Spectral Coarsening for Scalable Graph Embedding

Chenhui Deng(cd574) Ahmed Gashgash(atg74) Kailin Yang(ky362)

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 [1, 2, 3].

However, current graph embedding methods have several drawbacks in terms of either accuracy or scalability. On the one hand, random-walk-based embedding algorithms, such as DeepWalk [4] and node2vec [5], 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 smoothed over the entire graph [6]. 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 [7]. 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 [8]. For example, graph neural networks (GNNs) such as GraphSAGE [9] 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 cause drastic increase in the amount of computation among nodes, but also lead to high memory usage for storing the intermediate results.

The goal of simultaneously improving the scalability and accuracy of graph embedding methods has lately been of interest in literature. One way of improving the scalability of graph embedding models is to reduce the graph size via graph coarsening, which will significantly improve the execution time of graph embedding when training on the coarsest graph, and therefore improves the embedding scalability. Since graph coarsening may lose some graph structure information, a following question is: does the graph coarsening degrade the embedding quality? [10] has shown that the structure of real-world network contains some noises, which may be harmful to the embedding quality. For instance, the link between nodes with different labels is likely to degrade the accuracy for node classification task. Thus, not all structural information of the original graph is necessary for high-quality graph embedding, which means if we can coarsen the graph by removing the noisy structure while only preserve key graph properties for the underlying embedding model, it will not only improve the scalability by reducing graph size, but also enhance the embedding quality via removing the redundant/noisy structural information. Follow this idea, [11] proposed a multi-level framework to improve both embedding quality and scalability. Their coarsening algorithm, however, lacks of theory to guarantee that key graph properties are actually preserved in the coarsened graph. To theoretically preserve key spectral properties (e.g., first few non-trivial eigenvectors and eigenvalues of graph Laplacian), [12] proposed a spectral approach called GraphZoom, which consists of 4 kernels: (1) graph fusion, (2) spectral graph coarsening, (3) graph embedding, and (4) embedding refinement. Specifically, 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 much smaller graphs by spectral graph coarsening. GraphZoom allows any existing embedding methods to be applied to the coarsest graph, before it progressively refine the embeddings obtained at the coarsest level to increasingly finer graphs.

GraphZoom improves the embedding scalability by first reducing the graph size, and then applies graph embedding model (which is the most time-consuming kernel in GraphZoom) on the smallest graph. Moreover, it also enhances the embedding quality since its spectral coarsening algorithm preserves key graph spectral properties, while filtering out unnecessary structural information. More importantly, GraphZoom adopts a low-pass graph filter during embedding refinement stage to smooth the embeddings over the graph, which further improves embedding quality. Last but not least, the graph fusion kernel of GraphZoom fuses node features into graph structure, which enables embedding models such as DeepWalk and node2vec to incorporate node feature information during training. More details could be found in the original paper [12].

Although GraphZoom achieves quite impressive improvement for both embedding quality and scalability on various realistic datasets, the spectral coarsening algorithm of GraphZoom, which is based on the Lean Algebraic MultiGrid (LAMG) numerical approach, requires to iteratively perform some numerical methods in Matlab (e.g., Gauss-Seidel relaxations), which are not very friendly to hardware in terms of the computation (e.g. FPGA).

In this project, we aim to derive a more efficient spectral coarsening algorithm to replace the original version in GraphZoom. The new algorithm should be hardware-friendly while preserving key structure information for the underlying graph embedding model.

Besides, recent years people have tried to build all kinds of non-traditional computing engines to accelerate machine learning algorithms. Many among them utilize the idea of Processing-In-Memory (PIM) to combine the computing and storage logic in the same component, by leveraging the low-level computing capability of the SRAM, DRAM, or NVM arrays[13, 14, 15], or integrating simple processing units near the main memory[16, 17]. PIM is naturally favorable to graph-based algorithms, because it does not rely on the traditional data caching mechanism, and can provide much higher memory throughput compared to regular von Neumann architectures such as CPU and GPU. Thus it can mitigate the frequent and irregular memory accesses of graph algorithms, and utilize the substantial data-level parallelism.

In this project, besides the regular python-based implementation, we also implement an alternative version of our spectral coarsening algorithm using RISC-V instructions with standard vector extension [18], and execute it on a state-of-the-art CMOS-based PIM computing engine. The goal is to examine if spectral coarsening is a good candidate for the emerging PIM accelerators, so as to reach even faster execution time.

2 Related Work

There is a large and active body of research on graph embedding and graph coarsening, from which our work draws inspiration to derive efficient spectral coarsening for scalable unsupervised graph embedding. Specifically, we adopt the idea of low-pass graph filtering to develop our spectral coarsening algorithm. We are going to summarize some of the recent efforts in these areas.

Unsupervised graph embedding A large number of graph embedding approaches are proposed to learn node embeddings in an unsupervised manner, which avoids the nontrivial process for label generation and therefore can be considered as a generic task independent of downstream applications. [19, 20, 21] represent the connections between nodes in the form of a matrix and then factorize it to obtain the embedding, which are also known as matrix factorization based methods. Later, random walk based methods such as DeepWalk [4] and node2vec [5] are proposed to treat random walks in graph as sentences in a language model and exploit Skip-Gram to obtain nodes (words) embeddings. Then, Qiu et al. prove that DeepWalk and node2vec are essentially matrix factorization methods [22]. Hamilton et al. develop GraphSAGE to generate node embeddings by aggregating feature information from neighborhood [9]. GraphSAGE can learn node embeddings in an unsupervised way with the basic notion that nearby nodes have similar embeddings. Recently, Velickovic et al. adopt the idea of maximizing mutual information between "global" and "local" representations on graphs, and derive deep graph infomax (DGI) [23].

Multi-level graph coarsening The multi-level approach has been widely studied for efficient graph embedding. The key idea of multi-level algorithms is to coarsen the original graph into a series of much smaller ones, on which various graph embedding models can be performed. The embeddings are then refined from the coarse-grained graph back to the original graph in a iterative manner. For

example, [24, 25] generate a hierarchy of coarsened graphs and perform embedding from the coarsest level to the original one. [26] constructs and embed multi-level graphs through hierarchical, and the resulting embedding vectors are concatenated to obtain the final node embeddings for the original graph. These methods, however, only focus on improving embedding quality but not the scalability. Later, [27, 28] attempt to make graph embedding more scalable by only embedding on the coarsest graph. However, their methods lack proper refinement methods to generate high-quality embeddings for the original graph. [11] proposes MILE, which only trains the coarsest graph to obtain coarse embeddings, and leverages GCN as refinement method to improve embedding quality. However, MILE requires training a GCN model which is very time consuming for large graphs and leading to poor performance when multiple GCN layers are stacked together [29]. Recently, Deng et al. proposed GraphZoom, a multi-level spectral approach for high-quality and scalable graph embedding. We use GraphZoom in this work while replacing the original cooarsening algorithm in GraphZoom with our efficient spectral coarsening algorithm.

Graph filtering are direct analogs of classical filters in signal processing field, but intended for signals defined on graphs. [30] defines graph filters in both vertex and spectral domains, and apply graph filter on image denoising and reconstruction tasks. [31] leverages graph filter to simplify GCN model by removing redundant computation. More recently, [7] shows 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. [32] further derives two generalized graph filters and apply them on graph embedding models to improve their embedding quality for various classification tasks. In this work we adopt graph filter to properly smooth the node spectral embedding to preserve the first few non-trivial eigenpairs of graph Laplacian, which is crucial for high quality spectral coarsening.

3 Method

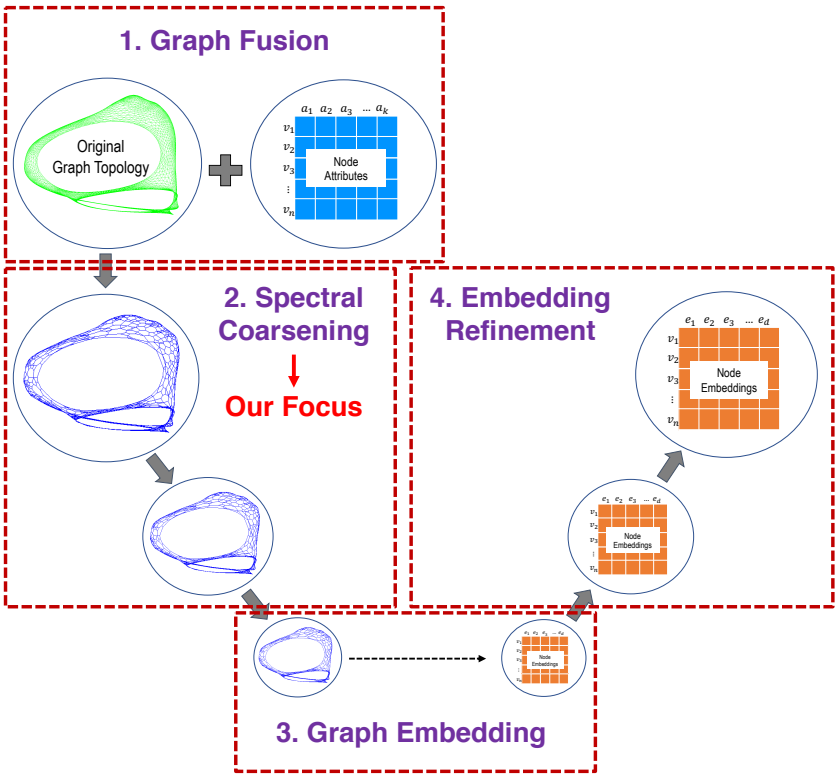


Figure 1: Overview of the GraphZoom framework and spectral coarsening.

In this work, we derive a new efficient spectral coarsening algorithm to reduce the graph size while preserving key spectral graph properties (e.g., first few non-trivial Laplacian eigenpairs) for graph embedding. We further integrate our algorithm with GraphZoom. The overview figure is shown in Fig. 1.

Coarsening is one type of graph reduction whose objective is to achieve computational acceleration by reducing the size (i.e., number of nodes) of the original graphs while maintaining the similar graph structure between the original graph and the reduced ones. For each coarsen level $i+1$, a surjection is defined between the original node set V_i and the reduced node set V_{i+1}, where each node $v \in V_{i+1}$ corresponds to a small set of adjacent vertices in the original node set V_i. Mapping operator H_i^{i+1} can be defined based on the mapping relationship between V_i and V_{i+1}. Note that the operator $H_i^{i+1} \in \{0,1\}^{|V_{i+1}| \times |V_i|}$ is a matrix containing only 0s and 1s. It has following properties:

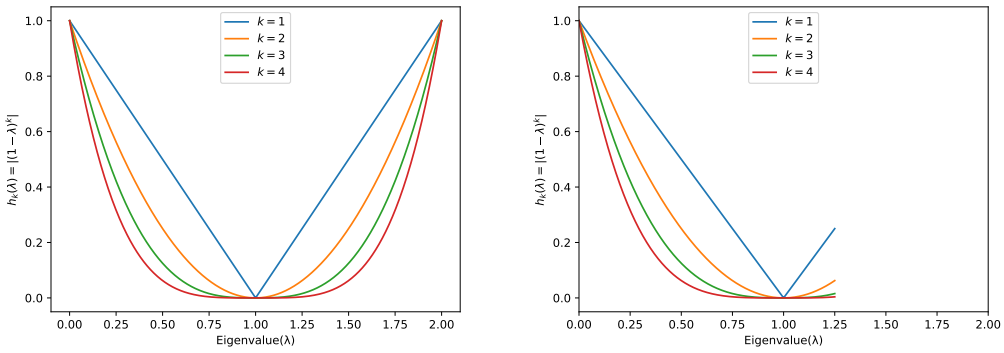
- The row (column) index of H_i^{i+1} corresponds to the node index in graph G_{i+1} (G_i).
- It is a surjective mapping of the node set, where $(H_i^{i+1})_{p,q} = 1$ if node q in graph G_i is aggregated to super-node p in graph G_{i+1}, and $(H_i^{i+1})_{p',q} = 0$ for all nodes $p' \in \{v \in V_{i+1}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

How to find a proper mapping operator H is the key to spectral coarsening. An intuitive way is to perform spectral clustering, which clusters spectrally similar nodes based on the first few non-trivial eigenvectors of the graph Laplacian. By mapping nodes in the same cluster of the i-th coarsen level graph G_i to the supernode in the next coarsen level graph G_{i+1}, we can actually obtain the corresponding mapping operator H_i^{i+1}, and the Laplacian matrix of G_{i+1} can be obtained via:

$$L_{i+1} = H_i^{i+1} L_i (H_i^{i+1})^T$$

By iteratively applying the above equation, we can progressively coarsen the graph until we obtain the desired graph size. However, the key issue of spectral clustering is that it requires to compute the eigenvectors of the graph Laplacian, which requires $O|V|^3$ time. Since the first few non-trivial Laplacian eigenvectors can be viewed as "low-frequency" signals over the graph, the goal of this project is to leverage a proper low-pass graph filter to obtain vectors that approximate those "low-frequency" Laplacian eigenvectors. Our spectral coarsening algorithm should be very efficient with time complexity up to $O(|V|\log(|V|) + |E|)$.

3.1 Low-Pass Graph Filter



(a) Effective range of $h_k(\lambda)$ without self-loop

(b) Effective range of $h_k(\lambda)$ by adding self-loop

Figure 2: Transform from band-pass filter to low-pass filter via self-loop trick

The key of our spectral graph coarsening algorithm is to preserve low-frequency components of the graph while filtering out the high-frequency components, which can be viewed as a low-pass graph filter. We are going to show the low-pass used in our method as follows.

Lemma 1. *Multiplying the normalized adjacency matrix corresponds to applying graph filter $h = 1 - \lambda$ in graph spectrum, where λ is the eigenvalue of graph Laplacian.*

Proof of Lemma 1. Let $y = \hat{A}x$, where x is an input graph signal, y is the output graph signal, and $\hat{A} = D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$ is the normalized adjacency matrix. Since we have $\hat{A} = I - \hat{L}$, where $\hat{L}$ is the normalized Laplacian matrix, we let $h(\hat{L}) = I - \hat{L} = \hat{A}$. Based on the eigen-decomposition of $\hat{L} = U\Lambda U^T$, we can get $h(\hat{L}) = U \begin{bmatrix} h(\lambda_1) & & \\ & \ddots & \\ & & h(\lambda_n) \end{bmatrix} U^T$, where $h(\lambda_i) = 1 - \lambda_i$. In other words, when multiplying the normalized adjacency matrix $y = \hat{A}x$, we can view it as $y = U \begin{bmatrix} h(\lambda_1) & & \\ & \ddots & \\ & & h(\lambda_n) \end{bmatrix} U^T x$, where $U^T x$ is the graph Fourier transform, $\hat{x} = \begin{bmatrix} h(\lambda_1) & & \\ & \ddots & \\ & & h(\lambda_n) \end{bmatrix} U^T x$ can be viewed as graph filtering in graph spectrum, and $U^T\hat{x}$ is the inverse graph Fourier transform to map $\hat{x}$ back to the spatial domain. Thus, multiplying $\hat{A}$ in spatial domain is equivalent to perform $h(\lambda) = 1 - \lambda$ in graph spectral domain.

Since the eigenvalues of the normalized Laplacian lie on the interval [0, 2], $h(\lambda) = 1 - \lambda$ resembles a band-stop filter that removes intermediate frequency components. In order to get the low-pass graph filter, we can use the trick of adding self-loop to the graph, which can be proved in Theorem 1.

Theorem 1. *Let $\lambda_i(\sigma)$ be the i-th smallest eigenvalue of $\tilde{L} = I - \tilde{A}$, where $\tilde{A} = A + \sigma I$, I is the identity matrix, and A is the normalized adjacency matrix. Then $\lambda_i(\sigma)$ is a non-negative number, and monotonically non-increasing in $\sigma > 0$. Moreover, $\lambda_i(\sigma)$ is strictly monotonically decreasing if $\lambda_i(0) \neq 0$.*

Proof of Theorem 1 can be found in the appendix of [7]. Theorem 1 shows that when adding self-loop, the largest eigenvalues (i.e., the ones closed to 2) will decrease. Thus, we can control the self loop parameter σ to make largest Laplacian eigenvalues closed the 1, which means $h(\lambda) = 1 - \lambda$ becomes a low-pass filter. We extend $\tilde{A}$ to $\tilde{A}^k$ to further enhance the low-pass property of $h(\lambda) = 1 - \lambda$ to $h_k(\lambda) = (1 - \lambda)^k$. The comparison of graph filters with/without adding self-loop is shown in Fig 2.

3.2 Efficient Spectral Coarsening

In this work, we leverage an efficient yet effective local spectral embedding scheme to identify node clusters based on emerging graph signal processing techniques [30]. 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 is expressed with a linear combination of eigenvectors u of the graph Laplacian. We adopt low-pass graph filters 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 x, we obtain a smoothed vector $\tilde{x}$, which is basically the linear combination of the first few eigenvectors:

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

More specifically, we apply the low-pass graph filter $\tilde{A}^k = (D^{-\frac{1}{2}}(A + \sigma I)D^{-\frac{1}{2}})^k$ 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 $\mathbf{1}^\top x^{(i)} = 0$, where A is the adjacency matrix, I is the identity matrix, and D is the corresponding diagonal degree matrix, σ controls the node self-loop, and p controls the power of low-pass filter.

Based on the smoothed vectors in T, we embed each node into a t-dimensional space such that nodes p and q are considered spectrally similar if their low-dimensional embedding vectors $x_p \in R^t$ and $x_q \in R^t$ are highly correlated. Here the node distance is measured by the spectral node affinity $a_{p,q}$ for neighboring nodes p and q [33, 34]:

$$a_{p,q} = \frac{|(T_{p,:}, T_{q,:})|^2}{\|T_{p,:}\|^2 \|T_{q,:}\|^2}, \quad (T_{p,:}, T_{q,:}) = \Sigma_{k=1}^{t}(x_p^{(k)} \cdot x_q^{(k)}) \tag{2}$$

Once the node aggregation schemes are determined, we can easily obtain the graph mapping operator H_i^{i+1} between two coarsening levels i and $i+1$. More precisely, H_i^{i+1} is a matrix of size $|V_{G_{i+1}}| \times |V_{G_i}|$. $(H_i^{i+1})_{p,q} = 1$ if node q in graph G_i is aggregated to (clustered) node p in graph G_{i+1}; otherwise, it is set to 0.

4 Dataset

Table 1: Statistics of datasets used in our experiments.

Dataset	Type	Nodes	Edges	Classes	Features	Train/Valid/Test
SBM	Synthetic network	12,000	47,986	6	NA	600/2,400/6,000
Cora	Citation network	2,708	5,429	7	1,433	140/500/1,000
Citeseer	Citation network	3,327	4,732	6	3,703	120/500/1,000
Pubmed	Citation network	19,717	44,338	3	500	60/500/1,000
ogbn-arxiv	Citation network	169,343	1,166,243	40	128	90,941/29,799/48,603
ogbn-proteins	Biology network	132,534	39,561,252	112	8	86,619/21,236/24,679

Synthetic benchmark. We first evaluate our approach on the synthetic dataset generated via the stochastic block model (SBM). The SBM is a stochastic graph model with planted clusters. Its widely used to study or understand the structure of a network. This model is generated by taking each pair of nodes and simulating the presence of a link based on a predefined probability. Edges between nodes within the same cluster would have a higher probability of occurrence than edges between nodes of different clusters. In our synthetic generated SBM graph, the edge probability within a cluster/block was set to 0.0015 and the edge probability between blocks was set to 0.0005. The statistics of the dataset and the Train/Valid/Test split are detailed in 1. The total split for each task was averaged between the clusters. For example, the training set contained 100 nodes from each cluster for a total of 600 nodes. Note that for this experiment no node features were considered. Also, the dataset split was randomized each time for 3 experiments and the average score was reported.

Popular benchmarks. In order to evaluate our spectral coarsening algorithm, we choose three popular real-world datasets of different sizes. Specifically we will consider the citation network datasets: Cora, Citeseer, and Pubmed to perform transductive node classification tasks. The statistics of each dataset are summarized in Table 1. Each node in the graph refers to a document/paper while the edges are citation links. Each dataset contains sparse bag-of-words feature vectors for each node and a list of citation links between documents. These citation links are represented in a binary symmetric adjacency matrix A. The feature vectors for each document are stored in a separate feature matrix X.

We choose the same dataset splits as [35]. Specifically, after obtaining the node embeddings, we use 20 nodes per class for training the classifier (e.g., logistic regression) for each dataset, and 1000 nodes for testing. More importantly, [36] has shown the pitfalls of these datasets, which are the fact that choosing different training/test split will largely change the node classification accuracy score. As suggested by [36], we randomly choose different training and test sets for 100 times, and report both the average classification accuracy as well as standard deviation during evaluation section.

More realistic benchmarks. To evaluate our approach on more realisitc benchmarks, we choose both "ogbn-arxiv" and "ogbn-proteins" datasets from open graph benchmark (OGB). The statistics of these two datasets are shown in Table 1.

- The dataset "ogbn-arxiv" is a directed graph, representing the citation network between all Computer Science (CS) arXiv papers. Each node is an arXiv paper and each directed edge indicates that one paper cites another one. Each paper comes with a 128-dimensional feature vector obtained by averaging the embeddings of words in its title and abstract. The embeddings of individual words are computed by running the skip-gram model. In addition, all papers are also associated with the year that the corresponding paper was published. Our goal is to classify each publication's areas and topics.
- The dataset "ogbn-proteins" is an undirected, weighted, and typed (according to species) graph. Nodes represent proteins, and edges indicate different types of biologically meaningful associations between proteins, e.g., physical interactions, co-expression or homology. All edges come with 8-dimensional features, where each dimension represents the strength of a single association type and takes values between 0 and 1 (the larger the value is, the stronger the association is). In our implementation, we use edge features as node features during graph embedding.

5 Evaluation

Table 2: Experimental results of GraphZoom with different levels of spectral coarsening over synthetic dataset

Stochastic Block Model		
Method	Acc	Speedup
DeepWalk	28.5	1×
1-level	**30.8**	2.4×
2-level	29.5	3.0×
3-level	28.3	4.1×
4-level	26.7	10.4×
5-level	25.8	17.8×

Table 3: Experimental results of GraphZoom with different levels of spectral coarsening over citation networks

	Cora		**Citeseer**		**Pubmed**	
Method	Acc ± Std	CPU Time/s	Acc ± Std	CPU Time/s	Acc ± Std	CPU Time/s
DeepWalk	70.2 ± 2.1	636	44.6 ± 2.6	697	70.1 ± 2.7	6157
1-level	72.5 ± 2.0	389 (1.6×)	47.7 ± 2.1	412 (1.7×)	73.2 ± 2.4	4564 (1.3×)
2-level	**73.4** ± 2.1	274 (2.3×)	48.4 ± 2.4	310 (2.2×)	73.4 ± 2.4	3707 (1.7×)
3-level	72.1 ± 1.7	191 (3.3×)	**48.5** ± 2.5	251 (2.8×)	73.7 ± 2.6	3048 (2.0×)
4-level	70.8 ± 1.9	137 (4.6×)	48.0 ± 2.4	205 (3.4×)	73.3 ± 2.1	2561 (2.4×)
5-level	70.4 ± 1.8	95 (6.7×)	48.0 ± 2.3	165 (4.2×)	**74.1** ± 2.3	2144 (2.9×)

5.1 Performance of GraphZoom with Spectral Coarsening

We examine the efficiency of GraphZoom using our algorithm for graph coarsening. For the synthetic dataset and citation networks, we choose DeepWalk [37] in the graph embedding phase in GraphZoom. And for the OGB datasets, we adopt node2vec [5]. We compare the classification accuracy and execution time of GraphZoom with at most five levels of spectral coarsening to naïve DeepWalk and node2vec approaches respectively. The execution time is measured on an 1.4GHz quad-core Intel Core i5 processor.

First, Table 2 shows the accuracy percentage and speedup with respective to the baseline of the synthetic dataset based on the stochastic block model. From the table, we find GraphZoom with

Table 4: Experimental results of GraphZoom with different levels of spectral coarsening over OGB datasets

	ogbn-proteins		**ogbn-arxiv**	
Method	AUC $\pm$ Std	Speedup	Acc $\pm$ Std	Speedup
node2vec	68.8 ± 0.65	$1\times$	70.1 ± 0.13	$1\times$
1-level	$\mathbf{70.7 \pm 0.60}$	$2.2\times$	$\mathbf{70.4 \pm 0.13}$	$2.5\times$
2-level	69.3 ± 0.73	$3.8\times$	69.9 ± 0.12	$3.6\times$
3-level	68.5 ± 0.37	$6.8\times$	69.0 ± 0.25	$5.8\times$
4-level	69.4 ± 0.28	$12.6\times$	67.7 ± 0.21	$8.6\times$
5-level	68.7 ± 0.24	$25.4\times$	67.0 ± 0.40	$13.8\times$

spectral coarsening can always achieve a comparable accuracy compared to naïve DeepWalk, even with multiple levels of coarsening. In the best case, it increases the accuracy by 2%. This proves that our coarsening algorithm can keep the key information of a randomly generated graph structure. As for the execution time, GraphZoom can always achieve speedup with respect to the baseline. Moreover, the speedup increases super-linearly as the level of coarsening increases. This trend proves that the time saved in graph embedding phase can effectively compensate the execution time of graph coarsening. Thus, our spectral coarsening algorithm is time efficient when incorporated in GraphZoom. As a conclusion, for a random graph, our coarsening algorithm can significantly simplify the graph structure, while still maintaining key information of the graph.

Table 3 gives the classification accuracy in percentage of the citation network datasets, the standard deviation, and the execution time. From the table, we find that GraphZoom can always achieve higher testing accuracy no matter how many levels of coarsening it performs for all three datasets. The accuracy increases by at most 3.2%, 3.9%, and 4% for Cora, Citeseer, and Pubmed, respectively (the bold italic numbers). Meanwhile, our approach also has smaller deviation compared to DeepWalk. These results show that our approach performs well for real world datasets. Besides, similar to the synthetic dataset, GraphZoom can always achieve speedup compared to the baseline. So our coarsening algorithm works properly for the citation networks. However, unlike the SBM, the speedup increases almost linearly as we apply deeper levels of coarsening for the citation networks. This phenomenon suggests that nodes in citation networks are less likely to have high spectral affinity compared to a random graph. Thus, our algorithm cannot coarsen their structure to the same extent as the SBM graph. But on the other hand, there is no accuracy degradation even when the level of coarsening is high.

Last, we give the results of the OGB datasets in Table 4. These two datasets have much higher node, edge, and class count than others. According to Table 4, GraphZoom still achieves comparable accuracy compared to the baseline. And the speedup also increases super-linearly with increased coarsening levels. For the ogbn-proteins graph, it even increases quadratically. Hence, our coarsening algorithm can efficiently reduce the graph topology yet still keep its key structure even when the graph is large.

5.2 Execution Time of Spectral Coarsening

Figure 3 presents the execution time of multiple levels of spectral coarsening algorithm on an 1.4GHz quad-core Intel Core i5 processor. We only depict the results of Cora and Pubmed dataset. Citeseer is very similar to Cora since they have a similar number of edges. From the plot, we see the execution time increases only linearly with the level count. Thus our algorithm is scalable with deep levels. Besides, by comparing Figure 3 with Table 3, we find that spectral coarsening takes up only no more than 1% of the total CPU time of GraphZoom. Therefore, the overhead of applying our spectral graph coarsening is negligible.

Additionally, we present the speedups of execution time on a state-of-the-art CMOS-based PIM engine with respect to the CPU, using a cycle-approximate system-level simulator that is widely used in computer systems research [38]. The key component of the engine is an array of SRAM banks with some peripherals, which can act simultaneously as a 4MB memory and a vector processing engine with 32 vector names and 32,768 lanes per vector[1]. The area of the engine is comparable to a single

[1]This work is currently under submission, so we cannot provide a citation or further details

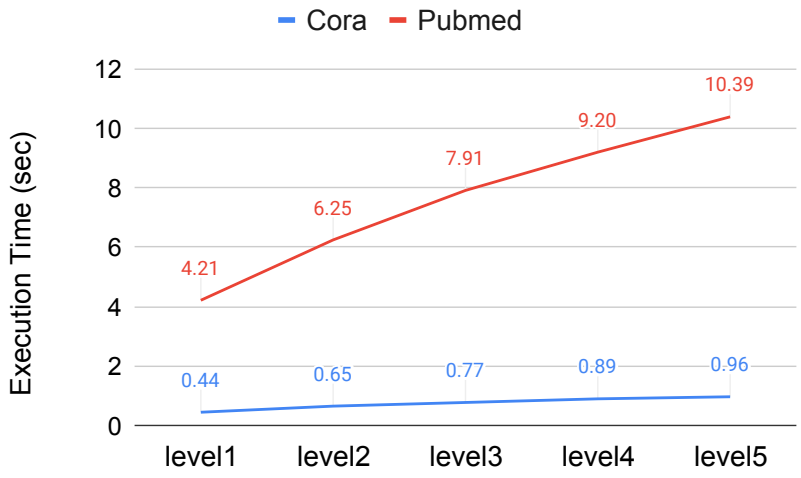


Figure 3: Execution time of different levels of spectral coarsening.

state-of-the-art out-of-order core. And we use the same main memory system as the CPU results for this part of the experiment (DDR3-2133, 17GBps peak bandwidth). We implemented an alternative version of our algorithm using C mixed with RISC-V instructions with its vector extension [18], and compile it using a modified version of RISC-V toolchain [39][2], so as to run on the simulator. For Cora, Citeseer, and Pubmed, the 1-level spectral coarsening achieves $\mathbf{16.4}\times$, $\mathbf{12.8}\times$, and $\mathbf{3.9}\times$ speedups with respect to the CPU time. The speedup is ascribed to two reasons. First, the PIM engine can perform 32k arithmetic operations simultaneously in several to three thousands of cycles. Thus it has a higher throughput than the conventional CPU when there is large data-level parallelism. Second, the PIM engine fetches data from DRAM with a much higher footprint compared to the last level cache of the CPU. Therefore, it can better utilize the memory bandwidth even with the same DRAM configuration. The speedup shown in our experiment proves that our spectral coarsening algorithm contains massive data-level parallelism. And it is feasible to program using lower-level programming language. Thus, our algorithm is a good candidate for the emerging PIM architectures.

6 Conclusion

This work introduces an efficient spectral coarsening algorithm and integrates it with GraphZoom, a multi-level framework for improving the accuracy and scalability of unsupervised graph embedding tasks. GraphZoom first fuses the node attributes and topology of the original graph to construct a new weighted graph. It then employs our efficient spectral coarsening algorithm to generate a hierarchy of coarsened graphs, where embedding is performed on the smallest graph at the coarsest level. Afterwards, proper graph filters are used to iteratively refine the graph embeddings to obtain the final result. Experiments show that spectral coarsening 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 our approach to support supervised graph embedding.

References

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

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

[2]The specific version we use is from a group in Cornell and is confidential

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

[4] 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*, pages 701–710, 2014.

[5] 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*, pages 855–864, 2016.

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

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

[8] Daokun Zhang, Jie Yin, Xingquan Zhu, and Chengqi Zhang. Network representation learning: A survey. *Transactions on Big Data (TBD)*, 2018.

[9] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Neural Information Processing Systems (NeurIPS)*, pages 1024–1034, 2017.

[10] Zhenyu Qiu, Wenbin Hu, Jia Wu, ZhongZheng Tang, and Xiaohua Jia. Noise-resilient similarity preserving network embedding for social networks. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pages 3282–3288. AAAI Press, 2019.

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

[12] Chenhui Deng, Zhiqiang Zhao, Yongyu Wang, Zhiru Zhang, and Zhuo Feng. Graphzoom: A multi-level spectral approach for accurate and scalable graph embedding. In *International Conference on Learning Representations*, 2020.

[13] Charles Eckert, Xiaowei Wang, Jingcheng Wang, Arun Subramaniyan, Ravi Iyer, Dennis Sylvester, David Blaaauw, and Reetuparna Das. Neural cache: Bit-serial in-cache acceleration of deep neural networks. *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, Jun 2018.

[14] Shuangchen Li, Dimin Niu, Krishna T. Malladi, Hongzhong Zheng, Bob Brennan, and Yuan Xie. Drisa: A dram-based reconfigurable in-situ accelerator. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-50 '17, page 288–301, New York, NY, USA, 2017. Association for Computing Machinery.

[15] Mohsen Imani, Saransh Gupta, Yeseong Kim, and Tajana Rosing. Floatpim: In-memory acceleration of deep neural network training with high precision. In *Proceedings of the 46th International Symposium on Computer Architecture*, ISCA '19, page 802–815, New York, NY, USA, 2019. Association for Computing Machinery.

[16] Junwhan Ahn, Sungpack Hong, Sungjoo Yoo, Onur Mutlu, and Kiyoung Choi. A scalable processing-in-memory accelerator for parallel graph processing. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, pages 105–117, 2015.

[17] L. Nai, R. Hadidi, J. Sim, H. Kim, P. Kumar, and H. Kim. Graphpim: Enabling instruction-level pim offloading in graph computing frameworks. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 457–468, 2017.

[18] Andrew Waterman, Yunsup Lee, David A. Patterson, and Krste Asanovic. The risc-v instruction set manual. volume i user-level isa. `https://www.amd.com/en/products/cpu/amd-epyc-7502`, 2014.

[19] Amr Ahmed, Nino Shervashidze, Shravan Narayanamurthy, Vanja Josifovski, and Alexander J Smola. Distributed large-scale natural graph factorization. *International Conference on World Wide Web (WWW)*, pages 37–48, 2013.

[20] Shaosheng Cao, Wei Lu, and Qiongkai Xu. GraRep: Learning graph representations with global structural information. *International on Conference on Information and Knowledge Management (CIKM)*, pages 891–900, 2015.

[21] Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. Asymmetric transitivity preserving graph embedding. *International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 1105–1114, 2016.

[22] Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Kuansan Wang, and Jie Tang. Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec. *International Conference on Web Search and Data Mining (WSDM)*, pages 459–467, 2018.

[23] Petar Veličković, William Fedus, William L Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. Deep graph infomax. *arXiv preprint arXiv:1809.10341*, 2018.

[24] Haochen Chen, Bryan Perozzi, Yifan Hu, and Steven Skiena. HARP: Hierarchical representation learning for networks. *The AAAI Conference on Artificial Intelligence (AAAI)*, 2018.
[25] Wenqing Lin, Feng He, Faqiang Zhang, Xu Cheng, and Hongyun Cai. Effective and efficient network embedding initialization via graph partitioning. *arXiv preprint:1908.10697*, 2019.
[26] Guoji Fu, Chengbin Hou, and Xin Yao. Learning topological representation for networks via hierarchical sampling. *arXiv preprint:1902.06684*, 2019.
[27] 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:1812.08972*, 2018.
[28] Esra Akbas and Mehmet Aktas. Network Embedding: on compression and learning. *arXiv preprint:1907.02811*, 2019.
[29] Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. *The AAAI Conference on Artificial Intelligence (AAAI)*, 2018.
[30] 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. *Signal Processing Magazine (SPM)*, 30(3):83–98, 2013.
[31] Felix Wu, Tianyi Zhang, Amauri Holanda de Souza Jr, Christopher Fifty, Tao Yu, and Kilian Q Weinberger. Simplifying graph convolutional networks. *International Conference on Machine Learning (ICML)*, pages 6861–6871, 2019.
[32] Qimai Li, Xiao-Ming Wu, Han Liu, Xiaotong Zhang, and Zhichao Guan. Label efficient semi-supervised learning via graph filtering. *The Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9582–9591, 2019.
[33] Oren E Livne and Achi Brandt. Lean Algebraic Multigrid (LAMG): Fast graph laplacian linear solver. *SIAM Journal on Scientific Computing (SISC)*, 34(4):B499–B522, 2012.
[34] Jie Chen and Ilya Safro. Algebraic distance on graphs. *SIAM Journal on Scientific Computing (SISC)*, 33(6):3468–3490, 2011.
[35] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks, 2016.
[36] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868*, 2018.
[37] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk. *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '14*, 2014.
[38] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. The gem5 simulator. *SIGARCH Computer Architecture News*, 2011.
[39] risc-v-gnu-toolchain. `https://github.com/riscv/riscv-gnu-toolchain`. Accessed: 2020-05-21.