Scalable Topology-Preserving Graph Coarsening: Concepts, Algorithms and Applications

Xiang Wu, Rong-Hua Li, Xunkai Li, Kangfei Zhao, Guoren Wang
Beijing Institute of Technology, China
xiangwubit@163.com, lironghuabit@126.com, {cs.xunkai.li, zkf1105, wanggrbit}@gmail.com

ABSTRACT

Graph coarsening reduces the size of a graph while preserving its certain properties. Most existing methods preserve either spectral or spatial characteristics. A recent study has shown that preserving topological features helps maintain the predictive performance of graph neural networks (GNNs) trained on the coarsened graph but suffers from exponential time complexity and fails to preserve persistent homology (PH), which is widely used in graph analysis and machine learning. To address these problems, we propose a scalable topology-preserving graph coarsening method (STPGC) by introducing the concepts of graph strong collapse and graph edge collapse extended from algebraic topology. STPGC comprises three new algorithms based on these two concepts. The first two eliminate *dominated* nodes and edges with graph strong collapse and graph edge collapse, respectively. The third algorithm reverses the graph edge collapse to insert dominated edges, creating new dominated nodes that can be subsequently reduced by graph strong collapse. We further introduce a novel application of STPGC to accelerate the PH computation by preserving it on the coarsened graphs. We theoretically prove the cases where STPGC guarantees PH preservation and propose efficient debiasing techniques to preserve PH otherwise. In addition, we apply STPGC in accelerating GNN training by preserving richer topological features than existing methods to improve GNN performance while achieving higher coarsening efficiency. Experiments on 14 real-life datasets in two applications demonstrate the efficiency and effectiveness of STPGC.

1 INTRODUCTION

Graph-structured data is ubiquitous in real-world scenarios, spanning applications such as social networks, recommender systems, and molecular graphs. As data volumes grow rapidly, analyzing large-scale graphs poses significant computational challenges. One prominent class of techniques to tackle this problem focuses on reducing graph size, including graph coarsening [38, 44, 52, 56], graph sparsification [19, 39, 62, 80, 84], and graph condensation [29, 31, 32, 42, 68]. Among these, graph coarsening has attracted considerable attention due to its solid theoretical foundations and practical effectiveness. It generates a downsized graph by merging specific nodes while preserving certain graph properties. Perhaps its most prevalent application is scaling up graph neural networks (GNNs) by training them on the coarsened graphs while maintaining inference performance on the original graphs [38]. Graph coarsening also finds applications in data augmentation [17, 50], graph partitioning [65], and beyond [35].

Most existing graph coarsening strategies preserve either the graph *spatial* or the *spectral* features. The former captures structural patterns such as diffusion of influence [61] or local connectivity [45]. The latter includes the eigenvalues or the eigenvectors of the graph Laplacian matrix. To preserve these features, they define various metrics to measure the differences between the coarsened and original graphs, such as reconstruction error for connectivity [45] and relative eigenvalue error for the top-k eigenvalues [52, 53], which are used as objective functions for optimization [44]. Additionally, greedy pairwise contraction of node pairs has also been shown to preserve spectral properties [18, 51–53].

Despite the effectiveness of these methods, none of them preserves the *topological* features of graphs [56]. In this context, the term *topological* stems from the theory of algebraic topology [28], characterizing the invariants of data under continuous deformation, such as stretching and compressing [23]. Specifically, graph topological features encompass, but are not limited to, connectivity, rings, and voids. These features capture the essential information of graphs and have been shown to benefit downstream applications [74–76, 87]. For example, rings information has improved the performance of graph representation learning [75, 76, 87], especially in biology and chemistry [1, 69]. Additionally, leveraging local topological structures contributes to improved node classification [20, 36], and link prediction [74]. Furthermore, a recent study has shown that preserving topological features in graph coarsening benefits node classification with GNNs, while disrupting topological structures leads to degraded performance [56].

Although existing studies have demonstrated the importance of topological features, the question of how to effectively preserve them in graph coarsening remains largely unexplored. The only existing method designed for this object is Graph Elementary Collapse (GEC) [56], which is based on *elementary collapse* [71]. Briefly speaking, operators that reduce the size of a graph while preserving its topological features are referred to as collapse, with elementary collapse being the most basic type [23]. To perform elementary collapse on a graph, GEC enumerates all the cliques and maximal cliques in the graph and computes the inclusion relationship between them. It then removes the cliques that are contained solely within another maximal clique. The worst-case time complexity of maximal clique enumeration is $O(3^{n/3})$ [41, 56], where n is the number of nodes. Due to its exponential time complexity, it faces scalability challenges on large-scale graphs. To reduce the time complexity, GEC partitions a graph into numerous small subgraphs and then applies elementary collapse to each subgraph separately [56]. The final coarsened graph is reconstructed by connecting these small subgraphs. However, GEC poses two major limitations. Firstly, the partitioning process inevitably disrupts the topological features of the graph. As the number of partitioned subgraphs increases with the graph size, the problem becomes rather severe on large graphs. Secondly, although n can be controlled by the size of partitioned subgraphs, the exponential time complexity remains undesirable, and the reconstruction introduces extra overhead. These problems limit the effectiveness and efficiency of GEC.

To address the above problems, we draw inspiration from the strong collapse and edge collapse in algebraic topology [4, 8–10]. These two concepts involve exclusively the nodes and edges in a simplicial complex, making it well suited for graph analysis. Building on this insight, we extend these concepts to graphs by introducing graph strong collapse and graph edge collapse. Specifically, the closed neighborhood of a node u is denoted as

$N[u] = N(u) \cup u$, and u is called dominated by another node v if and only if $N[u] \subseteq N[v]$. Removing a dominated node from a graph is called a graph strong collapse. Similarly, the closed neighborhood of an edge is defined as $N[x, y] = N[x] \cap N[y]$, and (x, y) is dominated by another node v if and only if $N[x, y] \subseteq N[v]$. Removing a dominated edge is called a graph edge collapse. We prove that both graph strong collapse and graph edge collapse preserve the topological features of the graph.

Unlike graph elementary collapse, graph strong and edge collapse eschew costly clique enumeration by directly identifying reducible nodes and edges through neighborhood inclusion, enabling the development of more efficient graph coarsening. Based on these two concepts, we introduce scalable topology-preserving graph coarsening (STPGC), consisting of three graph coarsening algorithms rigorously preserving the topological features. Specifically, we propose GStrongCollapse and GEdgeCollapse, which iteratively identify dominated nodes and edges and reduce them, respectively. However, the two algorithms are insufficient to achieve the desired reduction ability. To enable further coarsening beyond the two algorithms, we propose *neighborhood coning* algorithm to insert certain dominated edges between the neighbors of nodes to create new dominated nodes and further reduce them with graph strong collapse. The three algorithms are synergistically applied in downstream applications to ensure reduction ability and efficiency.

Application 1: Accelerating PH Computation. Persistent homology (PH) is a foundational technique in topological data analysis, with broad applications in graph analysis and machine learning [23, 74, 77]. In graph analysis, PH is employed to capture the distribution of cliques and rings [64, 67] and to facilitate graph comparison [22, 66]. In graph representation learning, it serves as a powerful topological descriptor for both graph-level [15, 40, 48, 63, 85] and node/link-level tasks [20, 74, 86]. However, common algorithms for computing PH rely on matrix reduction techniques [25], which have a time complexity of $O(\tilde{n}^3)$, where $\tilde{n}$ denotes the number of simplices (i.e., cliques). This substantial computational overhead limits the scalability of PH, hindering its applications on large-scale graphs, such as large protein structures [73], brain networks [66, 70], and complex systems [81]. To accelerate PH computation, we propose preserving PH on coarsened graphs of reduced sizes, supporting the use of any existing PH algorithms. While GEC does not guarantee to preserve PH, we theoretically prove that STPGC strictly preserves PH if the filtration value of a reduced node or edge is no less than that of its dominating node and the edges required for dominance. Otherwise, it may be biased. To correct this potential bias, we introduce a novel debiasing method that adjusts the filtration values of edges in the neighborhoods of dominated nodes or edges, ensuring PH is preserved in all cases.

Application 2: Accelerating GNN Training. Graph coarsening has become an essential technique for improving the efficiency and scalability of GNNs. Unlike most existing methods that focus on preserving spatial or spectral properties, we propose to improve the performance of GNNs trained on coarsened graphs by effectively retaining topological features. To this end, we first adopt an exact coarsening strategy to strictly preserve topological features with the combination of the proposed STPGC algorithms. However, the required coarsening ratio is sometimes difficult to achieve, as retaining full topological features while reducing the graph to its minimal requires finding an optimal collapse order—a problem that is NP-hard [27]. To address this problem, we further adopt approximate coarsening. If the desired coarsening ratio is not met in exact coarsening, we relax the dominance condition in graph strong collapse, allowing up to k nodes in $N[u]$ to be excluded from $N[v]$. While approximate coarsening does not offer strict guarantees, the degradation in topology occurs progressively, striking a balance between computational efficiency and topological fidelity. In contrast to GEC, which breaks topological integrity from the beginning of coarsening, our approach preserves topology throughout the exact coarsening stage, ensuring superior topological preservation while maintaining higher computational efficiency.

To summarize, our contributions are: (1) **Novel Concepts and Algorithms:** We introduce graph strong collapse and graph edge collapse, extended from algebraic topology to graph analysis. Building on the concepts, we propose two scalable, topology-preserving graph coarsening algorithms. Furthermore, we introduce neighborhood coning—a topology-preserving graph transformation technique enabling further reduction by inserting dominated edges within a node's neighborhood. (2) **Important Applications:** We study two applications of the proposed algorithms, accelerating persistent homology (PH) computation and accelerating GNN training. To our knowledge, we are the first to apply graph coarsening to accelerate PH computation on graphs. (3) **Extensive Experiments:** We conduct extensive experiments on the two applications. Results demonstrate: (i) In accelerating PH computation, STPGC achieves up to a 16.2x speedup in runtime and up to a 14x reduction in memory usage compared to computing PH on original graphs; (ii) in accelerating GNN training, STPGC outperforms state-of-the-art (SOTA) approaches in node classification while delivering up to a 37x runtime improvement over GEC. (4) **Reproducibility:** The source code is available at https://github.com/BITNEO/STPGC.

2 PRELIMINARIES

2.1 Notations and Concepts

Graph Coarsening. We denote a graph as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ and $\mathcal{E}$ are the sets of n nodes and m edges. If the nodes have d-dimensional features and labels, $\mathbf{X} \in \mathbb{R}^{n \times d}$ represents the feature matrix of the nodes, and $\mathbf{Y} \in \mathbb{N}^n$ represents the labels of the nodes. Graph coarsening aims to derive a graph $\mathcal{G}^c = (\mathcal{V}^c, \mathcal{E}^c)$ and the corresponding $\mathbf{X}^c$ and $\mathbf{Y}^c$, where $|\mathcal{V}^c| << |\mathcal{V}|$, while preserving the key characteristics of $\mathcal{G}$.

Simplex. In Euclidean space $\mathbb{R}^n$, a k-simplex τ is defined as the convex hull of $k+1$ affinely independent points, where k denotes its dimension [23]. In the context of graphs, a k-simplex corresponds to a complete subgraph of $k+1$ nodes: a 0-simplex represents a node, a 1-simplex represents an edge, a 2-simplex stands for a triangle, and higher-order simplices represent larger cliques [23].

Simplicial complex. An abstract simplicial complex $\mathcal{K}$ is a collection of subsets of a nonempty finite set, such that for every subset $\tau \in \mathcal{K}$, all subsets of τ are also included in $\mathcal{K}$ [9]. The elements of $\mathcal{K}$ are called simplices. Specifically, when a simplicial complex is constructed from nodes, edges, triangles, and higher-order cliques—where each subset of nodes forms a complete subgraph—it is referred to as a **clique complex**.

Elementary Collapse [71]. Given a k-simplex τ and an integer k' such that $0 \leq k' \leq k$, a k'-simplex σ is called a face of τ if $\sigma \subseteq \tau$. A simplex is said to be a maximal face if it is not a face of any other faces. If τ is a maximal face and σ is a $k-1$ face of τ, we call removing σ and τ an elementary collapse. We refer readers to [56] for illustrative examples.

Graph Elementary Collapse (GEC) [56] extends the concept of elementary collapse to graphs by transforming a graph to a clique complex by considering a $k+1$−clique as a k−simplex. To perform elementary collapse, GEC enumerates all cliques and

identifies inclusion relationships among them. However, clique enumeration has a worst-case time complexity of $O(3^{n/3})$, which becomes impractical for large graphs. In addition, GEC cannot guarantee to preserve persistent homology, which is widely used in graph analysis and machine learning [23, 74, 77]. To address these issues, we introduce two more scalable operators called graph strong collapse and graph edge collapse.

2.2 Graph Strong and Edge Collapses

1-skeleton. The 1-skeleton of a clique complex $\mathcal{K}$ is the subcomplex consisting of all simplices in $\mathcal{K}$ with dimension at most 1. In other words, it includes only the nodes and edges.

The strong and edge collapses on clique complexes, introduced by Boissonnat *et al.* [8, 9], can be directly applied to the 1-skeleton. In particular, when a graph $\mathcal{G}$ is transformed into its clique complex $\mathcal{K}$, the 1-skeleton of $\mathcal{K}$ is identical to $\mathcal{G}$ itself. This observation motivates us to extend the concepts of strong and edge collapses from algebraic topology to graphs. To this end, we first define relevant notions of neighborhoods and then formally introduce graph strong collapse and graph edge collapse.

The *open neighborhood* of a node $u \in \mathcal{V}$ is defined as $N(u) = \{v \in \mathcal{V} | (u, v) \in \mathcal{E}\}$. We denote $deg(u) = |N(u)|$ as the degree of u. The *closed neighborhood* of u is defined as $N[u] = \{u\} \cup N(u)$. The *open and closed neighborhood of an edge* $(x, y) \in \mathcal{E}$ is similarly defined as $N(x, y) = N(x) \cap N(y)$ and $N[x, y] = N[x] \cap N[y]$, respectively.

Definition 1. *(Graph strong collapse) On a graph $\mathcal{G}$, u is said to be dominated by a node v if and only if $N[u] \subseteq N[v]$, and u is called a **dominated** node if it is dominated by another node. Removing a dominated node and its connected edges from $\mathcal{G}$ is called a **graph strong collapse**.*

Example 1. *In Figure 1(a), the red leaf nodes are dominated by their parent nodes. We can remove the red nodes with graph strong collapse and their parent nodes then become dominated, which can be further removed via graph strong collapse.*

Definition 2. *(Graph edge collapse). An edge (x, y) is said to be dominated by a node v if and only if $N[x, y] \subseteq N[v]$, where $v \notin \{x, y\}$. (x, y) is called a **dominated** edge if it is dominated by another node. Removing a dominated edge from $\mathcal{G}$ is called a **graph edge collapse**.*

Example 2. *An example of dominated edges is illustrated in Figure 1(b), where all red edges are dominated. The three outer red edges are dominated by the central node, while each of the three inner red edges is dominated by one of the outer nodes. Reducing a dominated edge is a graph edge collapse. Once the three outer edges are eliminated, the inner edges are no longer dominated.*

Homotopy. An important property of collapses, including elementary collapse, strong collapse, and edge collapse—is their ability to preserve homotopy equivalence between a simplicial complex $\mathcal{K}$ and the reduced complex $\mathcal{L}$ obtained through these operations. Two simplicial complex are said to be homotopy equivalent if their corresponding topological spaces can be continuously deformed into one another [23]. Two graphs are homotopy equivalent if their corresponding clique complexes are homotopy equivalent. Homotopy-equivalent graphs retain key topological invariants—such as connected components, rings, voids, and Betti numbers—that capture both local and global connectivity structures [56]. The following Lemma shows that the graph obtained by graph strong collapse and graph edge collapse is homotopy equivalent to the original graph.

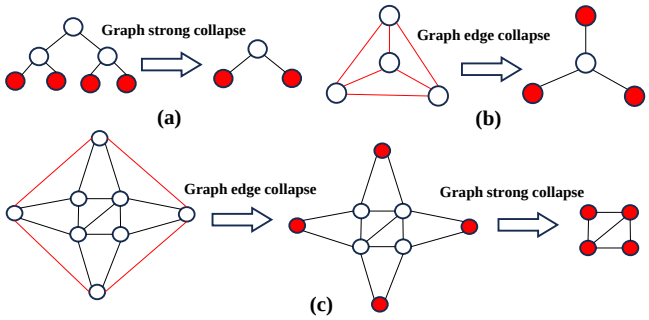

Figure 1: Examples of graph strong collapse and graph edge collapse. The dominated nodes and edges are shown in red.

Lemma 1. *(Homotopy Equivalent) Let $\mathcal{G}^c$ be a subgraph derived from $\mathcal{G}$ through a series of graph strong collapses and graph edge collapses, then $\mathcal{G}^c$ and $\mathcal{G}$ are homotopy equivalent.*

Proof. The proof is based on the one-to-one correspondence between a 1-skeleton and its clique complex. We denote the clique complex of $\mathcal{G}$ as $\mathcal{K}$. Therefore, $\mathcal{G}$ is the 1-skeleton of $\mathcal{K}$. If $\mathcal{G}^c$ is derived from a series of graph strong collapse and graph edge collapse from $\mathcal{G}$, and $\mathcal{K}^c$ is the clique complex derived through reducing the same nodes and edge via strong collapse and edge collapse, then $\mathcal{G}^c$ is also the 1-skeleton of $\mathcal{K}^c$, as $\mathcal{K}^c$ and $\mathcal{K}$ are homotopy equivalent, we have $\mathcal{G}^c$ and $\mathcal{G}$ are homotopy equivalent. □

Beyond homotopy, we show that graph strong collapse and graph edge collapse also preserve a stricter topological property called persistent homology (PH) that elementary collapse fails to maintain. PH not only preserves the number of topological features but also the saliency of these features, as will be detailed in Section 4.1.

3 GRAPH COARSENING ALGORITHMS

In this section, we present three topology-preserving graph coarsening algorithms. The first two, GStrongCollapse and GEdgeCollapse, implement the concepts of graph strong collapse and graph edge collapse. The third algorithm, NeighborhoodConing, inserts dominated edges within node neighborhoods to create new dominated nodes, enabling further reductions via graph strong collapse. We describe each algorithm in detail below.

3.1 Graph Strong Collapse

Our core idea is that if a node u is dominated by another node v, it can be removed via graph strong collapse by merging u into v to form a supernode. By iteratively eliminating the dominated nodes, the graph is progressively coarsened. Algorithm 1 implements the above idea by maintaining a queue to track nodes that need to be checked for dominance. For each node u, the algorithm iterates through its neighbors to check whether u is dominated by any of them (lines 7–10). If a dominating node v is found, u is removed and merged into v, forming a supernode. The loop exits once a dominating node is identified, avoiding redundant checks. Since the removal of u may alter the dominance status of its neighbors, all nodes in $N(u)$ are pushed into the queue for reevaluation (line 12). Practically, a node u can only be dominated if $deg(v) \geq deg(u)$ (line 8). During neighborhood inclusion checks (line 10), if there exists any element in $N[u]$ that is not in $N[v]$,

Algorithm 1: GStrongCollapse

Input: $\mathcal{G} = (\mathcal{V}, \mathcal{E})$.
Output: The coarsened graph $\mathcal{G}^c = \{\mathcal{V}^c, \mathcal{E}^c\}$.

```
1  NodeQueue ← push all the nodes in V;
2  G^c ← G;
3  while NodeQueue and the coarsening ratio is not achieved do
4      u ← pop(NodeQueue);
5      N[u] ← the closed neighborhood of u;
6      if deg(u) ≤ θ1 then
7          for v in N(u) do
8              if deg(v) ≥ deg(u) then
9                  N[v] ← the closed neighborhood of v;
10                 if N[u] ⊆ N[v] then
11                     delete u from G^c;
12                     Push each node in N(u) into NodeQueue if it's not in
                         NodeQueue;
13                     break;
14 return G^c
```

u can be immediately determined as not dominated by v. To accelerate the dominance checking, we construct a hash table for the neighborhood of each node. Furthermore, removing a dominated node by directly modifying a sparse matrix incurs a time complexity of $O(|N(u)| + |N(v)|)$. To improve efficiency, we employ a lazy deletion strategy with $O(1)$ time complexity by simply marking the node as deleted. Moreover, since high-degree nodes are rare and less likely to be dominated, we skip nodes whose degrees exceed a predefined threshold parameter θ_1. It is easy to see that Algorithm 1 correctly implements the graph strong collapse process as formulated in Definition 1.

Complexity Analysis. Given a graph $\mathcal{G}$ with m edges and an adjacency list where each node's neighbors are stored in a hash table, checking whether $N[u] \subseteq N[v]$ requires $O(min(deg(u), deg(v)))$ checks. Therefore, the worst-case time complexity of Algorithm 1 is $O\left(\sum_{(u,w)\in\mathcal{E}}(min(\deg(u), \deg(v)))\right) \leq O(m \times \theta_1)$ time, where θ_1 is a small constant on most real-world graphs. The space complexity of Algorithm 1 is $O(|\mathcal{V}| + |\mathcal{E}|)$.

3.2 Graph Edge Collapse

Although graph strong collapse effectively simplifies certain structures such as trees and cliques, it struggles with more complex structures. Figure 1(c) gives an example of such a situation, where none of the nodes are initially dominated. However, by first applying graph edge collapse to remove the dominated red edges, new dominated nodes emerge and become eligible for further reduction. Algorithm 2 depicts the procedure for coarsening a graph via graph edge collapse. Similar to Algorithm 1, it iteratively checks each edge for the dominance condition and deletes dominated edges. When an edge is deleted, its adjacent edges are added to the *EdgeQueue* as their open neighborhoods have changed. We also introduce the degree threshold θ_1 to enhance efficiency by skipping edges (x, y) for which $deg(x) + deg(y) > 2\theta_1$. Also, it is easy to show that Algorithm 2 correctly implements the graph edge collapse process as defined in Definition 2.

Complexity Analysis. The worst-case time complexity of computing the open neighborhood of all edges is $O\left(\sum_{(x,y)\in\mathcal{E}}(min(deg(x), deg(y))\right) \leq O(m \times \theta_1)$. Additionally, checking whether there exists a node $v \in N(x, y)$ such that $N(x, y) \subseteq N[v]$ requires examining all nodes in $N(u, v)$ in the worst case, resulting in $O(\sum_{(x,y)\in\mathcal{E}} |N(x, y)|^2) \leq O(\sum_{(x,y)\in\mathcal{E}} min(deg(x), deg(y))^2) \leq O(m\theta_1^2)$ checks.

Algorithm 2: GEdgeCollapse

Input: $\mathcal{G} = (\mathcal{V}, \mathcal{E})$.
Output: The coarsened graph $\mathcal{G}^c = \{\mathcal{V}^c, \mathcal{E}^c\}$.

```
1  EdgeQueue ← push all the edges in E;
2  G^c ← G;
3  while EdgeQueue do
4      (x, y) ← pop(EdgeQueue);
5      if deg(x) + deg(y) ≤ 2θ1 then
6          N(x, y) ← the open neighborhood of x and y;
7          for v in N(x, y) do
8              if N(x, y) ⊆ N[v] then
9                  delete (x, y) from G^c;
10                 Push each edge connected to x or y into EdgeQueue , if it's not in
                     EdgeQueue ;
11                 break;
12 return G^c
```

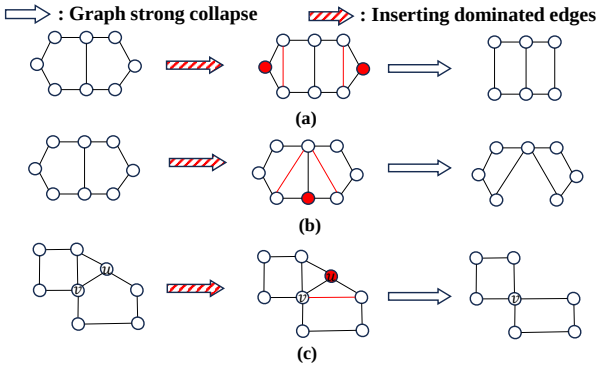

Figure 2: Examples of neighborhood coning.

Consequently, the worst-case time complexity of Algorithm 2 is $O(m\theta_1^2)$, where θ_1 is a small constant on most real-world graphs. The space complexity of Algorithm 2 is $O(|\mathcal{V}| + |\mathcal{E}|)$.

3.3 Neighborhood Coning

Although graph strong collapse and edge collapse effectively eliminate dominated nodes and edges, certain scenarios remain where they cannot be applied. For example, the left graph in Figure 2(a) contains neither dominated nodes nor edges, yet it can be further shortened while preserving its topological features. To explain this process, we can decompose it into two steps: first inserting dominated edges (the red edges) into the graph, and then creating new dominated nodes (the red nodes). Such a process is an *inverse* of a graph edge collapse. Here, the inverse means that after adding a dominated edge to the graph, we can obtain the original graph through a graph edge collapse, thereby the homotopy equivalence is also guaranteed. The red nodes can then be reduced through graph strong collapse. This approach is not restricted to creating 2-degree dominated nodes, nor is it limited to inserting just one edge (Figure 2(b), (c)). This insight naturally leads to a generalized strategy: we can create dominated nodes of arbitrary degree and subsequently reduce them, enabling further coarsening.

Building on the above analysis, we develop a topology-preserving graph transformation technique to create new dominated nodes and further reduce them with graph strong collapse. The central idea involves inserting dominated edges into the graph to create new dominated nodes. This raises two questions: (1) Where should the dominated edges be inserted? (2) How many dominated edges need to be inserted to transform a node dominated? Regarding the two questions, we have the following observation. Assume v is the node dominating u after

Algorithm 3: NeighborhoodConing

Input: $\mathcal{G} = (\mathcal{V}, \mathcal{E})$.
Output: The coarsened graph $\mathcal{G}^c = \{\mathcal{V}^c, \mathcal{E}^c\}$

```
NodeQueue ← Initialize a priority queue of nodes with ascending degrees;
G^c ← G;
while NodeQueue and the coarsening ratio is not achieved do
    u ← NodeQueue.pop();
    if deg(u) ≤ θ2 then
        for v in N(u) do
            Inserted ← True;
            InsertEdgeList ← An empty list;
            for w in N(u)/v do
                if (v,w) not exist then
                    if (v, w) is dominated then
                        Add (v, w) into InsertEdgeList;
                    else
                        Inserted ← False;
                        break;
            if Inserted is True then
                break;
        if Inserted is True then
            Insert edges in InsertEdgeList into G^c and Delete u;
            Update deg(v) in NodeQueue;
return G^c
```

the insertion. The inserted edges must connect v to u's neighbors; otherwise, they cannot establish dominance. Furthermore, since u can only be dominated by its neighbors, v must be a neighbor of u. Specifically, if a neighbor v of u can dominate u through edge insertion, then v must connect to all nodes in $N(u) \setminus \{v\}$ afterward. Thus, v can dominate u by inserting at most $\deg(u) - 1$ edges. This process resembles constructing a cone over the closed neighborhood of u, with v as the apex. We formalize it as neighborhood coning, as defined in Definition 3.

Definition 3. *(Neighborhood Coning) Given a node $u \in \mathcal{V}$, we say u can be neighborhood coned by a neighbor $v \in N(u)$ if for every $w \in N(u) \setminus \{v\}$, the edge (v, w) either: 1) exists in $\mathcal{G}$, or 2) can be inserted as a dominated edge. The process of adding these required dominated edges and subsequently reducing u is called a neighborhood coning on u.*

Example 3. *Figures 2(a)-(c) demonstrate examples of neighborhood coning through red arrows indicating the insertion of dominated edges, and red nodes indicating newly created dominated nodes. Figures 2(a) and (b) illustrate examples of neighborhood coning by inserting $\deg(u) - 1$ edges. In Figure 2 (c), one of the neighbors of the dominated node u is already connected to v, by inserting the red dominated edge, u becomes dominated by v.*

Degree Ascending Algorithm. While a straightforward approach to implement neighborhood coning is to examine each node $u \in \mathcal{V}$ for possible neighborhood coning, this method fails to maximize the coarsening potential. As illustrated in Figure 2 (a) and (b), different node selections lead to varying coarsening outcomes: the upper example reduces two nodes, while the lower one reduces only a single node. This is because higher-degree nodes often involve more complex topological features than lower-degree nodes. To maximize the number of reduced nodes, we process nodes in ascending degree order. The developed algorithm, called NeighborhoodConing, is described in Algorithm 3. For each node $u \in \mathcal{V}$, it identifies whether any neighbor v can dominate u via neighborhood coning by ensuring connections (either existing or through inserted dominated edges) to all of u's other neighbors. If such a neighbor is found, the dominated edges are inserted, and u is removed along with its adjacent edges. In addition, any inserted edge dominated by nodes other than u remains dominated after u's reduction, which can also be safely reduced.

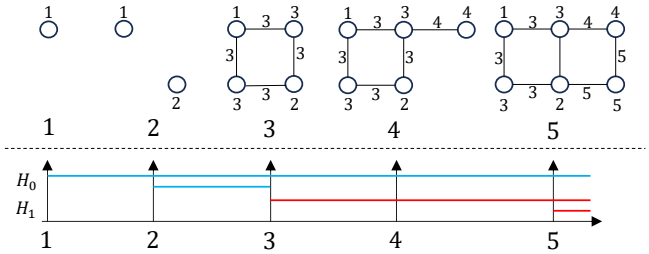

Figure 3: An example of a filtration and its persistent barcode.

Complexity Analysis. The neighborhood coning verification for a node u requires, in the worst case, examining $O(deg(u) \times (deg(u) - 1)/2)$ edge dominance. This leads to a worst-case time complexity of $O\left(deg(u) \times (deg(u) - 1)d_{\max}^2\right) = O\left(deg(u)^2 d_{\max}^2\right)$, where $d_{\max}$ is the maximum degree of nodes. This quadratic growth with respect to $deg(u)$ makes the verification process particularly expensive for high-degree nodes. In practice, such high-degree nodes not only appear infrequently in real-world graphs but also tend to have more complex neighborhood structures, making them less likely to be reducible via neighborhood coning. To improve scalability, we propose skipping nodes with degree greater than a predefined threshold θ_2. This reduces the per-node worst-case complexity from $O(\deg(u)^2 d_{\max}^2)$ to $O(\theta_2^2 d_{\max}^2)$. As a result, the worst-case time complexity of Algorithm 3 is $O(n\theta_2^2 d_{\max}^2)$. On real-world sparse graphs, the average cost of edge dominance checks can be amortized to $O(\bar{d}^2)$ per edge, where $\bar{d}^2$ denotes the average node degree. Thus, the overall complexity is amortized to $O(n\theta_2^2 \bar{d}^2)$, where $\bar{d} << d_{\max}$.

4 APPLICATIONS

We introduce two applications of the proposed algorithms, i.e, accelerating persistent homology (PH) computation and accelerating GNN training. In both applications, we employ a combination of the three algorithms to obtain a coarsened graph, on which the downstream tasks can be more efficiently performed.

4.1 Accelerating PH Computation

We first give a brief introduction to persistent homology on graphs and then introduce how we accelerate its computation. Persistent homology [25, 26] characterizes the saliency of parameterized topological features on a graph $\mathcal{G}$ with a persistent barcode [14, 34]. For this purpose, each simplex (node, edge, triangles...) is assigned a filtration value by a function f. As a common practice, we can first define the function $f : \mathcal{V} \to \mathbb{R}$ on nodes [21, 57]. Then, the filtration values for higher-order simplices, such as edges and triangles, are the maximum value of their nodes. Given the filtration value on simplices, a graph $\mathcal{G}_t$ is defined as $\{\tau | \tau \in \mathcal{G}, f(\tau) \leq t\}$. When t increases from $-\infty$ to $+\infty$, a sequence $\emptyset = \mathcal{G}_{-\infty} \subseteq \cdots \subseteq \mathcal{G}_{+\infty} = \mathcal{G}$ is obtained [74]. During this process, topological features gradually born and die. Persistent barcode records the birth and death times of these features to characterize their persistence. Features that are born, but do not die, are recognized as essential features. The number of k-dimensional essential features is called the k-Betti number of the graph.

Example 4. *The upper and lower part of Figure 3 illustrates a filtration and its persistent barcode. The left and right ends of a line in the barcode represent the birth and death of a topological feature. In the barcode, H_0 (blue lines) and H_1 (red lines) represent the 0-dimensional*

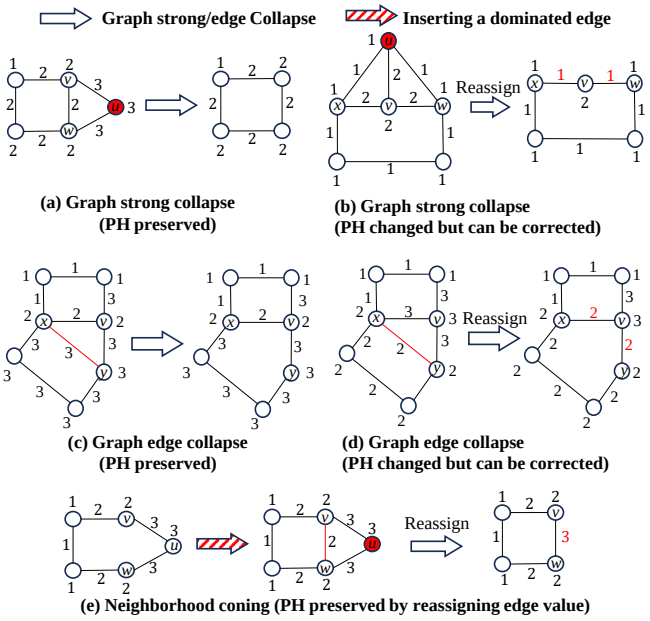

Figure 4: Examples of the bias of the three operators and debiasing PH. The red edge values denote the reassigned filtration values.

topological features (connected components) and the 1-dimensional topological features (rings), respectively. At filtration value = 1, a node appears, indicating the birth of a connected component. At filtration value = 2, another isolated node appears, giving rise to the second connected component. However, at filtration value = 3, it merges with the first connected component and thus dies. The first connected component persists throughout the entire scale. In H_1, two rings appear at filtration values = 3 and 5, respectively, and never die. Therefore, the 1-Betti number is 2.

We aim to preserve the same persistent barcode of $\mathcal{G}$ on its coarsened graph $\mathcal{G}^c$. Here, we focus on computing 1-dimensional barcodes (H_1), which captures rings, as H_0 can be computed very efficiently [5, 55]. Higher-dimensional features are rare in real-world graphs and are typically omitted [74, 77]. Notably, all rings are essential features. Moreover, H_1 finds wide applications in graph analysis [64, 67] and machine learning tasks, including enhancing the expressive power of GNNs for representation learning [40, 75, 87], node classification [1, 69], and link prediction [74].

Bias of the Coarsened Graph. Since GStrongCollpase, GEdgeCollapse, and NeighborhoodConing preserve the number of essential features (Betti numbers), the persistent barcodes computed on $\mathcal{G}^c$ have the same cardinality (i.e., the number of persistent lines) as that in $\mathcal{G}$. However, the persistent barcodes can be biased in $\mathcal{G}^c$.

Example 5. *As shown in Figure 4, for the graph strong collapse in (a), the 4-ring is born at the filtration value 2 both before and after the strong collapse, so H_1 remains unchanged. In (b), the 4-ring is born at value 1 before the collapse but shifts to 2 after a graph strong collapse. This bias can be corrected by reassigning the filtration values of the edges (x, v) and (v, w) to 1. For graph edge collapse, in (c), collapsing the red edge does not affect the birth times of the two rings, thereby preserving H_1. However, in (d), the birth time of the lower 5-ring changes from 2 to 3. This can be corrected by reassigning the filtration values of (x, v) and (y, v) to 2. In (e), before inserting the edge, the ring is born at 3. If we assign the red edge following the node induced filtration function, the ring will be born at 2. However, by assigning a value of 3 to (v, w), the persistent homology is preserved.*

The above examples demonstrate that while the three operators preserve persistent homology in some cases, they may introduce bias in others. This bias, however, can be corrected by reassigning filtration values to specific edges. In the following lemmas, we formally characterize the conditions under which persistent homology is preserved or biased, and subsequently propose debiasing methods based on filtration value reassignment to edges.

Lemma 2. *(PH preservation of graph strong collapse) Given a graph with node-induced filtration values, reducing a node u dominated by v with i) $deg(u) \leq 2$ or ii) $deg(u) > 2$ and $f(u) \geq f(v)$ via graph strong collapse preserves persistent homology. However, if $deg(u) > 2$ and $f(u) < f(v)$, the persistent homology can be biased.*

Proof. We define the filtration of a graph $\mathcal{G}$ as $\mathcal{G}_1 \subseteq \mathcal{G}_2 \subseteq \cdots \subseteq \mathcal{G}_{p-1} \subseteq \mathcal{G}_p = \mathcal{G}$, where the subscripts represent the indexes of snapshots, and p denotes the length of the filtration. Then, we introduce the diagram $\mathcal{D}$ shown below, in which each ψ_i and ψ_i^c, for $i = 1, 2, \ldots, p-1$, represent the inclusion relationships of a node (along with its associated edges), and each of $\phi_1, \phi_2, \ldots, \phi_p$ corresponds to a series of strong graph collapses. If all the relationships in the diagram hold, we say that the diagram commutes.

$$\begin{array}{ccccccc} \mathcal{G}_1 & \xrightarrow{\psi_1} & \mathcal{G}_2 & \xrightarrow{\psi_2} & \cdots & \longrightarrow \mathcal{G}_{p-1} & \xrightarrow{\psi_{p-1}} \mathcal{G}_p \\ \downarrow{\scriptstyle \phi_1} & & \downarrow{\scriptstyle \phi_2} & & & \downarrow{\scriptstyle \phi_{p-1}} & \downarrow{\scriptstyle \phi_p} \\ \mathcal{G}_1^c & \xrightarrow{\psi_1^c} & \mathcal{G}_2^c & \xrightarrow{\psi_2^c} & \cdots & \longrightarrow \mathcal{G}_{p-1}^c & \xrightarrow{\psi_{p-1}^c} \mathcal{G}_p^c \end{array}$$

Next, we map each $\mathcal{G}_i$ and $\mathcal{G}_i^c$ to their corresponding clique complexes $\mathcal{K}_i$ and $\mathcal{K}_i^c$, resulting in a new diagram $\mathcal{D}'$, as shown below. In $\mathcal{D}'$, χ_i and χ_i^c represent the same inclusion relationships as ψ_i and ψ_i^c, but they are now defined between clique complexes. Moreover, $\phi_1, \phi_2, \ldots, \phi_p$ are replaced by graph strong collapses applied to the graph (1-skeletons) of $\mathcal{K}_1, \mathcal{K}_2, \ldots, \mathcal{K}_p$. If $\mathcal{D}'$ commutes, then the filtration of $\mathcal{G}_p$ and $\mathcal{G}_p^c$ have the same PH (Theorem 2 in [10]). Since $\mathcal{G}_i$ and $\mathcal{G}_i^c$ are the 1-skeletons of $\mathcal{K}_i$ and $\mathcal{K}_i^c$, if $\mathcal{D}$ commutes, then $\mathcal{D}'$ also commutes. Therefore, we only need to analyze whether the conditions ensure that $\mathcal{D}$ commutes.

$$\begin{array}{ccccccc} \mathcal{K}_1 & \xrightarrow{\chi_1} & \mathcal{K}_2 & \xrightarrow{\chi_2} & \cdots & \longrightarrow \mathcal{K}_{p-1} & \xrightarrow{\psi_{p-1}} \mathcal{K}_p \\ \downarrow{\scriptstyle \gamma_1} & & \downarrow{\scriptstyle \gamma_2} & & & \downarrow{\scriptstyle \gamma_{p-1}} & \downarrow{\scriptstyle \gamma_p} \\ \mathcal{K}_1^c & \xrightarrow{\chi_1^c} & \mathcal{K}_2^c & \xrightarrow{\chi_2^c} & \cdots & \longrightarrow \mathcal{K}_{p-1}^c & \xrightarrow{\chi_{p-1}^c} \mathcal{K}_p^c \end{array}$$

We now analyze the conditions specified in the lemma. (1) If we reduce a node u where $deg(u) = 2$ through a graph strong collapse in $\mathcal{G}_p$, we always have $deg(u) \leq 2$ in previous graphs, so it is also dominated in previous graphs and can be reduced. Therefore, the inclusion relationships ψ_1^c,ψ_2^c,...,ψ_{p-1}^c are satisfied. (2) If $deg(u) > 2$ and $f(u) \geq f(v)$, u is also dominated since its appearance because v appears before u in a filtration. (3) However, if $deg(u) > 2$ and $f(u) < f(v)$, u is dominated in $\mathcal{G}_p$ does not guarantee that it is dominated in previous graphs. The left graph in Figure 4(b) gives such an example, where u is dominated at filtration value 2 but not dominated at filtration value 1. If such a node u exists, the above diagram fails to commute, as there exists a $i \leq p$, such that u is

Algorithm 4: STPGCForPH

Input: $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, f: filtration function.
Output: $\mathcal{G}^c$: Coarsened graph, f': the re-valued filtration function.

while *the number of iterations* $< \delta_1$ **do**
  $\mathcal{G}^c, f \leftarrow$ *PHGStrongCollapse*($\mathcal{G}^c, f$);
  $\mathcal{G}^c, f \leftarrow$ *PHGEdgeCollapse*($\mathcal{G}^c, f$);
$\mathcal{G}^c$ $f' \leftarrow$ *PHNeighborhoodConing*($\mathcal{G}^c, f$);
return $\mathcal{G}^c, f'$;
Procedure *DebiasReducedNode(u, v, f)*
  for *w in N(u)* **do**
    if $f(v, w) > f(u)$ **then**
      $f(v, w) = f(u)$
  return f;
Procedure *DebiasReducedEdge((x, y), v, f)*
  for *w in N[x,y]* **do**
    if $f(v, w) > f(x, y)$ **then**
      $f(v, w) = f(x, y)$
  return f;
Procedure *PHGStrongCollapse*($\mathcal{G}^c, f$)
  NodeQueue $\leftarrow$ *push* all the nodes in $\mathcal{V}$;
  while *NodeQueue* **do**
    Apply GStrongCollapse (Line 4-13 of Algorithm 1);
    if *u is deleted and* $deg(u) > 2$ *and* $f(u) < f(v)$ **then**
      DebiasReducedNode(u, v, f); /*u is dominated by v*/
Procedure *PHGEdgeCollapse*($\mathcal{G}^c, f$)
  EdgeQueue $\leftarrow$ *push* all the edges in $\mathcal{E}$;
  while *EdgeQueue* **do**
    Apply GEdgeCollapse as in ;
    if $(x, y))$ *is deleted and f(x,y)<f(v)* **then**
      DebiasReducedEdge(e,v,f); /*(x,y) is dominated by v*/
  return $\mathcal{G}^c, f'$;
Procedure *PHNeighborhoodConing*($\mathcal{G}^c, f$)
  Initialize as in lines 1-3 of Algorithm 3;
  while *NodeQueue* **do**
    EdgeQueue $\leftarrow$ *push* all the edges in $\mathcal{E}$;
    Apply NeighborhoodConing (lines 5–20 in Algorithm 3);
    if *u is deleted* **then**
      DebiasReducedNode(u, v, f); /*u is dominated by v*/
  return $\mathcal{G}^c, f$;

reduced via strong collapse in $\mathcal{G}_i$, but cannot be reduced in $\mathcal{G}_{i-1}$, violating the inclusion relationship of ψ^c_{i-1}.

□

Lemma 3. *(PH preservation of graph edge collapse) Reducing an edge (x, y) dominated by v preserves persistent homology if $f(x, y) \geq f(v)$, where v is the node dominating (x, y). However, if $f(x, y) < f(v)$, the persistent homology can be biased.*

Proof. The proof follows the same idea of Lemma 2, except that it builds upon Theorem 4 from [9], which replace with $\gamma_1, \gamma_2, ..., \gamma_p$ with edge collapse. If $f(x, y) \geq f(v)$, If $deg(u) > 2$ and $f(u) \geq f(v)$, u is also dominated since its appearance because v appears in the previous of the same as u in a filtration. So it's dominated in the later snapshots. So the inclusion relationship of edges $\phi_1, \phi_2, \ldots, \phi_p$ holds thus $\mathcal{D}$ commutes. If $f(x, y) < f(v)$, the $\mathcal{D}$ fails to commute, as there exists a $i \leq p$, such that (x, y) is reduced via graph edge collapse in $\mathcal{G}_i$, but cannot be reduced in $\mathcal{G}_{i-1}$, violating the inclusion relationship of ψ^c_{i-1}. The left graph in Figure 4(d) gives such an example, where (x, y) is dominated at filtration value 3 but not dominated at filtration value 2.

□

Lemma 4. *(PH preservation of neighborhood coning) Reducing a node u via neighborhood coning preserves persistent homology if $f(u) \geq f(v)$, where v is the node dominating u after the insertion of edges. If $f(u) < f(v)$, the persistent homology can be biased.*

Debiasing Persistent Homology. The theoretical foundations for debiasing persistent homology (PH) in graph strong collapse and graph edge collapse are presented in Lemma 5 and Lemma 6, respectively. The core idea is as follows: if a node u or an edge (x, y) appears earlier than the node v that dominates it in a filtration (i.e., $f(u) < f(v)$ or $f(x, y) < f(v)$), we debias PH by ensuring that u or (x, y) becomes immediately dominated upon its appearance. To achieve this, we advance the appearance of all necessary edges for establishing this dominance to $f(u)$ or $f(x, y)$, since node filtration values do not affect 1-dimensional barcodes. For graph strong collapse, the set of the required edges is $\{(u, w)|w \in N(u)\} \cup \{(v, w)|w \in N(u)\}$. Among these, the edges in $\{(u, w)|w \in N(u)\}$ are reduced in $\mathcal{G}^c$ along with u in a graph strong collaspe. Therefore, only the edges in $\{(v, w)|w \in N(u)\}$ need to be reassigned. This procedure is implemented in *DebiasReducedNode* in Algorithm 4. In the case of neighborhood coning, the scenario remains the same after new edges are inserted. Hence, after reducing a node, we also reassign filtration values to these edges using *DebiasReducedNode*. For graph edge collapse, the set of necessary edges to ensure that edge (x, y) is dominated by node v at $f(x, y)$ is $\{(v, w)|w \in N[x, y]\}$. We assign these edges the filtration value $f(x, y)$, as shown in *DebiasEdgeCollapse* in Algorithm 4.

Lemma 5. *(Debiasing of graph strong collapse) If a node u is dominated by v and $f(u) < f(v)$, the persistent homology is preserved after the reduction of u by ensuring that u is dominated by v at $f(u)$.*

Proof. We prove this lemma by leveraging the conclusion from the proof of Lemma 2, which establishes that it suffices to show the diagram $\mathcal{D}$ commutes in order to guarantee the preservation of persistent homology (PH). Consider the diagram $\mathcal{D}$ defined in Lemma 2, and let $\mathcal{G}_i$ denote the i-th snapshot in the filtration of $\mathcal{G}$. Suppose a node u is dominated by node v in $\mathcal{G}_i$, which implies that $f(u) = f(v)$ and $N[u] \subseteq N[v]$. When a new adjacent node w of u with filtration value $f(w) > f(u)$ appears in $\mathcal{G}_{i+1}$, it must also be adjacent to v, since $f(w) > f(v)$. Consequently, the neighborhood inclusion $N[u] \subseteq N[v]$ continues to hold in $\mathcal{G}_{i+1}$. By induction, u remains dominated in all subsequent graphs $\mathcal{G}_{i+1}, \mathcal{G}_{i+2}, \ldots, \mathcal{G}_p$. In other words, it is not possible to find any $\mathcal{G}_j$ with $j > i$ where u is dominated in $\mathcal{G}_j$ but not in $\mathcal{G}_{j-1}$. Therefore, the inclusion relationships $\psi^c_1, \psi^c_2, \ldots, \psi^c_{p-1}$ are preserved, and the diagram $\mathcal{D}$ commutes.

□

Lemma 6. *(Debiasing of graph edge collapse) If an edge (x, y) is dominated by v and $f(x, y) < f(v)$, the persistent homology is preserved by ensuring that (x, y) is dominated by v at $f(x, y)$.*

Since the debiasing procedure is applied immediately after each node or edge reduction, we modify GStrongCollapse, GEdgeCollapse, and NeighborhoodConing by integrating debiasing steps directly following each reduction. The revised procedures, named *PHGStrongCollapse*, *PHGEdgeCollapse*, and *PHNeighborhoodConing*, are depicted in Algorithm 4. To maximize reduction capability, we first iteratively perform *PHGStrongCollapse* and *PHGEdgeCollapse* for δ_1 iterations (lines 1-3). This loop can either continue until no further nodes or edges can be collapsed or terminate after the predefined number of iterations δ_1. Following this, we apply PHNeighborhoodConing to enable further coarsening. According to Lemma 7, no dominated nodes remain after neighborhood coning, thereby completing the coarsening process. It is evident that Algorithm 4 preserves PH since each node or edge reduction individually maintains PH. The resulting coarsened graph and its

associated filtration function can then be input to any PH algorithm to compute the same barcode as the original graph.

Lemma 7. *Reducing a dominated node by neighborhood coning does not create new dominated nodes.*

Proof. First, after inserting an edge (v, w), if (v, w) still exists, it indicates that it was only dominated by u during the insertion. Otherwise, if it is dominated by another node, it would remain a dominated edge after deletion and thus be removed along with u.

If a new dominated node is created, it must be either v or w , as only $N[v]$ and $N[w]$ have changed. However, if w dominates v (or v dominates w), $N(w, v)$ must be a non-empty set. This is because f w must be connected to all neighbors of v. Specifically we denote one of the neighbor is $a \in N[v, w]$. If a exists, then (v, w) cannot be only dominated by u, leading to a contradiction.

□

Complexity Analysis. The time complexity of *DebiasReducedNode* in Algorithm 4 for a reduced node u is $O(deg(u))$. Therefore, the total worst-case time complexity of applying *DebiasReducedNode* on the graph is $O(\sum_{u\in\mathcal{V}} deg(u)) = O(m)$. For *DebiasGEdgeCollapse*, the worst-case time complexity is $\sum_{(x,y)\in\mathcal{E}} O(|N[x, y]|) \leq O(\sum_{(x,y)\in\mathcal{E}}(min(deg(x), deg(y))) \leq O(md_{\max})$, where $d_{\max}$ is the maximum degree of nodes. Although this upper bound appears to be costly, the number of reduced edges is typically much smaller than m, and only a small subset of edges require reassignment in practice. As a result, the actual overhead remains negligible. Assuming the loop (lines 1-3) executes δ_1 times, the worst-case time complexity of Algorithm 4 is $O\left((\delta_1 m + n\theta_2^2)d_{\max}^2\right)$, and can be amortized $O\left((\delta_1 m + n\theta_2^2)\bar{d}^2\right)$ on real-world graphs, where $\bar{d}$ is the average node degree on $\mathcal{G}$.

Discussions. Although the loop (lines 1-3 in Algorithm 4) can be executed multiple times, we empirically observe that the majority of nodes and edges are reduced in the first iteration. While running the loop until no further nodes or edges are removed yields the maximum reduction, a single iteration strikes a better balance between runtime and reduction ability, often leading to the minimal total time (i.e., graph coarsening time plus PH computation time).

4.2 Accelerating GNN Training

In this application, our goal is to generate a coarsened graph that preserves the topological features of the original graph, which serves as the input for GNN training. To balance the preservation of topological features with algorithmic efficiency, we adopt a two-step strategy. Specifically, we first perform a procedure called *ExactCoarsening*, where topological features are strictly preserved. If the desired coarsening ratio is not achieved in *ExactCoarsening*, we then apply another procedure named *ApproximateCoarsening*, relaxing the collapse conditions to further reduce the graph. The reason for applying *ApproximateCoarsening* is that we cannot guarantee arbitrary coarsening ratios on any real-world graph with *ExactCoarsening*. This limitation arises from two main reasons: (1) each graph has a minimal scale below which its topological features cannot be preserved (assuming the graph contains topological features), and (2) achieving this minimal graph requires determining an optimal collapse order of nodes and edges, which is an NP-hard problem [27]. In practice, we typically adopt a fixed collapse order for efficiency. In *ExactCoarsening*, we organize these three algorithms into the same approach in STPGCForPH, as depicted in lines 5–10 of Algorithm 5.

Algorithm 5: STPGCForGNN

```
   Input: G = (V, E).
   Output: The coarsened graph G^c = {V^c, E^c}.
1  G^c ← G, r ← 0;
2  G^c ← ExactCoarsening(G^c);
3  G^c ← ApproximateCoarsening(G^c);
4  return G^c;
5  Procedure ExactCoarsening(G^c)
6      while the number of iterations < δ_2 do
7          G^c ← GStrongCollapse(G^c);
8          G^c ← GEdgeCollapse(G^c);
9      G^c ← NeighborhoodConing(G^c);
10     return G^c;
11 Procedure ApproximateCoarsening(G^c)
12     while the coarsening ratio is not reached do
13         G^c ← RelaxedStrongCollapse(G^c);
14         G^c ← GEdgeCollapse(G^c);
15     return G^c;
16 Procedure RelaxedStrongCollapse(G^c)
17     Apply modified graph strong collapse by relaxing the dominance condition as u is
         r-relaxed dominated by v;
18     if number of reduced nodes < θ_3 then
19         r ← r + 1;
20     return G^c
```

Relaxed Strong Collapse. In *ApproximateCoarsening*, we introduce an r-relaxed strong collapse, which allows up to r nodes in $N[u]$ to be absent from $N[v]$, as formalized in Definition 4. It's obvious that 0-relaxed collapse is essentially the graph strong collapse. Therefore, we can adapt Algorithm 1 by modifying the dominance condition in line 10 from $N[u] \subseteq N[v]$ to "u is r-relaxed dominated by v," as implemented in the *RelaxedStrongCollapse* procedure shown in Algorithm 5.

Definition 4. *(r-relaxed strong collapse) Given two adjacent nodes u, v, if $|N[v]| \geq |N[u]| > r$, and there exists a set S_u with $|S_u| \leq r$ and $u \notin S_u$ such that $N[u] \setminus S_u \subseteq N[v]$, then u is said to be r-relaxed dominated by v. Removing a r-relaxed dominated node is called an r-relaxed strong collapse.*

When *ExactCoarsening* can no longer reduce the graph, we extend the process by replacing the graph strong collapse with the r-relaxed strong collapse. Specifically, we iteratively apply relaxed strong collapse and edge collapse in the same manner as *ExactCoarsening*, repeating this loop until the desired coarsening ratio is reached (lines 11-15). We do not further reapply neighborhood coning, as we most ring-like structures have already been reduced, leaving little room for further reduction. As the collapse proceeds, the number of dominated nodes gradually decreases. To maintain progress, we increase r by 1 whenever the number of reduced nodes in the current iteration falls below a predefined threshold θ_3.

Efficient Implementation. In *ExactCoarsening*, it is unnecessary to check all nodes and edges for dominance beyond the first iteration, because only nodes or edges whose neighborhoods have changed can potentially change to dominated. Specifically, after the initial iteration, GStrongCollapse only examines nodes whose connected edge are removed in the previous edge collapse. Similarly, GEdgeCollapse only checks edges adjacent to nodes whose neighbors were removed during the previous GStrongCollapse.

Complexity Analysis. The worst-case time complexity of *RelaxedStrongCollapse* in Algorithm 5 is $O(m\theta_1)$, the same as Algorithm 1. We assume that *ExactCoarsening* and *ApproximateCoarsening* performs δ_2 and δ_3 iterations, therefore the worst-case time complexity of Algorithm 5 is $O\left((\delta_2 + \delta_3)m\theta_1^2 + n\theta_2^2 d_{\max}^2\right)$, and can be amortized to $O\left((\delta_2 + \delta_3)m\theta_1^2 + n\theta_2^2\bar{d}^2\right)$ on most real-world

Table 1: Datasets for computing persistent homology.

Dataset	# Avg Nodes	# Avg Edges	# Graphs	Domain
REDDIT-5K	2,473	3,094	76	Social Networks
PDB-100	10,030	11,407	100	Proteins
Oregon-1	10,901	22,570	9	Systems
Email-enron	36,692	183,831	1	Communication
p2p-Gnutella31	62,586	147,892	1	Transaction

sparse graph. The values of δ_2 and δ_3 are determined by the coarsening ratio and the property of the graph. In practice, δ_2 and δ_3 are small constants, e.g., $\delta_2 < 5$ and $\delta_3 < 10$ on most datasets.

Coarsening on Attributed Graphs. To incorporate node labels and features for GNN training, we define each supernode's feature vector as the average of the features of all nodes mapped to it, and assign its label as the most frequent label among those nodes following [56]. In the coarsening process described above, node pairs are merged into supernodes based solely on graph topology, without considering the label distribution. However, in labeled graphs, nodes sharing the same label are intuitively more likely to be in the same supernode. To reflect this, in Algorithm 1 and Algorithm 3, we prioritize checking neighbors in $N(u)$ that share the same label as u when checking for node dominance and whether they can be neighborhood coned.

Pruning Heterophilic Edges. Most existing GNNs adhere to the homophily principle, assuming that adjacent nodes often share the same labels [88]. However, during the coarsening process, similar nodes are merged into supernodes, causing the removal of homophilic edges (edges between nodes with the same label). This leads to a gradual decline to the homophily ratio (the proportion of homophilic edges) of a graph, introducing a discrepancy between the coarsened graph and the original graph. To mitigate this issue, we randomly prune q_{del} heterophilic edges (edges between nodes with different labels) at the end of the coarsening to increase the homophily ratio, where q_{del} is set as a parameter.

5 EXPERIMENTS

5.1 Accelerating PH Computation

Settings. We evaluate STPGC by comparing the time and memory consumption required to compute PH on both the coarsened and original graphs. For this purpose, we adopt Ripser [5], the SOTA algorithm for PH computation. In the experiments, Ripser is configured to compute only 1-dimensional barcodes by setting the maximum dimension to 1. While any filtration function can be applied within our framework, we use node IDs as filtration values. The rationale for using node IDs is that each node has a distinct filtration value, which generalize across various filtration functions. All experiments in Section 5 are conducted on a Linux server equipped with an Intel Xeon 5218R CPU, 256 GB memory, and an NVIDIA RTX 3090 GPU.

Datasets. We select 5 datasets across different scales from the domains where PH is applied, as summarized in Table 1. Among these domains, protein analysis is a key application of PH, but existing protein graph datasets are relatively small, averaging fewer than 100 nodes [37]. To address this limitation, we adopt a new protein structure graph dataset (PDB-100) collected from Protein Data Bank [7] consisting of 100 large proteins. In addition, we include REDDIT5K [78], Oregon-1 [46], Email-enron [46] (abbreviated as Enron) and p2p-Gnutella31 (p2p) [46]. For REDDIT5K, we filtered the graphs whose number of nodes is less than 2000 to focus exclusively on large graphs. More details on the datasets are provided in the full version [72].

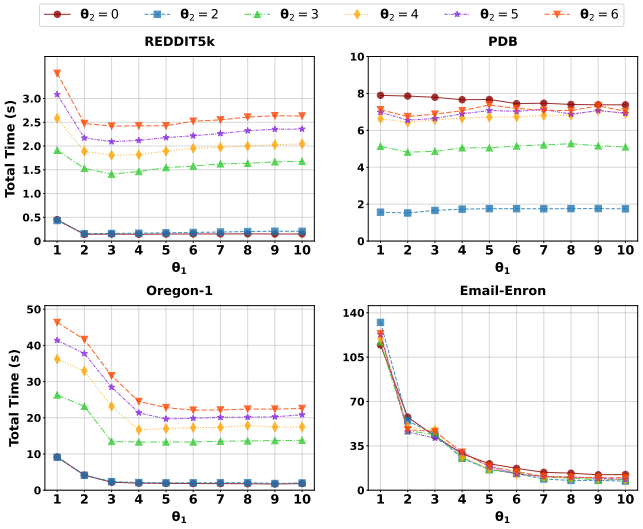

Figure 5: Impact of θ_1 and θ_2 on total runtime in computing PH. $\theta_2 = 0$ indicates neighborhood coning is not applied.

Efficiency Improvement. The results are summarized in Table 2. In this table, PH T, Coarsening T, and Total T represent the time required for persistent homology (PH) computation, graph coarsening (including filtration value reassignment), and their combined total, respectively. Mem and Coarsening Mem denote the peak memory usage during PH computation and graph coarsening, respectively. As illustrated, STPGC achieves consistent improvements in both runtime and memory usage across all datasets. In addition, the growth of Coarsening T with respect to graph size is significantly slower than that of PH T, highlighting the advantage of accelerating PH computation with STPGC. These results demonstrate STPGC effectively accelerates PH computation by reducing the graph size.

Parameters Experiment. The parameters in STPGCForPH include the degree thresholds θ_1 for *PHGStrongCollapse* and θ_2 for *PHNeighborhoodConing*. Larger values of θ_1 and θ_2 lead to smaller coarsened graphs, which reduces PH computation time but increases the runtime of graph coarsening. Therefore, it is crucial to choose appropriate parameters to achieve the minimal total runtime. Figure 5 illustrates how varying these parameters impacts the total runtime. Note that there is no $\theta_2 = 1$, since all 1-degree nodes are already reduced by graph strong collapse. On sparser datasets such as REDDIT5k and Oregon, the minimal runtime is achieved with relatively small θ_1 values (e.g., 2 or 3). In contrast, the denser and larger dataset (Enron) favors higher values (e.g., $\theta_1 = 10$). For the PDB dataset, the runtime is less sensitive to θ_1 due to the prevalence of rings in protein structures, which cannot be effectively reduced via graph strong and edge collapses alone, underscoring the importance of neighborhood coning. Regarding θ_2, on REDDIT5k, the minimal runtime is achieved when $\theta_2 = 0$, suggesting that graph strong and edge collapse alone are sufficient on small graphs. For others, $\theta_2 = 2$ yields the least total runtime. These results suggest that coarsening time dominates total runtime on smaller datasets, favoring smaller θ_1 and θ_2. For larger graphs, larger θ_1 and θ_2 lead to lower total runtime.

Effectiveness Analysis. To evaluate the effectiveness of STPGC, which combines *PHGStrongCollapse*, *PHGEdgeCollapse*, and *PHNeighborhoodConing*, we apply each procedure individually as well as in combination, and report the number of reduced simplices

Table 2: Efficiency improvement on computing persistence homology. Time is measured in second (s), and memory is measured in MB. The improvement over Ripser on the original graph is underlined.

Dataset	Original			Coarsened					
	Nodes / Edges / Simplices	PH T	Mem	Nodes / Edges / Simplices	Coarsening T	PH T	Total T	Coarsening Mem	Mem
REDDIT5K	2,473 / 3,094 / 5,709	0.371	1066	696 / 1,318 / 2,156	0.117	0.024	0.141 2.6x	732	735 1.5x
PDB-100	10,030 / 11,407 / 21,583	9.675	9,863	3,249/4,626 / 8,020	0.630	0.890	1.520 6.4x	743	1,305 7.6x
Oregon-1	10,901 / 22,570 / 51,330	9.235	7,410	3,974 / 10,368 / 27,521	0.763	0.974	1.737 5.3x	740	1,651 4.5x
Email-enron	36,692 / 183,831 / 947,567	121.250	78,129	8,884 / 113,304 / 757,566	1.941	5.596	7.509 16.2x	1,013	5,571 14.0x
p2p-Gnutella31	62,586 / 147,892 / 212,502	OOM	OOM	24,838 / 109,942.0 / 136,590	1.451	48.191	49.642 -	966	36,430 >7.2x

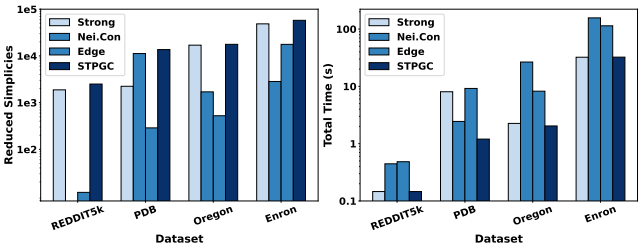

Figure 6: Number of reduced simplices and total runtime of *PHGStrongCollapse*(Strong), *PHGEdgeCollapse*(Edge), *PHNeighborhoodConing*(Nei.Con), and STPGC.

Table 3: Datasets for node classification and scalability analysis.

Dataset	# Nodes	# Edges	# Features	# Classes	Ave. Deg
Cora	2,703	5,429	1,433	7	3.88
Citeseer	3,312	4,732	3,703	6	2.84
DBLP	3,877	16,624	1,700	4	5.97
Ogbn-ArXiv	169,343	1,166,243	128	40	13.77
Ogbn-Products	2,449,029	61,859,140	100	47	50.52
LiveJournal	3,997,962	34,681,189	-	-	17.35
Youtube	1,134,890	2,987,624	-	-	5.27
cit-Patent	3,774,768	16,518,948	-	-	8.76
Flixter	2,523,386	9,197,338	-	-	7.30

and total computation time (Total T) in Figure 6. Note that since the optimal value of θ_2 on REDDIT5k is set to 0, no simplices are reduced by neighborhood coning. As illustrated, STPGC achieves the largest reduction in simplices and the shortest overall runtime, highlighting the benefit of integrating the three procedures. Specifically, graph strong collapse and graph edge collapse are more effective in eliminating tree-like and triangular substructures. In contrast, neighborhood coning excels at compressing rings and chain-like structures. As such, *PHNeighborhoodConing* serves as a complementary coarsening step after *PHGStrongCollapse* and *PHGEdgeCollapse*. By properly integrating them, STPGC demonstrates best acceleration performance across diverse graphs.

Scalability Analysis. We further evaluate the scalability of STPGCForPH (Algorithm 4) under varying graph sizes and coarsening ratios using four large-scale graph datasets: YouTube [79], LiveJournal [3], cit-Patent [47], and Flixster [82], as shown in Table 3. For each dataset, we construct subgraphs by randomly sampling nodes and edges under different ratios (Figure 7). We set $\theta_1 = \theta_2 = 50$. Across all datasets and configurations, STPGCForPH exhibits nearly linear growth in both runtime and memory usage, showing its strong scalability on large-scale graphs.

5.2 Accelerating GNN Training

Settings. We select 5 labeled benchmarks (Cora, Citeseer, Ogbn-ArXiv, Ogbn-Products) for the node classification task, and four additional unlabeled large datasets (Youtube, LiveJournal, cit-Patent, Flixter) for further scalability evaluation, as summarized in Table 3. Following [38, 56], we use GCN [43] and APPNP [33] for the GNN model. We select 7 graph coarsening baselines for a comprehensive comparison, including Variation Neighborhoods [38, 52], Variation Edges [38, 52], Algebraic JC [38, 52], Affinity GS [38, 52], Kron [38, 52], FGC [44], and the SOTA method GEC [56]. For parameters, we set θ_3 to 1% of the total number of nodes. More details on the datasets and baselines are provided in the full version [72].

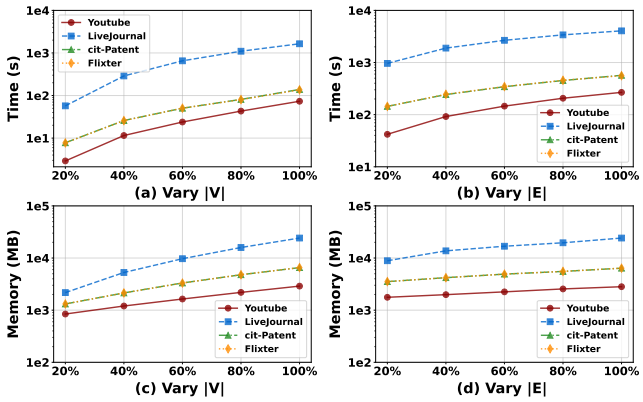

Figure 7: Scalability analysis of STPGCForPH.

Node Classification. The results are shown in Table 5, where STPGC consistently outperforms baselines across most datasets and coarsening ratios. In particular, topology-preserving methods (STPGC and GEC) outperform all other baselines, highlighting the importance of preserving topological features. Moreover, STPGC outperforms GEC by an average of 1.73% on five datasets, suggesting that STPGC more effectively preserve topological features. Although GEC is also designed to this object, its initial graph partition process inevitably disrupts the topological features (Figure 9). This may explain its inferior performance compared to STPGC. An exception is observed on the Ogbn-ArXiv dataset at $c = 0.1$ and $c = 0.2$. This may be attributed to the large number of reduced nodes, leading to loss of topological features in the coarsened graph.

Efficiency of Different Methods. Table 4 and Figure 8 present the runtime and memory consumption of different methods. As illustrated, STPGC consistently outperforms all baselines in both runtime and memory efficiency. In particular, compared to GEC, STPGC exhibits a much slower increase in runtime as the coarsening ratio c decreases, making it more efficient for lower coarsening ratios. For instance, when $c = 0.1$, STPGC is over 15x faster than GEC on the DBLP dataset and 8.7x faster on Ogbn-ArXiv.

Betti Number. Betti numbers quantify the topological features of a graph. To evaluate how effectively different methods preserve topology, we compute the 1-Betti numbers of the coarsened graphs under varying coarsening ratios. As shown in Figure 9, STPGC strictly

Table 4: Running time (s) of different coarsening methods.

Dataset	Coarsening Method	c=0.5	c=0.3	c=0.2	c=0.1
DBLP	Variation Neighborhoods	23.329	33.918	30.863	34.726
	Variation Edges	14.626	15.945	19.072	29.070
	Algebraic JC	22.048	34.683	35.978	44.321
	Affinity GS	23.627	18.394	19.846	18.638
	FGC	956.227	473.301	202.215	95.336
	GEC	8.261	18.820	36.026	100.417
	STPGC	3.649	4.821	5.519	6.371
Ogbn-ArXiv	Variation Neighborhoods	393.1	475.4	496.1	535.6
	Variation Edges	546.8	692.1	767.8	861.2
	Algebraic JC	557.9	710.7	815.1	995.2
	Affinity GS / FGC	Out of Memory (Over 256GB)			
	GEC	129.6	323.4	883.7	2474.5
	STPGC	75.4	178.6	207.8	286.3

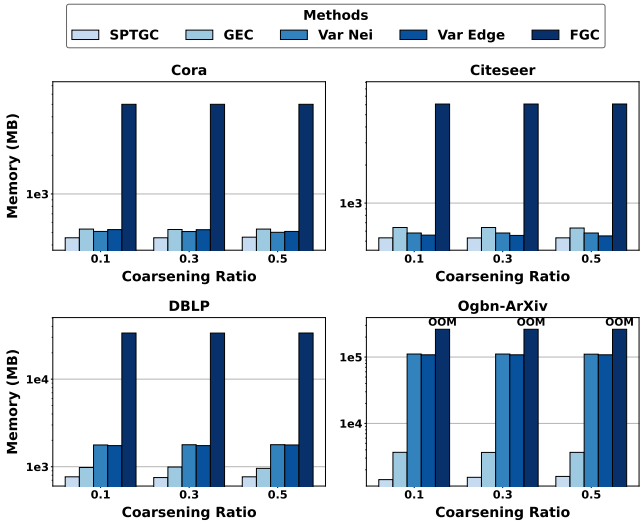

Figure 8: Memory overhead of STPGC and baseline methods.

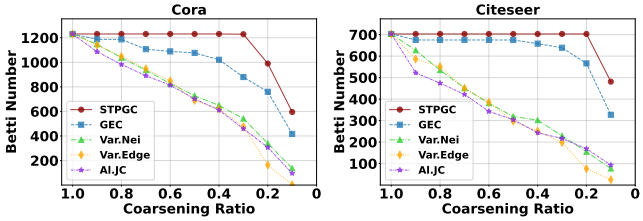

Figure 9: Betti number preserved by different methods.

maintains the number of topological features as the coarsening ratio decreases before *ExactCoarsening* exits. In contrast, other methods lose topological features at the beginning of coarsening. GEC shows a slower decline compared to non-topology-preserving methods, as it employs elementary collapse within partitioned subgraphs, which retains some local topology. However, its partitioning process inherently disrupts global topology, resulting in inferior preservation. These results highlight the superior ability of STPGC to maintain topological features.

Parameters Experiment. We investigate the impact of two key parameters—degree threshold (θ_1, θ_2) and the number of deleted edges q_{del}—on node classification accuracy and runtime. For simplicity, we set θ_1=θ_2. A larger value of θ_1 and θ_2 allows *ExactCoarsening* to reduce more nodes and edges, thereby preserving more topological features in the coarsened graph. However, this may also result in

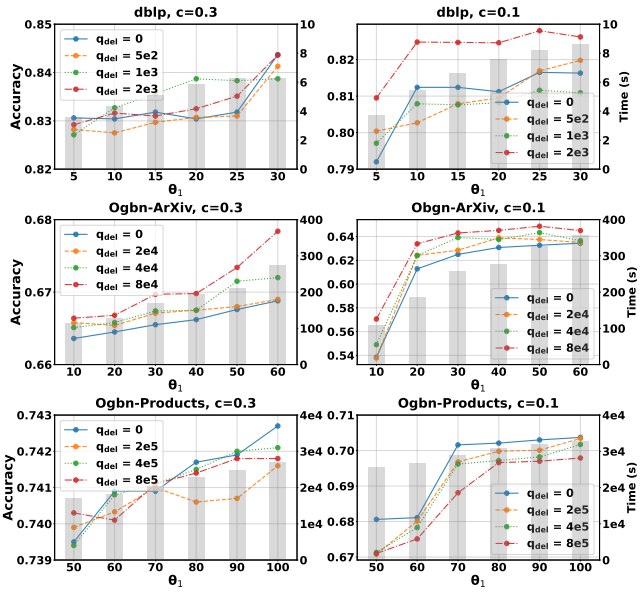

Figure 10: Impact of parameters on node classification accuracy and runtime of STPGC.

increased runtime. Additionally, deleting an appropriate number of edges can improve classification performance. Since edge deletion is highly efficient and has a negligible impact on runtime, we treat the runtime across different values of q_{del} as equivalent. The results on GCN are reported in Figure 10. Across all three datasets, we observe that classification accuracy generally increases with larger θ_1, suggesting that preserving more topological features enhances GNN performance. However, The accuracy ceases to improve after θ_1 exceeds the degree of most nodes. Overall, optimal results can be achieved with θ_1 set to approximately 2-6 times the average degree.

Scalability Analysis. We further evaluate the scalability of STPGC-ForGNN (Algorithm 5) on four large datasets (LiveJournal, Youtube, cit-Patent and Flixter). In the experiments varying the number of nodes and edges, we generate the same graphs as in the scalability analysis of STPGCForPH (Figure 11). The coarsening ratio is set to 0.3, and θ_1 is fixed at 50. We then compare STPGC (with varying degree thresholds θ_1) with the SOTA graph coarsening method GEC (with original parameters) on the cit-Patent dataset (Figure 12). Finally, we compare STPGC with GEC on four datasets with different coarsening ratios (Figure 13). Across different datasets, the runtime and memory usage of STPGC increase smoothly. An exception occurs when |V| increases from 80% to 100%, where the runtime growth accelerates. This is because STPGC's runtime largely depends on the number of edges, which grows more rapidly in this range than in previous intervals. Compared to GEC, STPGC achieves an order-of-magnitude speedup in most configurations, and delivers up to a 37x acceleration on the cit-Patent dataset. These results validate superior scalability of STPGC on large-scale graphs.

6 RELATED WORK

Graph Coarsening. Existing graph coarsening literature mainly focuses on preserving the spectral similarity of graphs [12, 52, 53]. Loukas *et al.* [53] introduced the restricted spectral similarity to show that graph coarsening preserves eigenvalue approximations. More recent researches resort to machine learning methods for data-specified graph coarsening. For example, Cai *et al.* [13] leveraged

Table 5: Accuracy on node classification. c denotes the coarsening ratio (number of nodes in the coarsened graph/ number of nodes in the original graph). The best results are bold, the second best results are underlined.

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	80.1±0.3	81.7±0.3	81.7±0.3	81.4±0.5	80.7±0.7	81.5±0.6	78.9±0.6	80.6±0.8	73.3±1.0	66.6±0.8
	Variation Edges			81.0±0.4	83.1±0.6	79.0±0.9	81.0±1.0	73.2±1.2	73.0±1.5	-	-
	Algebraic JC			81.3±0.3	82.5±0.5	79.5±0.6	80.1± 0.6	79.4 ±0.7	81.7±0.5	66.3±1.7	69.1±1.6
	Affinity GS			81.3±0.5	82.4±0.6	79.5±0.5	79.5±0.7	80.3 ±0.6	80.5±0.7	75.2±0.6	73.5±1.3
	Kron			81.4±0.5	82.6±0.8	79.8±0.7	80.1± 0.7	80.2±0.83	77.1 ± 0.7	65.0± 1.2	66.7±0.9
	FGC			79.9±1.9	78.7±1.3	77.6± 2.5	77.9±1.3	77.5±1.9	76.9±2.3	71.1 ±2.6	68.5 ±0.9
	GEC			80.7±0.4	81.3±0.7	80.9±0.4	82.0±0.7	81.0±0.3	81.7±0.9	80.8±0.7	81.7±0.5
	STPGC			**82.8**±0.4	**82.8**±0.3	**82.3**±0.5	**82.9**±0.29	**82.2**±0.4	**83.9**±0.3	**82.5**±0.3	**84.5**±0.3
Citeseer	Variation Neighborhoods	71.6±0.8	71.1±0.4	71.5±0.7	71.7±0.5	70.4±0.4	71.1±0.4	70.0±0.7	70.9±0.8	56.6±0.6	58.2±0.6
	Variation Edges			72.2±0.5	71.6±0.6	70.1±0.6	71.6±0.5	54.9±0.7	60.6±0.7	50.6 ±9.7	50.8±9.8
	Algebraic JC			71.2±0.5	71.4±0.8	70.2±0.4	**72.4**±0.5	57.1±1.6	67.2±1.2	60.7±5.9	60.9±6.6
	Affinity GS			70.4±0.7	71.3±0.4	70.3±0.5	71.2±0.5	69.6±0.7	70.7±0.7	63.7±5.6	62.8±6.1
	Kron			72.2±0.5	72.1±0.3	70.3±0.6	71.7±0.5	70.1±1.31	70.9±0.21	65.8 ±1.6	66.4±0.5
	FGC			70.1±1.4	71.4±1.9	69.1±1.7	70.5±1.3	68.1 ±2.3	69.9±2.1	67.5±1.6	68.1±2.3
	GEC			70.1±0.4	71.1±0.3	71.1±0.5	71.0±0.2	71.3±0.5	71.6±0.5	71.6±0.2	72.1±0.5
	STPGC			**72.3**±0.5	**72.3**±0.3	**72.4**±0.6	71.2±0.2	**71.4**±0.6	**72.2**±0.2	**73.2**±0.4	**73.5**±0.2
DBLP	Variation Neighborhoods	79.9±0.2	81.8±0.1	82.5±0.6	83.4±0.5	81.7± 0.6	82.4±0.6	80.2±0.4	82.3±0.8	79.6±0.5	80.4±0.5
	Variation Edges			82.2±0.8	84.2±0.7	80.6±0.3	82.6±0.8	80.2±0.5	82.1±0.8	79.4±0.5	81.9±0.6
	Algebraic JC			80.6±0.7	83.7±0.5	80.2±0.7	82.0±0.6	80.0±0.4	80.0±0.5	78.1±0.8	79.5±0.5
	Affinity GS			82.1±0.5	84.6±0.5	81.1±0.5	83.7±0.3	80.0±0.5	82.1±0.7	79.2±0.6	80.2±0.4
	Kron			80.6±0.6	84.0±0.4	80.6±0.7	81.9±0.8	79.6±0.4	80.6±0.6	77.8±0.4	78.8±0.8
	FGC			82.0±0.3	83.9±0.6	81.8±0.7	83.2±0.6	82.6±0.4	83.5±0.1	81.8±0.9	82.1±0.6
	GEC			83.0±0.7	84.3±0.3	82.4±0.6	83.4±0.2	83.5±1.0	83.8±0.2	80.5±2.9	78.9±0.8
	STPGC			**83.9**±0.2	**85.0**±0.1	**84.5**±0.1	**85.2**±0.1	**84.9**±0.1	**85.3**±0.1	**83.1**±0.1	**83.6**±0.2
Ogbn-ArXiv	Variation Neighborhoods	70.9±0.6	62.2±0.2	64.5±1.8	54.6±1.5	64.8±2.5	59.6±0.6	53.5±1.4	55.3±1.0	45.4±3.4	51.6±2.4
	Variation Edges			65.4±0.9	59.7±1.0	61.9±0.6	55.3±0.8	54.4±0.8	55.9±0.6	50.3±0.7	54.9±0.9
	Algebraic JC			65.3±0.7	60.9±0.6	61.1±0.6	56.4±0.6	54.0±1.5	51.8±0.8	48.5±1.8	57.3±0.5
	Affinity GS/FGC			Out of memory (Over 256GB)							
	Kron			64.4±0.6	61.1±0.7	56.1±0.6	58.5±0.7	56.4±2.2	55.1±0.9	57.7±0.5	58.6±0.9
	GEC			70.1±0.5	61.9±0.5	68.3±0.5	60.1±0.1	**67.6**±0.5	60.1±0.7	**65.1**±0.3	**59.4**±0.6
	STPGC			**70.4**±0.3	**62.4**±0.5	**68.5**±0.5	**60.9**±0.6	66.1±0.6	**60.8**±1.0	64.5±0.8	58.8±1.3
Ogbn-Products	Other methods	74.5±0.4	65.4±0.5	Out of memory (Over 256GB)							
	GEC			72.1±0.5	64.3±0.3	71.4±0.6	62.2±0.3	70.7±0.4	61.1±0.4	70.1±0.7	60.1±0.4
	STPGC			**75.2**±0.2	**65.6**±0.4	**74.3**±0.2	**64.7**±0.4	**73.2**±0.1	**62.4**±0.4	**70.4**±0.1	**60.4**±0.4

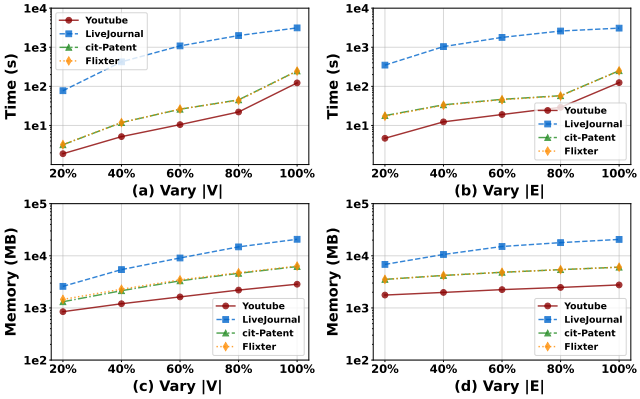

Figure 11: Scalability analysis of STPGC.

deep learning techniques to optimize edge weight assignment using graph neural networks. Given the challenges of training GNNs on large-scale graphs, some methods focus on preserving GNN performance as the goal of graph coarsening. Huang *et al.* [38] showed that graph coarsening not only enhances the scalability of GNNs but also serves as a regularization to improve training. ConvMatch [24] minimizes the change of graph convolution between the coarsened graph and the original graph. Recognizing that many real-world graphs contain node features, FGC [44] jointly optimizes both the adjacency matrix and feature matrix to preserve spectral and feature similarities. GEC [56] is the only existing method designed to preserve topological features. It employs the elementary collapse to remove structures called free faces [71].

Computing Persistent Homology. The computation of persistent homology (PH) is a fundamental problem in topological data analysis. The first algorithm to address this problem, an edge-pairing approach, was introduced in the seminal work by Edelsbrunner *et al.* [25]. To accelerate the computation, a matrix reduction-based variant was subsequently proposed [26]. Both of them have a time complexity of $O(\tilde{n}^3)$, where $\tilde{n}$ denotes the number of simplices. Although the theoretically fastest known algorithm achieves a time complexity of $O(\tilde{n}^\omega)$ with $\omega \approx 2.37$ [2, 58], it has yet to be implemented in practice. Beyond that, some methods leverage topology theory or parallel computing to optimize the computation of PH [5, 6]. Due to the high worst-case complexity of exact PH algorithms, recent studies have resorted to approximate methods based on neural networks. For example, some approaches reformulate the computation as an edge-pairing prediction task and approximate it using GNNs [48, 77]. Although these methods improve efficiency, they cannot guarantee the accuracy of results.

Neighborhood Inclusion. The neighborhood inclusion of nodes has important applications in graph analysis. Brandes *et al.* [11] studied the neighborhood inclusion to compute the preorders of nodes, resulting in a new threshold graph [54], which can be applied

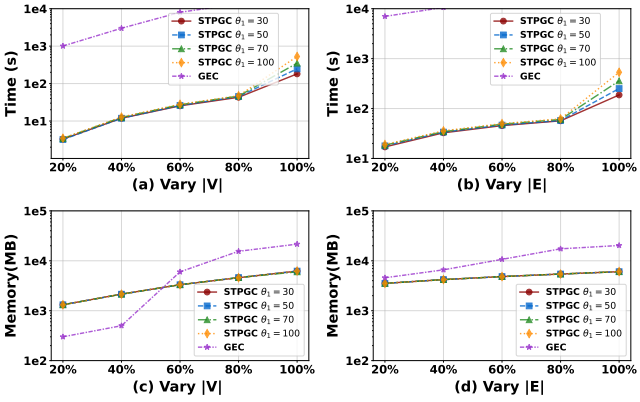

Figure 12: Scalability analysis of STPGC and GEC.

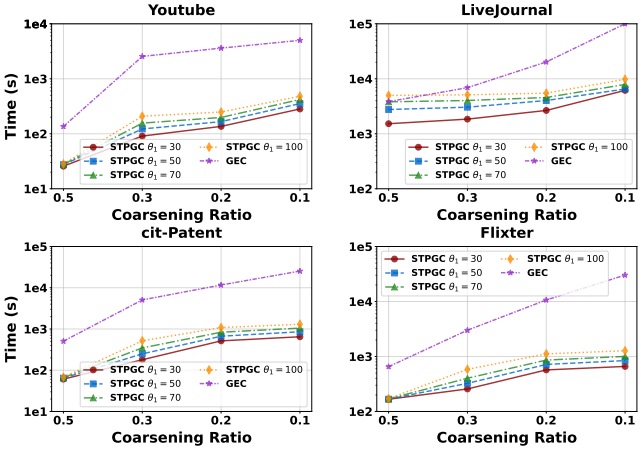

Figure 13: Results with varying coarsening ratios.

to problems such as minimum stretch trees [59] and characteristic polynomial computation [30]. Zhang *et al.* [83] introduced the concept of the Neighborhood Skyline, consisting of nodes whose open neighborhoods are not included by the open neighborhood of any other nodes. In maximum independent set computation [16, 60], a node can be safely removed from the graph if its closed neighborhood fully contains that of another node. In shortest distance queries, neighborhood inclusion defines an equivalence relation to compress the graph and reduce the index size [49].

7 CONCLUSION

In this paper, we propose scalable and effective graph coarsening algorithms (STPGC) to preserve graph topological features. STPGC is based on graph strong collapse and graph edge collapse, ensuring a theoretical guarantee of preserved homotopy equivalence and persistent homology on the coarsened graph. We study two applications of the proposed algorithms, i.e., accelerating persistent homology computation and accelerating graph neural network training. Extensive experiments on real-world datasets demonstrate the efficiency and effectiveness of our approach. As a future direction, we plan to apply STPGC to broader applications in graph data management, such as data pruning.

REFERENCES

[1] Manu Aggarwal and Vipul Periwal. 2023. Tight basis cycle representatives for persistent homology of large biological data sets. *PLOS Computational Biology* 19, 5 (2023), e1010341.

[2] Josh Alman and Virginia Vassilevska Williams. 2024. A refined laser method and faster matrix multiplication. *TheoretiCS* 3 (2024).

[3] Lars Backstrom, Dan Huttenlocher, Jon Kleinberg, and Xiangyang Lan. 2006. Group formation in large social networks: membership, growth, and evolution. In *KDD*. 44–54.

[4] Jonathan Ariel Barmak and Elias Gabriel Minian. 2012. Strong homotopy types, nerves and collapses. *Discrete & Computational Geometry* 47 (2012), 301–328.

[5] Ulrich Bauer. 2021. Ripser: efficient computation of Vietoris–Rips persistence barcodes. *Journal of Applied and Computational Topology* 5, 3 (2021), 391–423.

[6] Ulrich Bauer, Michael Kerber, and Jan Reininghaus. 2014. Distributed computation of persistent homology. In *2014 proceedings of the sixteenth workshop on algorithm engineering and experiments (ALENEX)*. SIAM, 31–38.

[7] Helen M Berman, John Westbrook, Zukang Feng, Gary Gilliland, Talapady N Bhat, Helge Weissig, Ilya N Shindyalov, and Philip E Bourne. 2000. The protein data bank. *Nucleic acids research* 28, 1 (2000), 235–242.

[8] Jean-Daniel Boissonnat and Siddharth Pritam. 2019. Computing Persistent Homology of Flag Complexes via Strong Collapses. In *SoCG 2019-International Symposium on Computational geometry*.

[9] Jean-Daniel Boissonnat and Siddharth Pritam. 2020. Edge collapse and persistence of flag complexes. In *36th International Symposium on Computational Geometry (SoCG 2020)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik.

[10] Jean-Daniel Boissonnat, Siddharth Pritam, and Divyansh Pareek. 2018. Strong Collapse for Persistence. In *ESA 2018-26th Annual European Symposium on Algorithms*. 67–1.

[11] Ulrik Brandes, Moritz Heine, Julian Müller, and Mark Ortmann. 2017. Positional dominance: Concepts and algorithms. In *Algorithms and Discrete Applied Mathematics: Third International Conference, CALDAM 2017, Sancoale, Goa, India, February 16-18, 2017, Proceedings 3*. Springer, 60–71.

[12] Gecia Bravo Hermsdorff and Lee Gunderson. 2019. A unifying framework for spectrum-preserving graph sparsification and coarsening. *NeurIPS* 32 (2019).

[13] Chen Cai, Dingkang Wang, and Yusu Wang. 2021. Graph coarsening with neural networks. *arXiv preprint arXiv:2102.01350* (2021).

[14] Gunnar Carlsson, Afra Zomorodian, Anne Collins, and Leonidas Guibas. 2004. Persistence barcodes for shapes. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*. 124–135.

[15] Mathieu Carrière, Frédéric Chazal, Yuichi Ike, Théo Lacombe, Martin Royer, and Yuhei Umeda. 2020. Perslay: A neural network layer for persistence diagrams and new graph topological signatures. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 2786–2796.

[16] Lijun Chang, Wei Li, and Wenjie Zhang. 2017. Computing a near-maximum independent set in linear time by reducing-peeling. In *Proceedings of the 2017 ACM International Conference on Management of Data*. 1181–1196.

[17] Haochen Chen, Bryan Perozzi, Yifan Hu, and Steven Skiena. 2018. Harp: Hierarchical representation learning for networks. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 32.

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

[19] Tianlong Chen, Yongduo Sui, Xuxi Chen, Aston Zhang, and Zhangyang Wang. 2021. A unified lottery ticket hypothesis for graph neural networks. In *ICML*. PMLR, 1695–1706.

[20] Yuzhou Chen, Baris Coskunuzer, and Yulia Gel. 2021. Topological relational learning on graphs. *NeurIPS* 34 (2021), 27029–27042.

[21] Zijian Chen, Rong-Hua Li, Hongchao Qin, Huanzhong Duan, Yanxiong Lu, Qiangqiang Dai, and Guoren Wang. 2022. Filtration-Enhanced Graph Transformation.. In *IJCAI*. 1987–1993.

[22] Tamal K Dey, Dayu Shi, and Yusu Wang. 2015. Comparing Graphs via Persistence Distortion. In *31st International Symposium on Computational Geometry (SoCG 2015)*. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 491–506.

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

[24] Charles Dickens, Edward Huang, Aishwarya Reganti, Jiong Zhu, Karthik Subbian, and Danai Koutra. 2024. Graph coarsening via convolution matching for scalable graph neural network training. In *Companion Proceedings of the ACM on Web Conference 2024*. 1502–1510.

[25] Edelsbrunner, Letscher, and Zomorodian. 2002. Topological persistence and simplification. *Discrete & computational geometry* 28 (2002), 511–533.

[26] Herbert Edelsbrunner and John L Harer. 2022. *Computational topology: an introduction*. American Mathematical Society.

[27] Ömer Eğecioğlu and Teofilo F Gonzalez. 1996. A computationally intractable problem on simplicial complexes. *Computational Geometry* 6, 2 (1996), 85–98.

[28] Samuel Eilenberg and Norman Steenrod. 2015. *Foundations of algebraic topology*. Vol. 2193. Princeton University Press.

[29] Junfeng Fang, Xinglin Li, Yongduo Sui, Yuan Gao, Guibin Zhang, Kun Wang, Xiang Wang, and Xiangnan He. 2024. Exgc: Bridging efficiency and explainability in graph condensation. In *Proceedings of the ACM Web Conference 2024*. 721–732.

[30] Martin Fürer. 2017. Efficient computation of the characteristic polynomial of a threshold graph. *Theoretical Computer Science* 657 (2017), 3–10.

[31] Xinyi Gao, Guanhua Ye, Tong Chen, Wentao Zhang, Junliang Yu, and Hongzhi Yin. 2025. Rethinking and accelerating graph condensation: A training-free approach with class partition. In *Proceedings of the ACM on Web Conference 2025*. 4359–4373.

[32] Xinyi Gao, Hongzhi Yin, Tong Chen, Guanhua Ye, Wentao Zhang, and Bin Cui. 2025. Robgc: Towards robust graph condensation. *IEEE Transactions on Knowledge and Data Engineering* (2025).

[33] Johannes Gasteiger, Aleksandar Bojchevski, and Stephan Günnemann. 2018. Predict then propagate: Graph neural networks meet personalized pagerank. *arXiv preprint arXiv:1810.05997* (2018).

[34] Robert Ghrist. 2008. Barcodes: the persistent topology of data. *Bull. Amer. Math. Soc.* 45, 1 (2008), 61–75.

[35] Mohammad Hashemi, Shengbo Gong, Juntong Ni, Wenqi Fan, B Aditya Prakash, and Wei Jin. 2024. A comprehensive survey on graph reduction: Sparsification, coarsening, and condensation. *arXiv preprint arXiv:2402.03358* (2024).

[36] Max Horn, Edward De Brouwer, Michael Moor, Yves Moreau, Bastian Rieck, and Karsten Borgwardt. [n.d.]. Topological Graph Neural Networks. In *ICLR*.

[37] 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. *NeurIPS* 33 (2020), 22118–22133.

[38] Zengfeng Huang, Shengzhong Zhang, Chong Xi, Tang Liu, and Min Zhou. 2021. Scaling up graph neural networks via graph coarsening. In *KDD*. 675–684.

[39] Bo Hui, Da Yan, Xiaolong Ma, and Wei-Shinn Ku. 2023. Rethinking graph lottery tickets: Graph sparsity matters. *arXiv preprint arXiv:2305.02190* (2023).

[40] Johanna Immonen, Amauri Souza, and Vikas Garg. 2023. Going beyond persistent homology using persistent homology. *NeurIPS* 36 (2023), 63150–63173.

[41] Shweta Jain and C. Seshadhri. 2020. The Power of Pivoting for Exact Clique Counting. In *Proceedings of the 13th International Conference on Web Search and Data Mining* (Houston, TX, USA) *(WSDM '20)*. Association for Computing Machinery, New York, NY, USA, 268–276. https://doi.org/10.1145/3336191.3371839

[42] Wei Jin, Lingxiao Zhao, Shichang Zhang, Yozen Liu, Jiliang Tang, and Neil Shah. 2021. Graph condensation for graph neural networks. *arXiv preprint arXiv:2110.07580* (2021).

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

[44] Manoj Kumar, Anurag Sharma, Shashwat Saxena, and Sandeep Kumar. 2023. Featured graph coarsening with similarity guarantees. In *ICML*. PMLR, 17953–17975.

[45] Kristen LeFevre and Evimaria Terzi. 2010. GraSS: Graph Structure Summarization. In *Proceedings of the SIAM International Conference on Data Mining, SDM 2010, April 29 - May 1, 2010, Columbus, Ohio, USA*. SIAM, 454–465. https://doi.org/10.1137/1.9781611972801.40

[46] Jure Leskovec and Andrej Krevl. 2014. SNAP Datasets: Stanford Large Network Dataset Collection. http://snap.stanford.edu/data.

[47] Jure Leskovec, Kevin J Lang, Anirban Dasgupta, and Michael W Mahoney. 2009. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *Internet Mathematics* 6, 1 (2009), 29–123.

[48] Hao Li, Hao Jiang, Jiajun Fan, Dongsheng Ye, and Liang Du. 2024. Dynamic Neural Dowker Network: Approximating Persistent Homology in Dynamic Directed Graphs. In *KDD*, Ricardo Baeza-Yates and Francesco Bonchi (Eds.). ACM, 1554–1564. https://doi.org/10.1145/3637528.3671980

[49] Wentao Li, Miao Qiao, Lu Qin, Ying Zhang, Lijun Chang, and Xuemin Lin. 2019. Scaling distance labeling on small-world networks. In *Proceedings of the 2019 International Conference on Management of Data*. 1060–1077.

[50] Jiongqian Liang, Saket Gurukar, and Srinivasan Parthasarathy. 2018. MILE: A Multi-Level Framework for Scalable Graph Embedding. *CoRR* abs/1802.09612 (2018). arXiv:1802.09612 http://arxiv.org/abs/1802.09612

[51] Oren E. Livne and Achi Brandt. 2012. Lean Algebraic Multigrid (LAMG): Fast Graph Laplacian Linear Solver. *SIAM J. Sci. Comput.* 34, 4 (2012). https://doi.org/10.1137/110843563

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

[53] Andreas Loukas and Pierre Vandergheynst. 2018. Spectrally approximating large graphs with smaller graphs. In *ICML*. PMLR, 3237–3246.

[54] Nadimpalli VR Mahadev and Uri N Peled. 1995. *Threshold graphs and related topics*. Elsevier.

[55] Clément Maria, Jean-Daniel Boissonnat, Marc Glisse, and Mariette Yvinec. 2014. The gudhi library: Simplicial complexes and persistent homology. In *Mathematical Software–ICMS 2014: 4th International Congress, Seoul, South Korea, August 5-9, 2014. Proceedings 4*. Springer, 167–174.

[56] Yuchen Meng, Rong-Hua Li, Longlong Lin, Xunkai Li, and Guoren Wang. 2024. Topology-Preserving Graph Coarsening: An Elementary Collapse-Based Approach. *Proceedings of the VLDB Endowment* 17, 13 (2024), 4760–4772.

[57] Yuchen Meng, Rong-Hua Li, Hongchao Qin, Xiang Wu, Huanzhong Duan, Yanxiong Lu, and Guoren Wang. 2024. Encoding Group Interests With Persistent Homology for Personalized Search. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 54, 9 (2024), 5606–5616.

[58] Nikola Milosavljević, Dmitriy Morozov, and Primoz Skraba. 2011. Zigzag persistent homology in matrix multiplication time. In *Proceedings of the twenty-seventh Annual Symposium on Computational Geometry*. 216–225.

[59] Stavros D Nikolopoulos and Charis Papadopoulos. 2004. The number of spanning trees in K n-complements of quasi-threshold graphs. *Graphs and Combinatorics* 20 (2004), 383–397.

[60] Chengzhi Piao, Weiguo Zheng, Yu Rong, and Hong Cheng. 2020. Maximizing the reduction ability for near-maximum independent set computation. *Proceedings of the VLDB Endowment* 13, 12 (2020), 2466–2478.

[61] Manish Purohit, B Aditya Prakash, Chanhyun Kang, Yao Zhang, and VS Subrahmanian. 2014. Fast influence-based coarsening for large networks. In *KDD*. 1296–1305.

[62] Noam Razin, Tom Verbin, and Nadav Cohen. 2023. On the ability of graph neural networks to model interactions between vertices. *NeurIPS* 36 (2023), 26501–26545.

[63] Bastian Rieck, Christian Bock, and Karsten Borgwardt. 2019. A persistent weisfeiler-lehman procedure for graph classification. In *ICML*. PMLR, 5448–5458.

[64] Bastian Rieck, Ulderico Fugacci, Jonas Lukasczyk, and Heike Leitte. 2017. Clique community persistence: A topological visual analysis approach for complex networks. *IEEE transactions on visualization and computer graphics* 24, 1 (2017), 822–831.

[65] Ilya Safro, Peter Sanders, and Christian Schulz. 2014. Advanced Coarsening Schemes for Graph Partitioning. *ACM J. Exp. Algorithmics* 19, 1 (2014). https://doi.org/10.1145/2670338

[66] Ann Sizemore, Chad Giusti, and Danielle S Bassett. 2017. Classification of weighted networks through mesoscale homological features. *Journal of Complex Networks* 5, 2 (2017), 245–273.

[67] Ann E Sizemore, Chad Giusti, Ari Kahn, Jean M Vettel, Richard F Betzel, and Danielle S Bassett. 2018. Cliques and cavities in the human connectome. *Journal of computational neuroscience* 44 (2018), 115–145.

[68] Qingyun Sun, Ziying Chen, Beining Yang, Cheng Ji, Xingcheng Fu, Sheng Zhou, Hao Peng, Jianxin Li, and Philip S Yu. 2024. Gc-bench: An open and unified benchmark for graph condensation. *NeurIPS* 37 (2024), 37900–37927.

[69] Jacob Townsend, Cassie Putman Micucci, John H Hymel, Vasileios Maroulas, and Konstantinos D Vogiatzis. 2020. Representation of molecular structures with persistent homology for machine learning applications in chemistry. *Nature communications* 11, 1 (2020), 3230.

[70] Nicholas L Turner, Thomas Macrina, J Alexander Bae, Runzhe Yang, Alyssa M Wilson, Casey Schneider-Mizell, Kisuk Lee, Ran Lu, Jingpeng Wu, Agnes L Bodor, et al. 2022. Reconstruction of neocortex: Organelles, compartments, cells, circuits, and activity. *Cell* 185, 6 (2022), 1082–1100.

[71] John Henry Constantine Whitehead. 1939. Simplicial spaces, nuclei and m-groups. *Proceedings of the London mathematical society* 2, 1 (1939), 243–327.

[72] Xiang Wu, Rong-Hua Li, Xunkai Li, Kangfei Zhao, and Guoren Wang. 2025. Scalable Topology-Preserving Graph Coarsening: Concepts, Algorithms and Applications. *Full version* (2025). https://github.com/BITNEO/STPGC/blob/main/STPGC.pdf

[73] Kelin Xia and Guo-Wei Wei. 2014. Persistent homology analysis of protein structure, flexibility, and folding. *International journal for numerical methods in biomedical engineering* 30, 8 (2014), 814–844.

[74] Zuoyu Yan, Tengfei Ma, Liangcai Gao, Zhi Tang, and Chao Chen. 2021. Link prediction with persistent homology: An interactive view. In *ICML*. PMLR, 11659–11669.

[75] Zuoyu Yan, Tengfei Ma, Liangcai Gao, Zhi Tang, and Chao Chen. 2022. Cycle Representation Learning for Inductive Relation Prediction. In *ICML, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA (Proceedings of Machine Learning Research)*, Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvári, Gang Niu, and Sivan Sabato (Eds.), Vol. 162. PMLR, 24895–24910.

[76] Zuoyu Yan, Tengfei Ma, Liangcai Gao, Zhi Tang, Chao Chen, and Yusu Wang. 2023. Cycle Invariant Positional Encoding for Graph Representation Learning. In *Learning on Graphs Conference, 27-30 November 2023, Virtual Event (Proceedings of Machine Learning Research)*, Soledad Villar and Benjamin Chamberlain (Eds.), Vol. 231. PMLR, 4.

[77] Zuoyu Yan, Tengfei Ma, Liangcai Gao, Zhi Tang, Yusu Wang, and Chao Chen. 2022. Neural approximation of graph topological features. *NeurIPS* 35 (2022), 33357–33370.

[78] Pinar Yanardag and SVN Vishwanathan. 2015. Deep graph kernels. In *KDD*. 1365–1374.

[79] Jaewon Yang and Jure Leskovec. 2012. Defining and evaluating network communities based on ground-truth. In *Proceedings of the ACM SIGKDD workshop on mining data semantics*. 1–8.

[80] Zhitao Ying, Dylan Bourgeois, Jiaxuan You, Marinka Zitnik, and Jure Leskovec. 2019. Gnnexplainer: Generating explanations for graph neural networks. *NeurIPS* 32 (2019).

[81] Ziqi Yuan, Haoyi Zhou, Tianyu Chen, and Jianxin Li. 2024. Phogad: Graph-based anomaly behavior detection with persistent homology optimization. In *Proceedings of the 17th ACM International Conference on Web Search and Data Mining*. 920–929.

[82] R. Zafarani and H. Liu. 2009. Social Computing Data Repository at ASU. http://socialcomputing.asu.edu

[83] Qi Zhang, Rong-Hua Li, Hongchao Qin, Yongheng Dai, Ye Yuan, and Guoren Wang. 2023. Neighborhood skyline on graphs: Concepts, algorithms and applications. In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*. IEEE, 585–598.

[84] Xikun Zhang, Dongjin Song, and Dacheng Tao. 2023. Ricci curvature-based graph sparsification for continual graph representation learning. *IEEE Transactions on Neural Networks and Learning Systems* (2023).

[85] Qi Zhao and Yusu Wang. 2019. Learning metrics for persistence-based summaries and applications for graph classification. *NeurIPS* 32 (2019).

[86] Qi Zhao, Ze Ye, Chao Chen, and Yusu Wang. 2020. Persistence enhanced graph neural network. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 2896–2906.

[87] 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. GNN: incorporating ring priors into molecular modeling. In *The Eleventh ICLR, ICLR*

2023, Kigali, Rwanda, May 1-5, 2023.

[88] Jiong Zhu, Yujun Yan, Lingxiao Zhao, Mark Heimann, Leman Akoglu, and Danai Koutra. 2020. Beyond homophily in graph neural networks: Current limitations and effective designs. *NeurIPS* 33 (2020), 7793–7804.