Graph-Skeleton: ~1% Nodes are Sufficient to Represent Billion-Scale Graph

Linfeng Cao*
The Ohio State University
Columbus, USA
cao.1378@osu.edu

Haoran Deng
Zhejiang University
Hangzhou, China
denghaoran@zju.edu.cn

Yang Yang†
Zhejiang University
Hangzhou, China
yangya@zju.edu.cn

Chunping Wang
FinVolution Group
Shanghai, China
wangchunping02@xinye.com

Lei Chen
FinVolution Group
Shanghai, China
chenlei04@xinye.com

ABSTRACT

Due to the ubiquity of graph data on the web, web graph mining has become a hot research spot. Nonetheless, the prevalence of large-scale web graphs in real applications poses significant challenges to storage, computational capacity and graph model design. Despite numerous studies to enhance the scalability of graph models, a noticeable gap remains between academic research and practical web graph mining applications. One major cause is that in most industrial scenarios, only a small part of nodes in a web graph are actually required to be analyzed, where we term these nodes as *target nodes*, while others as *background nodes*. In this paper, we argue that properly fetching and condensing the background nodes from massive web graph data might be a more economical shortcut to tackle the obstacles fundamentally. To this end, we make the first attempt to study the problem of massive background nodes compression for target nodes classification. Through extensive experiments, we reveal two critical roles played by the background nodes in target node classification: enhancing *structural connectivity* between target nodes, and *feature correlation* with target nodes. Following this, we propose a novel *Graph-Skeleton* model, which properly fetches the background nodes, and further condenses the semantic and topological information of background nodes within similar target-background local structures. Extensive experiments on various web graph datasets demonstrate the effectiveness and efficiency of the proposed method. In particular, for MAG240M dataset with 0.24 billion nodes, our generated skeleton graph achieves highly comparable performance while only containing 1.8% nodes of the original graph.

CCS CONCEPTS

• **Theory of computation → Data compression**.

*This work was done when the author was a visiting student at Zhejiang University.
†Corresponding author.

WWW '24, May 13–17, 2024, Singapore, Singapore

ACM ISBN 979-8-4007-0171-9/24/05…$15.00
https://doi.org/10.1145/3589334.3645452

KEYWORDS

Graph Mining, Graph Compression, Large-Scale Web Graph, Graph Neural Networks

ACM Reference Format:
Linfeng Cao, Haoran Deng, Yang Yang, Chunping Wang, and Lei Chen. 2024. Graph-Skeleton: ~1% Nodes are Sufficient to Represent Billion-Scale Graph. In *Proceedings of the ACM Web Conference 2024 (WWW '24), May 13–17, 2024, Singapore, Singapore.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3589334.3645452

1 INTRODUCTION

The ubiquity of graph data, especially in the form of web graphs, has made web graph mining a hot research topic. These web graphs are crucial for various applications, including web search [5, 13], social network [39, 40], blockchains [25–27], recommendation [28, 31], and more. However, it remains a big challenge to deploy the graph models on large-scale graphs. In practice, web graphs can be extremely large [42]. Take Facebook for instance, there are over 2.93 billion monthly active users [8]. Despite remarkable progress such as node sampling [3, 14, 42], model simplification [32, 35] to enhance the scalability of graph models on large-scale graph mining tasks, it remains a large gap between academic research and practical applications. One major reason is that in most industrial scenarios, not all nodes in a web graph are actually required to be analyzed [18]. We take DGraph, a real-world financial & social network dataset (users as nodes, social relationships between users as edges) [18], as an example to further illustrate this. Given a fraudster identification task among loan users, only the users with loan records needs to be classified, while the other users without loan behavior do not. Under this circumstance, we term the loan users as *target nodes*, while the others as *background nodes*. This target & background property is prevalent in web graph mining scenarios. Moreover, the number of background nodes is typically much larger than that of target nodes. For instance, to predict papers' subject areas in MAG240M [16], only 1.4 million Arxiv papers are concerned with classification among 240 million nodes.

Intuitively, it may not be an economical solution to deploy complex graphical models on massive web data just for a small number of node classification. It can pose significant challenges in terms of time and memory costs, as well as model design. Alternatively, a proper method to fetch and condense the useful background nodes from massive web data might be a shortcut to fundamentally tackle

the above obstacles. Nevertheless, how to fetch and condense the informative background nodes remains an open question at present. Background nodes can play diverse roles in target nodes classification, yet there is little understanding of how the background nodes impact the task performance. In this paper, we thus raise two questions: *Are background nodes necessary for target nodes classification? What roles do they play in the target classification task?*

To answer these questions, we conduct a comprehensive analysis using Graph Neural Network (GNN), one of the most popular graph models [38], for exploring the target-background issue. First of all, we observe a significant performance degradation when removing all background nodes, but a negligible impact of removing background nodes that are not neighboring the target nodes. Moreover, we find that the connection between target nodes plays a vital role in classification. Removing the background nodes that bridge between multiple target nodes results in a dramatic performance decline. Additionally, even background nodes neighboring a single target node would exhibit relatively higher feature correlation with their corresponding target nodes and contribute to the classification. Detail experimental results and analyses can be seen in Section 2.

This exploration reveals two key insights: First, the majority of background nodes are redundant, while the nodes neighboring the target nodes are crucial for target classification. Second, background nodes contribute to the target nodes classification primarily in two ways: i) enhance the *connectivity* between targets as *bridging node*; ii) neighboring to single target node but have *feature correlation* with the target node as the *affiliation node* (illustrated in Figure 1 (c)). With this inspiration, we argue that it is possible to generate a small and highly informative subgraph (with original target nodes and far fewer background nodes) from original web graph. The generated graph contains rich information for target nodes classification and is also friendly for graph model deployment and storage. However, we still face two challenges. (1) Extracting subgraph from original one would inevitably cause semantic and structural information loss. How to properly fetch the subgraph with useful background nodes? (2) The fetched subgraph would also contain redundant structural and semantic information. How to condense the subgraph while preserving the essential information for target classification?

In this paper, we propose a novel *Graph-Skeleton* to generate a small, synthetic and highly-informative graph for target node classification. Specifically, following the intuition of empirical analysis, we formulate a principle for background node fetching, ensures that the extracted subgraph maintains both the structural connectivity and feature correlation via bridging and affiliation background nodes. After subgraph extraction, we propose three graph condensation strategies to condense the redundant structural and semantic information. The condensed synthetic graph (term as *skeleton*) contains sufficient information for target node classification and enjoys the benefits of small scale. Our main contributions are summarized as follows: (1) We first address a common challenge in real-world web applications: compressing the massive background nodes for classifying a small part of target nodes, to ease data storage, GNNs deployment and guarantee the performance. Empirical analysis explicitly indicates the contributions of background nodes to the target classification, i.e., enhancing target *structural connectivity* and *feature correlation* with target nodes, which provides a valuable guidance for background nodes fetching. (2) We propose a novel *Graph-Skeleton* for massive background nodes compression. It properly fetches the useful background nodes from massive web graph and performs background node condensation to eliminate information redundancy. (3) Extensive experiments on various web graphs demonstrate the effectiveness and efficiency of the proposed method. In particular, for MAG240M dataset with 0.24 billion nodes, our generated skeleton graph achieves highly comparable performance while only contain 1.8% nodes of the original graph.

2 EMPIRICAL ANALYSIS

In this section, we conduct empirical analyses to explore the target-background problem, for answering two key questions we raise above: *Are the background nodes necessary for target nodes prediction? What roles do they play in the target classification task?* We first analyze the overall contribution of background nodes in target classification, and then we explore what kind of background nodes are essential and how they contribute to the performance.

To ensure the generality of our analysis, we employ GNNs (GraphSAGE [14], GAT [30], GIN [33]) with three representative aggregation mechanisms, including mean, weight-based and summation, as the backbones for target nodes classification. The task is conducted on two datasets: (1) Financial loan network DGraph [18]. We follow the same task setting as the original dataset, i.e., fraudster identification among loan users, so that the users with loan action are regarded as target nodes (~33%), while others are background nodes. (2) Academic citation network ogbn-arxiv [17]. We aim to predict the subject areas of papers published since 2018. In this case, papers published from 2018 are regarded as target nodes (~46%), while papers before 2018 are background nodes. The detailed experimental settings are provided in Appendix A.1.

Are Background Nodes Necessary for Target Classification? We first evaluate the contribution of background nodes to the overall performance. Specifically, we delete all the background nodes by cutting background-to-background edges (B-B) and target-to-background (T-B) edges. In this way, the information propagation of each background node will be cut-off. As comparison, the random edge cut (cut ratio spans from 0 to 1) is implemented. As results depicted in Figure 1 (a), when cutting B-B (▭), the performances of all GNNs show no significant decline and even presents slight improvement (DGraph) compared to the original graph (┈┈), indicating the background nodes are indeed highly redundant and even contain noise. However, when cutting T-B (▭), the performance presents a significant decline compared to the random edge cut (—). It reveals that the background nodes contain abundant information, which is essential to target node prediction.

Background Nodes Contribute to Structural Connectivity Between Target Nodes. For a comprehensive analysis, we additionally cut the target-to-target edges (T-T) to explore the dependency between the target nodes. One key observation is that the *structural connectivity* between target nodes plays an essential role in prediction. As shown in Figure 1 (a), the performance of cutting T-T (▭) presents a significant decline compared to the original graph and random edge cut (—). Then, what role of the background nodes play in the target node classification? Inspired by the above observations, we cut the T-B edges where background nodes act as the 1-hop bridging nodes between two target nodes (i.e., T–╳–B–╳–T,

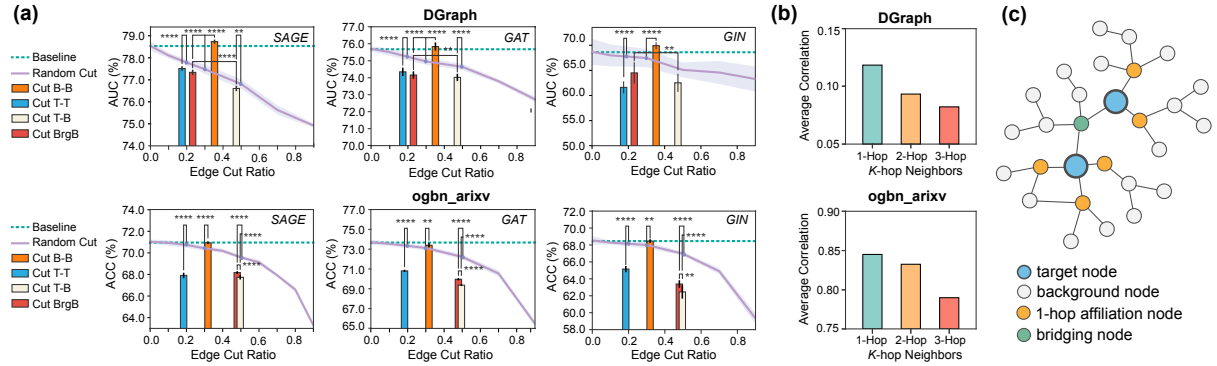

Figure 1: (a) Explorations of background nodes influences. Upper: results on DGraph [18]. Lower: results on ogbn-arxiv [17]. (**p < 1e-4, **p < 1e-2, paired t-tests; errorbars represent the standard deviation). (b) Feature correlation between target nodes and their neighboring background nodes. (c) Illustration of target nodes and the corresponding essential background nodes.**

BridB) to weaken the connectivity between targets. Consistent with T-T edge cutting, the performance of BridB cutting (▬) also declines significantly (Figure 1 (a)), indicating that enhancing target connectivity via bridging background nodes contributes to task.

Background Nodes Has Higher Feature Correlation with Neighboring Target Nodes. From the experimental results in Figure 1 (a), we can still observe a performance gap between BridB cutting (▬) and T-B edges cutting (▭), i.e., the performance of BridB cutting outperforms that of cutting all background nodes. This indicates that apart from the background node bridging multiple targets, the background node neighboring to a single target node also contributes to the task.

From previous studies, numerous representative GNNs [12, 14, 23, 30] employ a repeated propagation process to integrate the feature information from neighboring nodes [36, 37]. This process promotes the similarity of features among neighboring nodes, leading to the creation of synthetic and robust node representations. Following this, we hypothesize that the background nodes may exhibit higher *feature correlation* with their neighboring target nodes, enabling them to contribute during the propagation operation. To verify our hypothesis, we compute the Pearson correlation coefficient of features between target nodes and the background nodes with different neighboring hops. As the results shown in Figure 1 (b), the background nodes closer to the target nodes indeed have higher feature correlation, suggesting that the features of background nodes neighboring to a single target may correlated with feature of target itself, and thus contribute to the performance.

Intuitions. The analysis above provides us two key insights. First, the majority of background nodes are redundant, while the nodes neighboring to the target nodes are important to the target classification. Second, background nodes contribute to the target nodes classification primarily in two ways: i) enhance the *structural connectivity* between target nodes as *bridging node*; ii) neighbor to a solo target node with *feature correlation* to that target node as the *affiliation node* (illustrated in the Figure 1 (c)).

3 METHODOLOGY

Problem Definition. Given a large-scale graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and a specific node classification task, we can thus get a corresponding node set $\mathcal{T} = \{T_1, T_2, ..., T_n\}$, containing the nodes that are required to be classified, where $|\mathcal{T}| \ll |\mathcal{V}|$ in most of real-world scenarios. In this paper, we refer $\mathcal{T}$ as *target nodes*, and other nodes in $\mathcal{G}$ as background nodes $\mathcal{B} := \mathcal{V} \setminus \mathcal{T} = \{B_1, B_2, ..., B_{|\mathcal{V}|-n}\}$. The graph is also associated with node features $X \in \mathbb{R}^{|\mathcal{V}| \times d}$ and target node labels $Y \in \{0, ..., C-1\}^n$. Given that the majority of nodes are background nodes, our objective is to generate a synthetic graph $\mathcal{G}'$ that is highly informative while significantly reducing background nodes to alleviate the computational and storage burdens. This synthetic graph $\mathcal{G}'$ can be used to train graph models and classify the target nodes directly with comparable performance to the original graph $\mathcal{G}$. In this paper, we focus on the target nodes analysis in the real-world applications. Therefore we only compress the background nodes while preserving the whole original target nodes $\mathcal{T}$ in the generated synthetic graph $\mathcal{G}'$. This ensures that none of the target nodes are lost, which is crucial as it allows us to trace and retain the specific information associated with each individual target node in the compressed synthetic graph $\mathcal{G}'$.

Framework Overview. To tackle the problem, we propose a novel *Graph-Skeleton* framework to generate a synthetic skeleton subgraph from massive web graph with much smaller size but rich information for target classification. The framework is illustrated in Figure 2. It first fetches all target nodes and a subset of background nodes to construct a vanilla subgraph (Figure 2, left). For proper background nodes fetching, we formulate a fetching principle following the inspirations of *structural connectivity* and *feature correlation* in Section 2. Then Graph-Skeleton condenses the graph information of the vanilla subgraph (Figure 2, middle) to reduce redundancy. Specifically, we design three graph condensation strategies (i.e., α, β, γ) with condensation level ranging from low to high degree. The condensed graph (refer as *skeleton graph*) is highly informative and enjoys the benefits of small-scale for storage and graph model deployment (Figure 2, right).

3.1 Node Fetching

The observations in Section 2 reveal that the background nodes can be massive and highly redundant. To tackle this limitation, one natural idea is to reduce the graph size by fetching the essential background nodes and removing those that make little contribution to target classification. Inspired by the key observations of *structural connectivity* and *feature correlation*, we design a fetching principle

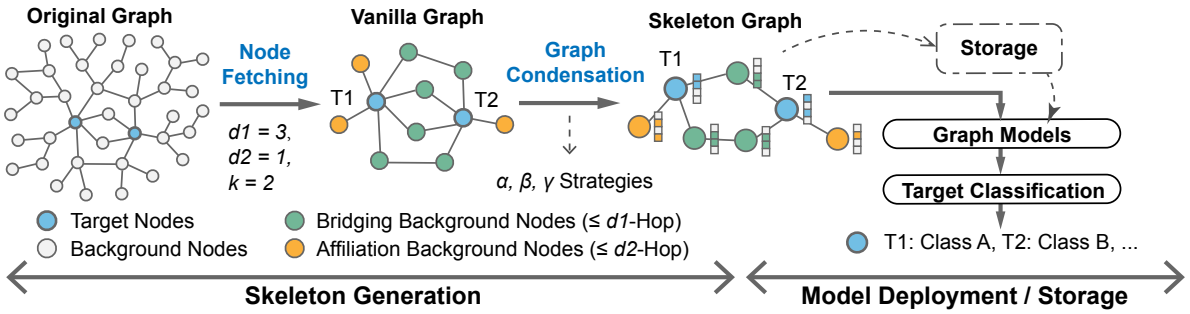

Figure 2: Graph-Skeleton framework: it generates a synthetic skeleton subgraph from original graph with rich information for target prediction while enjoying the benefits of small scale. It first fetches the essential background nodes under the guidance of *structural connectivity* and *feature correlation* (Left), then condenses the information of background nodes (Middle). The generated skeleton graph is highly informative and friendly for storage and graph model deployment (Right).

to properly fetch the *bridging* and *affiliation* background nodes from massive original data as the first phase. Utilizing these nodes, we can construct a vanilla subgraph containing all target nodes and a small subset of background nodes. To alleviate the over-expansion issue, we customize the fetching depth and width with $d_1, d_2, K \in \mathbb{N}$. The fetching principle is formulated as follows:

Principle of What Background Node Will be Fetched:
1. Structural connectivity: *Bridges two or more target nodes within d_1-hop as bridging node.*
2. Feature correlation: *K highest correlation background nodes neighboring to solo target node within d_2-hop as affiliation nodes.*

Bridging Background Node Fetching. Following principle.1, to fetch the bridging background nodes, our first step is to identify all *accessible background nodes* for each target node. Note that the *accessible nodes* for a node A refers to the nodes that can be reached from A along a path only composed of background nodes. We will use this notion in the following paper. To this end, we utilize the breadth first search (BFS) [7] to find the shortest paths from each target node to its all accessible background nodes. By traversing all target nodes, the shortest paths set for each background node to its accessible target nodes can also be obtained. Let B_j be one background node and is accessible to target nodes $T_{k_1}, T_{k_2}, ..., T_{k_i}$, with the corresponding shortest path set $PB_j = \{p_{k_1,j}, p_{k_2,j}, ..., p_{k_i,j}\}$. To verify whether B_j aligns with Principle.1, we calculate the length of each path $d(PB_j) = \{d(p_{k_1,j}), ..., d(p_{k_i,j})\}$ ($d(\cdot)$ is the distance function), and sum up the minimum and second-minimum distance values of these paths as $sd_j = \min[d(PB_j)] + \min_{2nd}[d(PB_j)]$. If $sd_j \leq d_1$, it indicates that the background nodes B_j bridges at least two target nodes within d_1-hop and will be regarded as the bridging node. By traversing all background nodes, a node subset $\mathcal{BR}$ containing all bridging background nodes can be obtained.

Affiliation Background Node Fetching. By conducting BFS for each target node T_i, we can obtain the shortest path set of T_i containing paths to its all accessible background nodes. Let $J := \{j_1, j_2, ..., j_k\}$ be the indices of accessible background nodes to T_i, the corresponding shortest path set is $PT_i := \{p_{i,j}\}, j \in J$, where $p_{i,j}$ refers to the shortest path from T_i to an accessible background node B_j. Following principle.2, we first pick the accessible background nodes with the shortest path distance within d_2, i.e., $\{B_m, m \in J, s.t., d(p_{i,m}) \leq d_2\}$. To fetch the most essential K background nodes, we compute the feature Pearson correlation coefficient (PCC) for each picked B_m with T_i, $PCC_{im} = \frac{cov(X[i]X[m])}{\sigma_{X[i]}\sigma_{X[m]}}$, where X is the node feature matrix, $cov(\cdot)$ refers to the covariance and σ refers to standard deviation. Then background nodes in $\{B_m\}$ with K largest PCC will be fetched as affiliation nodes of T_i. By traversing all target nodes, an affiliation background nodes subset $\mathcal{AF}$ can be obtained. Then we can construct a vanilla subgraph $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$ by preserving the target nodes $\mathcal{T}$ and the fetched background nodes $\mathcal{B}' = \{\mathcal{BR}, \mathcal{AF}\}$ within the original graph $\mathcal{G}$ (Figure 2 left, where d_1, d_2 and K are set to 3, 1 and 2 respectively).

3.2 Graph Condensation

To reduce information redundancy, we develop a condensation process for the constructed vanilla subgraph $\mathcal{G}'$, which effectively condenses both structural and semantic information. Specifically, we propose three graph condensation strategies, denoted as α, β, and γ, which provide varying degrees of condensation, ranging from low to high.

Strategy-α. Following previous studies [4, 35], the number of equivalence classes can be utilized to measure the richness of information. Under this inspiration, we leverage the equivalence relationship of node pairs as a hint of information redundancy, enabling us to condense the semantic and structural information in the vanilla subgraph $\mathcal{G}'$. To do so, we first introduce the notion of node pair equivalence relation [4] on a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, with X the node feature matrix.

Definition 3.1 (Node Pair Equivalence Class). Given a function family $\mathcal{F}$ on $\mathcal{G}$, define equivalence relation $\simeq_{\mathcal{F}}$ among all graph node pairs such that $\forall u, v \in \mathcal{V}$, $u \simeq_{\mathcal{F}} v$ iff $\forall f \in \mathcal{F}$, $f(\mathcal{G}, X) = f(\mathcal{G}, \tilde{X})$, where $\tilde{X} = X$ except $\tilde{X}[u] = X[v], \tilde{X}[v] = X[u]$.

Considering the equivalent node pairs share an identical structure within the graph, we argue that there is a large space for graph condensation. To this end, we propose a condensation strategy-α, which leverages *multiple structure-set* (*MSS*) to captures the local structural information of each fetched background node in vanilla subgraph $\mathcal{G}'$. It allows us to identify background nodes with similar structural information and condense them into a synthetic node (shown in Figure 3). Specifically, for one background node $B_j \in \mathcal{B}'$

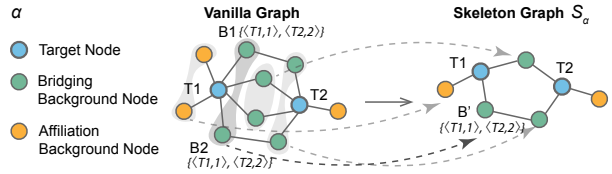

Figure 3: Illustration of Strategy-α, where the background nodes sharing a identical structural multiple-set (MSS) $\{\langle T, d\rangle\}$ (within a same shadow envelope) will be condensed into one synthetic node.

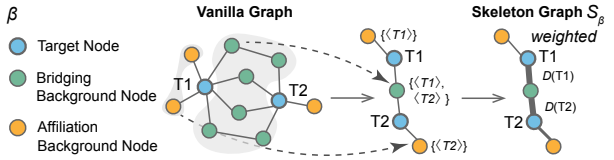

Figure 4: Illustration of Strategy-β. The background nodes sharing the same structural multiple-set MSS' $\{\langle T\rangle\}$ (within the same shadow envelope) will be condensed into one node. To maintain the relative distance information between different nodes, we encode the distance information of target nodes by weighting the edges of skeleton graph.

in $\mathcal{G}'$, we formulate the multiple structure-set MSS_j via its accessible target nodes and the corresponding shortest path distances:

$$MSS_j = \{\langle T_i, d_{i,j}\rangle, ..., \langle T_k, d_{k,j}\rangle\}, \tag{1}$$

where $T_i, ..., T_k$ are the accessible target nodes of B_j, $d_{i,j}$ represents the shortest path distance between B_j and T_i (For simplicity, we use this notation in the following paper). For the background nodes with the same MSS, we claim that these background nodes belong to the same linear message path passing (LMPP) equivalence class. The detailed definition is given below.

Definition 3.2 (Linear message passing operation). Given two connected nodes u, v, define the linear message passing operation $f_{lmp}(u, v)$ from u to v as:

$$X'[v] \leftarrow f_{lmp}(X[v], X[u]) = \text{AGGREGATE}(\{X[v], X[u]\})W, \tag{2}$$

where W is a transformation matrix, AGGREGATE can be formulated as element-wise mean or summation pooling.

Definition 3.3 (Linear message path passing). Given a path $p = \langle u_0, u_1, ..., u_\ell\rangle, \forall u_i \in \mathcal{V}$, define the linear path passing functions $f_{spp}(X, p)$ aggregating node feature from u_0 to u_ℓ over p as:

$$X'[u_i] \leftarrow f^i_{lmp}(X'[u_{i-1}], X[u_i])W^i, i \in 1, .., \ell. \tag{3}$$

Definition 3.4 (LMPP equivalence class). Given a family of linear message path passing functions $\mathcal{LMPP}$, two nodes $u, v \in \mathcal{V}$ and one common accessible node $K \in \mathcal{V}$, with the corresponding paths $p_{u,K} := \langle u, ..., K\rangle$ and $p_{v,K} := \langle v, ..., K\rangle$, let $P := p_{u,K}, p_{v,K}$, define LMPP equivalence relation $u \simeq_{\mathcal{LMPP},K} v$ iff $\forall f_{lmpp} \in \mathcal{LMPP}$, $f_{lmpp}(X, P) = f_{lmpp}(\tilde{X}, P)$ where $\tilde{X} = X$ except $\tilde{X}[u] = X[v], \tilde{X}[v] = X[u]$.

Here we relax the function family $\mathcal{F}$ in Definition 3.1 to linear message path passing functions $\mathcal{LMPP}$ over the paths for aggregation rather than the whole graph. Now we give a proposition to characterize the LMPP equivalence relation of background nodes.

Proposition 3.5. *With $T \in \mathcal{T}$ denoting a target node in G, $\mathcal{B}$ denotes the set of background nodes, $\forall u, v \in \mathcal{B}$, if $MSS_u = MSS_v \neq \emptyset$, then $u \simeq_{\mathcal{LMPP},T} v$.*

We provide the proof of Proposition 3.5 in Appendix B. Proposition 3.5 and the proof suggests that for background nodes with same MSS, sharing one same path for linear message passing operation delivers the same aggregated features at target node as using their own original paths. In this case the original multiple paths for linear message passing are actually redundant.

Following this, we argue that the background nodes with the same MSS may also leverage quite similar structure for information aggregation over the graph, and condensing this structural and feature information may also deliver similar aggregation results via nonlinear message passing operation. To this end, we condense the background nodes with the same MSS into one synthetic node for reducing information redundancy. As an example shown in Figure 3, both B_1, B_2 have the identical multiple structure-set contents, i.e., $MSS_1 = MSS_2 = \{\langle T_1, 1\rangle, \langle T_2, 2\rangle\}$, and will be condensed into one synthetic node B'. To preserve the semantic information of condensed background nodes, we generate the synthetic node feature via aggregating the original features of the corresponding condensed background nodes. Let $\mathcal{B}'_k = \{B_i, ..., B_j\}$ be the set of background nodes with same MSS to be condensed into a synthetic node B'_k, the feature of $x_{B'_k}$ is

$$x_{B'_k} \leftarrow \text{AGGREGATE}(\{x_v, \forall v \in \mathcal{B}'_k\}), \tag{4}$$

where AGGREGATE$(\cdot)$ can be element-wise mean or summation pooling. By condensing all sets of $\mathcal{B}'$, the condensed skeleton graph $\mathcal{S}_\alpha$ can be obtained for storage and graph model deployment.

Strategy-β. While the strategy-α effectively reduces the graph redundancy, its compression effect is limited as only a portion of background nodes strictly share the same MSS $\{\langle T, d\rangle\}$. To this end, we propose the second condensation strategy-β with stronger condensation capacity over vanilla subgraph $\mathcal{G}'$. As a compromise, for each background node, we only involve its accessible target nodes while omitting the corresponding distance information in MSS, i.e., $MSS' = \{\langle T_i\rangle, ..., \langle T_j\rangle\}$. Similarly, for the background nodes with the same MSS', they will be condensed into a synthetic node. As shown in Figure 4, the background nodes $B_1, B_2, B_3, B_4, B_5, B_6$ all have the same $MSS' = \{\langle T_1\rangle, \langle T_2\rangle\}$, which will be condensed into a synthetic node B'. The features generations of new condensed synthetic nodes follow the Eq 7.

However, it should be noted that in this strategy, the condensed graph would lose the relative distance information between nodes. For instance, the target nodes get closer when bridging background nodes are condensed together. Some target nodes that were originally several hops apart may be connected by a 1-hop bridging background node in the condensed graph.

To address this issue, we propose to encode the relative distance information between targets and backgrounds onto the edges in condensed graph. Let $\mathcal{B}'_k = \{B_i, ..., B_j\}$ be the set of background

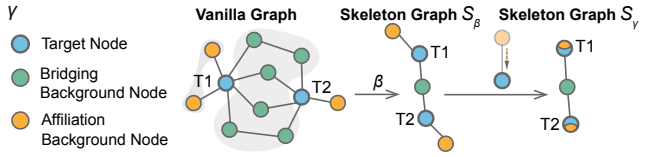

Figure 5: Illustration of Strategy-γ. Based on condensation-β, we further condense the affiliation nodes to the corresponding target node.

nodes in vanilla subgraph $\mathcal{G}'$ with the same $MSS'_k = \{\langle \mathcal{T}_k \rangle\}$, which will be condensed into a synthetic node B'_k. Given one accessible target node $T_m \in \mathcal{T}_k$ connected with B'_k via edge $e'_{m,k}$, and the shortest distance set of $\mathcal{B}'_k$ to T_m is $D_m := \{d_{m,i}, ..., d_{m,j}\}$. Then the weight of edge $e'_{m,k}$ is formulated as: $w'_{m,k} = \sum \frac{1}{d}, \forall d \in D_k$, which can be utilized to weight the features during aggregation in downstream graph models. To guarantee the scale consistency, normalization based on rows and columns is also required for the edge weights. After condensation for all sets of $\mathcal{B}'$, we can obtain the condensed skeleton graph $\mathcal{S}_\beta$.

Strategy-γ. Although the condensation-β strategy effectively reduces the size of bridging background nodes, there is still room for further condensation of affiliation background nodes. Numerous representative GNNs employ the propagation process to merge the information from neighbors [14, 23]. Intuitively, aggregating the features of affiliation nodes onto the neighboring target node directly would deliver a similar result with the recursive aggregation.

Under this inspiration, we propose the third strategy-γ for condensation. Given the vanilla subgraph $\mathcal{G}'$, we first perform strategy-β to condense the bridging background nodes, then we condense the affiliation nodes in the generated skeleton graph $\mathcal{S}_\beta$. Specifically, given one target node T_k in $\mathcal{S}_\beta$, with its corresponding affiliation background node set $\mathcal{AF}_k$, we update the feature of target T_k by incorporate its feature with the features of $\mathcal{AF}_k$, and then remove $\mathcal{AF}_k$ in $\mathcal{G}'$ (shown in Figure 5), to eliminate the massive affiliation nodes while maintaining most of the original correlation information from affiliation nodes. The condensed feature of T_k via aggregating the original features with $\mathcal{AF}_k$ is formulated below:

$$x_{T_k} \leftarrow \text{AGGREGATE}(\{x_{T_k} \cup \{x_u, \forall u \in \mathcal{AF}_k\}\}), \tag{5}$$

AGGREGATE$(\cdot)$ can be element-wise mean or summation pooling. Finlay, we can obtain the condensed skeleton graph $\mathcal{S}_\gamma$, which is highly informative and friendly to storage and model deployment. We provide the time complexity analysis in Appendix D.

4 EXPERIMENTS

Experimental protocol. To comprehensively evaluate the performance of the proposed *Graph-Skeleton*, we conduct the target nodes classification task on six web datasets: DGraph [18], ogbn-mag [17], ogbn-arxiv [17], MAG240M [16], DBLP [10] and IMDb [10], spanning across multiple domains. Based on the downstream task, we can obtain the corresponding target nodes (required to be classified) and background nodes. The basic information of datasets and how we select the target nodes are listed in Table 1. Note that in our study, we only compress the background nodes, and all target nodes are preserved in the generated skeleton subgraph. In this case, the generated skeleton and original graph contain the same target nodes for classification.

Table 1: Statistics of datasets.

Dataset	Nodes	Edges	Target Definition	Targets
DGraph	3,700,550	4,300,999	Loan Users	1,225,601
ogbn-arxiv	169,343	1,166,243	Papers (since 2018)	78,402
IMDB	11,616	17,106	Movies	4,278
DBLP	26,108	119,783	Authors	4,057
ogbn-mag	1,939,743	21,111,007	Papers	736,389
MAG240M	244,160,499	1,728,364,232	arxiv papers	1,398,159

Experimental Setup. We compare the downstream target node classification performance with original graph and other graph compression baselines including coreset methods (*Random*, Centrality Ranking [11] with PageRank centrality (*Central-P*) and degree centrality (*Central-D*)), graph coarsening methods (variation neighborhoods coarsening (*GC-VN*) [19], Algebraic JC coarsening (*GC-AJC*) [19], spectral coarsening with Schur complement (*Schur-C*) [41]), graph condensation methods (*GCond* [21], *DosCond* [20]) and graph active learning method (*GPA* [15]). Note that our goal is to compress the background node compression while maintain all target nodes. The compression rate is indicated by *background node compression rate* (*BCR*) (ratio of synthetic background nodes to original background nodes). For a fair comparison, we keep the *BCR* same across all methods.

After obtaining the compressed graphs by above methods, we adopt the GNNs to evaluate their target classification performance. Considering different datasets would be applicable to different GNNs, for DGraph, ogbn-axiv, ogbn-mag and MAG240M, we select the base GNNs on their respective official leaderboards for evaluation. For IMDB and DBLP, we adopt the most representative GNNs (GCN [23], GraphSAGE [14], GAT [30]) for evaluation. The target classification performance of DGraph is evaluated by AUC (%), and other datasets are evaluated by ACC (%). The code implementation is available at https://github.com/zjunet/GraphSkeleton.

4.1 Graph Compression Comparison

We first report target node classification results of compressed graphs under fixed compression rate (*BCR*) on six datasets in Table 2, where we compare the performance of Graph-Skeleton using condensation strategy-γ (Skeleton-γ in short, which has highest compression rate) to other baselines. As we can see, Skeleton-γ presents strong ability of scaling up GNNs to all datasets, including large-scale graph MAG240M with ~0.24 billion nodes. It also achieves superior target classification performance compared to other compression baselines under similar *BCR*. Besides, compared to other graph coarsening and compression methods being significantly hindered by heavy memory and computational loads, our method is more friendly for deployment on large-scale web graphs. Moreover, compared to the original graph, Skeleton-γ also presents highly comparable or even better target classification performance with a notably smaller number of background nodes.

Additionally, we report the target classification results of compressed graph with varied *BCR*s in Figure 6. By selecting different condensation strategies of Graph-Skeleton with different fetching depths ($d1, d2$), we can flexibly achieve different compression rates (details in Appendix A.2.4). It can be easily observed that our

Table 2: Target node classification comparisons on the original and the compressed graphs with fixed *BCR*s. "OOM": out of memory. DGraph uses AUC (%) for evaluation, other datasets use ACC (%).

Datasets	GNN	Original	Skeleton-γ	Random	Cent-P	Cent-D	Schur-C	GC-VN	GC-AJC	GCond	DosCond	GPA
DGraph BCR=15%	SAGE [14]	78.7±0.1	**78.2±0.1**	74.2±0.1	74.6±0.1	74.5 ±0.1	75.4±0.1	OOM	OOM	OOM	OOM	OOM
	GCN [23]	73.4±0.2	**74.4±0.1**	72.1±0.1	72.6±0.1	72.5±0.1	72.8±0.2	OOM	OOM	OOM	OOM	OOM
	GAT [30]	75.9±0.4	**76.6±0.6**	73.3±0.3	73.9±0.4	73.8±0.4	74.2±0.5	OOM	OOM	OOM	OOM	OOM
	NAGphormer [2]	76.5±0.1	**77.2±0.1**	73.7±0.1	74.49±0.1	74.21±0.1	75.1±0.2	OOM	OOM	OOM	OOM	OOM
ogbn_arxiv BCR=7%	SAGE [14]	71.0±0.5	68.8±0.4	66.7±0.2	68.0±0.3	67.9±0.3	**68.9±0.4**	67.5±0.5	68.3±0.4	64.2±0.3	61.2±0.3	OOM
	GCN [23]	71.3±0.2	**69.2±0.3**	67.8±0.3	68.1±0.2	69.0±0.3	68.0±0.3	68.1±0.2	68.0±0.2	64.3±0.4	61.5±0.5	OOM
	GAT [30]	72.1±0.1	**70.9±0.1**	70.1±0.1	70.3±0.2	70.2±0.1	70.5±0.2	70.1±0.2	70.3±0.2	62.4±0.5	59.3±0.4	OOM
	SIGN [9]	71.8±0.1	**69.3±0.2**	68.8±0.2	69.0±0.1	69.2±0.2	69.1±0.2	69.1±0.1	69.2±0.2	63.6±0.3	60.4±0.3	OOM
DBLP BCR=12%	SAGE [14]	84.1±0.4	**82.0±0.3**	79.3±0.2	80.0±0.4	79.8±0.5	78.3±0.4	77.9±0.8	79.8 ± 0.5	78.3±0.7	76.1±0.5	80.2±0.4
	GCN [23]	81.7±0.3	**82.1±0.3**	79.5±0.3	80.1±0.2	80.0±0.3	67.1±0.8	81.2±0.7	81.4±0.5	79.5±0.9	76.5±0.6	79.7±0.3
	GAT [30]	79.5±0.3	**79.3±0.3**	79.0±0.5	78.9±0.4	79.1±0.4	57.1±1.6	71.2±0.6	67.9±1.2	74.5±1.0	74.2±0.8	78.5±0.5
IMDB BCR=36%	SAGE [14]	55.2±0.6	55.6±1.2	48.8±0.8	51.1±0.8	51.3±0.8	53.1±0.7	55.4±0.5	**56.6±0.3**	49.1±0.8	53.2±1.0	51.9±0.9
	GCN [23]	56.9±0.7	**56.9±1.2**	49.8±0.8	51.6±0.7	51.8±0.7	53.4±0.8	55.5±0.5	56.4±0.5	50.6±0.6	53.5±0.8	52.5±0.7
	GAT [30]	57.2±0.6	**54.5±1.0**	48.2±0.7	50.7±0.8	51.1±0.7	51.6±1.8	52.3±0.6	52.8±0.6	48.9±0.7	50.2±0.6	49.4±0.8
ogbn_mag BCR=40%	R-GCN [29]	46.0±0.7	**46.2±0.4**	22.4±0.9	24.8±0.8	25.1±0.7	39.5±0.4	OOM	OOM	OOM	OOM	OOM
	GraphSaint [34]	43.2±0.5	**43.9±0.3**	9.1±1.1	13.6±1.5	11.2±1.2	36.7±0.5	OOM	OOM	OOM	OOM	OOM
	Cluster-GCN [6]	38.5±0.2	**39.5±0.2**	35.2±0.3	36.0±0.2	35.9±0.2	33.5±0.3	OOM	OOM	OOM	OOM	OOM
MAG240M BCR=1%	R-GAT [29]	70.0	**68.5**	59.0	59.8	60.1	OOM	OOM	OOM	OOM	OOM	OOM
	R-SAGE [29]	69.4	**68.2**	59.4	59.6	60.4	OOM	OOM	OOM	OOM	OOM	OOM

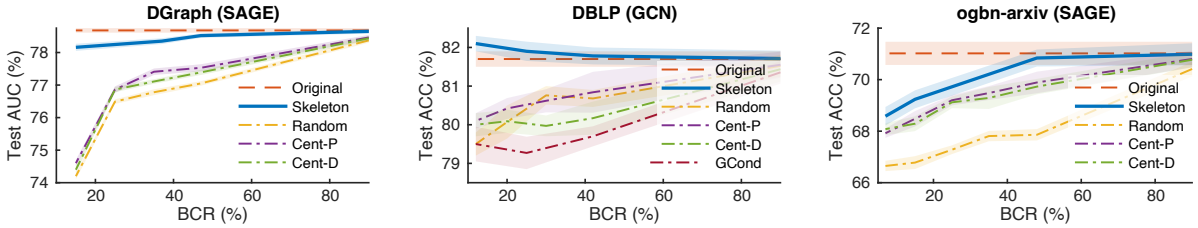

Figure 6: Comparisons of graph compression methods with varied *BCR*s for GNNs inference.

Table 3: Memory costs for data storage.

DGraph	Target Feat	Background Feat	Adj Matrix	AUC (SAGE)
Original	166.3 MB	336.6 MB	128.8 MB	78.7±0.1
Skeleton-γ (BCR: 15%)		50.1 MB	52.4 MB	78.2±0.1
ogbn-arixv	Target Feat	Background Feat	Adj Matrix	ACC (SAGE)
Original	40.1 MB	46.5 MB	37.0 MB	71.0±0.5
Skeleton-γ (BCR: 7%)		3.2 MB	12.3 MB	68.6±0.4
DBLP	Target Feat	Background Feat	Adj Matrix	ACC (GCN)
Original	1.0 MB	5.6 MB	3.8 MB	81.7±0.3
Skeleton-γ (BCR: 12%)		0.7 MB	0.2 MB	82.1±0.3
IMDB	Target Feat	Background Feat	Adj Matrix	ACC (GCN)
Original	52.4 MB	89.9 MB	0.6 MB	56.9±0.7
Skeleton-γ (BCR: 36%)		34.4 MB	0.4 MB	56.9±1.2
ogbn-mag	Target Feat	Background Feat	Adj Matrix	ACC (GraphSaint)
Original	377.0 MB	616.1 MB	674.9 MB	43.2±0.5
Skeleton-γ (BCR: 40%)		245.6 MB	505.8 MB	43.9±0.3
MAG24M	Target Feat	Background Feat	Adj Matrix	ACC (R-SAGE)
Original	2.05 GB	372.97 GB	55.95 GB MB	69.4
Skeleton-γ (BCR: 1%)		4.65 GB	2.69 GB	68.2

method significantly outperforms other methods in a wider range of compression rate.

4.2 Storage Costs

We report the memory costs for web graph storage of original graph and Skeleton-γ under the fixed *BCR* on six datasets. To present the results more intuitively, we decouple the storage costs of graph data into three main aspects: costs of target nodes features, background nodes features and graph adjacency matrix. The results are shown in Table 3. As we can see, Skeleton-γ significantly reduces the memory cost of background nodes features and graph adjacent matrix (green in Table 3). Since we preserve all the target node in compressed Skeleton-γ, the storage cost of target nodes feature keeps the same with the original graph. On the other hand, Skeleton-γ also achieves close or even better performance compared to the original data. This highlights the effectiveness of our proposed method in preserving the essential information for target node classification.

4.3 Studies on Three Condensation Strategies

In this section, we investigate the compression performance of three proposed condensation strategies α, β, γ of Graph-Skeleton. Specifically, we use the same vanilla subgraph as input and use three strategies for condensation. The results on four datasets are depicted in Figure 7, where the left axis (blue) presents the background nodes compression rate (*BCR*, bar) and right axis (red) presents the target node classification performance (dashed lines). As we can see, Skeleton-α, β, γ all present highly competitive target node classification performance with the original data, indicating the effectiveness of three proposed strategies for condensation. Generally the Skeleton-α presents the best classification performance within three strategies due to fewer information losses. On the other hand,

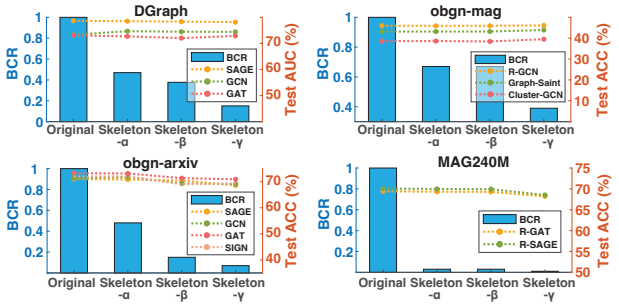

Figure 7: Compression performance of Graph-Skeleton with three condensation strategies: α, β, γ. The bars indicate the background nodes compression rate (BCR), dashed lines with points indicate target node classification performance.

Skeleton-γ also well approximates the original performance under all tested downstream GNNs while with a notably higher *BCR*, showing aggregating the features of affiliation background nodes onto the neighboring target node can well preserve the original background node information.

4.4 What Kinds of Background Nodes Are Essential in Real-World Cases?

Due to different properties of the web graph data, the condensed skeleton graph structure also vary dramatically from each other. Since none of the target nodes are lost in our compressed skeleton graph, it allows us to trace and investigate some important information and patterns relevant to the target classification upon the generated skeleton graphs. In this section, we utilize the distance information in each background node' multiple structure-set *MSS* in skeleton graph $\mathcal{S}_\alpha$ to represent the target-background structural patterns and leverage the attention mechanism to explore the importance of each structural pattern. Given background node B with $MSS = \{\langle T_i, d_i\rangle, \langle T_j, d_j\rangle\}$, define its structural patterns as $\{d_i, d_j\}$, indicating it has two accessible target nodes with shortest distances of d_i, d_j respectively. For each structural pattern class, we compute its importance weights by averaging the attention weight of background nodes with the corresponding pattern class in GNN.

Exploration on DGraph. We take DGraph [18] as a case study to investigate how the fetched background nodes contribute to the classification. Figure 8 presents the importance weights of different structural patterns maintained by the background nodes. For the top four structural patterns, i.e., {1,1,1}, {1,1,1,1,1}, {1,1} and {1,1,1,1}, the background nodes all act as the 2-hop bridging nodes, revealing the importance of connectivity between target nodes. This observation is also consistent with the exploration results in Section 1. Moreover, the attention scores of these four patterns are quite close, suggesting that the 2-hop bridging nodes with different degrees might play similar roles in the task. These observations can offer a good explanation for the classification task in the financial scenario where the social relations between users are crucial for fraud detection. Concretely, for one node connected with fraud users, the likelihood that it and its direct neighbors are fraudsters will increase significantly. Moreover, if most neighbors of one user are frauds, it is likely to be a fraud intermediary agency.

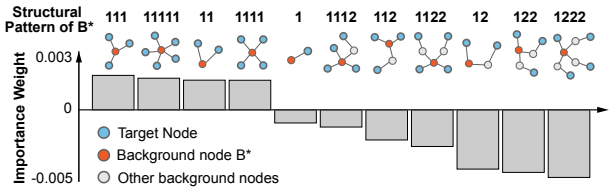

Figure 8: The importance weights of top 10 structural patterns of background nodes in $\mathcal{S}_\alpha$ of DGraph. The corresponding structural patterns are visualized upper the bars.

5 RELATED WORKS

Graph Coarsening. Graph coarsening reduces the number of graph nodes while preserving some important properties in the original graph. The theoretical approximation guarantees on spectrum or structure have been studies in some previous coarsening studies [1, 22, 24, 41]. Specifically, Zhu et al. propose a sparsifier which leverages schur complement construction to approximate the shortest distances between each pair of vertices in terminal set [41]. Nevertheless, these graph reduction methods only focus on the approximation of structure while ignoring the node features, which are not tailored to the node classification tasks in web mining. Huang et al. proposes a coarsening model for semi-supervised node classification, which merges the original nodes into super-nodes along with averaged node features for graph reduction [19].

Graph Condensation. Recently, graph condensation has also been studied. Jin et al. develop a graph condensation method based on gradient matching to imitate the GNNs training trajectory on large graph [21], and further extend the method to one-step gradient matching [20]. Nevertheless, these methods would inevitably lose original target nodes during reduction. For coarsening model [19], it is prone to merge the target nodes into one super-nodes. For the condensation methods [20, 21], they can only compress the target nodes since only the target nodes' labels are effective for gradient matching. However, in our study we aim to shrink the size of background nodes while maintaining all target nodes. Besides, these methods require large memory cost with higher time complexity during graph reduction, yet our method is efficient and effective, which is much easier to implement on very large graphs.

6 CONCLUSION

In this paper, We focus on a common challenge in web graph mining applications: compressing the massive background nodes for a small part of target nodes classification, to tackle the storage and model deployment difficulties on very large graphs. Empirical analysis explicitly reveals the contributions of critical background nodes to the target classification, i.e., enhancing target *structural connectivity* and *feature correlation* with target nodes. With these inspirations, we propose a novel *Graph-Skeleton*, which properly fetches and condenses the informative background nodes, so that the generated graph is small-scale but sufficient to trace, retain the information of each target node for classification. Extensive experiments well indicates the effectiveness of our proposed method.

ACKNOWLEDGMENTS

This work is supported by NSFC (No.62322606).

REFERENCES

[1] Daniele Calandriello, Alessandro Lazaric, Ioannis Koutis, and Michal Valko. 2018. Improved large-scale graph learning through ridge spectral sparsification. In *International Conference on Machine Learning*.

[2] Jinsong Chen, Kaiyuan Gao, Gaichao Li, and Kun He. 2023. NAGphormer: A tokenized graph transformer for node classification in large graphs. In *International Conference on Learning Representations*.

[3] Jie Chen, Tengfei Ma, and Cao Xiao. 2018. Fastgcn: fast learning with graph convolutional networks via importance sampling. In *International Conference on Learning Representations*.

[4] Lei Chen, Zhengdao Chen, and Joan Bruna. 2020. On Graph Neural Networks versus Graph-Augmented MLPs. In *International Conference on Learning Representations*.

[5] Yankai Chen, Yixiang Fang, Yifei Zhang, and Irwin King. 2023. Bipartite Graph Convolutional Hashing for Effective and Efficient Top-N Search in Hamming Space. In *Proceedings of the ACM Web Conference 2023*.

[6] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. 2019. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*.

[7] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. 2022. *Introduction to algorithms*. MIT press.

[8] DataReportal. 2022. FACEBOOK STATISTICS AND TRENDS. https://datareportal.com/essential-facebook-stats.

[9] Fabrizio Frasca, Emanuele Rossi, Davide Eynard, Ben Chamberlain, Michael Bronstein, and Federico Monti. 2020. Sign: Scalable inception graph neural networks. *arXiv preprint arXiv:2004.11198* (2020).

[10] Xinyu Fu, Jiani Zhang, Ziqiao Meng, and Irwin King. 2020. Magnn: Metapath aggregated graph neural network for heterogeneous graph embedding. In *Proceedings of The Web Conference 2020*.

[11] Li Gao, Hong Yang, Chuan Zhou, Jia Wu, Shirui Pan, and Yue Hu. 2018. Active discriminative network representation learning. In *IJCAI International Joint Conference on Artificial Intelligence*.

[12] Johannes Gasteiger, Aleksandar Bojchevski, and Stephan Günnemann. 2018. Predict then Propagate: Graph Neural Networks meet Personalized PageRank. In *International Conference on Learning Representations*.

[13] Siddharth Gollapudi, Neel Karia, Varun Sivashankar, Ravishankar Krishnaswamy, Nikit Begwani, Swapnil Raz, Yiyong Lin, Yin Zhang, Neelam Mahapatro, Premkumar Srinivasan, et al. 2023. Filtered-DiskANN: Graph Algorithms for Approximate Nearest Neighbor Search with Filters. In *Proceedings of the ACM Web Conference 2023*.

[14] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. *Advances in Neural Information Processing Systems* (2017).

[15] Shengding Hu, Zheng Xiong, Meng Qu, Xingdi Yuan, Marc-Alexandre Côté, Zhiyuan Liu, and Jian Tang. 2020. Graph policy network for transferable active learning on graphs. *Advances in Neural Information Processing Systems* (2020).

[16] Weihua Hu, Matthias Fey, Hongyu Ren, Maho Nakata, Yuxiao Dong, and Jure Leskovec. 2021. Ogb-lsc: A large-scale challenge for machine learning on graphs. *arXiv preprint arXiv:2103.09430* (2021).

[17] 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. *Advances in Neural Information Processing Systems* (2020).

[18] Xuanwen Huang, Yang Yang, Yang Wang, Chunping Wang, Zhisheng Zhang, Jiarong Xu, and Lei Chen. 2022. DGraph: A Large-Scale Financial Dataset for Graph Anomaly Detection. *Advances in Neural Information Processing Systems* (2022).

[19] Zengfeng Huang, Shengzhong Zhang, Chong Xi, Tang Liu, and Min Zhou. 2021. Scaling up graph neural networks via graph coarsening. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*.

[20] Wei Jin, Xianfeng Tang, Haoming Jiang, Zheng Li, Danqing Zhang, Jiliang Tang, and Bing Yin. 2022. Condensing graphs via one-step gradient matching. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*.

[21] Wei Jin, Lingxiao Zhao, Shichang Zhang, Yozen Liu, Jiliang Tang, and Neil Shah. 2022. Graph Condensation for Graph Neural Networks. In *International Conference on Learning Representations*.

[22] Yu Jin, Andreas Loukas, and Joseph JaJa. 2020. Graph coarsening with preserved spectral properties. In *International Conference on Artificial Intelligence and Statistics*.

[23] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *International Conference on Learning Representations* (2016).

[24] Huan Li and Aaron Schild. 2018. Spectral subspace sparsification. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science*.

[25] Yifan Mao, Soubhik Deb, Shaileshh Bojja Venkatakrishnan, Sreeram Kannan, and Kannan Srinivasan. 2020. Perigee: Efficient peer-to-peer network design for blockchains. In *Proceedings of the 39th Symposium on Principles of Distributed Computing*. 428–437.

[26] Yifan Mao and Shaileshh Bojja Venkatakrishnan. 2023. Less is more: Understanding network bias in proof-of-work blockchains. *Mathematics* 11, 23 (2023), 4741.

[27] Yifan Mao and Shaileshh Bojja Venkatakrishnan. 2023. Topiary: Fast, Scalable Publish/Subscribe for Peer-to-Peer (D) Apps. *arXiv preprint arXiv:2312.06800* (2023).

[28] Liang Qu, Ningzhi Tang, Ruiqi Zheng, Quoc Viet Hung Nguyen, Zi Huang, Yuhui Shi, and Hongzhi Yin. 2023. Semi-decentralized Federated Ego Graph Learning for Recommendation. In *Proceedings of the ACM Web Conference 2023*.

[29] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. 2018. Modeling relational data with graph convolutional networks. In *European semantic web conference*.

[30] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. In *International Conference on Learning Representations*.

[31] Yu Wang, Yuying Zhao, Yi Zhang, and Tyler Derr. 2023. Collaboration-Aware Graph Convolutional Network for Recommender Systems. In *Proceedings of the ACM Web Conference 2023*.

[32] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. 2019. Simplifying graph convolutional networks. In *International Conference on Machine Learning*.

[33] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2018. How Powerful are Graph Neural Networks?. In *International Conference on Learning Representations*.

[34] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. 2019. Graphsaint: Graph sampling based inductive learning method. *arXiv preprint arXiv:1907.04931* (2019).

[35] Shichang Zhang, Yozen Liu, Yizhou Sun, and Neil Shah. 2021. Graph-less Neural Networks: Teaching Old MLPs New Tricks Via Distillation. In *International Conference on Learning Representations*.

[36] Wentao Zhang, Zheyu Lin, Yu Shen, Yang Li, Zhi Yang, and Bin Cui. 2022. Deep and Flexible Graph Neural Architecture Search. In *Proceedings of the 39th International Conference on Machine Learning*.

[37] Wentao Zhang, Zeang Sheng, Ziqi Yin, Yuezihan Jiang, Yikuan Xia, Jun Gao, Zhi Yang, and Bin Cui. 2022. Model Degradation Hinders Deep Graph Neural Networks. In *Proceedings of the 28th ACM SIGKDD international conference on knowledge discovery & data mining*.

[38] Ziwei Zhang, Peng Cui, and Wenwu Zhu. 2020. Deep learning on graphs: A survey. *IEEE Transactions on Knowledge and Data Engineering* (2020).

[39] Xiaotian Zhou and Zhongzhi Zhang. 2023. Opinion Maximization in Social Networks via Leader Selection. In *Proceedings of the ACM Web Conference 2023*.

[40] Zhilun Zhou, Yu Liu, Jingtao Ding, Depeng Jin, and Yong Li. 2023. Hierarchical knowledge graph learning enabled socioeconomic indicator prediction in location-based social network. In *Proceedings of the ACM Web Conference 2023*.

[41] Chunjiang Zhu, Qinqing Liu, and Jinbo Bi. 2021. Spectral vertex sparsifiers and pair-wise spanners over distributed graphs. In *International Conference on Machine Learning*.

[42] Difan Zou, Ziniu Hu, Yewen Wang, Song Jiang, Yizhou Sun, and Quanquan Gu. 2019. Layer-dependent importance sampling for training deep and large graph convolutional networks. *Advances in Neural Information Processing Systems* (2019).

A DETAILED EXPERIMENTAL SETTINGS

A.1 Experimental Settings in Section 2

The GNNs we choose includes GraphSAGE [14], GAT [30] and GIN [33], which utilized three representative aggregation mechanisms (mean, weight-based, summation), as the backbone model, and deploy the model on two datasets for target nodes classification:

- Financial loan network DGraph, where we follow the task setting of the original dataset [18], i.e., fraudster identification among loan users. In this case the users with loan action are regarded as target nodes (~33%), while others are viewed as background nodes.
- Academic citation network ogbn-arxiv [17], where we aim to predict the subject areas of papers published since 2018. In this case, papers published from 2018 are regarded as target nodes (~46%), while papers before 2018 as the background nodes.

To analyze the contributions of background nodes to target nodes, we cut the edges between different types of nodes and deploy the cutted graphs on the GraphSAGE model. Specifically, we cut the edges within the original graph including: (1) Random Cut: randomly drop the edges within the graph with the edge drop rate spanning from 0 to 1. (2) B-B Cut: drop the edges between background nodes. (3) T-B Cut: drop the edges between target and background nodes. (4) BridB Cut: drop the edges where background nodes act as the 1-hop bridging nodes between two target nodes.

A.2 Experimental Settings in Section 4

A.2.1 Graph-Skeleton Settings in Fixed BCR Experiments. For the evaluation of proposed Gpaph-Skeleton, we first compare the performance of baselines in a fixed *BCR* manner in Section 4.1, Table 2. Specifically, we choose the Gpaph-Skeleton using condensation strategy-γ (Skeleton-γ in short, with highest compression rate) to compare with other baselines. During node fetching phase (Section 3.1), we set the fetching depth and width as $d_1 = 2, d_2 = 1, K = 5$ to generate the vanilla subgraph $\mathcal{G}^{'}$. For γ condensation, we choose mean operation as aggregation function in Equation 5. For baselines of Random, Centrality Ranking and GPA, we control the query budgets of background node selection to keep the *BCR* being same with that of skeleton graph $\mathcal{S}_\gamma$. For Graph Coarsening, GCond and DosCond, we control the compression rate to keep the synthetic graphs having the same total size (# Total after compression) with skeleton graph $\mathcal{S}_\gamma$. These synthetic graphs are considered to have the same *BCR* for comparison.

A.2.2 Downstream Classification Settings in MAG240M and ogbn-mag. For MAG240M and ogbn-mag, graphs contain various edge types (7 types in original dataset, which will be utilized for classification). Due to graph condensation would condense different nodes together and cause type inconsistency of generated edge, we re-define the edge type via the target and background nodes connection in original and compressed graphs (4 types in total: target-target, background-background, target-background, background-target) and implement node classification test.

A.2.3 Graph Models Settings. In our study, we evaluate target node classification performance in downstream tasks with various GNN models. Our experiments are implemented with PyTorch 1.10.0, CUDA v12.1 on NVIDIA Quadro RTX 6000 GPU. Each experiment is repeated for 10 trials on all datasets except MAG240M.

A.2.4 Varied Background Node Compression Rate Control. To obtain the synthetic graphs with different compressed rates, we can use different condensation strategies of Graph-Skeleton with varied fetching depth (d_1, d_2) for background nodes compression. Specifically, the fetching depth (d_1, d_2) controls the number of fetched background nodes in the vanilla subgraph $\mathcal{G}^{'}$. Increasing the fetching depth results in a more comprehensive collection of information, but it also leads to a larger size of the vanilla subgraph. On the other hand, three condensation strategies control the level of condensation of redundant information in vanilla subgraph $\mathcal{G}^{'}$, where Strategy-α preserves the greatest amount of the original structural and semantic information, while strategy-γ delivers the lowest *BCR*.

A.2.5 Settings of Three Strategies Analyses. To investigate the condensation performance of three condensation strategies, we conduct the experiments and analyze the results in Section 4.3. During compression, we keep the fetching depths being same ($[d_1, d_2]$ = [2,1]) for vanilla subgraph generation and use three strategies for condensation to generate the synthetic skeleton graphs respectively.

B PROOF OF OF PROPOSITION 1

Before the proof, let's revisit the definition of Linear message path passing on a single path (Section 3.2, Definition 3.3), then we extend the Linear message path passing function to multiple paths.

Definition B.1 (Linear message path passing). Given a path $p = \langle u_0, u_1, ..., u_\ell \rangle$, we define the linear single path passing functions $f_{spp}(X, p)$ which aggregates node feature starting from u_0 to u_ℓ over p as:

$$X^{'}[u_i] \leftarrow f^i_{lmp}(X^{'}[u_{i-1}], X[u_i])W^i, i \in 1, .., \ell, \tag{6}$$

where X is the node feature matrix, $f_{lmp}(u, v)$ denotes the linear message passing operation (defined in equation 2) from u to v, W is a transformation matrix.

Given m paths with same end node K: P:=$\{p_{u^0_0,K} = \langle u^1_0, u^1_1, ..., u^1_*, K\rangle$, $p_{u^1_0,K} = \langle u^2_0, u^2_1, ..., u^1_*, K\rangle$,...,$p_{u^m_0,K} = \langle u^m_0, u^m_1, ..., u^m_*, K\rangle\}$, define the linear message path passing $f_{lmpp}(X, P)$ over m paths to end node K as:

$$\begin{aligned} X^{'}[K] &\leftarrow \text{AGGREGATE}(X[K], \{X^{'}[u^i_*], \forall i \in 1, ..., m\})W^K \\ &= \text{AGGREGATE}(X[K], \{f_{spp}(X, p_{u^i_0,u^i_*})[u^i_*], \forall i \in 1, ..., m\})W^K \end{aligned} \tag{7}$$

Proof. Let X denotes node feature matrix, $f_{spp} = f^m_{lmp} \circ ... \circ f^2_{lmp} \circ f^1_{lmp}$ denotes a single path passing function, where f_{lmp} is the linear message passing operation. Given $u, v \in \mathcal{B}$, s.t., $MSS_u = MSS_v = \{\langle \mathcal{T}, \mathcal{D} \rangle\}$. Pick one target $T \in \mathcal{T}^{'}$ with the corresponding distance d, and two shortest paths $p_{u,T} = \langle u, u_1, ..., u_*, T\rangle$ and $p_{v,T} = \langle v, v_1, ..., v_*, T\rangle$ from u,v to T respectively, where $|p_{u,T}| = |p_{v,T}| = d$. Assuming $\{p_{u,u_*}\} \cap \{p_{v,v_*}\} = \emptyset$, let $P := \{p_{u,T}, p_{v,T}\}$, the linear message path passing over paths to T is formulated as:

$$X^{'}[T] = f_{lmpp}(X, P)[T] = \text{AGGREGATE}(X[T], \{X^{'}[u_*], X^{'}[v_*]\})W^T \tag{8}$$

$$X^{'}[u_*] = f_{spp}(X, p_{u,u_*})[u_*] = f_{lmp}^{d-2}(...f_{lmp}^{1}(X[u], X[u_1])W^1..., X[u_*])W^{d-2} \tag{9}$$

$$X^{'}[v_*] = f_{spp}(X, p_{v,v_*})[v_*] = f_{lmp}^{d-2}(...f_{lmp}^{1}(X[v], X[v_1])W^1..., X[v_*])W^{d-2} \tag{10}$$

Since f_{lmp} is linear, the aggregated feature $X^{'}[u_*]$ and $X^{'}[v_*]$ can be decoupled as:

$$X^{'}[u_*] = \frac{X[u]\cdot W^1\cdot ...\cdot W^{d-2}}{c^{d-2}} + \frac{X[u_1]\cdot W^1\cdot ...\cdot W^{d-2}}{c^{d-2}} + \frac{X[u_2]\cdot W^2\cdot ...\cdot W^{d-2}}{c^{d-3}} + ... + \frac{X[u_*]\cdot W^{d-2}}{c} \tag{11}$$

$$X^{'}[v_*] = \frac{X[v]\cdot W^1\cdot ...\cdot W^{d-2}}{c^{d-2}} + \frac{X[v_1]\cdot W^1\cdot ...\cdot W^{d-2}}{c^{d-2}} + \frac{X[v_2]\cdot W^2\cdot ...\cdot W^{d-2}}{c^{d-3}} + ... + \frac{X[v_*]\cdot W^{d-2}}{c} \tag{12}$$

where c is the division index, $c = 2$ for mean pooling aggregation in Eq.(2), for summation pooling aggregation $c = 1$. Combining Eq.(8) and Eq.(11),(12) we have

$$\begin{aligned} X^{'}[T] &= \text{AGGREGATE}(X[T], X^{'}[u_*], X^{'}[v_*])W^T \\ &= \frac{(X[u]+X[v])\cdot W^1\cdot ...\cdot W^{d-2}\cdot W^T}{c^{d-2}c^{\dagger}} \\ &+ \frac{(X[u_1]+X[v_1])\cdot W^1\cdot ...\cdot W^{d-2}\cdot W^T}{c^{d-2}c^{\dagger}} \\ &+ \frac{(X[u_2]+X[v_2])\cdot W^2\cdot ...\cdot W^{d-2}\cdot W^T}{c^{d-3}c^{\dagger}} \\ &+ ... \\ &+ \frac{(X[u_*]+X[v_*])\cdot W^{d-2}\cdot W^T}{cc^{\dagger}} \\ &+ \frac{(X[T])\cdot W^T}{c^{\dagger}} \end{aligned} \tag{13}$$

where $c^{\dagger} = 3$ for mean pooling aggregation in Eq.(8), $c^{\dagger} = 1$ for summation pooling aggregation. Similarly, for $\tilde{X}$ where $\tilde{X} = X$ except $\tilde{X}[u] = X[v], \tilde{X}[v] = X[u]$, we have

$$X^{'}[T] = \tilde{X}^{'}[T] \Leftrightarrow u \simeq_{\mathcal{LMPP},T} v \tag{14}$$

□

This indicates that the aggregated information via linear message path passing is only related to the path length, but not the other nodes on the path. The proposition also holds when $\{p_{u,u_*}\} \cap \{p_{v,v_*}\} \neq \emptyset$ and each f_{lmp}^{m} in $f_{spp} = f_{lmp}^{m} \circ ... \circ f_{lmp}^{2} \circ f_{lmp}^{1}$ employs different aggregation, we omit the proof since it is similar.

Algorithm 1: Bridging Background Node Fetching

Input: Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, target nodes $\mathcal{T} = (T_1, T_2, ..., T_n)$, d_1
Output: The bridging node set *Brid*

```
1  Initialize a queue Q with all target nodes T;
2  Brid ← ∅;
3  while Q is not empty do
4      u ← Q.dequeue();
5      foreach v is a neighbor of u do
6          if edge (u, v) has been accessed less than twice then
7              Accese edge (u, v);
               // Note that the shortest and
                  2nd-shortest path are maintained in
                  the update process so that they
                  start from different target nodes.
8              Update the shortest and 2nd-shortest path of v
                by u;
9              Q.enqueue(v);
10 foreach u ∈ V/T do
11     if u's length of shortest path + 2nd-shortest path ≤ d1
        then
12         Brid ← Brid ∪ u
13 return Brid
```

C ALGORITHMS IMPLEMENTATION DETAILS

C.1 Background Node Fetching

Bridging Background Node. We show the detailed process of bridging background node fetching in Algorithms 1. It searches for all accessible nodes starting from the target nodes and update their corresponding shortest and 2nd-shortest paths to target nodes (Line 3-9). Then we select the background nodes under our proposed fetching principle 1 of distance d_1 in Section 3.1 as the bridging background nodes *Brid* (Line 10-12).

Affiliation Background Node. Affiliation background node fetching process is shown in Algorithms 2. It aims to identify the bridging background nodes within the input graph $\mathcal{G}$ given the corresponding target node set $\mathcal{T}$, and fetching depth d_2 and width K.

C.2 Graph Condensation

We show the detailed algorithm of graph condensation. The condensation strategy-α and strategy-β are shown in Algorithms 3 and 4. For condensation strategy-γ, we omit the detailed pseudocode since it is similar to strategy-β but with the last step of merging the affiliation background nodes to their corresponding target nodes using Equation 5.

D TIME COMPLEXITY ANALYSIS

Our method is divided into two parts, node fetching and graph condensation. For the first part, we drop the node where the length of the shortest path is greater than d_2 or the length of the shortest path + the second short path is greater than d_1. This can be achieved

Algorithm 2: Affiliation Background Node Fetching

Input: Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, feature matrix X, target nodes $\mathcal{T} = (T_1, T_2, ..., T_n)$, d_2, K
Output: The affiliation node set $Affil$

Initialize an queue Q with all target nodes $\mathcal{T}$;
Initialize the an array dis with ∞ for background nodes and 0 for target nodes;
$Affil \leftarrow \emptyset$;
$Affil^{'} \leftarrow \emptyset$;
while *Q is not empty* **do**
  $u \leftarrow Q$.dequeue();
  foreach *v is a neighbor of u* **do**
    if *$dis[v]$ is ∞* **then**
      $dis[v] \leftarrow dis[u] + 1$;
      Q.enqueue(v);
foreach $u \in \mathbf{V}/\mathcal{T}$ **do**
  if $dis[u] \leq d_2$ **then**
    $Affil^{'} \leftarrow Affil^{'} \cup u$
foreach $u \in \mathcal{T}$ **do**
  $pcc \leftarrow \emptyset$;
  $affil \leftarrow \emptyset$;
  foreach *v is a neighbor of u* **do**
    if $v \in Affil^{'}$ **then**
      $affil \leftarrow affil \cup v$;
      $pcc \leftarrow pcc \cup coorelation(X_u, X_v)$;
      sort $affil$ according to pcc;
      $Affil \leftarrow Affil \cup affil[1:K]$;
return $Affil$

Algorithm 3: Condensation-α

Input: Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, target nodes $\mathcal{T} = (T_1, T_2, ..., T_n)$, Affiliation and Bridging node set $Affil, Brid$, ego-network's hop k
Output: Condensed skeleton graph $\mathcal{S}_\alpha$

Set $mask[u]$ to 0 for every node u;
foreach *target node $u \in \mathcal{T}$* **do**
  for $d \leftarrow 1$ *to* k **do**
    $key[u, d] \leftarrow$ an unique random 256-bit integer;
  foreach *node v in u's k-hop ego-network* ***and*** $v \in Affli \cup Brid$ **do**
    Let d be the distance between u and v;
    `// ⊙ denote Binary Exclusive Or(xor).`
    $mask[v] \leftarrow mask[v] \odot key[u, d]$;
Sort the node by node's *mask* with Radix Sort;
Merge nodes with the same mask into one node and get $\mathcal{S}_\alpha$;
return $\mathcal{S}_\alpha$

Algorithm 4: Condensation-β

Input: Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, target nodes $\mathcal{T} = (T_1, T_2, ..., T_n)$, Affiliation and Bridging node set $Affil, Brid$, ego-network's hop k
Output: Condensed skeleton graph $\mathcal{S}_\beta$

Assign each target node u with an unique random 256-bit integer $key[u]$;
Set $mask[u]$ to 0 for every node u;
foreach *target node $u \in \mathcal{T}$* **do**
  foreach *node v in u's k-hop ego-network* ***with*** $v \in Affli \cup Brid$ **do**
    `// ⊙ denote Binary Exclusive Or(xor).`
    $mask[v] \leftarrow mask[v] \odot key[u]$;
Sort the node by node's *mask* with Radix Sort;
Merge nodes with the same mask into one node;
Update edge weights by original distance information of merged nodes and get $\mathcal{S}_\beta$;
return $\mathcal{S}_\beta$

through a multi-source shortest path problem (all target nodes as the source). In this algorithm, each node will be accessed at most twice, each node will be enqueued at most twice, and their adjacent edges will be enumerated twice. And the time complexity of this step is $O(|E|)$. Specifically, to find shortest paths in an unweighted graphs via BFS, the complexity is $O(|E| + |V|) = O(|E|$ (assuming no isolated nodes, $|E| > |V|$) [7]. For weighted graphs, we use the same algorithm. Please note that we aim to find the minimum hop paths while not the minimum weight paths in weighted graphs, so the complexity is also $O(|E|)$. After that, the nodes in the k-order neighborhood of each target node are accessed to calculate PCC. Let the average number of edges in each background node's k-hop ego-network be $e(k)$, then the time complexity of the above steps is $O(e(k)|V|)$. The total time complexity of the first part is $O(|E| + e(k)|V|)$.

For the second part, we propose three graph condensation methods ($-\alpha$, $-\beta$ and $-\gamma$ respectively). All these methods need to calculate the distance between each background node and the target nodes in its k-hop ego-network. We use the hash method to merge similar nodes, so the subsequent merging steps can be completed with $O(1)$ for each node. This makes the time complexity of the whole process determined by the complexity of previous distance calculating, that is, $O(e(k)|V|)$

The first two methods (α, β) end after merging nodes with the same hash value, while the last method(γ) requires an additional step. For Graph Condensation-γ, merging of affiliation nodes for each target node only requires traversing the neighbors of these target nodes, which means that the time complexity of this additional step is $O(|E|)$. Therefore, the time complexity of these three graph condensation methods for the second part is $O(|E| + e(k)|V|)$. To sum up, the total time complexity is $O(|E| + e(k)|V|)$.