Topology-preserving Graph Coarsening: An Elementary Collapse-based Approach

Yuchen Meng
Beijing Institute of Technology
Beijing, China
meng_yc@163.com

Rong-Hua Li
Key Laboratory of Intelligent Supply Chain Technology, Shenzhen, China
Beijing Institute of Technology
Beijing, China
lironghuabit@126.com

Longlong Lin
Southwest University
Chongqing, China
longlonglin@swu.edu.cn

Xunkai Li
Beijing Institute of Technology
Beijing, China
cs.xunkai.li@gmail.com

Guoren Wang
Beijing Institute of Technology
Beijing, China
wanggrbit@126.com

ABSTRACT

Graph coarsening techniques aim at simplifying the graph structure while preserving key properties in the resulting coarsened graph, have been widely used in graph partitioning and graph neural networks (GNNs). Existing graph coarsening techniques mainly focus on preserving cuts or graph spectrums. In this paper, we propose a new method that focuses on preserving graph topological features. In particular, we develop a novel graph coarsening approach, called Graph Elementary Collapse (GEC), by extending the concept of elementary collapse in algebraic topology to graph analysis. With this novel method, we can ensure a kind of equivalence relationship called *homotopy equivalence* of the graph during the coarsening process, thereby preserving numerous topological properties, including connectivity, rings, and voids. To enhance the scalability, we also propose several carefully-designed optimization techniques to reduce the time and memory consumption of our approach. Extensive experiments on several real-world datasets demonstrate the effectiveness and efficiency of our proposed method across various GNN prediction tasks.

PVLDB Reference Format:
Yuchen Meng, Rong-Hua Li, Longlong Lin, Xunkai Li, and Guoren Wang. Topology-preserving Graph Coarsening: An Elementary Collapse-based Approach. PVLDB, 17(13): 4760 - 4772, 2024.
doi:10.14778/3704965.3704981
PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/Shiina-Mashiro-M/GEC.

1 INTRODUCTION

Graph coarsening refers to the operation of reducing a graph to a smaller one while preserving certain graph characteristics. This technique finds applications in diverse fields including graph partitioning [20, 40] and scaling up graph neural networks (GNNs) training [23, 29]. Perhaps, the current most prevalent application of graph coarsening is to enhance the scalability of GNNs.

Indeed, GNNs have achieved remarkable performance in numerous real-world applications, including node classification, link prediction, and graph clustering [6, 12, 19, 31–33, 37, 45, 54]. However, most existing methods cannot handle massive graphs because of the prohibitively high time and space costs. For instance, the vanilla GNNs [27] rely heavily on the unwieldy message-passing mechanism between neighboring nodes for iteratively updating each node embedding vector. On top of that, the vanilla GNNs also require storing redundant intermediate node embedding vectors for subsequent training, which consumes huge GPU memory spaces. Therefore, how to extend the vanilla GNNs to massive graphs has become a hot topic for both industrial and academic communities.

Many strategies have been proposed to improve GNN's scalability. One crucial direction is to implement message-passing only between the neighbors within a sampled mini-batch, thereby reducing the receptive fields. For example, the seminal GraphSAGE [18] aggregates a limited subset of neighboring nodes, usually between 10 and 25, per target node. Subsequently, several follow-up works have attempted to enhance GraphSAGE through optimized sampling processes, improved stochastic estimations, and other extensions [7, 57]. Another related technique is subgraph sampling, where a small subgraph is carefully sampled in each training iteration, followed by full-batch training on this subgraph [10, 55]. However, performing random sampling in each epoch requires multiple random memory accesses, which is not GPU-friendly [39]. The second approach is to decouple the feature propagation and transformation by removing the intermediate nonlinear activation functions [49]. Thus, this approach can simplify the vanilla GNNs to a graph diffusion process followed by a classifier. The graph diffusion process can be pre-computed and stored, facilitating the training of the classifier using naive stochastic optimization [2]. However, the decoupling of graph diffusion and nonlinear feature transformation restricts the full expressive power of GNNs [50].

In addition to the aforementioned methods which aim to improve the scalability of GNNs at the model level, other researchers have explored a more *direct and generic* approach to further improve the scalability of GNNs by reducing the size of the graph (Section

Proceedings of the VLDB Endowment, Vol. 17, No. 13 ISSN 2150-8097.
doi:10.14778/3704965.3704981

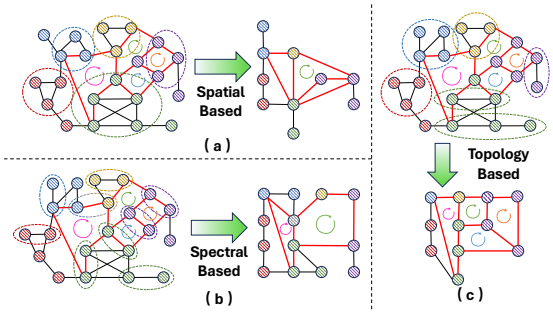

Figure 1: Illustration of different graph coarsening methods. The nodes enclosed by dashed lines in the original graph are mapped to a super node in the coarsened graph.

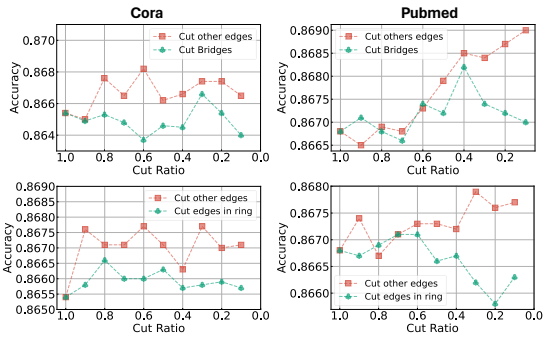

Figure 2: Experiments on topological structures.

6). By combining this approach with those above sampling-based or decoupled methods, they can achieve greater improvements in scalability [26, 56]. Among them, the graph coarsening technique has gained significant attention recently due to its nice structure properties and solid theoretical foundation [23, 42, 46, 47]. By employing graph coarsening techniques, they can generate a coarsened graph that approximately preserves some of the spectral or spatial properties of the original graph. Subsequently, they train a GNN exclusively on this coarsened graph and transfer the trained model parameters from this downsized model to the GNN defined on the original graph to perform inference.

Despite the significant success, we observe that most existing graph coarsening approaches mainly focus on specific graph properties, such as cuts, eigenvectors, or eigenvalues [23, 34] while neglecting several effective topological properties (e.g. connectivity, rings, and voids). An example is illustrated in Figure 1. For the Spatial method shown in Figure 1(a), we employed the well-known "Variational Neighborhood" [23] approach, while the "kron" algorithm [34] is picked as the Spectral method as illustrated in Figure 1(b). "Variational Neighborhood" and "kron" lost rings (formed by the red edges in the figure) during the coarsening process leading to the loss of critical topological structures. On the other hand, the coarsening result of our proposed topology-based method is illustrated in Figure 1(c), which perfectly preserves all rings. These overlooked topological structures hold significant functions. For example, in the field of algebra, they can be viewed as the basis of groups formed by data points and play a crucial role in data analysis [38]. Besides, many researchers have empirically demonstrated the importance of these topological features for downstream tasks. For example, [51, 52] demonstrated that rings can facilitate more accurate graph representation learning and relationship prediction; [1, 9, 59] showcased the powerful capability of topological structures in graph classification tasks, especially in the field of biomedical research. Furthermore, Figure 2 also provides evidence for the importance of these topological features. Specifically, in Figure 2, the "Cut Bridges" means we randomly removed a certain proportion of bridges in the graph to disrupt the connectivity between nodes; the "Cut edges in ring" means we randomly prune edges to breach a certain proportion of rings in the graph; the "Cut other edges" served as the control group, where the same number of edges were randomly removed. The training performance of the "Cut other edges" is apparently better than the results in the other two experiments , thereby demonstrating the importance of the topological structures.

Therefore, preserving the graph's topological properties during the graph coarsening may enhance performance but also bring challenges. To address this dilemma, we propose a novel **G**raph **E**lementary **C**ollapse (*GEC*) operator based on topological theory [36], which can achieve a continuous transformation on the graph, allowing the graph to maintain a kind of equivalence relationship called *homotopy equivalence* with the original graph while reducing the graph scale. Homotopy equivalence is an equivalence relationship widely employed in the field of algebraic topology that can maintain a series of homotopy type invariance, including connectivity, rings, and voids. By ensuring homotopy equivalence between the coarsened graph and the original graph during the graph coarsening process, we can achieve a more comprehensive preservation of properties within the graph.

To our knowledge, we are the first to develop a topological method for graph coarsening. Moreover, to accommodate massive graphs, we develop several carefully-designed optimization strategies for the graph elementary collapse operator. These optimizations have resulted in a significant reduction in the required runtime. In a nutshell, we highlight our main contributions as follows: **(1) Novel Perspective.** We propose a new graph coarsening method that focuses on preserving graphs' topological structures. Our method introduces a novel topological perspective, which is commonly used in the field of algebraic topology, known as homotopy equivalence to ensure the preservation of structural information in the graph. To the best of our knowledge, our approach is the first graph coarsening method that can preserve the topological features of the original graph. **(2) New Operator and its Optimizations.** With the help of topological theory, we propose a novel graph analysis operator, called graph elementary collapse, which can achieve continuous changes on the graph and guarantee the preservation of the graph's high-dimensional topological structures. Besides, we also optimize the memory and time consumption of the proposed graph elementary collapse operator to enhance its scalability for massive graphs. **(3) Extensive Experiments.** We extensively evaluate our solutions on various real-world networks, demonstrating their superiority over existing state-of-the-art graph coarsening methods across diverse applications. Notably, our method achieves 5% improvement in accuracy and 100× reduction in memory usages compared to the state-of-the-art methods.

2 PRELIMINARIES

2.1 Basic Concepts

A graph is denoted as $\mathcal{G} = (V, E)$, where $V = \{v_1, v_2, ..., v_n\}$ represents the node set and $E \in \mathbb{R}^{|V|\times|V|}$ is the edge set. Let $n = |V|$ and $m = |E|$ be the number of the nodes and edges of $\mathcal{G}$ respectively. In the context of graph learning methods, a graph dataset is represented by $\mathcal{T} = (\mathcal{G}, X, Y)$, where $X \in \mathbb{R}^{|V|\times f}$ is a matrix of node features, f denotes the dimension of feature and $Y \in \{0, ..., C-1\}^{|V|}$ denotes the label of nodes. The constants f and C are the dimensions of node features, and the number of classes respectively.

Graph coarsening can generate a smaller graph $\mathcal{G}' = (V', E')$ with $n'(<< n)$ nodes while preserving some essential characteristics of the original graph $\mathcal{G}$ as much as possible. To illustrate this process, we provide a toy example as shown in Figure 1. A distinctive characteristic of graph coarsening is that each node $v_i \in V$ is mapped to a specific node $v'_i \in V'$ and forms a super node. Previous studies on graph coarsening focus primarily on preserving some of the spectral or spatial graph properties of the coarsened graph (Section 6). In this work, we mainly focus on the problem of graph coarsening by preserving the topological features of the input graph. In the following, we introduce several important topological definitions that will be frequently used in our method.

2.2 Elementary Collapse on Simplicial Complex

Definition 2.1. (**Simplex** [13]) In the Euclidean space $\mathbb{R}^n$, for any $k > 0$, if a set P of $k + 1$ affinely independent points can form a convex hull, it is called a *k-simplex*, k is called the dimension of the simplex. For $0 \le k' \le k$, a *k'-face* of *k-simplex* σ is a *k'-simplex* that is the convex hull of a nonempty subset of P.

Definition 2.2. (**Simplicial Complex** [13]) A *simplicial complex* $\mathcal{K}$ is a set containing finitely many simplices that satisfy the following two restrictions: $\mathcal{K}$ contains every face of each simplex in $\mathcal{K}$; For any two simplices $\sigma, \tau \subseteq \mathcal{K}$, their intersection $\sigma \cap \tau$ is either empty or a face of both σ and τ.

The dimension of $\mathcal{K}$, denoted by $dim(\mathcal{K})$, is determined by the maximum dimension among the simplices in $\mathcal{K}$. When the maximum dimension of the simplices in $\mathcal{K}$ is k, we denote it as *simplicial k-complex*. Another crucial concept in our method is the free face.

Definition 2.3. (**Free Face**) Let $\mathcal{K}$ be a simplicial complex. If there are two simplices $\sigma, \tau \in \mathcal{K}$ satisfy the following two conditions: $\tau \subseteq \sigma$ and $dim(\tau) < dim(\sigma)$; σ is a maximal face (i.e. a face is not contained by any other face) of $\mathcal{K}$ and no other maximal face of $\mathcal{K}$ contains τ. then τ is called a free face.

Definition 2.4. (**Elementary Collapse**) For a maximal face $\sigma \in \mathcal{K}$, if we remove two simplices τ and σ where $\tau \subseteq \sigma$ is a free face and $dim(\tau) = dim(\sigma) - 1$, it is called an elementary collapse. We say $\mathcal{K}$ collapses to $\mathcal{L}$ if $\mathcal{L}$ can be obtained from $\mathcal{K}$ via a sequence of elementary collapse.

In the rest of this paper, the term "collapse" specifically refers to elementary collapse (Definition 2.4). Such a collapse method possesses a crucial characteristic, as stated in the following lemma.

Lemma 2.5. *(**Homotopy Equivalence** [36]) Let $\mathcal{L} \subset \mathcal{K}$ be a subcomplex collapsed from $\mathcal{K}$ via an elementary collapse, we can derive that $\mathcal{L}$ and $\mathcal{K}$ are homotopy equivalent.*

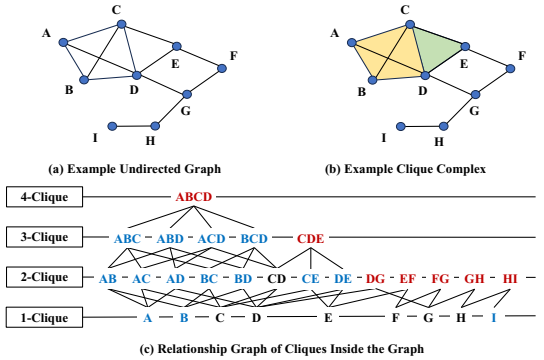

Figure 3: An example of the definitions used in our proposed graph elementary collapse method.

We now provide a simple explanation of homotopy equivalence. Homotopy equivalence refers to a type of weak equivalence observed between topological spaces. When two topological spaces are homotopy equivalent, they share certain invariant topological properties, from neighborhood relationships and global connectivity to Betti numbers, fundamental groups, and homology groups. From a graph perspective, we can roughly describe it as follows: if two graphs are homotopy equivalent, it means that any corresponding routes between the two graphs can be transformed into each other through continuous topological deformations. While the individual characteristics of specific nodes may vary between these spaces, the preserved high-dimensional features mentioned above enable the collapsed graph to retain essential information about the original graph. This preservation allows us to study the neighborhood relationships and global connectivity of the nodes by examining the collapsed graph.

3 THE PROPOSED SOLUTIONS

In this section, we first propose a novel operator called *Graph Elementary Collapse (GEC)*, which extends the abstract concepts involved in elementary collapse to graphs (Section 3.1). Then, we explain how GEC is applied to graph coarsening (Section 3.2).

3.1 Graph Elementary Collapse

In the previous section, we provided a generalized definition of Elementary Collapse. To apply them as graph coarsening and to facilitate readers' understanding of the concepts involved, we will use the Clique Complex (a specific common type of simplicial complex) for subsequent discussions. Specifically, given a graph $\mathcal{G} = (V, E)$, we can transform $\mathcal{G}$ into a clique complex $\mathcal{K}$ according to the following rules: The 0-simplices of $\mathcal{K}$ correspond to the nodes in V; The 1-simplices of $\mathcal{K}$ correspond to the edges in E; For any $k > 1$, a k-simplex σ of $\mathcal{K}$ corresponds to the *(k+1)*-clique in $\mathcal{G}$.

By defining clique complex on the graph, we can treat all $(k+1)$-cliques as k-simplices. To facilitate readers' understanding, we directly use the term 'clique' to interpret the definitions in the following paper. Next, we use several examples to illustrate how to combine the concept of cliques with the definitions in Section 2.2.

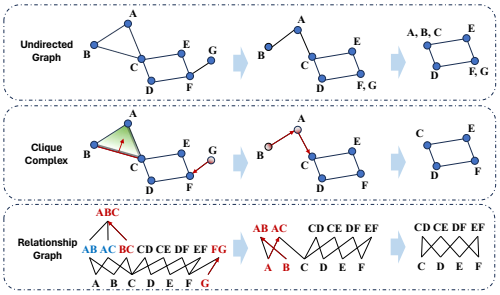

Figure 4: Example of graph elementary collapse. The "Undirected Graph" part shows the change of $\mathcal{G}$ during the collapse process. The "Clique Complex" part represents the clique complex $\mathcal{K}$ corresponding to graphs. The "Relationship Graph" shows the inclusion relationships between cliques.

Algorithm 1: Build up the Relationship Graph

Input: $\mathcal{G}$ =(V, E)
Output: The relationship graph $\mathcal{R}$ of the cliques in $\mathcal{G}$.

Initialize relationship graph $\mathcal{R} \leftarrow (V_R = \emptyset, E_R = \emptyset)$;
Initialize nodes in current clique $Q \leftarrow \emptyset$;
Initialize common neighbors of the nodes in the clique $\mathcal{N} \leftarrow V$;
$\mathcal{R} \leftarrow Building(\mathcal{G}, \mathcal{R}, Q, V)$;
Procedure $Building(\mathcal{G}, \mathcal{R}, Q, \mathcal{N})$
  Let $\mathcal{N}$ be $\{v_1, v_2, ...\}$, $\mathcal{R}$ be (V_R, E_R);
  for $i = 1$ *to* $|\mathcal{N}|$ **do**
    $Q' \leftarrow Q \cup \{v_i\}$;
    Let $\mathcal{N}_{v_i}$ be the neighbors of v_i ;
    $\mathcal{N}' \leftarrow \mathcal{N} \cap \mathcal{N}_{v_i} \setminus \{v_1, ..., v_i\}$;
    /* Update the nodes and edges in the relationship graph, (Q', Q) means that Q is a face of Q' */
    $V_R \leftarrow V_R \cup \{Q'\}$;
    $E_R \leftarrow E_R \cup \{(Q', Q)\}$;
    $\mathcal{R} \leftarrow Building(\mathcal{G}, \mathcal{R}, Q', \mathcal{N}')$;
    return $\mathcal{R}$

Example 3.1. Figure 3(a) displays an undirected graph $\mathcal{G}$ =(V, E), while Figure 3(b) illustrates the corresponding Clique Complex of graph $\mathcal{G}$. The large cliques $\{A, B, C, D\}$ and $\{C, D, E\}$ correspond to the yellow 3-simplex and the green 2-simplex in Figure 3(b). In Figure 3(c), we enumerate all cliques that exist in the graph $\mathcal{G}$. Figure 4 illustrates the process of elementary collapse on the clique complex. We can observe that not all maximal cliques can undergo collapse. For example, $\{C, D\}$ cannot collapse because its faces $\{C\}$ and $\{D\}$ are also faces of $\{A, B, C\}$ and $\{D, F\}$, respectively. In the first graph, there are two maximal cliques with free faces: $\{F, G\}$ which has only one free face $\{G\}$ and $\{A, B, C\}$ that have three free faces including $\{A, B\}$, $\{A, C\}$, and $\{B, C\}$. The cliques undergoing collapse are marked with red arrows.

Build Up the Clique Relationship. From Example 3.1, we can see that the graph elementary collapse relies heavily on the relationship graph of cliques. Therefore, we propose Algorithm 1 to construct the relationship graph of cliques. Note that Algorithm 1 is self-explanatory, so we omit its explanation for brevity.

Once the graph $\mathcal{G}$ has been transformed into a relationship graph $\mathcal{R}$, we can proceed with a series of elementary collapses on $\mathcal{R}$. During this process, we observe a simultaneous reduction in the number of nodes and edges while ensuring homotopy equivalence.

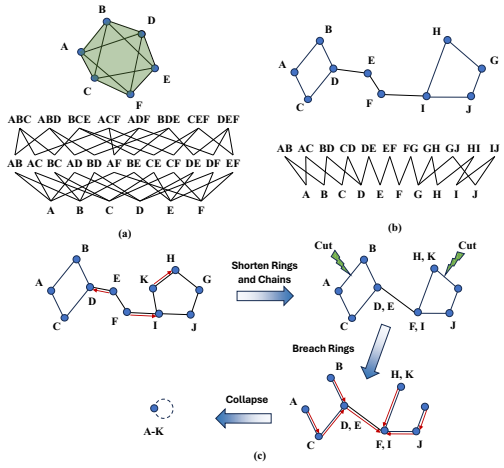

Figure 5: Illustration of possible reasons why collapse cannot proceed further and how we address this obstacle.

Additionally, since our goal is graph coarsening, during the collapse process, when we are going to remove a node from the graph, we will map it to its only neighbor, resulting in a smaller graph $\mathcal{G}' = (V', E')$ with supernodes. This process directly provides the mapping relationship which is useful for graph coarsening applications shown in the following sections. The "Undirected Graph" part in Figure 4 shows how we build up the supernode. When we are going to collapse the clique pair $\{A, B\}$ and $\{B\}$, we also map the node B to its neighbor A, forming a super node.

3.2 GEC-based Graph Coarsening

Section 3.1 has introduced the key ideas and steps of the proposed graph During the collapse process, based on the theoretical support provided by Lemma 2.5, our GEC method can effectively preserve rings and voids of the original graph, enabling more effective preservation of the graph's topological features. However, such preservation also means GEC cannot arbitrarily reduce the number of nodes and edges in $\mathcal{G}$ to any desired percentage. The following example illustrates such situations.

Example 3.2. Figure 5 illustrates situations that may prevent further collapse by two graphs that cannot be collapsed further. We can see the 2-cliques in Figure 5(b) compose rings $ABCD$ and $GHIJ$ in the graph, leading to an absence of free faces. Similarly, larger cliques can also compose such "rings", referred to as voids, which impede further collapse of the graph. Figure 5(a) shows one of such voids. It can be observed that all the 2-cliques belonging to two maximal cliques. Furthermore, as illustrated in Figure 5(b), the rings in the graph also impede the collapse of the chains connecting them. We can see although DE, EF, and FG are not included in the ring, none of them have a free face to collapse with.

Therefore, in scenarios where all nodes are either part of rings or voids, or they lie on chains connecting them, it indicates that further collapse of nodes is no longer possible. Moreover, even if there exists an optimal collapse sequence that allows collapsing down $\mathcal{G}$ to a single node, we are usually unable to quickly determine this

sequence, because determining an optimal collapse sequence of a k-complex ($k \geq 3$) is NP-hard [14]. To overcome the aforementioned obstacle, we can shorten the ring structure by collapsing nodes inside it and identify the chain-like structures between the rings or voids and collapse nodes inside them. The graph obtained through these operations also remains homotopy equivalent to the original graph, thereby further enhancing the graph coarsening capability. We also employ a strategy to disrupt the integrity of rings, voids, or chains by isolating a few edges within the graph. The information of these isolated edges can be preserved and regained once the collapse of the graph is complete if desired. By formally relaxing the restrictions of the elementary collapse operation, we can overcome the constraint mentioned above.

Example 3.3. Figure 5(c) demonstrates how we further collapse the graph when there are no nodes to collapse. This graph contains two rings and a chain between them that cannot be collapsed. To further collapse the graph, we first shorten the large ring ($IJGHK$) and the chain structure ($DEFI$) inside the graph by collapsing the nodes inside it to their neighbors. After that, we isolate two of the 2-cliques to break the rings. As a consequence, the graph can now be collapsed into a single node.

GEC on Attributed Graph. Since many real-world graphs have node features X and labels Y, we also propose a method for GEC to handle attributed graphs. Specifically, for the resulting graph $\mathcal{G}' = (V', E')$ with super nodes, the features of the super nodes are defined as the average of all the features of the nodes mapped to them, while the labels are determined as the label that occurs most frequently among all nodes mapped to the supernodes.

With the aforementioned modifications, we can now use GEC as a graph coarsening method. However, we will find that this method cannot be applied directly to large graphs at present. The primary reason for such limitation is that the relationship graph $\mathcal{R}$ can become excessively large, containing an overwhelming number of cliques that our method struggles to handle. To address this challenge, we propose two powerful strategies to limit the number of cliques within the clique complex and $\mathcal{R}$, which enable our GEC-based coarsening to be applied to large-scale graphs.

Reduce the Maximum Dimension. One intuitive approach is to limit the maximum size of the clique, which sets a lower dimension as the maximum dimension for the clique complex. Specifically, if we set the maximum dimension as d, we consider a k-clique in the graph as a *(k-1)*-simplex if $k\text{-}1 \leq d$. For k-clique with $k\text{-}1 > d$, we treat it as a void surrounded by d-simplex and will not add it into the relationship graph $\mathcal{R}$. Yet the following example illustrates that setting a low maximum dimension may hinder the collapse.

Example 3.4. Figure 6 illustrates a simple example of how a low maximum dimension hinders the collapse. When setting the maximum dimension of the clique complex to 1, the 3-clique in the graph cannot be collapsed because we consider it as a ring. But, if we change the maximum dimension to 2 or higher, we can see they all belong to a 2-simplex (shown in green), enabling further collapse of the graph. That's how reducing the value of d may result in misjudgments of the collapsibility of cliques in the graph.

Thus, in practical applications, it is crucial to select an appropriate maximum dimension that is neither too high, resulting in an excessive number of cliques in $\mathcal{R}$, nor too low, hindering the collapse process by causing misjudgments of the collapsibility.

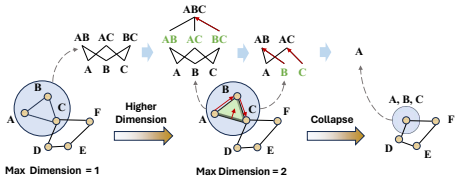

Figure 6: An example of how a low dimension may impede the graph elementary collapse.

Algorithm 2: Graph Splitting

Input: $\mathcal{G}$ =(V, E), one integer $\tilde{n}$
Output: Subgraphs $G = \{\mathcal{G}_1, ..., \mathcal{G}_g\}$

```
1  Initialize G ← ∅, num ← 1 ;
2  Let V be {v_1, v_2, ...} ;
3  for i = 1 to |V| do
4      if v_i not in any subgraph in G then
5          Initialize G_num ←(V_num = ∅, E_num = ∅) and an empty queue q ;
6          Enqueue v_i into q;
7          while q is not empty do
8              Get a node u from q ;
9              V_num ← V_num ∪ {u} ;
10             if |V_num| == ñ then
11                 break;
12             Let N_{v_i} be the neighbors of v_i;
13             for w in N_{v_i} do
14                 if w not in any subgraph in G or V_num then
15                     Enqueue w into the queue q;
16         for edge(u, w) in E do
17             if u, w ∈ V_num then
18                 E_num ← E_num ∪ {(u, w)} ;
19         G ← G ∪ {G_num} ;
20         num ← num + 1 ;
```

Constrain the Graph Size. Although we have limited the number of cliques in the graph by setting a maximum dimension, it may not be sufficient in practical applications. For instance,a graph with ten thousand nodes and 50 million edges can leads to over 50 million cliques in the relationship graph $\mathcal{R}$. To address this challenge, we propose a solution that randomly splits the graph based on its connectivity into multiple subgraphs containing at most $\tilde{n}$ nodes. Specifically, we employ the BFS coloring method (Algorithm 2) to perform graph splitting, and we find it does achieve good results as observed in our empirical results. Therefore, we directly adopt the BFS coloring method for graph partitioning instead of using more cumbersome approaches. This allows us to strike a balance between efficiency and effectiveness. Note that Algorithm 2 is self-explanatory, we omit its explanation for brevity. Once the collapse is partially complete, we can easily reconnect the subgraphs with edges between them.

With the two strategies for limiting clique quantity mentioned above, we present our GEC-based graph coarsening algorithm, as shown in Algorithm 3. Specifically, we first split the graph into subgraphs $G = \{\mathcal{G}_1, ..., \mathcal{G}_g\}$ and get the corresponding relationship graphs $R = \{\mathcal{R}_1, ..., \mathcal{R}_g\}$ with a maximum dimension d (Lines 1-2). Then we initialize the map relationship M (Lines 3-4). Subsequently, we can proceed to implement the GEC on each $\mathcal{R}_i$. Specifically, we

Algorithm 3: GEC-based Graph Coarsening (GEC-B)

Input: $\mathcal{G} = (V, E)$, the maximum dimension of clique complex d, the maximum number of nodes $\tilde{n}$ for each subgraph
Output: Coarsened graph $\mathcal{G}' = (V', E')$, map relationship M.

1 Split $\mathcal{G}$ into a collection of subgraphs $G = \{\mathcal{G}_1, ..., \mathcal{G}_g\}$ with each has at most $\tilde{n}$ nodes;
2 Get relationship graph $R = \{\mathcal{R}_1, ..., \mathcal{R}_g\}$ including at most $(d+1)$-clique of graphs in G;
3 **for** *node* u *in* V **do**
4 $M[u] \leftarrow u$;
5 **for** $\mathcal{R}_i$ *in* R **do**
6 **while** *coarsening ratio has not been achieved* **do**
7 **for** *each pair of* (j+1)*-clique* $\sigma \in \mathcal{R}_i$ *and* j*-clique* $\tau \subseteq \sigma$ **do**
8 **if** τ *and* σ *meet the conditions of collapse* **then**
9 $\mathcal{G}_i, M, \mathcal{R}_i \leftarrow Update(\mathcal{G}_i, M, \mathcal{R}_i, \tau)$;
10 $\mathcal{G}_i, M, \mathcal{R}_i \leftarrow Update(\mathcal{G}_i, M, \mathcal{R}_i, \sigma)$;
11 **if** $\mathcal{R}_i$ *unchanged during the collapse in this iteration* **then**
12 $\mathcal{G}_i, M, \mathcal{R}_i \leftarrow ShortenRingandChain(\mathcal{G}_i, M, \mathcal{R}_i)$;
13 Uniformly pick a 2-clique σ from $\mathcal{R}_i$;
14 **for** *clique* τ *that have* $\sigma \subseteq \tau$ **do**
15 $\mathcal{R}_i \leftarrow \mathcal{R}_i \setminus \{\tau\}$;
16 Let σ be $\{v_1, v_2\}$, $\mathcal{G}_i$ be (V_i, E_i) ;
17 $E_i \leftarrow E_i \setminus \{(v_1, v_2)\}$;
18 $\mathcal{G}' \leftarrow Rebuild(\mathcal{G}, G)$;
19 **Procedure** $ShortenRingandChain(\mathcal{G}, M, \mathcal{R})$
20 **for** *each node* u *that have only two neighbor* v,w **do**
21 **if** v,w *are not neighbors of each other and have only one coneighbor* **then**
22 Let σ be $\{u\}$;
23 $\mathcal{G}, M, \mathcal{R} \leftarrow Update(\mathcal{G}, M, \mathcal{R}, \sigma)$;
24 Let $\mathcal{G} = (V, E)$, $\mathcal{R}$ be (V_R, E_R);
25 $V_R \leftarrow V_R \cup \{\{v, w\}\}$, $E \leftarrow E \cup \{(v, w)\}$;
26 $E_R \leftarrow E_R \cup \{(\{v, w\}, \{v\}), (\{v, w\}, \{w\})\}$;
27 **return** $\mathcal{G}, M, \mathcal{R}$;
28 **Procedure** $Update(\mathcal{G}, M, \mathcal{R}, \sigma)$
29 Let $\mathcal{G}$ be (V, E) ;
30 $\mathcal{R} \leftarrow \mathcal{R} \setminus \{\sigma\}$;
31 **if** $|\sigma| == 2$ **then**
32 Let σ be $\{v_1, v_2\}$;
33 $E \leftarrow E \setminus \{(v_1, v_2)\}$;
34 **if** $|\sigma| == 1$ **then**
35 Let σ be $\{v_1\}$, v_2 be one of the neighbor of v_1;
36 $V \leftarrow V \setminus \{v_1\}$;
37 **for** u *have* $M[u] == v_1$ **do**
38 $M[u] \leftarrow v_2$;
39 **return** $\mathcal{G}, M, \mathcal{R}$
40 **Procedure** $Rebuild(\mathcal{G} = (\mathrm{V}, \mathrm{E}), G = \{\mathcal{G}_1, ..., \mathcal{G}_g\})$
41 Let $\mathcal{G}_i$ be (V_i, E_i), $V' \leftarrow V_1 \cup V_2 \cup ... \cup V_g$, $E' \leftarrow E_1 \cup E_2 \cup ... \cup E_g$;
42 **for** *edge* (u, w) *in* E **do**
43 **if** u, w *in different subgraph in* G **then**
44 Let u, w been mapped to u', w' respectively;
45 $E' \leftarrow E' \cup (u', w')$;
46 **return** (V', E')

check all the clique pairs in $\mathcal{R}_i$ to find whether they can be collapsed (Lines 7-10). If they can be collapsed, we remove them from $\mathcal{R}_i$ and simultaneously update the $\mathcal{G}_i$ and M (Lines 29-40). If no collapse is possible, we first shorten chains and rings in the graph (Lines 12, 19-28). After that, we uniformly pick a 2-clique from $\mathcal{R}_i$ and remove all the cliques that take it as their face (Lines 13-17). After achieving the desired coarsening ratio (the proportion of nodes in the coarsened graph relative to the original graph), we rebuild the graph with subgraphs and edges between them (Lines 18, 41-49).

Complexity Analysis of Algorithm 3. Let $\tilde{n} = \max\{|V(\mathcal{G}_i)||\mathcal{G}_i \in G\}$ be the maximum number of nodes in the subgraph set G and d be the maximum dimension. Thus, there are most $n/\tilde{n}$ subgraphs in G, where n is the number of nodes in the original graph. We know that the worst-case time complexity on each subgraph $\mathcal{G}_i$ is $O(\tilde{n}^d|E|)$, this is because there are at most $\tilde{n}^d$ cliques stored in $\mathcal{R}_i$ and each clique can be checked at most $|E|$ times. The space complexity on each subgraph $\mathcal{G}_i$ is $O(d\tilde{n}^d)$, this is beacue there are at most $\tilde{n}^d$ cliques stored in $\mathcal{R}_i$ and each k-clique store at most d $(k\text{-}1)$-cliques as its faces. As a result, the overall time complexity is $O(n\tilde{n}^{(d-1)}|E|)$ and the space complexity is $O(nd\tilde{n}^{(d-1)})$.

4 OPTIMIZATIONS

So far, we have introduced the GEC-based graph coarsening method. However, it still has several limitations when applied to massive graphs due to its prohibitive time and memory cost. Specifically, the expensive complexity arises from two key factors: (1) The maximum dimension constraint d poses difficulties in eliminating large cliques through collapse. As the coarsening ratio increases, the graph may tend to be denser (mainly because GEC eliminates many chain-like structures, leading to tighter cohesion of the cliques that cannot be collapsed), resulting in the significant proliferation of cliques. (2) After the random isolation of edges on the graph, traversing all cliques becomes necessary to determine the possibility of further collapse which leads to high time and space complexity. Therefore, we propose a more efficient collapse method, called **Bottom-Up GEC**, to further optimize the scalability of GEC.

4.1 Relationship Graph with Maximal Clique

Recall that since the limitations of elementary collapse, we need expensive space and time to enumerate all *1* to $(d+1)$ cliques and form the relationship graph. Although we partition the graph into many smaller subgraphs and limit the size of the cliques to be searched, it is still not very efficient as shown in our experiments. On the other hand, if we reduce the value of d, it would lead to misjudgments of clique collapsibility as described in Example 3.4, thereby significantly affecting the effectiveness of the proposed method. To alleviate these issues, we propose a new relationship graph that enables us to perform GEC more effectively and precisely.

Before introducing the new optimizations, we need to briefly extend the graph elementary collapse. In previous sections, we mentioned that graph elementary collapse requires two cliques τ and σ to satisfy $|\tau| = |\sigma| - 1$. However, we discover that for any k-clique τ that is a free face of maximal clique σ, even if σ is not a $(k+1)$-clique, when we remove τ and all other cliques containing τ, this operation can always be decomposed into a series of elementary collapses. This means when we want to collapse a k-clique τ, we can simply check whether τ belongs to exactly one maximal clique and then collapse τ with all the cliques that contain it. We can use maximal cliques to reconstruct a simpler relationship graph based on this new collapse approach. In this new graph, we set edges only between the k-cliques ($k = 1, ..., d + 1$) and the maximal cliques they belong to. The problem of maximal clique enumeration is challenging, which has received much attention and many fast algorithms have been developed [5, 11, 15, 25], allowing us to quickly find all maximal cliques. Here, we choose the pivote-based Bron-Kerbosch algorithm [25], one of the fastest algorithms, to enumerate the maximal cliques.

Example 4.1. Figure 7 shows how we build up the relationship graph $\mathcal{R}$ with maximal cliques when set $d = 1$. In Figure 7, the

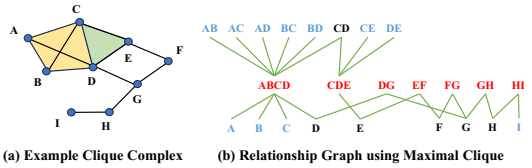

Figure 7: An example of how we build the relationship graph using only node, edge, and maximal clique.

maximal cliques are highlighted in red and we associate the maximal cliques with their faces, indicated by green lines. With this simplified relationship graph, we significantly reduce the size of the Relationship graph that needs to be stored.

Note that when using the maximal clique to build the relationship graph $\mathcal{R}$, if we use the original collapse sequence (i.e., from $(d+1)$-cliques to 1-cliques) may introduce several new problems. For example, whenever we attempt to collapse an existing k-clique τ as a free face of maximal clique σ, it leads to the splitting of σ into at most *k-1* new maximal cliques, resulting in the need to preserve an increasing number of maximal cliques. In such cases, collapsing cliques starting from $(d+1)$-cliques to 1-cliques would lead to a significant increase in the number of maximal cliques which goes against our intention to reduce the memory cost. Thus, we choose to reverse the order of collapse, starting from 1 cliques and collapsing the existing cliques. This approach aims to minimize the increase in maximal cliques caused by the collapse of larger cliques. Also, another benefit using maximal cliques to store the relationship graph is that we can accurately describe the current clique complex without the need to store all cliques. As mentioned before, setting a maximum dimension d may lead to misjudgments of the collapsibility of cliques (see Example 3.4). Taking an extreme example of a complete graph with 100 nodes and setting $d = 1$, when using maximal cliques in the relationship graph, we no longer consider the 100-clique as a series of rings where no collapse can be performed. Instead, we can easily realize each clique within it belongs to only one maximal clique, thereby easily collapsing the graph into a single node. On the other hand, even though the use of maximal cliques helps avoid misjudgments of collapsibility, it does not mean that reducing the value of d has no impact on the effectiveness. It simply makes the occurrence of negative impacts more stringent, thus reducing their frequency. Figure 8 provides an example to elaborate on such a situation. Thus, limiting d to a low value still results in potential loss of collapse for certain nodes or edges. In practical applications, we have found that setting the value of d to 2 still yields excellent empirical results, thereby significantly reducing memory consumption.

4.2 Finding Collapse Pairs

Since traversing a large number of cliques can directly lead to an increase in time complexity, this subsection proposes the approach of collapse pairs to narrow down the range of cliques that need to be traversed after removing an edge. The details of this approach are provided in Lines 18-33 of Algorithm 4. Specifically, given the graph $\mathcal{G}$, relationship graph $\mathcal{R}$, map relationship M, maximum dimension d and the k-simplex γ need to be removed or isolated,

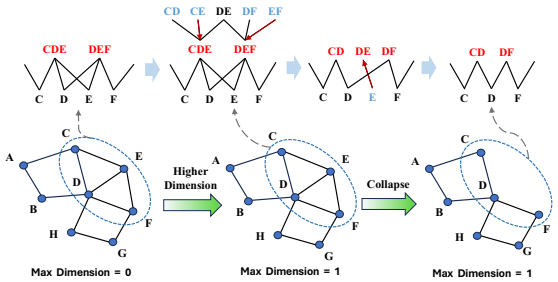

Figure 8: An example of how a low dimension may impede the graph's elementary collapse when using maximal cliques.

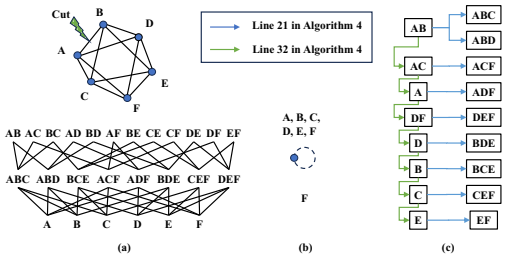

Figure 9: An example of how Bottom-up GEC find collapse pairs. Figures (a-b) show the graph and the relationship graph before and after invoking Algorithm 4. Figure(c) shows the order in which Algorithm 4 removes each pair of cliques.

we first remove the clique γ from $\mathcal{R}$ (Line 19). Subsequently, for each maximal clique λ with $\gamma \subseteq \lambda$, we first remove λ from $\mathcal{R}$ and split λ into new maximal cliques, then add them into $\mathcal{R}$ (Lines 20-27). After that, we check each clique τ as λ's face, if it can be collapsed as a free face, we further clear the collapsible clique τ (Lines 28-32). By employing the aforementioned methods, we can significantly reduce the running time of the Bottom-up GEC method.

Example 4.2. Figure 9 illustrates how Bottom-up GEC traverses all possible collapse pairs. In Figure 9(a), there is no clique pair available for collapse. After an edge removed, it can be observed that no new maximal clique is generated in this step. Next we check if there is any new free face within the cliques and collapse it. The order of collapse is shown in Figure9(c). It is worth noting some cliques, such as $\{A, D\}$, do not appear in the sequence. This is because when we collapse $\{A\}$ and $\{A, D, F\}$, $\{A, D\}$ also collapses simultaneously, but it is omitted in Figure 9(c). Through the process shown in Figure 9(c), we can efficiently traverse all cliques and collapse them into a single super node, as shown in Figure 9(b).

Based on the above in-depth analysis, we can now provide the pseudocode of the Bottom-up GEC algorithm, outlined in Algorithm 4. Specifically, given the graph $\mathcal{G}$, the maximum dimension of simplex d, the maximum number of nodes $\tilde{n}$ for each subgraph, we first split $\mathcal{G}$ into subgraphs G and build up the map relationship M and relationship graph $\mathcal{R}_i$ including only maximal cliques and nodes (Lines 1-4). After that, we iteratively add the k-cliques ($k = 1$, ..., $d+1$) into $\mathcal{R}_i$ and collapse the collapsable cliques (Lines 5-11). Subsequently, we shorten all chain-like structures (Line 14). When

Algorithm 4: Bottom-up Graph Elementary Collapse (Bottom-up GEC)

Input: $\mathcal{G}$ = *(V, E)*, the maximum dimension of clique complex d, the maximum number of nodes $\tilde{n}$ for each subgraph
Output: Coarsened graph $\mathcal{G}'$ = *(V', E')*, map relationship M.

1 Split $\mathcal{G}$ into a collection of subgraphs $G = \{\mathcal{G}_1, ..., \mathcal{G}_g\}$ with each has at most $\tilde{n}$ nodes;
2 Get relationship graph $R = \{\mathcal{R}_1, ..., \mathcal{R}_g\}$ including at most *(d+1)*-clique of graphs in G;
3 **for** *node v in* V **do**
4 $M[u] \leftarrow u$;
5 **for** $\mathcal{R}_i$ *in* R **do**
6 **for** j *in 1, ...,* d **do**
7 **if** *j-cliques are not included in* $\mathcal{R}_i$ **then**
8 Add j-clique and its relationship into $\mathcal{G}_i$;
9 **for** *each j-clique* τ **do**
10 **if** τ *is a subset of exactly one maximal clique in* $\mathcal{R}_i$ **then**
11 $\mathcal{G}_i, M, \mathcal{R}_i \leftarrow MaximalClearSimplex(\mathcal{G}_i, M, \mathcal{R}_i, \tau, j)$;
12 **while** *coarsening ratio has not been achieved* **do**
13 **for** $\mathcal{R}_i$ *in* R **do**
14 $\mathcal{G}_i, M, \mathcal{R}_i \leftarrow ShortenRingandChain(\mathcal{G}_i, M, \mathcal{R}_i)$;
15 Pick a *2*-clique σ from $\mathcal{R}_i$;
16 $\mathcal{G}_i, M, \mathcal{R}_i \leftarrow MaximalClearSimplex(\mathcal{G}_i, M, \mathcal{R}_i, \sigma, d)$;
17 $\mathcal{G}' \leftarrow Rebuild(\mathcal{G}, G)$;
18 **Procedure** $MaximalClearSimplex(\mathcal{G} = (\mathrm{V}, \mathrm{E}), M, \mathcal{R}, \gamma, d)$
19 $\mathcal{G}, M, \mathcal{R} \leftarrow Update(\mathcal{G}, M, \mathcal{R}, \gamma)$;
20 **for** *maximal clique* λ *that have* $\gamma \subseteq \lambda$ **do**
21 $\mathcal{G}, M, \mathcal{R} \leftarrow Update(\mathcal{G}, M, \mathcal{R}, \lambda)$;
22 **for** *node v in* γ **do**
23 $\epsilon \leftarrow \lambda \setminus \{v\}$;
24 **if** ϵ *is a maximal face* **then**
25 $\mathcal{R} \leftarrow \mathcal{R} \cup \{\epsilon\}$;
26 **for** $\tau \subseteq \epsilon$ *that have* $|\tau| \leq d+1$ **do**
27 $\mathcal{R} \leftarrow \mathcal{R} \cup \{(\epsilon, \tau)\}$;
28 **for** *maximal clique* λ *that have* $\gamma \subseteq \lambda$ **do**
29 $C \leftarrow \{\tau \in \mathcal{R} | \tau \subseteq \lambda, |\tau| \leq d+1\}$;
30 **for** τ *in* C **do**
31 **if** τ *is a free face of the maximal clique* μ **then**
32 $\mathcal{G}, M, \mathcal{R}, \leftarrow MaximalClearSimplex(\mathcal{G}, M, \mathcal{R}, \tau, d)$
33 **return** $\mathcal{G}, M, \mathcal{R}$

the collapse cannot go on, we uniformly pick a 2-clique from $\mathcal{R}_i$ and remove all the cliques and take it as a face (Lines 15-16) to make further collapse possible. After we achieve our coarsening ratio, we can rebuild the graph with the subgraphs and edges between them (Lines 17). Through this procedure, Algorithm 4 can minimize memory consumption and computational time as much as possible, while providing a more accurate collapse sequence.

Complexity Analysis of Algorithm 4. We let $\tilde{n} = \max\{|V(\mathcal{G}_i)|| \mathcal{G}_i \in G\}$ be the maximum number of nodes in the subgraph set G. Thus, there are most $n/\tilde{n}$ subgraphs in G, where n is the number of nodes in the original graph. For each subgraph $\mathcal{G}_i$, the time complexity of maximal clique enumeration is $O(3^{\tilde{n}/3})$ [25]. Besides, let q be the number of maximal cliques of $\mathcal{G}_i$ and d be the maximum dimension, the time complexity of the collapse process is $O(q\tilde{n}^d)$. This is because there are at most $\tilde{n}^d$ cliques stored in $\mathcal{R}_i$ and each clique need to be checked at most q times for its collapsibility. As a result, the time complexity of Bottom-up GEC is $O(n3^{\tilde{n}/3}/\tilde{n} + nq\tilde{n}^{(d-1)})$. In terms of the space complexity, the space complexity of the collapse process on each subgraph $\mathcal{G}_i$ is $O(q\tilde{n}^d)$. This is because at most q maximal cliques are stored for each subgraph $\mathcal{G}_i$ and each maximal clique needs to be stored at most $\tilde{n}^d$ times by its faces. Thus, the space complexity of Bottom-up GEC is $O(nq\tilde{n}^{(d-1)})$.

Table 1: Statistics of datasets

Datasets	$\|V\|$	$\|E\|$	Ave. Degree	#Features	#Classes
Cora	2,708	5,429	3.88	1,433	7
Citeseer	3,327	4,732	2.84	3,703	6
DBLP	17,716	52,867	5.97	1639	4
PubMed	19,717	44,338	4.50	500	3
Coauthor Physics	34,493	247,962	14.38	8,415	5
Ogbn-ArXiv	169,343	1,166,243	13.77	128	40
Reddit	232,965	57,307,946	491.98	602	41
Ogbn-products	2,449,029	61,859,140	50.52	100	47
com-youtube	1,134,890	2,987,624	5.27	-	-
cit-Patent	3,774,768	16,518,948	8.75	-	-
dblp-v5	1,572,277	2,084,019	2.65	-	-
dblp-v7	2,244,021	4,354,534	3.88	-	-

5 EXPERIMENTS

5.1 Experimental Setup

Datasets. We evaluate our solutions on several publicly available real-world datasets with different statistics (Table 1). First eight datasets are attributed graphs widely used for evaluating GNN's performance [3, 21, 27, 41]. The last four graphs are non-attributed. The first two are different versions across time for the DBLP citation network (dblp-v5, dblp-v7) [44], and the last two are from SNAP [30]. For node classification on Ogbn-ArXiv and Ogbn-products, we use the split from [22]. while others use the split from [53]. For the link prediction task, we randomly sampled 85% edges for training, 5% for validation, and the remaining 10% for testing.

Tasks and Evaluation Metrics. The main objective is to validate the performance of our proposed GEC-B and Bottom-up GEC as a graph coarsening method for the node classification task. To assess the applicability of Bottom-up GEC across different downstream tasks, we also perform experiments on the link prediction task. For both tasks, we conduct training and validation processes on the coarsened graph $\mathcal{G}'$ and evaluate the training results on the original graph $\mathcal{G}$. We utilize accuracy as the main evaluation metric for the node classification task and ROC-AUC for the link prediction task.

Baselines. To establish a comprehensive comparison, we select several baselines for our experiments. Specifically, for the graph coarsening methods, we choose Variation Neighborhoods [23, 34], Variation Edges [23, 34], Algebraic JC [23, 34], Affinity GS [23, 34], kron [23, 34], and FGC [29] as the state-of-the-art methods. We also compare our method with graph condensation methods and graph sparsification methods. For graph condensation, we select GCOND [26] and SFGC [58] as the baseline methods. For graph sparsification, we choose UGS [8] and random split as the baselines.

Models and Hyper-parameter Settings. Following previous research [16, 23, 29, 41], we also adopt the Graph Convolutional Network (GCN) [27] and APPNP [28] as our evaluated models, and use their corresponding default settings. For GEC-B and Bottom-up GEC, unless otherwise stated, we set $d = 6$ and $\tilde{n} = 1000$.

5.2 Empirical Results

Exp-1: Node Classification with Graph Coarsening Methods. Table 2 reports the experimental results. An issue we need to mention is that, although we can obtain a graph $\mathcal{G}' = (V', E')$ using the GEC-B and Bottom-up GEC, the graph $\mathcal{G}'$ may not meet the

Table 2: Node classification with graph coarsening methods. Each value is shown in terms of average classification accuracy and standard deviation (in percent) over 20 runs on different datasets. The horizontal line (i.e., -) in the table signifies that the method cannot be applied. The best and second-best results in each metric are marked in bold and underlined, respectively.

Dataset	Coarsening Method	c=1.0		c=0.5		c=0.3		c=0.2		c=0.1	
		GCN	APPNP	GCN	APPNP	GCN	APPNP	GCN	APPNP	GCN	APPNP
Cora	Variation Neighborhoods	81.1±0.4	83.3±0.4	**81.7±0.4**	81.9±0.3	80.5±0.6	81.7±0.6	78.5±0.8	81.0±0.8	72.9±1.3	66.4±0.8
	Variation Edges			81.6±0.4	**83.4±0.5**	79.0±1.0	81.3±1.0	72.5±1.5	72.8±1.7	–	–
	Algebraic JC			81.3±0.3	82.7±0.5	79.5±0.5	80.3±0.9	79.0±0.4	82.1±0.6	66.5±0.9	69.7±2.0
	Affinity GS			81.4±0.5	82.5±0.5	79.8±0.5	79.5±1.0	80.1±0.7	80.3±0.7	74.0±1.0	70.7±1.7
	kron			81.4±0.3	83.0±0.7	79.8±0.6	80.0±0.7	79.9±0.8	76.9±0.7	64.2±0.7	66.9±0.4
	FGC			79.8±2.2	78.7±1.4	77.6±2.4	77.8±2.5	77.1±1.4	76.8±2.4	70.7±1.8	68.8±3.5
	GEC-B			79.1±0.5	81.4±0.7	79.8±0.7	81.2±0.6	80.2±0.5	82.2±0.6	79.7±0.6	81.5±0.7
	Bottom-up GEC			80.7±0.4	82.2±0.3	**80.9±0.5**	**83.1±0.3**	**81.0±0.7**	**82.6±0.6**	**81.2±0.4**	**82.8±0.7**
Citeseer	Variation Neighborhoods	71.6±0.4	71.9±0.4	71.8±0.5	71.6±0.6	70.3±0.5	71.1±0.3	70.1±0.7	70.6±0.6	56.8±7.0	58.2±6.0
	Variation Edges			**72.2±0.5**	71.6±0.6	70.0±0.5	72.3±0.4	54.8±6.1	60.9±4.8	47.0±11.	47.6±17.
	Algebraic JC			71.2±0.5	71.3±0.7	70.2±0.4	**72.7±0.5**	56.9±1.3	67.2±1.1	60.0±7.3	58.4±8.2
	Affinity GS			70.3±0.7	71.5±0.4	70.3±0.2	71.2±0.5	69.5±0.9	70.7±0.7	59.2±7.5	59.9±4.9
	kron			72.4±0.5	**72.2±0.1**	70.3±0.6	71.2±0.4	70.1±1.4	70.6±0.2	63.6±1.2	66.4±0.4
	FGC			70.1±1.4	71.5±1.9	68.8±1.7	70.7±1.3	67.3±2.1	70.5±2.2	66.9±1.5	66.8±2.1
	GEC-B			70.7±0.5	71.1±0.3	70.6±0.4	71.3±0.3	70.2±0.9	70.8±0.7	70.9±0.4	71.0±0.8
	Bottom-up GEC			70.2±0.4	70.9±0.6	**71.1±0.5**	71.1±0.3	**71.5±0.7**	**71.6±0.5**	**71.6±0.7**	**72.0±0.4**
Ogbn-ArXiv	Variation Neighborhoods	70.4±1.1	64.7±1.2	64.8±2.0	56.8±1.2	65.1±3.4	60.9±0.4	57.0±1.3	56.5±0.8	44.2±2.9	53.6±0.6
	Variation Edges			65.7±1.1	62.2±0.7	61.6±0.9	56.4±0.3	54.7±1.4	56.2±1.1	51.4±3.4	55.8±1.0
	Algebraic JC			66.2±0.5	64.2±0.9	60.4±1.7	60.2±0.8	53.4±1.4	51.6±0.8	47.3±1.7	60.5±0.8
	kron			64.5±1.5	63.5±0.2	56.2±3.1	60.3±0.6	56.4±2.7	54.9±0.9	57.7±0.5	60.7±0.4
	Affinity GS / FGC			Out of Memory (Over 400GB)							
	GEC-B			69.1±0.4	**64.8±0.1**	66.5±0.7	62.3±0.5	65.6±1.1	61.4±0.4	Out of Time (Over 1 day)	
	Bottom-up GEC			**70.4±0.3**	64.3±0.3	**69.1±0.2**	**62.6±0.4**	**68.0±0.4**	**61.7±0.4**	**65.3±0.1**	**60.8±0.7**
Reddit	Variation Neighborhoods	94.1±1.1	93.9±0.8	93.5±2.0	93.4±1.7	92.6±3.4	92.0±1.3	91.9±1.3	90.4±2.0	90.3±2.9	90.1±0.9
	Var. Edges / Algebraic JC / kron			Out of Time (Over 1 day)							
	Affinity GS / FGC / GEC-B			Out of Memory (Over 400GB)							
	Bottom-up GEC ($\tilde{n} = 100$)			**94.2±0.3**	**94.0±1.1**	**93.1±0.4**	**93.7±1.1**	**93.1±0.3**	**93.0±1.1**	**90.5±0.5**	**90.2±0.9**
Ogbn-products	Other methods	71.8±1.1	67.5±0.8	Out of Memory (Over 400GB)							
	Bottom-up GEC ($\tilde{n} = 100$)			70.9±2.0	66.2±1.1	70.7±1.0	66.1±1.1	70.6±2.0	66.0±1.1	69.8±2.4	65.4±0.9

Table 3: The HE and RE values obtained by Variation Neibor-hoods , Algebraic JC, FGC, and our proposed Bottom-up GEC on different coarsening ratios.

Dataset	Metric	c	Var. Nei.	Alg. JC	FGC	Bottom-up GEC
Cora	HE	0.7	1.40	1.67	**0.64**	1.48
		0.5	2.30	2.34	**1.08**	1.85
		0.3	2.94	3.05	**1.98**	2.40
		0.2	3.56	3.57	**2.26**	2.83
		0.1	4.43	4.30	**2.76**	3.52
	RE in log(·)	0.7	2.93	3.34	**1.73**	3.13
		0.5	3.65	3.68	**2.59**	3.37
		0.3	3.77	3.79	**3.48**	3.61
		0.2	3.81	3.82	**3.72**	**3.72**
		0.1	3.84	3.83	3.79	**3.78**

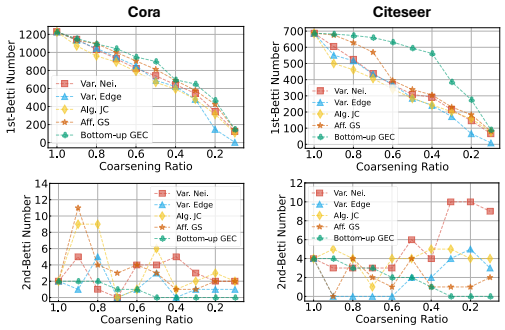

Figure 10: Betti number obtained by various methods.

requirements of graph coarsening. This is because, in addition to mapping nodes to super nodes, we may also remove additional edges during the collapse process which is prohibited in the coarse step. For a fair comparison, we get the coarse result only using the mapping relationship M instead of using $\mathcal{G}'$. Here, 'c' denotes the coarsening ratio, indicating the proportion of nodes in the coarsened graph relative to the original graph. As can be seen, we have GEC-B and Bottom-up GEC consistently outperformed the other methods in most cases. Also, these two methods maintain compatible performance even at higher coarsening ratios, where other methods have significant degradation. Specifically, on the Ogbn-ArXiv, Bottom-up GEC achieves more than 5% improvement compared with other methods. We also examine the differences between GEC-B and Bottom-up GEC: Bottom-up GEC, which constructs the relationship graph using maximal cliques, eliminates misjudgments of clique collapsibility. This enables the Bottom-up GEC method to achieve slightly better results than GEC-B in most cases. Besides, Bottom-up GEC has demonstrated great scalability. As observed, only Bottom-up GEC and the Variational Neighborhoods can run on the relatively dense dataset Reddit. When facing a larger dataset ogbn-products, Bottom-up GEC becomes the only capable method. Therefore, we primarily use Bottom-up GEC as our method for comparison with other baselines.

Exp-2: RE, HE, and Betti Number Analysis. We also employ Reconstruction Error (RE), Hyperbolic Error (HE), and Betti number to further evaluate the performance of Bottom-up GEC method. RE and HE are the metrics used in previous work [29] to evaluate coarsening algorithms. Due to the space limits, we only report the results on Cora (Table 3). Other datasets can also obtain similar results. Although our algorithm is not a spectral-based method, it still achieves favorable outcomes in traditional evaluation metrics (RE and HE). The Betti number is an evaluation metric to validate the ability of the coarsening method to preserve rings and voids in the

Table 4: Evaluation with other reduction method

Dataset	c	GCOND	SFGC	FGC	Bottom-up GEC
Cora	0.3	81.5±0.6	82.3±0.3	**85.7±0.2**	84.7±0.2
	0.2	79.3±0.6	80.5±0.2	82.3±1.3	**83.7±0.3**
	0.1	81.3±0.4	79.8±0.5	81.4±0.7	**82.5±0.3**
Citeseer	0.3	72.4±0.9	68.8±0.1	74.6±1.3	**75.3±0.2**
	0.2	72.0±0.4	69.5±0.4	72.1±0.7	**76.6±0.3**
	0.1	70.4±0.4	66.8±0.4	**73.3±0.5**	**73.3±0.3**
PubMed	0.05	78.1±0.3	79.2±0.2	**80.7±0.4**	79.5±0.8
	0.03	78.0±0.4	78.2±0.5	79.9±0.3	**80.1±0.7**
	0.01	77.2±0.2	78.6±0.1	78.4±0.4	**78.9±0.6**
Co-Phy	0.05	93.0±0.2	**94.3±0.4**	94.2±0.2	**94.3±0.0**
	0.03	92.8±0.3	92.6±0.5	92.6±0.2	**93.7±0.2**
	0.01	92.7±0.4	**93.1±0.3**	65.6±0.3	76.5±0.3

graph. Specifically, the 1^{st}-Betti and 2^{nd}-Betti numbers represent the quantity of rings and voids in the graph, respectively. In Figure 10, we present the 1^{st}-Betti number and 2^{nd}-Betti number of each method on Cora and Citeseer. It can be observed that our Bottom-up GEC method can effectively preserve the rings during the coarsening process. Meanwhile, unlike other methods, Bottom-up GEC does not generate additional voids during the coarsening.

Exp-3: Evaluation with Other Reduction Method. Graph Condensation methods [17, 26, 58] have emerged as a graph reduction technique for GNNs in recent years. They generate new nodes by leveraging the downstream task information from GNN instead of mapping nodes into others. We select GCOND [26], FGC [29], and SFGC [58] as the baseline methods, following the experimental settings outlined in [29]. By Table 4, we can observe that Bottom-up GEC consistently achieves superior performance in most cases when contrasted with SFGC and GCOND. However, both FGC and Bottom-up GEC may experience performance degradation when confronted with extremely high coarsening ratios. Through analysis, we discovered that in such cases, the label distribution of nodes in the coarsened graph may exhibit extreme patterns, resulting in poor training effectiveness for certain labels, leading to inferior overall results. The graph sparsification method is another graph reduction method. Since it only modifies the edges without reducing the number of nodes, we only compare the result of the node classification task on GCN with the same number of edges in the reduced graph. Specifically, we compare Bottom-up GEC with UGS (200 epochs) [8] and random pruning method with a two-layer GCN [27] on PubMed dataset. Other datasets have similar trends. The accuracy and the running time of each method are shown in Figure 11. It is evident that Bottom-up GEC consistently outperforms random pruning and gets similar or even better results than UGS while consuming much less time to reduce the number of nodes.

Exp-4: Link Prediction with Graph Coarsening Methods. We also conduct experiments of the link prediction task with four other graph coarsening methods to evaluate the effectiveness of Bottom-up GEC facing variant downstream tasks. The experiment result is shown in the left part of Figure 12. Based on the experimental results, it is evident that while the performance of all methods is similar at low coarsening ratios, akin to the node classification task, the Bottom-up GEC method outperforms other approaches significantly at higher coarsening ratios, particularly when c=0.1/0.2. This

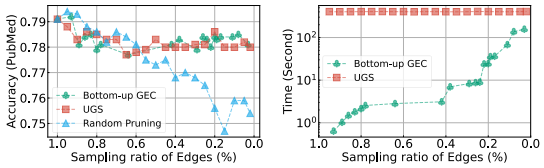

Figure 11: Comparison of the proposed Bottom-up GEC and the graph sparsification methods on PubMed dataset.

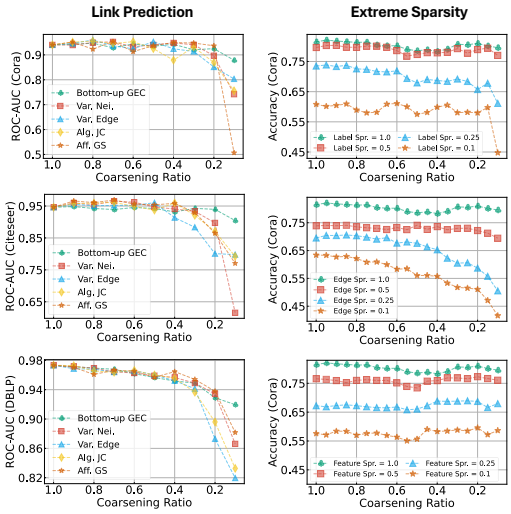

Figure 12: Experiment results on link prediction with graph coarsening methods and extreme sparsity scenarios.

provides substantial evidence for the effectiveness of Bottom-up GEC when confronted with various downstream tasks.

Exp-5: Experiments with Extreme Sparsity. In practical application scenarios, we often encounter situations where graphs exhibit extreme sparsity, such as missing features or a low number of labeled nodes. To further evaluate the effectiveness of Bottom-up GEC, we investigate the impact of such sparsity on Bottom-up GEC and whether it can still achieve satisfactory results in high-sparsity scenarios. We consider three types of sparsity in our experiments: label sparsity (ratio of labeled nodes in the sparsified graph to the full graph), feature sparsity (ratio of remaining features in the sparsified graph to the full graph), and edge sparsity (ratio of remaining edges in the sparsified graph to the full graph). We present the results in the right part of Figure 12.

Through our experimental analysis, we observe that as the graph's sparsity varies, there is a decline in node classification accuracy due to missing information from the original graph. Interestingly, when facing high feature sparsity, the performance on the coarsened graph even surpasses the performance on the original sparsified graph at high coarsening ratios. However, when facing high-edge sparsity, its performance is severely affected. These outcomes can be attributed to Bottom-up GEC's emphasis on the graph's topological structure, where the influence of node labels and features is

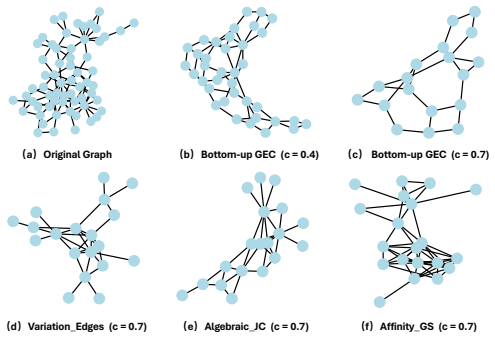

Figure 13: Visualization of Bottom-up GEC and other methods with different coarsening ratios.

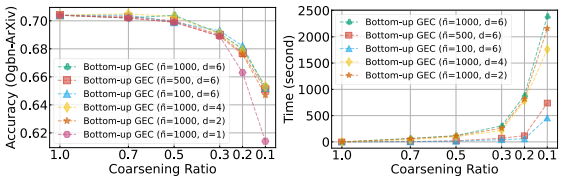

Figure 14: Impact of different hyperparameters.

minimal. When facing feature sparsity, the absence of features is partially compensated by other nodes that are coarse with it, resulting in improved outcomes. On the other hand, high edge sparsity may cause significant alterations in the topological characteristics of the graph, leading to poor results.

Exp-6: Visualization of Different Coarsened Methods. We also validate Bottom-up GEC through a case study. We selected the Dolphins dataset for visualization and the results are shown in Figure 13. It can be observed that during the collapsing process, our method preserves a significant number of rings in the graph, these rings even become more prominent after collapse. For other methods, at higher coarsening ratios, they contain several nodes located outside of rings, and significant loss of ring is also observed.

Exp-7: Running Time and Memory Overhead of Different Coarsening Methods. Firstly, we investigate the efficiency of the proposed Bottom-up GEC by comparing the processing time with other graph coarsening methods. Table 5 presents the processing time on Cora and Ogbn-ArXiv. Other datasets can also obtain similar results. We can observe that the computational efficiency of Bottom-up GEC is close to or even surpasses the existing state-of-the-art methods in most coarsening ratios. Memory cost is also evaluated in our experiments. We compare the memory cost of Bottom-up GEC with different coarsening ratios against other baselines. By Figure 15, we find that Bottom-up GEC has better performance and achieves similar or even far more lower memory costs compared to other methods. Furthermore, it can be observed that compared to GEC-B, Bottom-up GEC has shown significant improvement in both runtime and memory cost, thereby demonstrating the effectiveness of the optimization introduced in Section 4.

Table 5: Running time of different coarsening methods.

Dataset	Coarsening Method	Time (second)				
		c=0.7	c=0.5	c=0.3	c=0.2	c=0.1
Cora	Variation Neighborhoods	2.670	1.752	1.638	1.658	1.628
	Variation Edges	1.168	1.251	1.124	1.156	1.172
	Algebraic JC	0.907	0.907	0.909	0.900	0.944
	Affinity GS	3.213	3.186	3.218	3.247	3.157
	FGC(50 loops)	78.352	41.445	18.026	11.415	6.456
	GEC-B	12.551	20.539	27.553	28.775	29.235
	Bottom-up GEC	0.119	0.646	6.286	10.819	15.655
Ogbn-ArXiv	Variation Neighborhoods	353.9	357.1	433.6	429.4	447.7
	Variation Edges	415.3	546.8	692.1	767.8	861.2
	Algebraic JC	392.8	557.9	710.7	815.1	995.2
	Affinity GS / FGC	Out of Memory (Over 400GB)				
	GEC-B	1144.8	1890.2	2171.1	3719.6	> 1day
	Bottom-up GEC	65.67	118.9	306.4	883.7	2388.9

Exp-8: Impact of Different Hyperparameters. We also conducted experiments to study the impact of different hyperparameters. The processing time is shown in Figure 14(a), and the accuracy results are shown in Figure 14(b). We can observe that splitting the graph into subgraphs does not affect the effectiveness while significantly improve the efficiency. This characteristic of Bottom-up GEC ensures its effectiveness when applied to larger graphs. On the other hand, as discussed in Section 4.1, when using the maximal clique in the relationship, reducing the maximum dimension d of the clique complex from 6 to 2 has little effect on the effectiveness. Moreover, as discussed earlier, reducing d to 1 can significantly affect the pruning process of Bottom-up GEC, leading to low accuracy.

Exp-9: Scalability Analysis. We further evaluate the scalability of Bottom-up GEC on 4 different large datasets which contain millions of nodes. For each dataset, we generate subgraphs by randomly sampling nodes from 20% to 100% and randomly sampling edges from 20% to 100%. Figure 16 shows the processing time and memory cost of Bottom-up GEC on these graphs. The coarsening ratio is set to be 0.3, and similar results can also be observed with other coarsening ratios. As can be seen, the runtime and the memory cost of Bottom-up GEC increase smoothly as the graph size increases on massive graphs. Moreover, all the curves are nearly linear, indicating that our algorithm scales very well in practice.

To facilitate comparison with other graph coarsening methods, we conduct a scalable analysis on a relatively small dataset ogbn-arxiv which most method can deal with. The coarsening ratio is set to be 0.3, and the results are illustrated in Figure 17. Affinity GS and FGC exhibit missing data points due to the space limit (400G). We can observe that Bottom-up GEC exhibits faster runtime as $|E|$ increases. This is because it can quickly collapse a k-clique. However, in scenarios where only a few edges are removed, the large clique transform into a complex structure composed of multiple cliques, leading to longer collapse time. Some other methods also exhibit faster runtime as $|E|$ increases. These methods perform the coarse operation on each connected component of the graph. As number of edges decreases, the number of connected components increases. The computation time of each connected component adds up, resulting an increase in computation time. From the experiment, we can see that our method demonstrates similar scalability in terms of time compared to other methods, while significantly outperforming all methods around 100× in terms of space scalability.

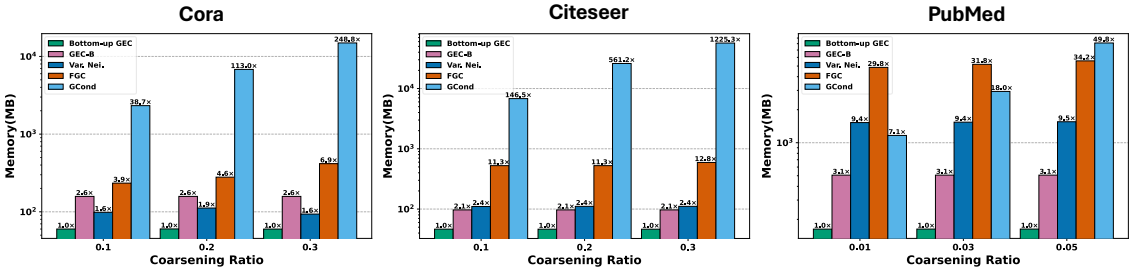

Figure 15: Memory overhead of different coarsening methods with varying coarsening ratios.

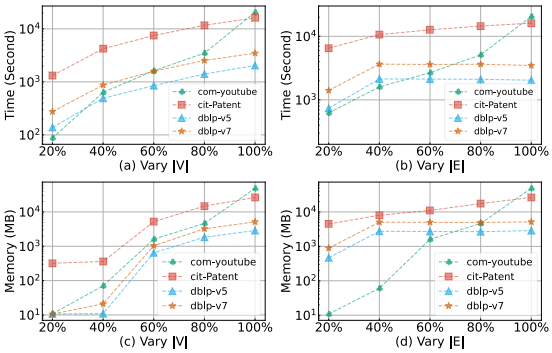

Figure 16: Scalability analysis of Bottom-up GEC

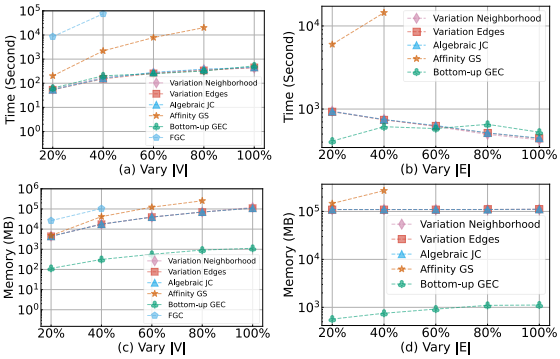

Figure 17: Scalability analysis of graph coarsening methods

6 FURTHER RELATED WORK

Graph Coarsening. Existing graph coarsening methods mainly focused on preserving a graph's spectral or spatial graph properties. For example, Loukas et al. [35] focused on the restricted spectral approximation to provide spectrum approximation guarantees during coarse. Bravo et al. [4] developed a probabilistic framework to preserve the inverse graph Laplacian matrix. To boost the scalability of GNNs, Huang et al. [23] first apply graph coarsening to scale up GNNs and analyze the impact of coarsening on GNNs. Kumar et al. [29] achieved the state-of-the-art performance by incorporating node features into the coarsening process. However, these methods do not preserve the topological properties of the graph (such as rings, and voids), which may lead to the loss of critical topological structures and obtain unpleasant performance.

Graph Sparsification. Graph sparsification reduces the size of graphs by sampling nodes or edges from the graph. Conventional approaches focus on preserving properties of spectrum or centrality. For example, Wickman et al.[48] defined a reinforcement learning process and adjusted reward functions to preserve pairwise distance. Daniel et al. [43] sampled edges using normalized effective resistance as probabilities. Recently, methods aiming to preserve the GNN performance have also emerged in the literature. For example, Neural Sparse [56] extracts weakly relevant edges by receiving feedback from downstream tasks. UGS [8, 24] applies the Lottery Ticket Hypothesis to the graph to find the less important edges. However, the selection of nodes or edges inevitably loss of some information in the graph, leading to poor performance.

Maximal Clique Enumeration. Maximal clique enumeration is computationally challenging due to the combinatorial explosion of vertices. Numerous approaches have been proposed to address this problem. For example, the Bron-Kerbosch algorithm [5] is a recursive search algorithm with three sets. Three sets are maintained in each recursive call: a partially constructed clique, a set of candidate vertices, and a set of excluded vertices. Pivoting [25] reduces the search space of the Bron-Kerbosch algorithm based on the co-occurrence of neighboring vertices in maximal cliques. Efficiency can also be improved by sorting vertices by increasing degeneracy [15] to limits the size of the right neighborhood of a vertex, or create graph cutouts in each recursive call that can reduce the length of adjacency lists [11].

7 CONCLUSION

In this paper, we propose a novel graph elementary collapse operator for graph coarsening which extends a concept of *elementary collapse* from algebraic topology to the field of graph analysis. Our method incorporates elementary collapse operations to maintain the homotopy equivalence of the graph topology, which can preserve the topological properties of the graph. We also devise several nontrivial optimization techniques to improve the efficiency of our solution. Finally, we conduct extensive experiments on diverse graph datasets to evaluate our approach, and the results demonstrate the high efficiency and effectiveness of our method for various graph analysis applications.

ACKNOWLEDGMENTS

This work was supported by NSFC Grants U2241211, 62402399 and 62072034. Rong-Hua Li is the corresponding author of this paper.

REFERENCES

[1] Cristian Bodnar, Fabrizio Frasca, Yuguang Wang, Nina Otter, Guido F Montufar, Pietro Lió, and Michael Bronstein. 2021. Weisfeiler and Lehman Go Topological: Message Passing Simplicial Networks. In *Proceedings of the 38th International Conference on Machine Learning*. 1026–1037.

[2] Aleksandar Bojchevski, Johannes Gasteiger, Bryan Perozzi, Amol Kapoor, Martin Blais, Benedek Rózemberczki, Michal Lukasik, and Stephan Günnemann. 2020. Scaling Graph Neural Networks with Approximate PageRank. In *KDD*.

[3] Aleksandar Bojchevski and Stephan Günnemann. 2018. Deep Gaussian Embedding of Graphs: Unsupervised Inductive Learning via Ranking. In *ICLR*.

[4] Gecia Bravo Hermsdorff and Lee Gunderson. 2019. A Unifying Framework for Spectrum-Preserving Graph Sparsification and Coarsening. In *NeurIPS*.

[5] Coenraad Bron and Joep Kerbosch. 1973. Finding All Cliques of an Undirected Graph (Algorithm 457). *Commun. ACM* (1973), 575–576.

[6] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. 2013. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203* (2013).

[7] Jie Chen, Tengfei Ma, and Cao Xiao. 2018. FastGCN: Fast Learning with Graph Convolutional Networks via Importance Sampling. In *ICLR*.

[8] Tianlong Chen, Yongduo Sui, Xuxi Chen, Aston Zhang, and Zhangyang Wang. 2021. A Unified Lottery Ticket Hypothesis for Graph Neural Networks. In *ICMR*. 1695–1706.

[9] Zhengdao Chen, Soledad Villar, Lei Chen, and Joan Bruna. 2019. On the equivalence between graph isomorphism testing and function approximation with GNNs. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*. 15868–15876.

[10] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. 2019. Cluster-GCN: An Efficient Algorithm for Training Deep and Large Graph Convolutional Networks. In *KDD*. 257–266.

[11] Naga Shailaja Dasari, Desh Ranjan, and Mohammad Zubair. 2014. pbitMCE: A bit-based approach for maximal clique enumeration on multicore processors. In *IEEE*. 478–485.

[12] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. *NeurIPS* (2016).

[13] Tamal Krishna Dey and Yusu Wang. 2022. *Computational topology for data analysis*. Cambridge University Press.

[14] Ömer Eğecioğlu and Teofilo F Gonzalez. 1996. A computationally intractable problem on simplicial complexes. *Computational Geometry* (1996).

[15] David Eppstein, Maarten Löffler, and Darren Strash. 2010. Listing All Maximal Cliques in Sparse Graphs in Near-Optimal Time. In *ISAAC*. 403–414.

[16] Matthias Fey and Jan Eric Lenssen. 2019. Fast Graph Representation Learning with PyTorch Geometric. *CoRR* (2019).

[17] Xinyi Gao, Tong Chen, Yilong Zang, Wentao Zhang, Quoc Viet Hung Nguyen, Kai Zheng, and Hongzhi Yin. 2023. Graph Condensation for Inductive Node Representation Learning.

[18] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *NeurIPS*.

[19] Yue He, Longlong Lin, Pingpeng Yuan, Ronghua Li, Tao Jia, and Zeli Wang. 2024. CCSS: Towards conductance-based community search with size constraints. *Expert Syst. Appl.* 250 (2024), 123915.

[20] Bruce Hendrickson, Robert W Leland, et al. 1995. A Multi-Level Algorithm For Partitioning Graphs. *SC* (1995), 1–14.

[21] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. 2020. Open Graph Benchmark: Datasets for Machine Learning on Graphs. In *NeurIPS*. 22118–22133.

[22] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. 2020. Open Graph Benchmark: Datasets for Machine Learning on Graphs. *arXiv preprint arXiv:2005.00687* (2020).

[23] Zengfeng Huang, Shengzhong Zhang, Chong Xi, Tang Liu, and Min Zhou. 2021. Scaling Up Graph Neural Networks Via Graph Coarsening. In *KDD*. 675–684.

[24] Bo Hui, Da Yan, Xiaolong Ma, and Wei-Shinn Ku. 2023. Rethinking Graph Lottery Tickets: Graph Sparsity Matters. In *The Eleventh ICLR*.

[25] Shweta Jain and C. Seshadhri. 2020. The Power of Pivoting for Exact Clique Counting. In *WSDM '20: The Thirteenth ACM International Conference on Web Search and Data Mining, Houston, TX, USA, February 3-7, 2020*, James Caverlee, Xia (Ben) Hu, Mounia Lalmas, and Wei Wang (Eds.). 268–276.

[26] Wei Jin, Lingxiao Zhao, Shichang Zhang, Yozen Liu, Jiliang Tang, and Neil Shah. 2022. Graph Condensation for Graph Neural Networks. (2022).

[27] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*.

[28] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. 2019. Predict then Propagate: Graph Neural Networks meet Personalized PageRank. In *ICLR*.

[29] Manoj Kumar, Anurag Sharma, Shashwat Saxena, and Sandeep Kumar. 2023. Featured Graph Coarsening with Similarity Guarantees. In *ICML*.

[30] Jure Leskovec and Andrej Krevl. 2014. SNAP Datasets: Stanford Large Network Dataset Collection.

[31] Longlong Lin, Tao Jia, Zeli Wang, Jin Zhao, and Rong-Hua Li. 2024. PSMC: Provable and Scalable Algorithms for Motif Conductance Based Graph Clustering. *CoRR* abs/2406.07357 (2024).

[32] Longlong Lin, Ronghua Li, and Tao Jia. 2023. Scalable and Effective Conductance-Based Graph Clustering. In *AAAI*. 4471–4478.

[33] Longlong Lin, Pingpeng Yuan, Rong-Hua Li, Chun-Xue Zhu, Hongchao Qin, Hai Jin, and Tao Jia. 2024. QTCS: Efficient Query-Centered Temporal Community Search. *Proc. VLDB Endow.* 17, 6 (2024), 1187–1199.

[34] Andreas Loukas. 2019. Graph Reduction with Spectral and Cut Guarantees. *Journal of Machine Learning Research* (2019), 1–42.

[35] Andreas Loukas and Pierre Vandergheynst. 2018. Spectrally Approximating Large Graphs with Smaller Graphs. In *ICML*. 3237–3246.

[36] John Milnor. 1966. Whitehead torsion. *Bull. Amer. Math. Soc.* (1966).

[37] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodola, Jan Svoboda, and Michael M Bronstein. 2017. Geometric deep learning on graphs and manifolds using mixture model cnns. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 5115–5124.

[38] James R. Munkres. 1984. *Elements of algebraic topology*. Addison-Wesley.

[39] Morteza Ramezani, Weilin Cong, Mehrdad Mahdavi, Anand Sivasubramaniam, and Mahmut Kandemir. 2020. GCN meets GPU: Decoupling "When to Sample"from "How to Sample". In *NeurIPS*.

[40] Ilya Safro, Peter Sanders, and Christian Schulz. 2014. Advanced Coarsening Schemes for Graph Partitioning. *ACM J. Exp. Algorithmics* 19, 1 (2014).

[41] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. 2018. Pitfalls of Graph Neural Network Evaluation. *CoRR* (2018).

[42] David I Shuman, Mohammad Javad Faraji, and Pierre Vandergheynst. 2016. A Multiscale Pyramid Transform for Graph Signals. *IEEE Transactions on Signal Processing* (2016).

[43] Daniel A. Spielman and Nikhil Srivastava. 2011. Graph Sparsification by Effective Resistances. *SIAM J. Comput.* (2011), 1913–1926.

[44] Jie Tang, Jing Zhang, Limin Yao, Juanzi Li, Li Zhang, and Zhong Su. 2008. ArnetMiner: extraction and mining of academic social networks. In *SIGKDD*. 990–998.

[45] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *ICLR*.

[46] Chris Walshaw. 2003. A Multilevel Algorithm for Force-Directed Graph-Drawing. *Journal of Graph Algorithms and Applications* (2003).

[47] Lu Wang, Yanghua Xiao, Bin Shao, and Haixun Wang. 2014. How to partition a billion-node graph. In *ICDE*. 568–579.

[48] Ryan Wickman, Xiaofei Zhang, and Weizi Li. 2022. A Generic Graph Sparsification Framework using Deep Reinforcement Learning. In *ICDM*. 1221–1226.

[49] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. 2019. Simplifying Graph Convolutional Networks. In *ICML*.

[50] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How Powerful are Graph Neural Networks?. In *ICLR*.

[51] Zuoyu Yan, Tengfei Ma, Liangcai Gao, Zhi Tang, and Chao Chen. 2022. Cycle Representation Learning for Inductive Relation Prediction. In *ICML*. 24895–24910.

[52] Zuoyu Yan, Tengfei Ma, Liangcai Gao, Zhi Tang, Chao Chen, and Yusu Wang. 2023. Cycle Invariant Positional Encoding for Graph Representation Learning. *CoRR* (2023).

[53] Zhilin Yang, William Cohen, and Ruslan Salakhudinov. 2016. Revisiting Semi-Supervised Learning with Graph Embeddings. In *ICML*. 40–48.

[54] Yunfeng Yu, Longlong Lin, Qiyu Liu, Zeli Wang, Xi Ou, and Tao Jia. 2024. GSD-GNN: Generalizable and Scalable Algorithms for Decoupled Graph Neural Networks. In *ICMR*.

[55] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. 2020. GraphSAINT: Graph Sampling Based Inductive Learning Method. In *ICLR*.

[56] Cheng Zheng, Bo Zong, Wei Cheng, Dongjin Song, Jingchao Ni, Wenchao Yu, Haifeng Chen, and Wei Wang. 2020. Robust graph representation learning via neural sparsification. In *ICML*. 11458–11468.

[57] Da Zheng, Chao Ma, Minjie Wang, Jinjing Zhou, Qidong Su, Xiang Song, Quan Gan, Zheng Zhang, and George Karypis. 2020. DistDGL: Distributed Graph Neural Network Training for Billion-Scale Graphs. In *2020 IEEE/ACM 10th Workshop on Irregular Applications: Architectures and Algorithms (IA3)*.

[58] Xin Zheng, Miao Zhang, Chunyang Chen, Quoc Viet Hung Nguyen, Xingquan Zhu, and Shirui Pan. 2023. Structure-free Graph Condensation: From Large-scale Graphs to Condensed Graph-free Data. In *NeurIPS*.

[59] Jinhua Zhu, Kehan Wu, Bohan Wang, Yingce Xia, Shufang Xie, Qi Meng, Lijun Wu, Tao Qin, Wengang Zhou, Houqiang Li, and Tie-Yan Liu. 2023. $\mathcal{O}$-GNN: incorporating ring priors into molecular modeling. In *The Eleventh International Conference on Learning Representations*.