embedding refinement as total run time of GraphZoom. We also perform fine-tuning on the hyper-parameters. For both DeepWalk and node2vec, we use 10 walks with a walk length of 80, a window size of 10, and an embedding dimension of 128; we further set $p = 1$ and $q = 0.5$ in node2vec. For DGI, we choose early stopping strategy with a learning rate of 0.001, an embedding dimension of 512. For GraphSAGE, we train a two-layer model for one epoch, with a learning rate of 0.00001, an embedding dimension of 128, and a batch size of 256.

Table 3: Summary of results in terms of micro-averaged F1 score and CPU time for inductive task, on PPI and Reddit datasets — The baselines are GraphSAGE with four different aggregation functions. GZoom and GSAGE denote GraphZoom and GraphSAGE, respectively; l means the graph coarsening level.

	PPI		Reddit	
Method	Micro-F1	Time(mins)	Micro-F1	Time(hours)
GraphSAGE-GCN	0.601	9.6	0.908	10.1
GZoom(GSAGE-GCN, l=1)	**0.621**	4.8(2.0×)	**0.923**	3.4(3.0×)
GZoom(GSAGE-GCN, l=2)	0.612	**1.8(5.2×)**	0.917	**1.6(6.3×)**
GraphSAGE-mean	0.598	11.1	0.897	8.1
GZoom(GSAGE-mean, l=1)	0.614	5.2(2.2×)	**0.925**	2.6(3.1×)
GZoom(GSAGE-mean, l=2)	**0.617**	**1.8(6.2×)**	0.919	**1.2(6.8×)**
GraphSAGE-LSTM	0.596	387.3	0.907	92.2
GZoom(GSAGE-LSTM, l=1)	0.614	151.8(2.6×)	**0.920**	39.8(2.3×)
GZoom(GSAGE-LSTM, l=2)	**0.615**	**52.5(7.4×)**	0.917	**14.5(6.4×)**
GraphSAGE-pool	0.602	144.9	0.892	84.3
GZoom(GSAGE-pool, l=1)	0.611	66.0(2.2×)	**0.921**	27.0(3.1×)
GZoom(GSAGE-pool, l=2)	**0.614**	**23.4(6.2×)**	0.912	**12.4(6.8×)**

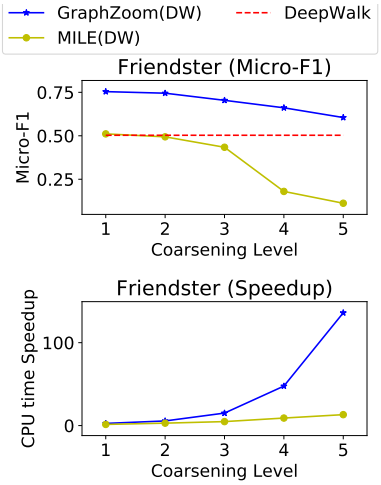


Figure 2: Comparisons of GraphZoom and MILE on Friendster dataset.

Comparing GraphZoom with baseline embedding methods. We show the results of GraphZoom with coarsening level varying 1 to 3 for transductive learning and 1 to 2 for inductive learning. Results with larger coarsening level are available in Figure 3 (blue curve) and the Appendix I. Our results demonstrate that GraphZoom is agnostic to underlying embedding methods and capable of boosting the accuracy and speed of state-of-the-art unsupervised embedding methods on various datasets. More specifically, for transductive learning task, GraphZoom improves classification accuracy upon both DeepWalk and node2vec by a margin of 8.3%, 19.4%, and 10.4% on Cora, Citeseer, and Pubmed, respectively, while achieving up to 40.8x run-time reduction. In regards to comparing with DGI, GraphZoom achieves comparable or better accuracy with speedup up to 11.2×. Similarly, GraphZoom outperforms all the baselines by a margin of 3.4% and 3.3% on PPI and Reddit for inductive learning task, respectively, with speedup up to 7.6x. Our results indicate that reducing graph size while properly retaining the key spectral properties of graph Laplacian and smoothing embeddings will not only boost the embedding speed but also lead to high embedding quality.

Comparing GraphZoom with multi-level frameworks. As shown in Table 2, HARP only slightly improves and sometimes even worsens the classification accuracy while significantly increasing the CPU time. Although MILE improves both accuracy and CPU time compared to baseline embedding methods in some cases, the performance of MILE becomes worse with increasing coarsening levels (e.g., the classification accuracy of MILE drops from 0.708 to 0.618 on Pubmed dataset with node2vec as the embedding kernel). GraphZoom achieves a better accuracy and speedup compared to MILE with the same coarsening level across all datasets. Moreover, when increasing coarsening levels, namely, decreasing number of nodes on the coarsened graph, GraphZoom still produces comparable or even a better embedding accuracy with much shorter CPU times. This further confirms

GraphZoom can retain the key graph structure information to be utilized by underlying embedding models to generate high-quality node embeddings.

GraphZoom for large graph embedding. To show GraphZoom can significantly improve performance and scalability of underlying embedding model on large graph, we test GraphZoom and MILE on Friendster dataset, which contains 8 million nodes and 400 million edges, using DeepWalk as the embedding kernel. As shown in Figure 2, GraphZoom drastically boosts the Micro-F1 score up to 47.6% compared to MILE and 49.9% compared to DeepWalk with speedup up to $119.8\times$. When increasing coarsening level, GraphZoom achieves a higher speedup while the embedding accuracy decreases gracefully.

4.3 Analysis on GraphZoom Kernels

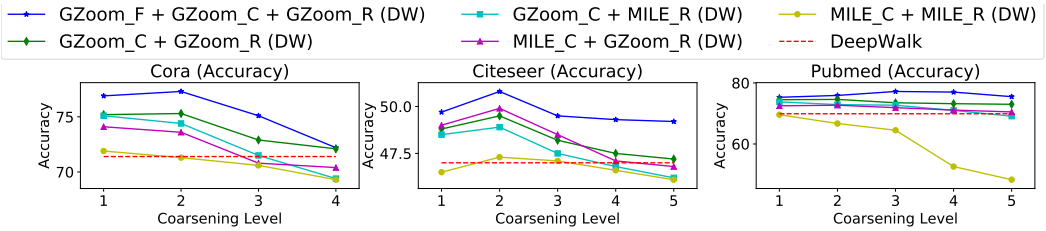


Figure 3: Comparisons of different kernel combinations in GraphZoom and MILE in classification accuracy on Cora, Citeseer, and Pubmed datasets — We choose DeepWalk (DW) as the embedding kernel. GZoom_F, GZoom_C, GZoom_R denote the fusion, coarsening, and refinement kernels proposed in GraphZoom, respectively; MILE_C and MILE_R denote the coarsening and refinement kernels in MILE, respectively; The blue curve is basically GraphZoom and the yellow one is MILE.

To study the effectiveness of our proposed GraphZoom kernels separately, we compare each of them against the corresponding kernel in MILE with other kernels fixed. As shown in Figure 3, when fixing coarsening kernel and comparing refinement kernel of GraphZoom with that of MILE (shown in purple curve and yellow curve), GraphZoom refinement kernel can improve embedding results upon MILE refinement kernel, especially when the coarsening level is large, which indicates that our proposed graph filter in refinement kernel can successfully filter out high frequency noise from graph to improve embedding quality. Similarly, when comparing coarsening kernels in GraphZoom and MILE with refinement kernel fixed (shown in light blue curve and yellow curve), GraphZoom coarsening kernel can also improve embedding quality upon MILE coarsening kernel, which shows that our spectral graph coarsening algorithm can indeed retain key graph structure for underlying graph embedding models to exploit. When combining GraphZoom coarsening kernel and refinement kernel (green curve), we can achieve better classification accuracy compared with the ones using any kernel in MILE (i.e., light blue curve, purple curve and yellow curve), which means that GraphZoom coarsening kernel and refinement kernel play different roles to boost embedding performance and their combination can further improve embedding result. Moreover, when adding graph fusion kernel with the combination of GraphZoom coarsening and refinement kernels (blue curve, which is our GraphZoom framework), it improves classification accuracy by a large margin, which betokens that graph fusion can properly incorporate both graph topology and node attribute information and lifts the embedding quality of downstream embedding models. Results of each kernel CPU time and speedup comparison are available in Appendix F and Appendix H.

5 Conclusion

In this work we propose GraphZoom, a multi-level framework to improve embedding quality and scalability of underlying unsupervised graph embedding techniques. GraphZoom encodes graph structure and node attribute in a single graph and exploiting spectral coarsening and refinement methods to remove high frequency noise from the graph. Experiments show that GraphZoom improves both classification accuracy and embedding speed on a number of popular datasets. An interesting direction for future work is to derive a proper way to propagate node labels to the coarsest graph, which would allow GraphZoom to support supervised graph embedding models.

REFERENCES

Esra Akbas and Mehmet Aktas. Network embedding: on compression and learning. *arXiv preprint arXiv:1907.02811*, 2019.

Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural computation*, 15(6):1373–1396, 2003.

Hongyun Cai, Vincent W Zheng, and Kevin Chen-Chuan Chang. A comprehensive survey of graph embedding: Problems, techniques, and applications. *IEEE Transactions on Knowledge and Data Engineering*, 30(9):1616–1637, 2018.

Haochen Chen, Bryan Perozzi, Yifan Hu, and Steven Skiena. Harp: Hierarchical representation learning for networks. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Jie Chen and Ilya Safro. Algebraic distance on graphs. *SIAM Journal on Scientific Computing*, 33 (6):3468–3490, 2011.

Zhuo Feng. Similarity-aware spectral sparsification by edge filtering. In *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, pp. 1–6. IEEE, 2018.

Guoji Fu, Chengbin Hou, and Xin Yao. Learning topological representation for networks via hierarchical sampling. *arXiv preprint arXiv:1902.06684*, 2019.

Palash Goyal and Emilio Ferrara. Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems*, 151:78–94, 2018.

Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 855–864. ACM, 2016.

Caglar Gulcehre, Misha Denil, Mateusz Malinowski, Ali Razavi, Razvan Pascanu, Karl Moritz Hermann, Peter Battaglia, Victor Bapst, David Raposo, Adam Santoro, and Nando de Freitas. Hyperbolic attention networks. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=rJxHsjRqFQ`.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pp. 1024–1034, 2017.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

Johannes Klicpera, Aleksandar Bojchevski, and Stephan Gnnemann. Combining neural networks with personalized pagerank for classification on graphs. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=H1gL-2A9Ym`.

Qimai Li, Xiao-Ming Wu, Han Liu, Xiaotong Zhang, and Zhichao Guan. Label efficient semi-supervised learning via graph filtering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 9582–9591, 2019.

Jiongqian Liang, Saket Gurukar, and Srinivasan Parthasarathy. Mile: A multi-level framework for scalable graph embedding. *arXiv preprint arXiv:1802.09612*, 2018.

Wenqing Lin, Feng He, Faqiang Zhang, Xu Cheng, and Hongyun Cai. Effective and efficient network embedding initialization via graph partitioning. *arXiv preprint arXiv:1908.10697*, 2019.

Oren E Livne and Achi Brandt. Lean algebraic multigrid (lamg): Fast graph laplacian linear solver. *SIAM Journal on Scientific Computing*, 34(4):B499–B522, 2012.

Andreas Loukas. Graph reduction with spectral and cut guarantees. *Journal of Machine Learning Research*, 20(116):1–42, 2019.

Takanori Maehara. Revisiting graph neural networks: All we have is low-pass filters. *arXiv preprint arXiv:1905.09550*, 2019.

Richard Peng, He Sun, and Luca Zanetti. Partitioning well-clustered graphs: Spectral clustering works! In *Annual Conference on Learning Theory*, pp. 1423–1455, 2015.

Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 701–710. ACM, 2014.

David I Shuman, Sunil K Narang, Pascal Frossard, Antonio Ortega, and Pierre Vandergheynst. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE signal processing magazine*, 30(3):83–98, 2013.

Petar Velikovi, William Fedus, William L. Hamilton, Pietro Li, Yoshua Bengio, and R Devon Hjelm. Deep graph infomax. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=rklz9iAcKQ`.

Jaewon Yang and Jure Leskovec. Defining and evaluating network communities based on ground-truth. *Knowledge and Information Systems*, 42(1):181–213, 2015.

Zhilin Yang, William W Cohen, and Ruslan Salakhutdinov. Revisiting semi-supervised learning with graph embeddings. *arXiv preprint arXiv:1603.08861*, 2016.

Daokun Zhang, Jie Yin, Xingquan Zhu, and Chengqi Zhang. Network representation learning: A survey. *IEEE transactions on Big Data*, 2018a.

Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Maosong Sun, Zhichong Fang, Bo Zhang, and Leyu Lin. Cosine: Compressive network embedding on large-scale information networks. *arXiv preprint arXiv:1812.08972*, 2018b.

Zhiqiang Zhao and Zhuo Feng. Effective-resistance preserving spectral reduction of graphs. In *Proceedings of the 56th Annual Design Automation Conference 2019*, pp. 109. ACM, 2019.

Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, and Maosong Sun. Graph neural networks: A review of methods and applications. *arXiv preprint arXiv:1812.08434*, 2018.

APPENDIX A DETAILS OF DATASETS

Transductive task. We follow the experiments setup in Yang et al. (2016) for three standard citation network benchmark datasets: Cora, Citeseer, and Pubmed. In all these three citation networks, nodes represent documents and edges correspond to citations. Each node has a sparse bag-of-word feature vector and a class label. We allow only 20 labels per class for training and $1,000$ labeled nodes for testing. In addition, we further evaluate on Friendster dataset (Yang & Leskovec, 2015), which contains 8 million nodes and 400 million edges, with 2.5% of the nodes used for training and 0.3% nodes for testing. In Friendster, nodes represent users and a pair of nodes are linked if they are friends; each node has a class label but is not associated with a feature vector.

Inductive task. We follow Hamilton et al. (2017) for setting up experiments on both protein-protein interaction (PPI) and Reddit dataset. PPI dataset consists of graphs corresponding to human tissues, where nodes are proteins and edges represent interaction effects between proteins. Reddit dataset contains nodes corresponding to users' posts: two nodes are connected through an edge if the same users comment on both posts. We use 60% nodes for training, 40% for testing on PPI and 65% for training and 35% for testing on Reddit.

APPENDIX B DETAILS OF BASELINES

DeepWalk first generates random walks based on graph structure. Then, walks are treated as sentences in a language model and Skip-Gram model is exploited to obtain node embeddings.

node2vec is different from DeepWalk in terms of generating random walks by introducing the return parameter p and the in-out parameter q, which can combine DFS-like and BFS-like neighborhood exploration.

Deep Graph Infomax (DGI) is an unsupervised approach that generates node embeddings by maximizing mutual information between patch representations (local information) and corresponding high-level summaries (global information) of graphs .

GraphSAGE embeds nodes in an inductive way by learning an aggregation function that aggregates node features to obtain embeddings. GraphSAGE supports four different aggregation functions: GraphSAGE-GCN, GraphSAGE-mean, GraphSAGE-LSTM and GraphSAGE-pool.

HARP coarsens the original graph into several levels and apply underlying embedding model to train the coarsened graph at each level sequentially to obtain the final embeddings on original graph. Since the coarsening level is fixed in their implementation, we run HARP in our experiments without changing the coarsening level.

MILE is the state-of-the-art multi-level unsupervised graph embedding framework and similar to our GraphZoom framework since it also contains graph coarsening and embedding refinement kernels. More specifically, MILE first uses its heuristic-based coarsening kernel to reduce the graph size and trains underlying unsupervised graph embedding model on coarsest graph. Then, its refinement kernel employs Graph Convolutional Network (GCN) to refine embeddings on the original graph. We compare GraphZoom with MILE on various datasets, including Friendster that contains 8 million nodes and 400 million edges (shown in Table 2 and Figure 2). Moreover, we further compare each kernel in GraphZoom and MILE in Figure 3.

APPENDIX C GRAPH SIZE AT DIFFERENT COARSENING LEVEL

The details of graph size at different coarsening level on all six datasets are shown in Table 4.

Table 4: Number of nodes at different GraphZoom coarsening levels. GZoom-0 means GraphZoom with 0 coarsening level (i.e., without coarsening), GZoom-1 means GraphZoom with 1 coarsening level and so forth.

Dataset	**GZoom-0**	**GZoom-1**	**GZoom-2**	**GZoom-3**	**GZoom-4**	**GZoom-5**
Cora	2,708	1,169	519	218	100	45
Citeseer	3,327	1,488	606	282	131	58
Pubmed	19,717	7,903	3,562	1,651	726	327
PPI	14,755	5,061	1,815	685	281	120
Reddit	232,965	84,562	30,738	11,598	4,757	2,117
Friendster	7,944,949	2,734,483	1,048,288	409,613	134,956	44,670

APPENDIX D GRAPHZOOM ALGORITHM

Algorithm 1: GraphZoom algorithm

Input: Adjacency matrix $\boldsymbol{A}_{topo} \in \mathbb{R}^{N \times N}$; node feature matrix $\boldsymbol{X} \in \mathbb{R}^{N \times K}$; base embedding function $l(\cdot)$; coarsening level m

Output: Node embedding matrix $\boldsymbol{E} \in \mathbb{R}^{N \times D}$

$\boldsymbol{A}_0 = graph_fusion(\boldsymbol{A}_{topo}, \boldsymbol{X})$;
for $i = 1...m$ **do**
  $\boldsymbol{A}_i, \boldsymbol{H}_{i-1}^i = spectral_coarsening(\boldsymbol{A}_{i-1})$;
end
$\boldsymbol{E}_L = l(\boldsymbol{A}_m)$;
for $i = m...1$ **do**
  $\hat{\boldsymbol{E}}_{i-1} = (\boldsymbol{H}_{i-1}^i)^T \boldsymbol{E}_i$;
  $\boldsymbol{E}_{i-1} = refinement(\hat{\boldsymbol{E}}_{i-1})$;
end
$\boldsymbol{E} = \boldsymbol{E}_0$;

APPENDIX E SPECTRAL COARSENING

Note that the mapping operator $\boldsymbol{H}_i^{i+1} \in \{0,1\}^{|\mathcal{V}_{i+1}| \times |\mathcal{V}_i|}$ is a matrix containing only 0s and 1s. It has following properties:

- The row (column) index of $\boldsymbol{H}_i^{i+1}$ corresponds to the node index in graph $\mathcal{G}_{i+1}$ ($\mathcal{G}_i$).
- It is a surjective mapping of the node set, where $(H_i^{i+1})_{p,q} = 1$ if node q in graph $\mathcal{G}_i$ is aggregated to super-node p in graph $\mathcal{G}_{i+1}$, and $(H_i^{i+1})_{p',q} = 0$ for all nodes $p' \in \{v \in \mathcal{V}_{i+1} : v \neq p\}$.
- It is a locality-preserving operator, where the coarsened version of $\mathcal{G}_i$ induced by the non-zero entries of $(H_i^{i+1})_{p,:}$ is connected for each $p \in \mathcal{V}_{i+1}$.

Algorithm 2: spectral_coarsening algorithm

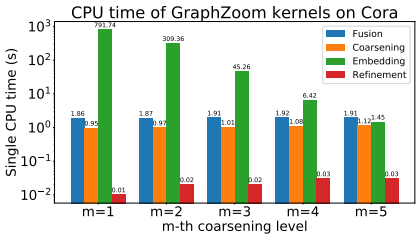
Input: Adjacency matrix $\boldsymbol{A}_i \in \mathbb{R}^{|\mathcal{V}_i|\times|\mathcal{V}_i|}$

Output: Adjacency matrix $\boldsymbol{A}_{i+1} \in \mathbb{R}^{|\mathcal{V}_{i+1}|\times|\mathcal{V}_{i+1}|}$ of the reduced graph $\mathcal{G}_{i+1}$, mapping operator $\boldsymbol{H}_i^{i+1} \in \mathbb{R}^{|\mathcal{V}_{i+1}|\times|\mathcal{V}_i|}$

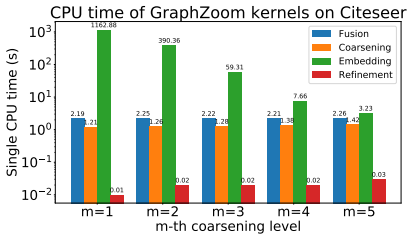
$n = |\mathcal{V}_i|$, $n_c = n$;
[graph reduction ratio] $\gamma_{max} = 1.8$, $\delta = 0.9$;
for *each edge* $(p, q) \in \mathcal{E}_i$ **do**
  [spectral node affinity set] $\mathbb{C} \leftarrow a_{p,q}$ defined in Eq. 2 ;
end
for *each node* $p \in \mathcal{V}_i$ **do**
  $d(p) = \left|(A_i)_{p,:}\right|$, $d_m(p) = \text{median}\left(\left|(A_i)_{q,:}\right| \text{ for all } q \in \{q|(p,q) \in \mathcal{E}_i\}\right)$;
  if $d(p) \geq 8 \cdot d_m(p)$ **then**
    [node aggregation flag] $\boldsymbol{z}(p) = 0$;
  else
    [node aggregation flag] $\boldsymbol{z}(p) = -1$;
  end
end
$\gamma = 1$;
while $\gamma < \gamma_{max}$ **do**
  $\mathbb{S} = \emptyset$, $\mathbb{U} = \emptyset$;
  [unaggregated node set] $\mathbb{U} \leftarrow p \in \{p|z(p) == -1 \ \forall p \in \mathcal{V}_i\}$;
  $\mathbb{S} \leftarrow p \in \{p|a_{p,q} \geq \delta \cdot \max_{s\neq p,q}\left(\max_{(p,s)\in \mathcal{E}_i}\left(a_{p,s}\right), \max_{(q,s)\in \mathcal{E}_i}\left(a_{q,s}\right)\right) \ \forall p \in \mathcal{V}_i\}$;
  for *each node* p *in* $\mathbb{S} \cap \mathbb{U}$ **do**
    if $\boldsymbol{z}(p) == -1$ **then**
      $q = \arg\max_{(p,q)\in \mathcal{E}_i}\left(a_{p,q}\right)$;

      if $\boldsymbol{z}(q) == -1$ **then**
        $\boldsymbol{z}(q) = 0, \boldsymbol{z}(p) = q$, $\hat{q} = q$;
      else if $\boldsymbol{z}(q) == 0$ **then**
        $\boldsymbol{z}(p) = q$, $\hat{q} = q$;
      else
        $\boldsymbol{z}(p) = \boldsymbol{z}(q)$, $\hat{q} = \boldsymbol{z}(q)$;
      end
      update smoothed vectors in T with $x_p^{(k)} = x_{\hat{q}}^{(k)}$ for $k = 1, \cdots, t$
      $n_c = n_c - 1$;
    end
  end
  $\gamma = n/n_c$, $\delta = 0.7 \cdot \delta$;
  $\boldsymbol{z}(p) = p$ for node $p \in \{p|\boldsymbol{z}(p) == 0\}$
end
form $\boldsymbol{H}_i^{i+1}$ and $\boldsymbol{A}_{i+1}$ based on $\boldsymbol{z}$

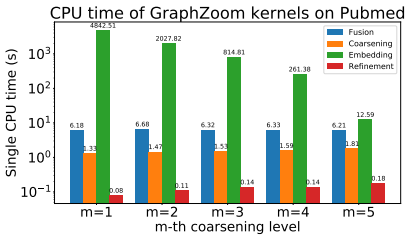
APPENDIX F CPU TIME OF EACH GRAPHZOOM KERNEL



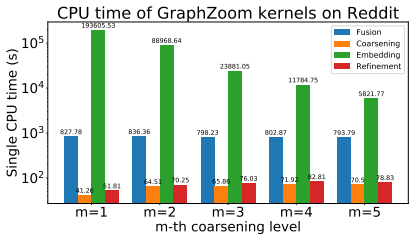
(a) GraphZoom with DeepWalk as embedding kernel



(b) GraphZoom with DeepWalk as embedding kernel



(c) GraphZoom with DeepWalk as embedding kernel



(d) GraphZoom with GraphSAGE as embedding kernel

Figure 4: CPU time of GraphZoom kernels

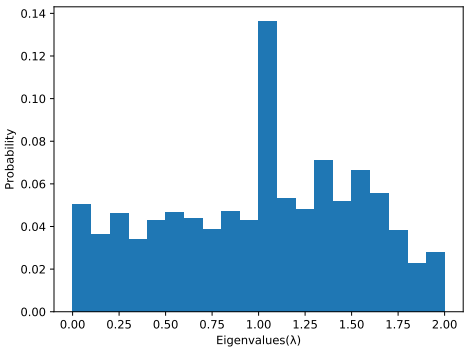
As shown in Figure 4 (note that the y axis is in logarithmic scale), the GraphZoom embedding kernel dominates the total CPU time, which can be more effectively reduced with a greater coarsening level L. All other kernels in GraphZoom are very efficient, which enable the GraphZoom framework to drastically reduce the total graph embedding time.

APPENDIX G GRAPH FILTERS AND LAPLACIAN EIGENVALUES

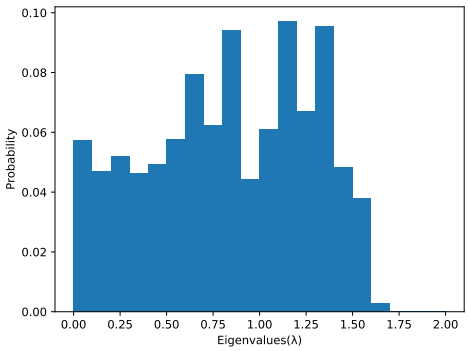
Figure 5a shows the original distribution of graph Laplacian eigenvalues which also can be interpreted as frequencies in graph spectral domain (smaller eigenvalue means lower frequency). The proposed graph filter for embedding refinement (as shown in Figure 5e) can be considered as a band-stop filter that passes all frequencies with the exception of those within the middle stop band that is greatly attenuated. Therefore, the band-stop filter may not be very effective for removing high-frequency noises from the graph signals. Fortunately, it has been shown that by adding self-loops to each node in the graph as follows $\tilde{A} = A + \sigma I$ (shown in Figure 5b, 5c, 5d, where $\sigma = 0.5, 1.0, 2.0$), the distribution of Laplacian eigenvalues can be squeezed to the left (towards zero) (Maehara, 2019). By properly choosing σ such that large eigenvalues will mostly lie in the stop band (e.g., $\sigma = 1.0, 2.0$ shown in Figure 5c and 5d), the graph filter will be able to effectively filtered out high-frequency components (corresponding to high eigenvalues) while retaining low-frequency components, which is similar to a low-pass graph filter as shown in Figure 5f. It is worth noting that if σ is too large, then most eigenvalues will be very close to zero, which makes the graph filter less effective for removing noises. In this work, we choose $\sigma = 2.0$ for all our experiments.

APPENDIX H SPEEDUP OF GRAPHZOOM KERNELS COMPARED TO MILE

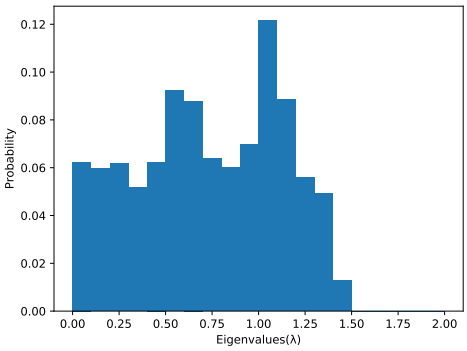
As shown in Figure 6, the combination of GraphZoom coarsening and refinement kernels can always achieve the greatest speedups (green curves); adding GraphZoom fusion kernel (blue curves) will lower the speedups by a small margin but further boost the embedding quality, showing a clear trade-off between embedding quality and runtime efficiency: to achieve the highest graph embedding quality, the graph fusion kernel should always be included.



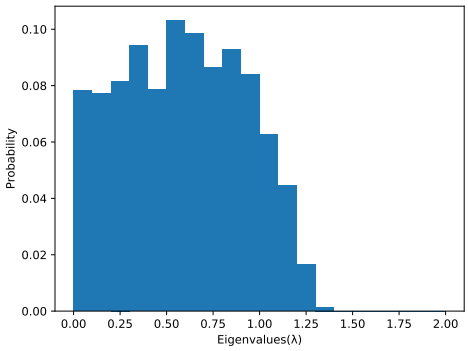
(a) Original eigenvalues with $\tilde{\boldsymbol{A}} = \boldsymbol{A}$



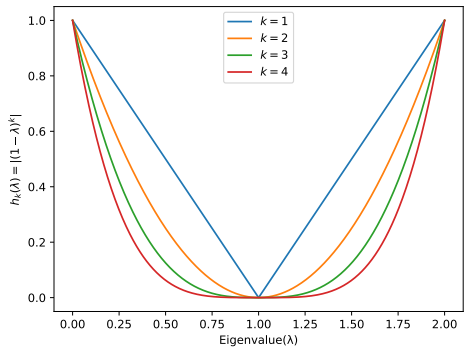
(b) Squeezed eigenvalues with $\tilde{\boldsymbol{A}} = \boldsymbol{A} + 0.5\boldsymbol{I}$



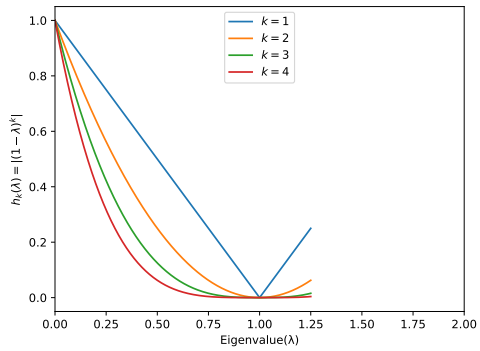
(c) Squeezed eigenvalues with $\tilde{\boldsymbol{A}} = \boldsymbol{A} + 1.0\boldsymbol{I}$



(d) Squeezed eigenvalues with $\tilde{\boldsymbol{A}} = \boldsymbol{A} + 2.0\boldsymbol{I}$



(e) Filters for original laplacian eigenvalues



(f) Filters for squeezed laplacian eigenvalues

Figure 5: Distribution of graph laplacian eigenvalues with different self-loops on Cora

APPENDIX I MORE RESULTS ON PPI DATASETS

As shown in Figure 7, the embedding results by GraphZoom with increasing number of coarsening levels are still better than the ones by GraphSAGE with different aggregation functions. It is worth noting that although the level-5 (coarsened) graph contains only 120 nodes (as shown in C), GraphZoom embedding results can still beat the GraphSAGE baseline obtained with the original graph containing $14,755$ nodes.

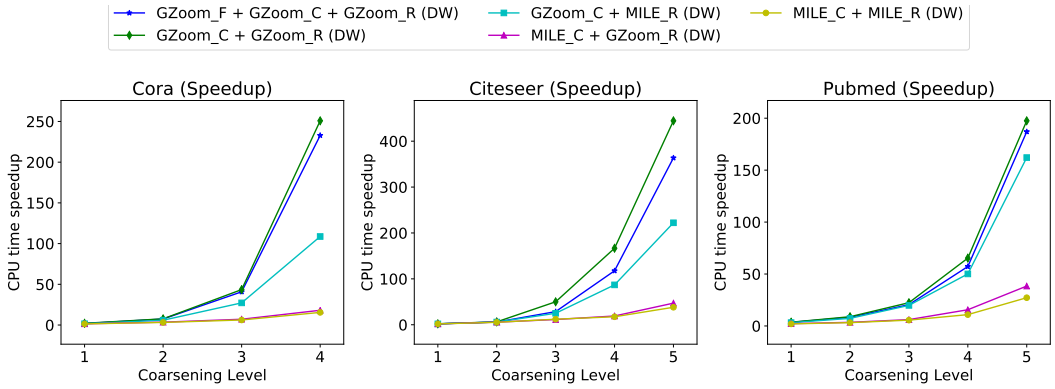
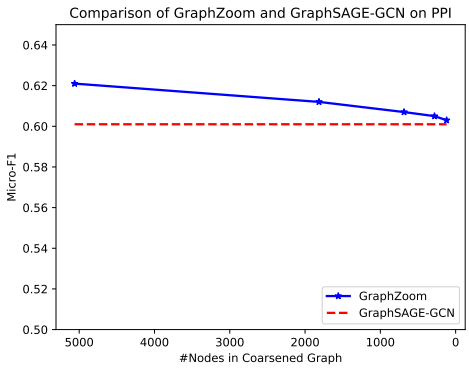
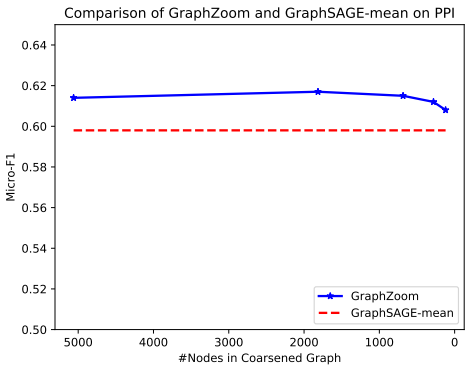


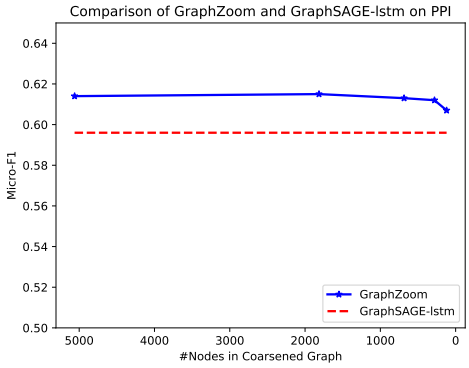
Figure 6: Comparisons of different combinations of kernels in GraphZoom and MILE in terms of CPU time speedup on Cora, Citeseer, and Pubmed datasets — We choose DeepWalk (DW) as basic embedding method. GZoom_F, GZoom_C, GZoom_R, MILE_C and MILE_R represent GraphZoom fusion kernel, GraphZoom coarsening kernel, GraphZoom refinement kernel, MILE coarsening kernel and MILE refinement kernel, respectively.



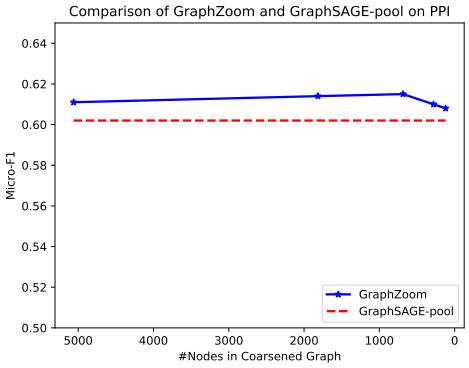
(a) GraphSAGE with GCN as aggregation function



(b) GraphSAGE with mean as aggregation function



(c) GraphSAGE with LSTM as aggregation function



(d) GraphSAGE with pool as aggregation function

Figure 7: Comparisons of GraphZoom and GraphSAGE on PPI