Linear Complexity Framework for Feature-Aware Graph Coarsening via Hashing

Anonymous Author(s)
Affiliation
Address
email

Abstract

Large-scale graphs are increasingly common in various applications, leading to significant computational challenges in data processing and analysis. To address this, coarsening algorithms are employed to reduce graph size while preserving key properties. However, existing methods for large-scale graphs are computationally intensive, undermining the coarsening goal. Additionally, real-world graphs often contain node-specific features or contexts, which current coarsening approaches overlook, focusing solely on structural information like adjacency matrices. This limitation may not suit downstream tasks reliant on node features. In this paper, we introduce a **F**eature-**A**ware graph **C**oarsening algorithm via **H**ashing, called FACH, inspired by locality sensitive hashing to coarsen the graph based on the node features. To our knowledge, this is the first-ever method that coarsens a graph with node features in linear time. FACH is over 7× faster than the quickest and around 150× faster than the existing techniques for datasets like Coauthor Physics which has 34,493 nodes. We also demonstrate the efficacy of the proposed framework in terms of superior run-time complexity. The coarsened graph obtained by our method also preserves the spectral properties of the original graph while achieving massive improvement in time-complexity of coarsening which is the primary goal of our study. We showcase the effectiveness of FACH for the downstream task by evaluating the performance on scalable training of graph neural networks using coarsened data on benchmark real-world datasets.

1 Introduction

Graphs are a useful representational tool for a wide range of natural and man-made structures. They can be used to represent a wide variety of relations and process dynamics in physical, biological, and social systems as well as in computer science. Increasingly the applications of graphs in different domains are increasing like maps, transport networks, social network graphs, citation networks, etc. These graphs in such varied applications are also ever-growing. As a result, it is only fair to look for ways to simplify the graph with the least information loss [1]. Coarsening is one of the most widely used approaches for reducing graphs. Instead of solving a vast graph problem in its native domain, coarsening entails solving a similar problem of a smaller size at a lower cost. It is used in a number of algorithms for partitioning [2, 3, 4, 5, 6] and visualizing big graphs in a computationally efficient manner [7, 8]. It's also been used to make coarse-grained diffusion maps [9], multi-scale wavelets [10], and pyramids [11], which are multi-scale representations of graph-structured data. Coarsening has also been used as a pooling operation in graph convolutional networks [12]. The spatial size of each layer's output is reduced by combining the values of nearby vertices, preventing overfitting and facilitating representations' hierarchical scaling.

Submitted to 37th Conference on Neural Information Processing Systems (NeurIPS 2023). Do not distribute.

Real-world nodes are typically distinguished by a number of different characteristics in addition to their connections. In social networks, nodes' attributes might include a person's age, gender, or nationality; in protein-interaction networks, nodes' attributes might include the quantity of proteins in a cell participating in those networks; or nodes' attributes might include the location of airports connected by directed flights. Integrating information about the features of nodes is crucial in performing many operations on graphs. However, the vast majority of graph coarsening algorithms [13, 14, 5, 15, 16] depend solely on the structural information of the graph and do not make use of the node attributes throughout the coarsening process. All these methods cannot be straightforwardly extended to incorporate the node information from the graph. Therefore, when employing graph coarsening for the purpose of learning enriched smaller representations, it is essential to incorporate the node information. Moreover, the computational cost of the existing coarsening methods is high as they are dependent on the graph's adjacency matrix as input. Furthermore, the existing works cannot generalize well to the unseen/new nodes since these methods use the adjacency matrix. In this paper, we present a novel feature-aware graph coarsening algorithm, FACH, for graphs that exploits the structure of the graphs using node features. Our method is motivated by locality-sensitive hashing (LSH) to address the issues mentioned above. We first cluster the nodes of the graph based on node features using the principles of locality sensitive hashing [17] such that nodes having similar features get grouped together which also allows us to coarsen the graph to any desired coarsening ratio and then use the adjacency matrix of the graph to learn the edges that connect those clusters (supernodes). Since we start with node features, we get the merit of handling large nodes at once to cluster which is done linearly and then to learn the edges connecting the supernodes we perform a matrix multiplication which is a one time process. This makes our algorithm faster.

Our Main contributions are summarized below:

- We proposed a novel linear time complexity framework that is extremely fast compared to other state-of-the-art methods for graph coarsening.
- In our framework, node features are taken into account to coarsen the graph, and the adjacency matrix of the original graph is used to rewire the edges between these nodes. The suggested framework is essentially a first linear time method that takes into account both node attributes and the structure of the original graph.
- The proposed framework is also shown to be helpful for graph-based downstream tasks. We have benchmarked our framework on standard graph classification instances, demonstrating its effectiveness with extensive computational experiments. Results on classification tasks using GCN show the efficacy of the proposed framework.

2 Background

In this section, we give a brief background of graphs, graph coarsening, and some properties of the original graph to be preserved in the coarsened graph. Then we give background about locality-sensitive hashing and describe the existing related graph coarsening algorithms and their limitations.

2.1 Graph and Graph Coarsening

A graph is denoted by $G = (V, E, A, X)$ where $V = \{v_1, v_2, \cdots, v_{|k|}\}$ is the set of k vertices , E is set of edges (v_i, v_j) which is a subset of $(V \times V)$. $A \in \mathbb{R}^{k\times k}$ is the adjacency matrix, non-zero entries of A i.e A_{ij} denotes a edge between node i and j in the graph. We also have a feature matrix $X \in \mathbb{R}^{k\times d}$, every row of X is a feature vector $X_i \in \mathbb{R}^d$ associated with ith node of G. A degree matrix D with diagonal entries $D_{ii} = \sum_j A_{ij}$ denotes the degree of nodes of G. Along with the adjacency matrix, we also have Graph Laplacian L matrix. L is defined as $D - A$ [18] . Matrix $L \in \mathbb{R}^{k\times k}$ belongs to the following set: $S_L = \left\{ L \in \mathbb{R}^{k\times k} | L_{ij} = L_{ji} \leq 0 \forall i \neq j; L_{ii} = -\sum_{j\neq i} L_{ij} \right\}$.

Through graph coarsening, we want to reduce our input Graph $G(V, E, X)$ which has p-nodes and $X \in \mathbb{R}^{p\times d}$ into a new graph $G_c(\tilde{V}, \tilde{E}, \tilde{X})$ which has k-nodes and $\tilde{X} \in \mathbb{R}^{k\times d}$ where $k << p$ nodes. The graph coarsening problem can be posed as a problem where we want to learn a coarsening matrix $C \in R^{k\times p}$, which is a linear mapping from $V \to \tilde{V}$. The linear mapping ensures that the similar

nodes in G are mapped to a super-node in G_c such that $\tilde{X} = CX$. Every non-zero entry in C_{ij} in C denotes mapping of i^{th} node to j^{th} supernode.

This coarsening matrix belongs to the set

$$S = \left\{ C \in \mathbb{R}^{p \times k}, \langle C_i, C_j \rangle = 0, \forall i \neq j, \langle C_l, C_l \rangle = d_i, ||C_i||_0 \geq 1 \right\} \tag{1}$$

where d_i is degree of i^{th}-node. $\langle C_i, C_j \rangle = 0$ will make sure that every node in G is only mapped to one single super-node. Also, every super-node should have at least one node in it, $||C_i||_0 \geq 1$ term looks after it.

2.2 Preserving properties of G in coarsened Graph.

It is desired that coarsened graph G_c is similar to the original graph. There are some graph similarities that are widely used in literature [16] to describe this measure of similarity which we describe below:

Spectral Similarity. This similarity measure, also called Relative Eigen Error (REE), also used in papers [16, 19] gives the means to quantify the measure of the eigen properties of the original graph that are preserved in the coarsened graph. REE is defined as: $REE(L, L_c, k) = \frac{1}{k} \sum_{i=1}^{k} \frac{|\tilde{\lambda_i} - \lambda_i|}{\lambda_i}$, where λ_i and $\tilde{\lambda_i}$ are top k eigenvalues of Laplacian original (L) and Laplacian coarsened graph(L_c) matrix respectively. For the best approximation, we need REE to be close to zero.

Hyperbolic error (HE). It is reasonable to think about how relevant linear operators treat the same input vector when contracting a graph G to its reduced form G_c. Such a measure is given by hyperbolic error defined as

$$HE = arccosh\left(\frac{||(L - L_{\text{lift}})X||_F^2 ||X||_F^2}{2trace(X^T L X) trace(X^T L_{\text{lift}} X)} + 1\right) \tag{2}$$

where L and $X \in \mathbb{R}^{k \times d}$ are the Laplacian, and X is the feature matrix of the original input graph, and L_{lift} is the lifted Laplacian matrix defined as $L_{\text{lift}} = C^T L_c C$ where $C \in \mathbb{R}^{k \times p}$ is the coarsening matrix and L_c is the Laplacian of coarsened graph. The lifted Laplacian matrix is the reconstruction of the original space from the coarsened space.

REE value indicates the similarity between the eigenspace of G and G_c. A low value of REE is desired for higher spectral similarity. HE indicates the structural similarity between G and G_c with the help of a lifted matrix along with the feature matrix X of the original graph. Even though each of these characteristics has a unique sense of similarity, coarsening is better when these error levels are lower across all of them. Extensive results have been shown in Section 4.

2.3 Locality Sensitive Hashing

Locality sensitive hashing (LSH) has been widely used for efficient similarity search methods for higher dimensional data [20, 21, 22, 23]. LSH aims to map a higher dimensional vector to a representation in lower dimension space by ensuring that the probability of two vectors colliding is equal to their similarity under the given measure. LSH is this defined as a distribution on a family H of hash functions that operate on vectors such that for any two vectors in the vector collection, u, v, $P_{r_h \in H}[h(u) = h(v)] = sim(u, v)$ where sim(u,v) is a similarity measure defined on the collection of vectors. The LSH algorithm is parameterized by k and L where $k << d$, d is the dimension of the vectors in the original space, k is the reduced dimension of vectors, and L is the number of functions randomly, and independently sampled from H. The larger the k is, the higher the precision. For the recall to be higher, the LSH algorithm independently and randomly samples L functions from H. The data is now replicated in L hash tables, where each vector is mapped to L buckets. Upon query, the search is carried out in L buckets, increasing recall at the expense of more processing and storage. In simpler terms and for Euclidean distance as the similarity measure, LSH involves scalar projection given by $h_i(x) = \left\lfloor \frac{<x, a_i> - b_i}{w} \right\rfloor$, where a_i is selected at random from a probability distribution, for example, sampled from Gaussian distribution $\mathcal{N}(0, 1)$, x is the data in higher dimensional space,

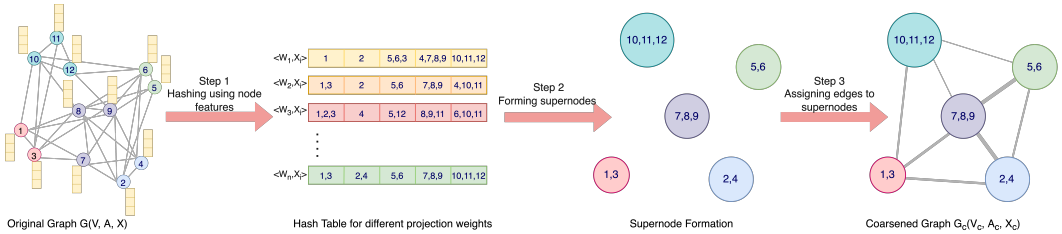


Figure 1: The figure shows an illustration of our proposed graph coarsening algorithm FACH that aims to learn a reduced graph $G_c(V_c, A_c, X_c)$ from the larger original graph $G(V, A, X)$. In Step (1), we hash the node features of the graph and assign their bins. In Step (2), we cluster the graph nodes occurring in the same bin to form supernodes. After forming the supernodes, the edge weights are assigned by aggregating the weight of the cross edges in the original graph G as seen in Step (3).

w is the width (bin-width) of each quantization bin and offset b_i is a random variable uniformly distributed between 0 and w. Because of the clustering nature of LSH, it has been widely used for image retrieval, similarity search algorithms, duplicate webpage detection, etc.

2.4 Related Works

Several graph reduction methods aim to decrease the graph size by reducing the number of nodes through vertex selection, re-combination schemes, or aggregation. Loukas proposed advanced spectral graph coarsening algorithms based on local variation to preserve the spectral properties of the original graph [19]. Two variants, edge-based local variation and neighborhood-based local variation, select contraction sets with small local variation in each stage. However, these methods have limitations in achieving arbitrary levels of coarsening [19]. Another technique, heavy edge matching (HEM), determines the contraction family by computing a maximum-weight matching based on the weight of each contraction set [5, 15]. The Algebraic Distance method calculates the weight of each candidate set using an algebraic distance measure [15, 14]. The affinity method, inspired by algebraic distance, uses a vertex proximity heuristic [13]. Kron reduction method selects a set of vertices based on the positive elements of the final eigenvector of the Laplacian matrix [24], but it suffers from high time complexity for large networks. These methods do not consider node features during graph coarsening and often require large computing memory. To address these limitations, we propose a method that efficiently utilizes node features to coarsen the graph while maintaining computational speed.

3 Proposed Feature Aware Coarsening via Hashing

In this section, we formalize our problem setting and introduce the notation followed in the paper. Then we describe our algorithm, the parameters of our method, and time complexity analysis.

3.1 Problem Formulation

The input is a graph $G = (V, A, X)$, where $V = \{v_1, v_2, \cdots, v_{|N|}\}$ is the set of vertices, $A \in \mathbb{R}^{|N|\times|N|}$ is the adjacency matrix and $X \in \mathbb{R}^{|N|\times d}$ is the node feature matrix with d being the dimension of node feature, and the $i-$th row of X denotes the feature vector of node v_i in the graph G. Our goal is to construct an appropriate "coarser" graph $G_c = (V_c, A_c, X_c)$, such that the set of vertices of the coarsen graph $|V_c| << |V|$ and the principal eigenvalues and eigenspaces of Laplacian matrices of the original and coarsened graph are comparable. The coarsening ratio is defined as $r = 1 - \frac{n}{N}$ where N is the number of nodes in the original graph G and n is the number of the nodes in the coarsened graph G_c. The first step is to define a surjective mapping using hashing $\pi : V \to V_c$ such that for any node $v_c \in V_c$, all nodes $\pi^{-1}(v_c) \subset V$ are mapped to this supernode v_c in the coarsened graph G_c. Next, we define the edge weights, A_c, of the coarsened graph equal to the sum of weights of crossing edges in the original graph. Similarly, we define the features of the supernodes, v_c, based on the features of all nodes $\pi^{-1}(v_c) \subset V$ that maps to this supernode v_c in the G_c according to the defined surjective mapping, π. Figure 1 represents the overview of FACH. The goal is to group together the graph nodes by applying projection hashing on their features such that the nodes that are close in feature space are grouped together in the same bins upon hashing.

Algorithm 1 FACH: Feature-Aware Graph Coarsening via Hashing

Require: Input $G(V, A, X)$, $L \leftarrow \#$ of Projectors, $h \leftarrow$ bin-width, $N \leftarrow \#$ of nodes in graph G
Ensure: Coarsen Graph $G_c(V_c, A_c, X_c)$
for every projector $\ell \in \{1, 2, ..., L\}$ **do**
    $h^\ell \leftarrow N$ size array for Hash indices
    $P^\ell \leftarrow d$ dimensional Projection weight, $b^\ell \leftarrow$ Scalar for generated bias
end for
for $\ell \in \{1, 2, ..., L\}$ **do**
    $P_i^\ell \sim U[0, 1]\ \forall i \in \{1, 2, 3, ..., d\}$
    $b^\ell \sim U[-h, h]$
    $h_i^\ell \leftarrow \lfloor \frac{1}{h} \times (\sum_{j=1}^{j=d} (X_i^j \times P_j^\ell) + b^\ell) \rfloor\ \forall i \in \{1, 2, 3, ..., N\}$
end for
$h \leftarrow N$ size array for Supernode indices.
$h_i \leftarrow$ maxOccurence$\{h_i^\ell; \ell \in \{1, 2, 3, ..., L\}\} \forall i \in \{1, 2, 3, ..., N\}$
$n \leftarrow \#$ of distinct hash indices/# of nodes in the coarsen graph G_c.
$\pi \leftarrow$ Dictionary mapping every node in G to supernode $\in \{1, 2, ..., n\}$ in G_c
$C \in R^{n \times N} \leftarrow$ Coarsening Matrix
for every node v in V **do**
    $C[v, \pi[v]] \leftarrow 1$
end for
$A_c(i, j) \leftarrow \sum_{(u \in \pi^{-1}(\tilde{v}_i), v \in \pi^{-1}(\tilde{v}_j))} A_{uv}, \forall i, j \in \{1, 2, ..., n\}$
$X_c(i) \leftarrow \frac{1}{|\pi^{-1}(\tilde{v}_i)|} \sum_{u \in \pi^{-1}(\tilde{v}_i)} X_u, \forall i \in \{1, 2, ..., n\}$
return $G_c(V_c, A_c, X_c)$, n

3.2 Construction of Surjective Mapping

Let $X_i \in \mathbb{R}^d$ represent the graph node feature corresponding to node v_i. Let $P \in \mathbb{R}^{L \times d}$ and $b \in \mathbb{R}^L$ be the hashing matrices employed in the method, with L denoting the number of projectors used for clustering the graph nodes. This is shown as Step 1 in Figure 1. The hash index that has the maximum occurrence among the hash indices generated by the hash functions is the hash value assigned to the graph node. So, the hash value for node v_i is given by $h_i = \max\{\lfloor \frac{1}{h} * (P \cdot X_i + b) \rfloor\}$, where h is a hyperparameter called the bin-width. The hyperparameter h controls the size of the coarsened graph G_c and "$\cdot$" represents matrix multiplication between matrix P and the feature vector X_i. It is found out empirically that increasing the value of h decreases the size of the G_c. Now, all the nodes that have been assigned the same hash value refer to the same supernode in the G_c as also shown in Step 2 of Figure 1. And if we assign an index to the cluster of nodes of the G_c, we have the set $\{\tilde{v}_1, \tilde{v}_2, ..., \tilde{v}_n\}$ where n is the number of nodes in the G_c.

3.3 Construction of Coarsened Graph

Let $G_c = (V_c, A_c, X_c)$ represent the coarsened graph that is to be built. Now, any of the supernodes, say $\tilde{v_i}$ and $\tilde{v_j}$, in the coarsened graph G_c are connected to each other, if any of the nodes say $u \in \pi^{-1}(\tilde{v_i})$ has an edge to any of the nodes, say $v \in \pi^{-1}(\tilde{v_j})$ in the original graph, i.e., $\exists\ u \in \pi^{-1}(\tilde{v_i}), v \in \pi^{-1}(\tilde{v_j})$ such that $e_{u,v} \in E$ where E is the edge set of the graph G. The coarsened graph G_c is weighted, and the weight assigned to the edge between nodes $\tilde{v_i}$ and $\tilde{v_j}$, a parameter $A_c(ij) = \sum_{(u \in \pi^{-1}(\tilde{v_i}), v \in \pi^{-1}(\tilde{v_j}))} A_{uv}$, where A_{uv} refers to the element (u, v) in the adjacency matrix A of graph G, is incorporated, which reflects the strength of the connection between them. The features of super-nodes are taken to be the average of the features of the nodes in the super-node, i.e., $X_c(i) = \frac{1}{|\pi^{-1}(\tilde{v_i})|} \sum_{u \in \pi^{-1}(\tilde{v_i})} X_u$. The supernode's label is chosen as the class that has the most instances. The partition can be represented by a matrix $\tilde{C} \in \{0, 1\}^{N \times n}$, where N is the number of nodes in the original graph and $\tilde{C}_{ij} = 1$ if and only if vertex v_i in the original graph belongs to cluster $\tilde{v_j}$. Also, note that any node in G can only be assigned a single h_i value, i.e., each node has a single supernode mapping and bin-width hyperparameter (h) ensures that every supernode has at least one node; more details about h is presented in the forthcoming section. Thus, each row of $\tilde{C}$ contains exactly one nonzero entry, and columns of $\tilde{C}$ are pairwise orthogonal. Hence our partition

matrix satisfies constraints described in the equation 1. From this C matrix, we can calculate the adjacency A_c matrix of G_c. Because each super-edge combines multiple edges from the original graph, the number of edges in the coarse graph is also much less than m. It means that the adjacency matrix A_c has a substantially smaller number of non-zero elements than A. The pseudo-code for FACH is listed down in Algorithm 1.

3.4 Relation between Bin-width Parameter h and Coarsening Ratio r

The parameter bin-width h decides the size of the coarsened graph G_c. For a particular coarsening ratio r, we find the corresponding h by divide and conquer approach on the real axis, which is similar to binary search. Algorithm 2 in the Appendix shows the method by which we find the h for any given r for G_c. Figure 5 in the Appendix shows the relation of h with r for two datasets: Cora & Coauthor CS. It is observed that the r decreases as the h increases. The h for any r can be obtained by running a bin-width finder as stated in Algorithm 2. For each dataset Bin-width finder is a hyper-parameter that needs to be run only once, and hence it is not included in the reported time complexity.

3.5 Time Complexity Analysis of FACH

FACH offers notable advantage of linear time complexity, $O(NLd)$, where N is the number of nodes, L is the number of projectors used in constructing the partition matrix C, and d is the feature dimension. The algorithm 1 performs three passes over the graph nodes. In the 1^{st} pass, nodes are individually hashed to bins based on their features, taking $O(N)$ operations. The overall time for initializing the weight matrix is $L \times d$. The 2^{nd} pass involves constructing the supernodes for the G_c using node accumulation in the bins. These two phases contribute to the time complexity of $O(NLd) \equiv O(NC)$, where C is a constant. Obtaining the partition matrix takes $O(N)$ time. In the 3^{rd} phase, the features of the coarsened graph's supernodes are computed. Edge weights are calculated by iterating over the edges of the original graph and incrementing the corresponding edge's weight between supernodes in G_c using the surjective mapping $\pi : V \to V_c$. The computational cost for this operation is $O(m)$, with m is the number of edges in the original graph.

4 Experiments

In this section, we conduct extensive experiments to evaluate the proposed FACH against the existing graph coarsening algorithms. The experiments are unfolded by answering the following questions: (i) How does FACH perform against other algorithms in terms of run-time? (ii) Is FACH able to preserve the spectrum of the original graph in the coarsened graph, i.e., retain information in the coarsened graph? (iii) How does FACH perform for real-world graph applications when we reduce the size of the original graph under question?

4.1 Experimental Protocol

4.1.1 Baselines.

We compare our proposed algorithm with the following coarsening algorithms: two variation methods based on edges and neighborhood [16], Algebraic Distance [14], Affinity [13], Heavy Edge [5, 15] and Kron [24]. We show that time complexity wise FACH is better than all of these methods. All the experiments conducted for this research were performed on an Intel Xeon W-295 CPU and 64GB of RAM desktop using the Python environment.

4.1.2 Datasets

We perform experiments on widely-used benchmarks with class-labeled nodes, including citation networks like Cora, CiteSeer, Coauthor CS, Coauthor Physics, DBLP, and PubMed, which feature sparse bag-of-words document vectors and citation links. Additionally, we utilize FACH to preprocess large datasets such as Flickr, Reddit, and Yelp, which was previously infeasible using existing techniques. Estimating eigen error for preserving spectral features in these datasets is beyond the scope of this paper. Refer to Table 7 in the Appendix for dataset specifics.

Data/ Method	CiteSeer	Cora	CS	PubMed	DBLP	Physics	Flickr	Reddit	Yelp
Var. Neigh.	8.72	6.64	23.43	24.38	22.79	58.0	OOM	OOM	OOM
Var. Edges	7.37	5.34	16.72	18.69	20.59	67.16	OOM	OOM	OOM
Var. Cliques	9.8	7.29	24.59	61.85	38.31	69.80	OOM	OOM	OOM
Heavy Edge	1.41	0.70	7.50	12.03	8.39	39.77	OOM	OOM	OOM
Alg. Distance	1.55	0.93	9.63	10.48	9.67	46.42	OOM	OOM	OOM
Affinity GS	2.53	2.36	169.05	168.38	110.95	924.75	OOM	OOM	OOM
Kron	1.37	0.63	5.81	6.37	7.09	34.53	OOM	OOM	OOM
FACH	**0.54**	**0.29**	**2.9**	**1.12**	**1.38**	**5.9**	**4.7**	**12.49**	**136.03**

Table 2: Summary of run-time in seconds averaged over 5 runs taken by other coarsening algorithms to reduce the graph to 50 percent coarsening ratios.

Dat/ Method	CiteSeer	Cora	CS	PubMed	DBLP	Physics	Flickr	Reddit	Yelp
Var. Neigh.	0.1807	0.1211	0.2488	0.1087	0.1179	0.2737	OOM	OOM	OOM
Var. Edges	0.1363	0.1293	0.0498	0.9654	0.1355	0.0424	OOM	OOM	OOM
Var. Cliques	0.0640	0.0850	**0.0263**	1.2089	0.0826	0.0394	OOM	OOM	OOM
Heavy Edge	0.0434	0.0713	0.0467	0.8343	0.0863	0.0310	OOM	OOM	OOM
Alg. Distance	0.1117	0.1079	0.0872	0.4039	**0.0471**	0.1176	OOM	OOM	OOM
Affinity GS	0.0571	0.0950	0.0633	**0.0637**	0.0735	0.0520	OOM	OOM	OOM
Kron	**0.0287**	**0.0695**	0.0564	0.3781	0.0601	0.0641	OOM	OOM	OOM
FACH	0.3408	0.2244	0.2080	0.1791	0.1455	**0.0168**	**0.0140**	EOOM	EOOM

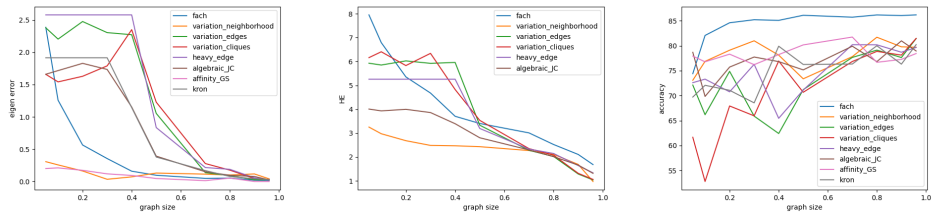
Table 3: Relative Eigen Error for other methods at 50 percent coarsening.

4.2 Experiments for Run-time analysis.

The main contribution of our algorithm is in terms of computational time. The time needed to obtain the coarsening matrix using FACH for different datasets is summarized in Table 1. It is clear from Table 1 that the run-time complexity of our algorithm is linear with respect to the coarsening ratio. Table 2 demonstrates the superiority of FACH by showing that it outperforms all existing algorithms across all datasets by significant margins. FACH is over $7\times$ faster than the quickest and around $150\times$ faster than the slowest technique for large datasets like Physics which has 34,493 nodes. While other methods fail at large datasets, FACH is able to coarsen down massive datasets like Yelp, which has 716,847 nodes, which was previously not possible. It should be emphasized that the time taken by FACH on the Reddit dataset which has $7\times$ more number of nodes compared to Physics is one-third the time taken by the fastest state-of-the-art methods on the Physics dataset.

Dataset/ Coarsening Ratio	0.1	0.3	0.5	0.7	0.9
Yelp	138.35	137.19	136.03	123.9	122.8
Reddit	12.8	12.48	12.49	12.44	12.55
Coauthor CS	2.91	2.98	2.9	2.97	2.93
Flickr	4.72	4.76	4.79	4.78	4.7
Cora	0.28	0.29	0.29	0.29	0.29
PubMed	1.13	1.12	1.12	1.13	1.1
DBLP	1.35	1.41	1.38	1.4	1.37
Physics	5.87	5.93	5.95	6.03	5.99
CiteSeer	0.52	0.55	0.54	0.54	0.52

Table 1: Summary of run-time in seconds averaged over 5 runs taken by FACH to coarsen the graph to corresponding to different coarsening ratios.



(a) Comparison of REE. (b) Comparison of HE. (c) Comparison of GCN accuracy.

Figure 2: This figure shows the comparison of all graph coarsening methods in terms of REE, HE, and GCN accuracy on the PubMed dataset. FACH's REE performance, while not the best, is comparable to other methods. FACH outperforms these methods significantly in terms of GCN accuracy.

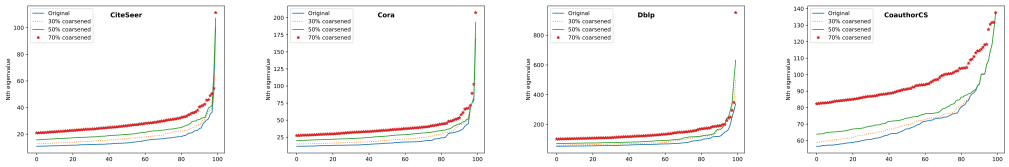
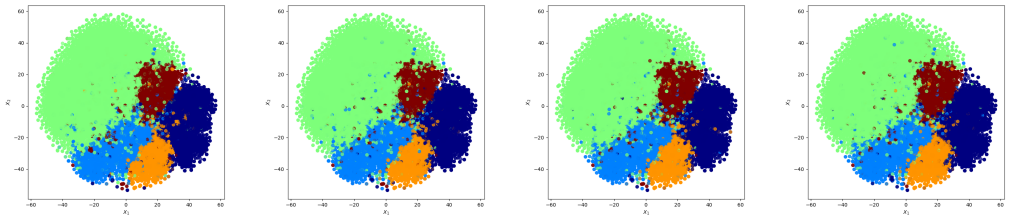


Figure 3: Top 100 eigenvalues of the original and coarsened graph at coarsening ratios: 30%, 50% and 70%. We can observe that the spectral property is maintained across all coarsening ratios for all coarsened graphs. For a lower coarsening ratio, this approximation (REE) is more accurate.



(a) Original graph. (b) 30% coarsened graph (c) 50% coarsened graph (d) 70% coarsened graph

Figure 4: Visualization of GCN predicted nodes when training is done using coarsened graph.

4.3 Spectral properties preservation.

We evaluated our coarsened graph using relative eigen error (REE) and hyperbolic error (HE) as metrics for spectral and smoothness similarity. Figure 3 demonstrates the preservation of eigenvalues, showing that even a 70% coarsened graph maintains spectral properties for most datasets. However, the accuracy of the approximation decreases as the coarsening ratio increases, leading to higher REE. Table 3 compares REE values for different approaches at a fixed 50% coarsening ratio, where FACH performs comparably. For larger datasets like Yelp and Reddit, eigen error calculation was not feasible due to memory limitations (EOOM), whereas other methods failed to generate the coarsened graph (OOM). Figure 2 and Table 6 illustrate the effect of varying coarsening ratios on eigen error, hyperbolic error, and GCN accuracy. FACH consistently achieves higher GCN accuracy than other methods for three out of six datasets. Notably, FACH achieves the highest GCN accuracy for the PubMed dataset, despite ranking third in terms of REE (Table 6). This emphasizes the need for further investigation into the relationship between GNNs, downstream tasks, and graph spectral properties. While REE values for FACH may not always be the best, Figure 2 demonstrates that FACH successfully preserves spectral properties. Considering the significant speedup offered by our framework compared to other approaches in the literature, these REE values are reasonable.

4.4 Invariant to Hash Functions.

As discussed in Section 3, we have employed dot-product i.e., $\frac{WX+B}{h}$ as the hash function for all of the above results. Here, we demonstrate FACH's compatibility with various hash functions. We have also used two other Hash functions a) ℓ_1-norm i.e $\frac{||W-X||_1}{h}$ b) ℓ_2 norm i.e $\frac{||W-X||_2}{h}$ where h is the bin-width we used to control coarsening ratio r and W and B are the randomly generated matrix. X is the feature matrix we discussed in previous sections. Table 4 summarizes GCN accuracy when we use ℓ_2 norm and ℓ_1 norm hash functions to coarsen down our original graph matrix. Since various datasets may exhibit varying types of similarities,

Dataset	ℓ_2 norm	ℓ_1 norm
Physics	94.70	93.80
CS	74.19	73.57
Cora	77.92	83.43
PubMed	85.65	85.12
DBLP	75.50	74.21
Citeseer	68.48	68.03

Table 4: This table reports the GCN accuracy on the coarsened graphs for ℓ_2 and ℓ_1 hash functions at 50 percent coarsening.

our algorithm is designed to be adaptable, so that we can incorporate any suitable hash function respecting the properties of any given dataset, encoding the similarities unique to a dataset.

4.5 Application of Coarsened Graph

4.5.1 Scalable Training of Graph Neural Networks (GNNs).

Graph neural networks (GNNs) are advanced deep learning architectures designed for handling non-Euclidean input [18, 25, 26, 27, 28], with applications spanning various domains involving graph structures [29, 30, 31, 32]. Despite their massive success, one of the biggest issues in graph machine learning is the scalability of GNNs. The receptive fields increase exponentially because the representation of a node is derived by recursively aggregating and transforming representation vectors of its nearby nodes from previous layers, rendering typical stochastic optimization strategies ineffective [27, 33, 34]. [35] proposes a generic method to apply off-the-shelf graph coarsening algorithms to scale up the training of GNNs. We experiment to check how well our proposed graph coarsening algorithm performs on the scalable training of GNNs.

Ratio/ Data	Cite-Seer	Cora	CS	Pub-Med	DBLP	Physics
30	68.48	81.63	79.54	85.82	76.0	94.8
50	66.97	74.92	74.19	85.65	75.5	94.7
70	62.27	69.76	67.29	84.82	72.5	94.43

Table 5: Accuracy of GCN on node classification after coarsening by FACH at different ratios.

4.5.2 Experimental Details.

We employed a single hidden layer GCN model with standard hyper-parameters values [18]. Coarsened data is used to train the GCN model and all the prediction is being done on original graph data. The idea is to coarsen the original graph $G(V, A, X)$ to a smaller graph $G_c(V_c, A_c, X_c)$ which is then used for training a GCN. The learned weights on G_c, are then used for making predictions on G. Table 5 lists the outcomes of the classification performance by the GCN when we coarsen down datasets with different coarsening ratios. It is evident that for most of the datasets, accuracy is on par with all the above-mentioned techniques. Even at a 70% coarsening ratio, accuracy on most of the datasets is maintained. Table 6 compares the accuracy among all the approaches with all datasets when they are coarsened down by 50%. We have used t-SNE [36] algorithm for visualization of predicted node labels shown in Figure 4. It is evident that even with 70% coarsened data training GCN model is able to maintain its accuracy. Very few of the data points are mis-classified (mostly outliers) when we increase our coarsening ratio to reduce the original graph.

5 Conclusion

In this paper, we presented a framework **FACH** for efficient graph coarsening using a hashing of node features inspired by Locality Sensitive Hashing (LSH). FACH exhibits linear time complexity, making it the fastest graph coarsening algorithm to the best of our knowledge. Our experiments on large graphs like Reddit and Yelp demonstrate its scalability and efficiency. We've also shown that FACH preserves spectrum and smoothness properties. When applied to training graph neural networks, FACH maintains performance even after substantial coarsening, enabling scalable training on complex graphs. In conclusion, FACH is a significant contribution to graph coarsening, providing a fast, efficient solution for simplifying large networks. Our future research will explore different hash functions and novel applications for the framework.

Ratio/ Data	Cite-Seer	Cora	CS	DBLP	Pub-Med	Physics
Var.Neigh.	69.54	79.75	87.90	77.05	77.87	93.74
Var.Edges	70.60	81.57	**88.74**	**79.93**	78.34	93.86
Var.Clique	68.81	80.92	85.66	79.15	73.32	92.94
Heavy Edge	**71.11**	79.90	69.54	77.46	74.66	93.03
Alg. Dis.	70.09	79.83	83.74	74.51	74.59	93.94
Aff. GS	70.70	80.20	87.15	78.15	80.53	93.06
Kron	69.00	80.71	85.35	77.79	74.89	92.26
FACH	66.97	77.92	74.19	75.50	**85.65**	**94.70**

Table 6: Accuracy of GCN after Coarsening by other methods with 50 percent coarsen ratios

References

[1] Sandeep Kumar, Jiaxi Ying, José Vinícius de Miranda Cardoso, and Daniel P Palomar. A unified framework for structured graph learning via spectral constraints. *J. Mach. Learn. Res.*, 21(22):1–60, 2020.

[2] Bruce Hendrickson and Robert Leland. A multilevel algorithm for partitioning graphs. In *Proceedings of the 1995 ACM/IEEE Conference on Supercomputing*, Supercomputing '95, page 28–es, New York, NY, USA, 1995. Association for Computing Machinery.

[3] George Karypis and Vipin Kumar. Kumar, v.: A fast and high quality multilevel scheme for partitioning irregular graphs. siam journal on scientific computing 20(1), 359-392. *Siam Journal on Scientific Computing*, 20, 01 1999.

[4] Dan Kushnir, Meirav Galun, and Achi Brandt. Fast multiscale clustering and manifold identification. *Pattern Recognition*, 39(10):1876–1891, 2006. Similarity-based Pattern Recognition.

[5] Inderjit S. Dhillon, Yuqiang Guan, and Brian Kulis. Weighted graph cuts without eigenvectors a multilevel approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(11):1944–1957, 2007.

[6] Lu Wang, Yanghua Xiao, Bin Shao, and Haixun Wang. How to partition a billion-node graph. In *2014 IEEE 30th International Conference on Data Engineering*, pages 568–579, 2014.

[7] Harel, David, Koren, and Yehuda. A fast multi-scale method for drawing large graphs. *Journal of Graph Algorithms and Applications*, 6(3):179–202, 2002.

[8] Chris Walshaw. A multilevel algorithm for force-directed graph drawing, 01 2003.

[9] S. Lafon and A.B. Lee. Diffusion maps and coarse-graining: a unified framework for dimensionality reduction, graph partitioning, and data set parameterization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(9):1393–1403, 2006.

[10] Matan Gavish, Boaz Nadler, and Ronald R. Coifman. Multiscale wavelets on trees, graphs and high dimensional data: Theory and applications to semi supervised learning. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, page 367–374, Madison, WI, USA, 2010. Omnipress.

[11] David I Shuman, Mohammad Javad Faraji, and Pierre Vandergheynst. A multiscale pyramid transform for graph signals. *IEEE Transactions on Signal Processing*, 64(8):2119–2134, 2016.

[12] Michael M. Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: Going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

[13] Oren E. Livne and Achi Brandt. Lean algebraic multigrid (lamg): Fast graph laplacian linear solver, 2011.

[14] Jie Chen and Ilya Safro. Algebraic distance on graphs. *SIAM J. Scientific Computing*, 33:3468–3490, 12 2011.

[15] Dorit Ron, Ilya Safro, and Achi Brandt. Relaxation-based coarsening and multiscale graph organization, 2010.

[16] Andreas Loukas and Pierre Vandergheynst. Spectrally approximating large graphs with smaller graphs. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, Proceedings of Machine Learning Research, 2018.

[17] Omid Jafari, Preeti Maurya, Parth Nagarkar, Khandker Mushfiqul Islam, and Chidambaram Crushev. A survey on locality sensitive hashing algorithms and their applications, 2021.

[18] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks, 2016.

[19] Andreas Loukas. Graph reduction with spectral and cut guarantees. *J. Mach. Learn. Res.*, (116), 2019.

[20] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, page 604–613, New York, NY, USA, 1998. Association for Computing Machinery.

[21] Brian Kulis and Kristen Grauman. Kernelized locality-sensitive hashing for scalable image search. In *2009 IEEE 12th international conference on computer vision*, pages 2130–2137. IEEE, 2009.

[22] Jeremy Buhler. Efficient large-scale sequence comparison by locality-sensitive hashing. *Bioinformatics*, 17(5):419–428, 2001.

[23] Venu Satuluri and Srinivasan Parthasarathy. Bayesian locality sensitive hashing for fast similarity search. *arXiv preprint arXiv:1110.1328*, 2011.

[24] Florian Dorfler and Francesco Bullo. Kron reduction of graphs with applications to electrical networks. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 60(1):150–163, 2013.

[25] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs, 2013.

[26] Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and deep graph convolutional networks, 2020.

[27] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs, 2017.

[28] Meng Liu, Hongyang Gao, and Shuiwang Ji. Towards deeper graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & amp; Data Mining*, KDD '20, page 338–348, New York, NY, USA, 2020. Association for Computing Machinery.

[29] Chang Li and Dan Goldwasser. Encoding social information with graph convolutional networks forPolitical perspective detection in news media. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2594–2604, Florence, Italy, July 2019. Association for Computational Linguistics.

[30] Aditya Paliwal, Felix Gimeno, Vinod Nair, Yujia Li, Miles Lubin, Pushmeet Kohli, and Oriol Vinyals. Reinforced genetic algorithm learning for optimizing computation graphs, 2019.

[31] Tobias Pfaff, Meire Fortunato, Alvaro Sanchez-Gonzalez, and Peter W. Battaglia. Learning mesh-based simulation with graph networks. 2020.

[32] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, jul 2018.

[33] Morteza Ramezani, Weilin Cong, Mehrdad Mahdavi, Anand Sivasubramaniam, and Mahmut T. Kandemir. Gcn meets gpu: Decoupling "when to sample" from "how to sample". In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS'20, Red Hook, NY, USA, 2020. Curran Associates Inc.

[34] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks?, 2018.

[35] Zengfeng Huang, Shengzhong Zhang, Chong Xi, Tang Liu, and Min Zhou. Scaling up graph neural networks via graph coarsening, 2021.

[36] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

A Relation between Bin-width Parameter h and Coarsening Ratio r

Algorithm 2 Bin-width Finder

Require: Input $G(V, A, X)$, $L \leftarrow \#$ of Projectors, $c \leftarrow$ Coarsening Ratio, $p \leftarrow$ precision of coarsening, $N \leftarrow \#$ of nodes in the graph G
Ensure: bin-width h
$h \leftarrow 1, ratio \leftarrow 1$
while $|c - ratio| > p$ **do**
  if ratio > c **then**
    $h \leftarrow h * 0.5$
  else
    $h \leftarrow h * 1.5$
  end if
  $\_, n \leftarrow \text{FACH}(G, L, h, N)$
  $ratio \leftarrow (1 - \frac{n}{N})$
end while
return h

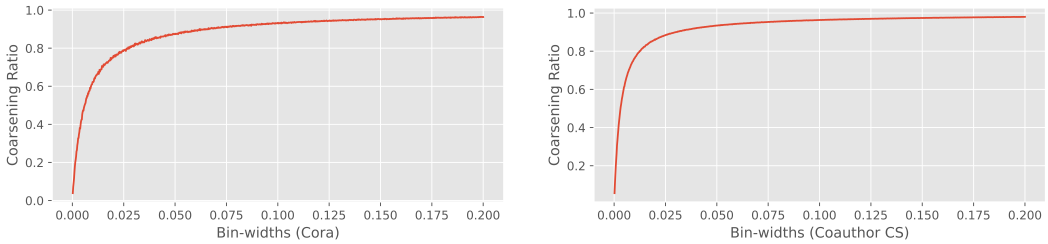


Figure 5: This figure shows the trend of coarsening ratio as the bin-width increases on two datasets: Cora and Coauthor CS.

B Datasets Summary

Dataset	Nodes	Edges	Features	Classes
Cora	2,708	5,429	1,433	7
Citeseer	3,327	9,104	3,703	6
DBLP	17,716	52,867	1,639	4
Coauthor CS	18,333	163,788	6,805	15
PubMed	19,717	44,338	500	3
Coauthor Phy.	34,493	247,962	8,415	5
Flickr	89,250	899,756	500	7
Reddit	232,965	114,615,892	602	41
Yelp	716,847	13,954,819	300	100

Table 7: Summary of the datasets.