machine learning & knowledge extraction

MDPI

Article

Node-Centric Pruning: A Novel Graph Reduction Approach

Hossein Shokouhinejad *, Roozbeh Razavi-Far, Griffin Higgins and Ali A. Ghorbani

Canadian Institute for Cybersecurity, University of New Brunswick, Fredericton, NB E3B 5A3, Canada; roozbeh.razavi-far@unb.ca (R.R.-F.); griffin.higgins@unb.ca (G.H.); ghorbani@unb.ca (A.A.G.)
* Correspondence: hossein.shokouhinejad@unb.ca

Abstract: In the era of rapidly expanding graph-based applications, efficiently managing large-scale graphs has become a critical challenge. This paper introduces an innovative graph reduction technique, Node-Centric Pruning (NCP), designed to simplify complex graphs while preserving their essential structural properties, thereby enhancing the scalability and maintaining performance of downstream Graph Neural Networks (GNNs). Our proposed approach strategically prunes less significant nodes and refines the graph structure, ensuring that critical topological properties are maintained. By carefully evaluating node significance based on advanced connectivity metrics, our method preserves the topology and ensures high performance in downstream machine learning tasks. Extensive experimentation demonstrates that our proposed method not only maintains the integrity and functionality of the original graph but also significantly improves the computational efficiency and preserves the classification performance of GNNs. These enhancements in computational efficiency and resource management make our technique particularly valuable for deploying GNNs in real-world applications, where handling large, complex datasets effectively is crucial. This advancement represents a significant step toward making GNNs more practical and effective for a wide range of applications in both industry and academia.

Keywords: graphneural networks (GNNs); graph reduction; node-centric pruning (NCP); topology preservation

Citation: Shokouhinejad, H.; Razavi-Far, R.; Higgins, G.; Ghorbani, A.A. Node-Centric Pruning: A Novel Graph Reduction Approach. *Mach. Learn. Knowl. Extr.* **2024**, *6*, 2722–2737. https://doi.org/10.3390/make6040130

Academic Editor: Massimo Ferri

Received: 26 October 2024
Revised: 14 November 2024
Accepted: 20 November 2024
Published: 22 November 2024

1. Introduction

A graph is a fundamental data structure in mathematics and computer science, consisting of nodes (vertices) connected by edges (links). Graphs are used to model relationships and interactions in various fields such as computer science, biology, social sciences, and engineering. In social networks, graphs help to analyze relationships and identify influential nodes. In biology, they model interactions between biomolecules, aiding in understanding diseases. In engineering, graphs can optimize network design and improve efficiency in communication and transportation systems. Their importance lies in their ability to abstract and simplify complex systems, making them essential for both theoretical research and practical applications.

Graph reduction refers to the process of simplifying a graph, while preserving its essential properties and structural characteristics. This technique is crucial for managing the complexity of large-scale graphs, making them more tractable for analysis and computation. In practical applications, graph reduction is often necessary in order to improve the efficiency, reduce computational resources, and enhance the scalability of solutions. For instance, in network analysis, reduced graphs can facilitate faster detection of critical nodes or bottlenecks, enabling timely and effective interventions. In the context of machine learning, graph reduction can help in preprocessing of data, where redundant or less significant nodes and edges are eliminated, thus, streamlining the learning process and improving the model performance. By employing reduction techniques, researchers and practitioners can ensure that graph-based analyses remain feasible, accurate, and applicable to real-world

applications. There are mainly three graph reduction methods: graph coarsening, graph condensation, and graph sparsification [1].

Graph coarsening reduces the graph by merging nodes and edges, forming a smaller graph that approximates the original structure. This method often involves creating supernodes that represent clusters of nodes from the original graph, making it useful for multilevel algorithms [2–7]. The authors in [5] investigate the problem of reducing the size of a graph, while preserving its essential properties using a leveraging graph coarsening approach. The study introduces a new perspective through restricted spectral approximation and derives sufficient conditions for a smaller graph to approximate a larger one and proposes nearly linear time algorithms for coarsening that improve the quality of the reduced graphs. The paper [6] addresses GNN scalability by reducing graph size through coarsening. This approach, which simplifies training and cuts memory costs, also acts as a form of regularization, potentially enhancing model generalization. Furthermore, the authors in [7] introduce an optimization-based framework for graph coarsening that takes into account both the graph structure and node features. The proposed framework jointly learns a coarsened graph matrix and feature matrix, while ensuring the coarsened graph maintains a similarity to the original. This method guarantees that the coarsened graph is ϵ-similar to the original graph, ensuring the preservation of key properties. While graph coarsening can reduce the size of the original graph and potentially speed up computations, fine-tuning the coarsening process to balance between computational efficiency and accuracy can be difficult. It often requires domain-specific knowledge and extensive experimentation, which can be computationally costly. Additionally, although a coarsened graph is smaller and can result in faster subsequent computations, the graph coarsening process itself might be too time-consuming for certain algorithms. In such cases, the overhead of coarsening may surpass its advantages.

Graph condensation aims to synthesize a smaller yet highly representative graph, enabling GNNs to achieve performance levels comparable to those attained when trained on the larger original graph [8–20]. The paper [16] presents a graph condensation method called CrafTing RationaL trajectory (CTRL). This method addresses the high cost and storage concerns associated with training on large-scale graphs. The CTRL method improves upon traditional graph condensation techniques by optimizing both the gradient direction and magnitude, thereby minimizing accumulated errors and enhancing the performance of the condensed graphs. In addition, Reference [17] introduced a method to address inefficiencies in existing graph condensation techniques, particularly when dealing with large-scale graph datasets like web data. The authors identify two main inefficiencies: the concurrent updating of a vast parameter set and pronounced parameter redundancy. To mitigate these issues, they propose an Efficient and eXplainable Graph Condensation (EXGC) method. EXGC employs the Mean-Field variational approximation to accelerate convergence and integrates leading graph explanation techniques, such as GNNExplainer and Graph Stochastic Attention (GSAT), to remove redundant parameters and enhance efficiency. The paper [18] also proposes a graph condensation framework that addresses the computational inefficiencies of existing methods. By reformulating graph condensation as a Kernel Ridge Regression (KRR) task and introducing a Structure-based Neural Tangent Kernel (SNTK) to capture graph topologies, the proposed method achieves efficient graph condensation. Furthermore, the work [19] introduces a method called DisCo for graph condensation. DisCo separates the condensation process for nodes and edges, addressing the scalability issues of existing methods. This disentangled approach reduces GPU memory requirements and enhances the ability to condense large-scale graphs. The paper [20] proposes a method called Graph Condensation via Expanding Window Matching (GEOM). This method aims to achieve lossless graph condensation by employing a curriculum learning strategy to gather diverse supervision signals from the original graph. GEOM uses expanding window matching to transfer rich information efficiently and introduces a new loss function to extract knowledge from expert trajectories. While graph condensation offers a promising method for managing large graphs, its effectiveness can be highly sensitive to various

parameters and hyperparameters, posing challenges in identifying the optimal settings. Even slight adjustments in parameters can lead to substantial variations in the resulting condensed graph and the performance of GNN. Additionally, the condensation process is often computationally intensive and complex. Creating an effective condensation algorithm that retains crucial information, while minimizing computational demands, is a significant challenge. Furthermore, condensation can diminish the model's and the graph's interpretability, making it harder to discern underlying patterns and relationships. Important nodes and edges may be merged or omitted, reducing the graph's structural clarity.

Graph sparsification, on the other hand, involves reducing the number of edges or nodes in the graph, typically by removing edges or nodes with the least significance or by using algorithms that retain a sparsified version of the original graph that approximates certain properties, such as spectral properties or connectivity [21–25]. Reference [21] introduces a method for sparsification of weighted graphs. A t-spanner is a subgraph, where distances between nodes are at most t times the original graph's distances. The authors present a simple polynomial-time algorithm to construct these sparse spanners, which reduces both the number of edges and total edge weight, while preserving approximate distances. The authors in [23,25] proposed sparsification techniques intended to preserve the performance of downstream tasks when using GNNs for graph embeddings in the reduced graph. The study [23] introduces a Unified GNN Sparsification (UGS) framework that prunes both the graph adjacency matrix and the model weights simultaneously. By extending the Lottery Ticket Hypothesis (LTH) to GNNs, the paper demonstrates that it is possible to identify highly sparse and independently trainable sub-networks that match the performance of the original dense models. Additionally, the authors in [25] introduced the concept of separation rank to quantify these interactions and revealed that the ability of GNNs to model interactions is primarily determined by the partition's walk index, which is a graph-theoretical characteristic defined by the number of walks originating from the boundary of the partition. The paper also presents an edge sparsification algorithm named Walk Index Sparsification (WIS), which aims to preserve the expressive power of GNNs, while removing edges. In general, graph sparsification is often less computationally expensive compared to graph coarsening or graph condensation. Nonetheless, UGS and WIS are intricate algorithms, and their implementation for large graphs can become computationally expensive and time-consuming.

In this paper, we introduce a novel two-step graph sparsification technique named Node-Centric Pruning (NCP), specifically designed to maintain the performance of graph classification. The first step of NCP focuses on pruning nodes with minimal connections, effectively removing non-essential elements that do not significantly influence the overall graph structure. In the second step, we employ a Jaccard similarity-based algorithm to further refine the graph. This method meticulously preserves the core topology by selectively maintaining nodes that contribute meaningfully to the graph's structure. Both steps of NCP are straightforward to implement, computationally efficient, and quick to execute, making it a practical choice for real-world applications. Our experiments demonstrate that NCP outperforms WIS, particularly in enhancing the accuracy and efficiency of GNNs in graph classification tasks.

The main contributions of this study are as follows:

- We propose a novel, two-step graph sparsification technique that effectively balances graph reduction with classification performance preservation for GNNs.
- The proposed method is computationally efficient and simple to implement, offering a practical alternative to more complex methods like WIS, which can be resource-intensive for large graphs.
- Experimental results demonstrate that our approach enhances both the accuracy and efficiency of GNNs in graph classification tasks compared to WIS.

The remainder of this paper is organized as follows: Section 2 provides the necessary preliminaries and notations, Section 3 details the proposed NCP algorithm, Section 4 presents experimental results comparing NCP with the state-of-the-art method, WIS, on

standard datasets, and Section 5 concludes with a discussion on the implications of our findings and potential areas for future research.

2. Preliminaries

2.1. Basic Notations

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ represent a graph, where $\mathcal{V} = \{v_1, \ldots, v_n\}$ denotes the set of nodes, and $\mathcal{E} \subseteq \{(v_i, v_j) \mid v_i, v_j \in \mathcal{V}\}$ represents the set of edges. The reduced graph is denoted by $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$.

2.2. Jaccard Similarity

In the context of graph theory, the Jaccard similarity between two nodes v_i and v_j is a measure of the similarity between their sets of neighbors. It is defined as the size of the intersection of the neighbors of v_i and v_j divided by the size of their union.

Let $\mathcal{N}(v_i)$ and $\mathcal{N}(v_j)$ denote the sets of neighbors of nodes v_i and v_j, respectively. The Jaccard similarity $\mathcal{J}(v_1, v_2)$ is given by

$$\mathcal{J}(v_i, v_j) = \frac{|\mathcal{N}(v_i) \cap \mathcal{N}(v_j)|}{|\mathcal{N}(v_i) \cup \mathcal{N}(v_j)|} \tag{1}$$

where:

- $\mathcal{N}(v_i) \cap \mathcal{N}(v_j)$ represents the set of common neighbors of v_i and v_j;
- $\mathcal{N}(v_i) \cup \mathcal{N}(v_j)$ represents the set of all unique neighbors of v_i and v_j.

The Jaccard similarity $\mathcal{J}(v_i, v_j)$ ranges from 0 to 1, where 0 indicates no shared neighbors and 1 indicates identical sets of neighbors.

2.3. Graph Neural Networks

Graph Neural Networks (GNNs) are a powerful class of neural networks designed to operate on graph-structured data. They exploit the inherent relationships and dependencies within the graph to perform tasks such as node classification, link prediction, and graph clustering [26]. The core mechanism of GNNs involves iteratively updating node representations through a series of message-passing layers, where each layer aggregates information from a node's neighbors. This iterative process allows GNNs to capture both local and global graph structures, depending on the number of layers, which is commonly referred to as the network's depth [27].

The depth of a GNN, denoted as $\mathcal{D}$, which corresponds to the number of message-passing layers, is a critical parameter as it dictates the extent of information propagation across the graph. Let h_i^l represent the node embeddings for node i at layer l. Each GNN layer takes as input the node embeddings. The node representations for node i at each layer $l + 1$ are updated using the following formula:

$$h_i^{(l+1)} = f\left(h_i^l, \sum_{j \in \mathcal{N}(i)} g(i, j)\right) \tag{2}$$

where f and g are learnable functions and $\mathcal{N}(i)$ are the neighbors of node i. The depth of a GNN directly influences the receptive field of each node, effectively determining the number of hops or neighbors from which information can be aggregated.

3. Method

Efficient graph reduction techniques are essential for enhancing the scalability and learning performance of GNNs by simplifying complex network structures. Our proposed NCP algorithm strategically reduces graph complexity while ensuring topology preservation, meaning the retention of the graph's essential structural characteristics and core connectivity patterns. This preservation is achieved by applying advanced connectivity metrics to assess each node's significance within the network. Specifically, in NCP, these

metrics involve analyzing walks of a specified length, $\mathcal{L}$, originating from each node. These walks measure reachability by counting the unique nodes accessible within this range, which indicates each node's connectivity and influence. Nodes with high reachability counts demonstrate significant connectivity and play central roles in maintaining the graph's core pathways.

NCP further distinguishes between more and less significant nodes based on their connectivity and role within the overall structure. Nodes deemed less significant contribute minimally to the graph's primary structure and are thus removed, while more significant nodes—those with strong connections and higher structural relevance—are retained. By systematically identifying and removing less critical nodes, NCP reduces the graph's size and complexity, ensuring that the pruned structure remains highly representative of the original. This balanced reduction process creates a streamlined graph suited for efficient GNN processing without sacrificing essential structural information.

The NCP algorithm is based on certain assumptions regarding the nature of the graph data. Primarily, NCP is designed for graphs that contain distinguishable connectivity patterns or structural hubs, where key nodes demonstrate significant reachability within a defined neighborhood. Additionally, the algorithm assumes that the graph data are large and complex enough for node pruning to yield computational benefits without compromising essential structural features.

3.1. Node Categorization and Sparse Node Removal

The first step of the algorithm focuses on identifying nodes that are critical to the graph's structure, termed as Nexus Nodes. This is achieved by exhaustively analyzing all possible walks of a fixed length, denoted by $\mathcal{L}$, starting from each node in the graph. Nodes that do not meet the criteria to be categorized as Nexus Nodes are further examined to determine whether they should be retained as Connector Nodes or removed as Sparse Nodes, based on their direct connections to Nexus Nodes. As shown in Figure 1b, after identifying and categorizing nodes, Sparse Nodes that do not significantly contribute to the graph's structure are removed.

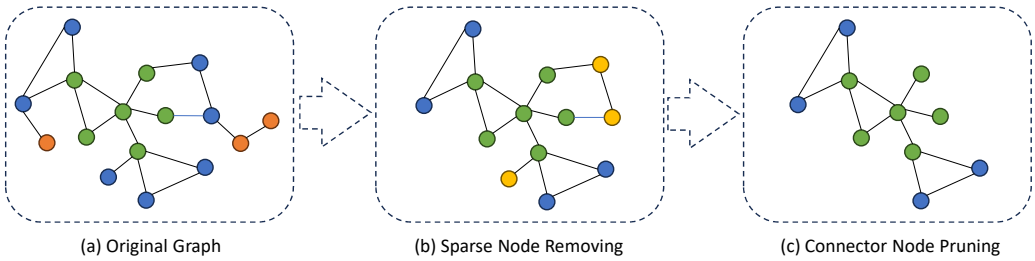

Figure 1. Example of graph reduction using the NCP algorithm: (**a**) Original graph. (**b**) Removal of Sparse Nodes with $\mathcal{L} = 2$ and $\beta = 8$. (**c**) Pruning of Connector Nodes with $\theta = 0.15$.

To categorize the nodes, the NCP calculates the connectivity level of each node v by considering all possible walks originating from v. These walks are systematically generated to include every possible path of length $\mathcal{L}$. The collection of such walks is represented as $\text{Walks}(v, \mathcal{L})$, which can be mathematically defined as

$$\text{Walks}(v, \mathcal{L}) = \{w \mid |w| = \mathcal{L}, w_0 = v\} \tag{3}$$

where:

- w represents a specific walk within the graph.
- $|w| = \mathcal{L}$ indicates that the length of the walk w is exactly $\mathcal{L}$ steps.
- $w_0 = v$ specifies that the starting point of the walk w is the node v.

From these generated walks, the algorithm extracts the set of unique nodes encountered, referred to as $\text{NodeSet}(v)$. This set encompasses all nodes that appear in any walk originating from v and provides a measure of the node's reachability within the graph:

$$\text{NodeSet}(v) = \{u \mid u \in w, w \in \text{Walks}(v, \mathcal{L})\} \tag{4}$$

The cardinality of this set, $\text{NodeCount}(v)$, represents the number of unique nodes accessible from v through walks of length $\mathcal{L}$. This count is used to determine whether v qualifies as a Nexus Node:

$$\text{NodeCount}(v) = |\text{NodeSet}(v)| \tag{5}$$

A node v is determined as a Nexus Node, denoted as v_n, if $\text{NodeCount}(v)$ meets or exceeds a predefined threshold β. Nexus Nodes are characterized by their significant connectivity within the graph, as evidenced by the large number of unique nodes they connect to within the specified walk length.

Nodes that do not satisfy the Nexus Node criteria are further evaluated for their connectivity to Nexus Nodes. Specifically, a node v that is not categorized as a Nexus Node ($v \notin$ NexusNodes) is labeled as a Connector Node, denoted as v_c, if it has a direct edge connecting it to at least one Nexus Node. If no such direct connection exists, the node is identified as a Sparse Node (v_s).

Sparse Nodes, along with their associated edges, are removed from the graph, which leads to a reduction in the graph's overall size and complexity. The sets of remaining nodes and edges after this removal are denoted by $\hat{\mathcal{V}}$ and $\hat{\mathcal{E}}$, respectively:

$$\hat{\mathcal{V}} = \mathcal{V} - \{v \mid v = v_s\} \tag{6}$$

$$\hat{\mathcal{E}} = \mathcal{E} - \{\text{all edges connected to } v_s\} \tag{7}$$

This step effectively eliminates nodes that do not contribute significantly to the graph's structural integrity, thereby streamlining the graph for further analysis.

3.2. Connector Node Pruning Using Jaccard Similarity

After categorizing nodes and removing Sparse Nodes in the first step, the NCP algorithm further refines the graph by evaluating the Connector Nodes. This evaluation is based on their structural similarity to Nexus Nodes, using the previously defined Jaccard similarity measure. Figure 1c demonstrates the pruning of Connector Nodes based on Jaccard similarity.

For each Connector Node v_c, the algorithm calculates the Jaccard similarity with each of its directly connected Nexus Nodes v_n. The goal is to determine the extent to which a Connector Node shares its neighborhood with Nexus Nodes. The maximum Jaccard similarity score for each Connector Node v_c with its Nexus Node neighbors is computed as:

$$\mathcal{J}_{\max}(v_c) = \max_{v_n \in \text{NexusNodes}} \mathcal{J}(v_c, v_n) \tag{8}$$

where $\mathcal{J}(v_c, v_n)$ is the Jaccard similarity between Connector Node v_c and Nexus Node v_n. This score $\mathcal{J}_{\max}(v_c)$ reflects the strongest structural relationship that the Connector Node v_c has with the Nexus Nodes.

To decide whether to retain or prune a Connector Node, the algorithm compares $\mathcal{J}_{\max}(v_c)$ against a predefined threshold θ. If $\mathcal{J}_{\max}(v_c)$ is less than θ, it indicates that the Connector Node v_c has insufficient similarity to the Nexus Nodes to be considered structurally significant. Consequently, such nodes and their associated edges are pruned from the graph:

$$\mathcal{V}' = \hat{\mathcal{V}} - \{v_c \mid \mathcal{J}_{\max}(v_c) < \theta\} \tag{9}$$

$$\mathcal{E}' = \hat{\mathcal{E}} - \{\text{all edges connected to pruned } v_c\} \tag{10}$$

Here, $\mathcal{V}'$ and $\mathcal{E}'$ represent the sets after pruning based on the Jaccard similarity. This step ensures that the remaining Connector Nodes are those that have significant connections with the core structure of the graph, as represented by the Nexus Nodes. By retaining only the most structurally relevant nodes, the reduced graph $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$ retains the essential properties of the original graph, facilitating more efficient analysis and interpretation. Algorithm 1 presents the pseudo-code for our proposed graph reduction technique.

Algorithm 1: NCP Algorithm.

Input: Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, Walk Length $\mathcal{L}$, β, θ;
Step 1: Node Categorization and Sparse Node Removal
NexusNodes $\leftarrow \emptyset$;
ConnectorNodes $\leftarrow \emptyset$;
for *each node* $v \in \mathcal{V}$ **do**
  Walks $\leftarrow$ generateAllWalks(v, $\mathcal{L}$);
  NodeSet $\leftarrow$ getUniqueNodes(Walks);
  NodeCount $\leftarrow$ size(NodeSet);
  if *NodeCount* $\geq \beta$ **then**
    Add v to NexusNodes;
  end
end
for *each node* $v \notin$ *NexusNodes* **do**
  if *v is directly connected to any Nexus Node* **then**
    Add v to ConnectorNodes;
  else
    Remove v from $\mathcal{V}$;
    Remove edges connected to v from $\mathcal{E}$;
  end
end
Step 2: Connector Node Pruning Using $\mathcal{J}$
for *each node* $v_c \in$ *ConnectorNodes* **do**
  max_similarity $\leftarrow 0$;
  for *each node* $v_n \in$ *NexusNodes connected to* v_c **do**
    similarity $\leftarrow \mathcal{J}(v_c, v_n)$;
    if *similarity* $>$ *max_similarity* **then**
      max_similarity $\leftarrow$ similarity;
    end
  end
  if *max_similarity* $< \theta$ **then**
    Remove v_c from $\mathcal{V}$;
    Remove edges connected to pruned v_c from $\mathcal{E}$;
  end
end
Output: Reduced Graph $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$;

Our node pruning algorithm aims to maintain the performance of GNNs by strategically reducing the graph's size, while retaining critical structural and feature-related information. This reduction is achieved through the retention of "NexusNodes", which are identified based on walks of a specified length, $\mathcal{L}$. These nodes are integral to the graph's connectivity and are likely to hold central roles in information flow, making them particularly valuable for learning tasks. By pruning less significant nodes, our algorithm not only reduces computational demands on GNN—thereby decreasing training and inference times—but also minimizes noise, potentially enhancing GNN's ability to generalize from training data.

The second step of the NCP algorithm involves the process of pruning the Connector Nodes using the Jaccard similarity, which ensures that only those Connector Nodes that share a significant overlap in their neighborhoods with the Nexus Nodes are retained. This step is critical for maintaining the integrity of the graph structure, while still eliminating redundant connections that do not contribute meaningfully to the network's information processing capabilities. By applying a threshold-based similarity measure, this step helps in further refining the graph to a structure that is highly representative of the original graph yet significantly more efficient for processing by a GNN.

Moreover, the relationship between the walk length $\mathcal{L}$ in our pruning algorithm and the depth $\mathcal{D}$ of a GNN is meticulously calibrated to ensure that the pruned graph is optimally structured for the GNN's receptive field. For clarity, let $\mathcal{D}$ represent the depth of a GNN, which dictates how far the network aggregates information from its nodes. In the case of a GNN with a depth $\mathcal{D} = 3$, the layers of information aggregation are defined as follows:

- $\mathcal{D} = 1$: Processes the node's own features.
- $\mathcal{D} = 2$: Aggregates information from the node's immediate neighbors.
- $\mathcal{D} = 3$: Aggregates information from the neighbors of the node's immediate neighbors (two-hop neighborhood).

Given this setup, to align the pruning process effectively with GNNs' capabilities, the walk length $\mathcal{L}$ in our algorithm is set to $\mathcal{D} - 1$. This means for a GNN with a depth of 3, $\mathcal{L}$ would be 2. This setting ensures that the structural and feature information retained in the pruned graph matches the depth of information processing in the GNN up to its second depth. By configuring $\mathcal{L} = \mathcal{D} - 1$, our algorithm enhances the efficiency and effectiveness of a GNN, focusing the network's learning on the most crucial features and connections within the graph, which are essential for capturing meaningful patterns and interactions.

Figure 1 illustrates this process, showing (a) the original graph, (b) the removal of Sparse Nodes with $\mathcal{L} = 2$ and $\beta = 8$, and (c) the pruning of Connector Nodes with $\theta = 0.15$.

3.3. Time and Space Complexity Analysis of the NCP Algorithm

To address the computational complexity of the proposed NCP algorithm, we analyze both time and space requirements for each phase. Here, $|\mathcal{V}|$ denotes the number of nodes in the graph, while $|\mathcal{E}|$ represents the number of edges in the graph.

In Step 1, all possible walks of a specified length $\mathcal{L}$ are generated for each node. For a node with degree d, the number of potential unique walks of length $\mathcal{L}$ grows approximately as $d^{\mathcal{L}}$, where each additional step in the walk introduces up to d branching choices. Generating these walks for each node in the graph yields a time complexity of $O(|\mathcal{V}| \cdot d^{\mathcal{L}})$. After generating these walks, we count the unique nodes encountered, which further requires examining each of the $d^{\mathcal{L}}$ walks to identify distinct nodes. Thus, the complexity of counting unique nodes also remains $O(|\mathcal{V}| \cdot d^{\mathcal{L}})$ in time. For categorizing nodes into Nexus Nodes and Connector Nodes, a simple check is applied to each node, contributing an additional $O(|\mathcal{V}|)$ complexity in time. Therefore, the overall time complexity for Step 1 is $O(|\mathcal{V}| \cdot d^{\mathcal{L}})$.

In Step 2, each Connector Node is evaluated by computing the Jaccard similarity between its neighborhood and that of connected Nexus Nodes. Each similarity calculation takes $O(d)$ time, and each Connector Node has up to d neighbors, leading to a time complexity of $O(|\mathcal{V}| \cdot d^2)$ for this step. The final check to determine if the maximum similarity meets the threshold θ incurs negligible additional time complexity. Thus, the overall time complexity for Step 2 is $O(|\mathcal{V}| \cdot d^2)$.

Combining the complexities of the two steps, the overall time complexity of the NCP algorithm is $O(|\mathcal{V}| \cdot d^{\mathcal{L}} + |\mathcal{V}| \cdot d^2)$.

The space complexity of the NCP algorithm is determined by the storage requirements for the graph, the generated walks, and any additional structures used in the algorithm. Storing the original graph requires $O(|\mathcal{V}| + |\mathcal{E}|)$ space. In Step 1, all generated walks of length $\mathcal{L}$ must be stored for each node, leading to an additional $O(|\mathcal{V}| \cdot d^{\mathcal{L}})$ space complexity, as there can be up to $d^{\mathcal{L}}$ walks from each node. Counting unique nodes in these walks requires no additional space beyond what is needed to store the walks themselves. Storing

node categories (Nexus Nodes, Connector Nodes) requires only $O(|\mathcal{V}|)$ space. In Step 2, similarity values between nodes are computed and discarded as needed, so no extra space is required beyond $O(|\mathcal{V}|)$ to temporarily store these values during processing. Thus, the overall space complexity for Step 1 is $O(|\mathcal{V}| \cdot d^{\mathcal{L}})$ and for Step 2 is $O(|\mathcal{V}|)$.

Combining these factors, the overall space complexity of the NCP algorithm is $O(|\mathcal{V}| \cdot d^{\mathcal{L}} + |\mathcal{V}| + |\mathcal{E}|)$.

3.4. Extension of NCP to Weighted and Directed Graphs

The NCP algorithm, as presented, is designed for undirected, unweighted graphs. However, extending NCP to weighted or directed graphs could provide valuable insights in cases where edge weights reflect varying connection strengths or directionality indicates information flow. Below, we outline potential adaptations for both graph types.

In weighted graphs, where edge weights signify the strength or significance of connections, additional criteria for Nexus Node identification could incorporate weight thresholds or aggregate connection strengths. For instance, in the walk generation phase, weighted connections could be prioritized, allowing the algorithm to favor nodes with stronger overall connectivity. This adaptation might require modifying the Nexus threshold (β) to consider cumulative edge weights instead of solely relying on the number of connections, potentially refining Nexus Node identification in graphs where connection strengths vary significantly.

In directed graphs, where edges indicate information flow, additional modifications are necessary for effective pruning. To adapt the Nexus Node identification approach for directed graphs, we propose the following changes. First, we generate two distinct sets of walks for each node: Incoming Walks, which capture paths leading into a node and represent how influence or information flows toward that node, and Outgoing Walks, which capture paths leading out from a node, indicating how influence or information propagates outward. After generating the incoming and outgoing walks for each node, we calculate the unique nodes reached within each set and define two separate thresholds: β_{in} for incoming connectivity and β_{out} for outgoing connectivity. A node qualifies as a Nexus Node if it meets either the incoming or outgoing threshold, allowing the algorithm to identify nodes that are either influential sources or significant recipients within the graph's directional flow.

For pruning Connector Nodes in directed graphs, we calculate Jaccard similarity between each Connector Node and its connected Nexus Nodes separately for incoming and outgoing edges. Connector Nodes with low similarity in both incoming and outgoing connections relative to Nexus Nodes are pruned, ensuring that only those Connector Nodes with meaningful directed connections remain in the pruned graph.

These adaptations allow NCP to account for the additional structural nuances present in both weighted and directed graphs. In future work, we aim to test these modifications across a range of weighted and directed graph datasets to evaluate their effectiveness and impact on classification tasks.

3.5. Parameter Sensitivity and Robustness of NCP

The performance and quality of the NCP algorithm can vary based on its key parameters: walk length ($\mathcal{L}$), Nexus threshold (β), and similarity threshold (θ). These parameters impact the level of reduction and preservation of essential graph structures, and tuning them for specific graph types can be beneficial:

- Walk Length ($\mathcal{L}$): Increasing the walk length enables NCP to capture more extensive connections, which is particularly useful in dense graphs with higher clustering coefficients. In contrast, for sparser graphs, shorter walks may be more appropriate to avoid excessive inclusion of distant nodes that may not contribute significantly to core structure.
- Nexus Threshold (β): The Nexus threshold controls which nodes are classified as Nexus Nodes, depending on their connectivity. Graphs with high-degree nodes (e.g., social networks with hubs) may benefit from a higher β to avoid designating too many Nexus Nodes, while low-density graphs may require a lower β to ensure that essential connections are retained.

- Similarity Threshold (θ): The similarity threshold influences Connector Node pruning, and its effect can vary by graph type. In graphs with uneven node degree distributions (e.g., citation networks), a lower θ may help retain essential connections, while a higher θ could be suitable for more uniform graphs like biological networks, where local structure is critical.

In future work, we intend to conduct a thorough sensitivity analysis on different types of graphs (e.g., social networks, biological networks, citation networks) to evaluate the robustness of NCP across varying structural characteristics such as density, node degree distribution, and clustering coefficient. This analysis will help to establish general guidelines for parameter selection based on graph characteristics.

3.6. Limitations and Disadvantages of the NCP Method

While the NCP method offers notable efficiency and performance benefits, it also has certain limitations:

- Dependence on Threshold Parameters: The effectiveness of NCP depends on threshold parameters β (for node categorization) and θ (for Jaccard similarity-based pruning). These parameters need careful tuning to balance graph reduction and classification performance. Choosing suboptimal values for these thresholds can lead to over-pruning, where critical information may be lost, or under-pruning, where the graph remains too dense for efficient processing. As optimal threshold values can vary across datasets, selecting appropriate values may require parameter tuning or experimentation on a per-dataset basis.
- Applicability Limited to Certain Graph Types: NCP is particularly suited for graphs with a clear structure, including distinguishable hubs and connectors. However, in sparse or highly irregular graphs, the categorization criteria and pruning approach may not fully capture the nuances of the graph's structure, potentially limiting the method's effectiveness. This could result in suboptimal classification performance on such datasets, where the method may struggle to distinguish between critical and non-essential nodes effectively.
- Impact of NCP on Interpretability of GNN Models: The reduction achieved by NCP could potentially alter information pathways within the graph, affecting how information flows through the GNN layers. This may lead to the exclusion of nodes that, although less connected, play an important role in specific interpretations of the model. To mitigate this risk, we propose the following strategies:
    1. Before applying NCP, nodes identified as important for interpretability (e.g., nodes with high centrality in explanations or those frequently involved in known pathways) can be retained by setting a lower pruning threshold for these nodes or including an additional criterion based on interpretability relevance.
    2. After applying NCP, the pruned graph can be analyzed for interpretability impact by running a comparative analysis on explanation metrics (e.g., feature importance scores or attention weights) before and after pruning. If critical nodes or edges were removed, NCP thresholds can be adjusted iteratively to retain interpretability without significantly increasing graph complexity.
    3. In applications where specific node and edge relationships must be preserved, domain-specific constraints can be incorporated into the pruning process to retain key nodes and edges. For example, in biological networks, nodes representing essential genes or pathways can be exempted from pruning.

    In future work, we plan to further investigate how NCP can be adapted to maintain interpretability by retaining nodes and edges essential for producing explanations, especially for applications where model transparency is critical. These adaptations will help balance the goals of graph reduction and interpretability, ensuring that essential interpretative structures are preserved.

4. Experiments

NCP's ability to reduce graph complexity while preserving essential structural information can be especially valuable in various real-world applications where large and complex graphs challenge the efficiency and interpretability of GNNs. For example, in social network analysis, NCP can streamline extensive social graphs, retaining influential nodes and key connections, which can improve the detection of communities and enhance models analyzing influence propagation. In biological networks, such as protein interaction or gene regulatory networks, NCP can focus GNNs on the most critical interactions, facilitating the identification of significant regulatory pathways or protein interactions. In traffic and transportation networks, NCP can reduce the complexity of urban traffic graphs, making it easier for GNNs to predict and optimize traffic patterns by retaining essential intersections and pathways. Additionally, in graph-based malware detection, where graphs represent API calls or code flows, NCP can simplify these structures by retaining core malicious behavior patterns, allowing GNNs to more effectively identify and generalize patterns of malicious activity across different samples. These use cases demonstrate NCP's potential to enhance GNNs' effectiveness in analyzing large, real-world datasets by providing a computationally efficient and interpretable graph reduction approach.

In this section, we evaluate the performance of our proposed graph reduction algorithm and compare it with the WIS method, which focuses solely on edge pruning. We carried out experiments on two datasets—the CFG dataset (comprising 50 benign and 50 malicious samples selected from the PE Malware Machine Learning Dataset [28] and DikeDataset [29], with control flow graphs dynamically captured using the angr library) and the PROTEINS dataset [30,31] to evaluate the effectiveness of the algorithms in graph classification tasks using Graph Convolutional Networks (GCNs) with $\mathcal{D} = 3$. For these experiments, the walk length ($\mathcal{L}$) is set to 2.

Table 1 provides an overview of the datasets used in our experiments, including the number of samples, average nodes and edges per graph, and the breakdown of sample types. The CFG dataset contains graphs with a larger average size, allowing us to assess the scalability of NCP in handling complex structures, while the PROTEINS dataset provides a more compact graph representation suitable for testing classification performance on relatively smaller graphs.

Table 1. Characteristics of Datasets Used in Experiments.

Dataset	Number of Samples	Average Nodes per Graph	Average Edges per Graph	Sample Types	Number of Samples per Type
PROTEINS	1113	39.05	72.81	Enzyme/Non-enzyme	450 Enzyme, 663 Non-enzyme
CFG	100	11,223.01	18,945.09	Benign/Malicious	50 Benign, 50 Malicious

We selected the PROTEINS dataset for our experiments because it is a widely recognized benchmark dataset that contains sufficiently large graphs, making it ideal for evaluating the performance of graph reduction techniques. Additionally, the CFG dataset was chosen due to the limited availability of datasets containing large graphs suitable for runtime evaluation. The CFG dataset's intricate and variable structure provides a unique challenge, making it well suited for assessing the computational efficiency and scalability of our proposed graph reduction algorithm.

To ensure that our experiments were conducted under robust and controlled conditions, they were performed on a high-performance computing platform. The experiments utilized an Intel Xeon Platinum 8253 CPU with 32 cores at 3.000 GHz and an NVIDIA Quadro RTX 6000/8000 GPU, which are particularly suited for handling the extensive computational demands of graph neural network processing. The system was equipped with 128 GB of memory, facilitating efficient management of large datasets and complex

computations. The software stack was primarily based on Python, with critical dependencies on PyTorch Geometric specifically for implementing and training the GCN used in our experiments. NetworkX v2.8.8 was used for graph manipulation and analysis.

4.1. Experiment on the PROTEINS Dataset

The PROTEINS dataset is a widely used benchmark for graph classification tasks. In this dataset, each graph represents a protein, which is categorized into one of two classes: enzyme and non-enzyme. To assess the classification we use accuracy as our primary performance metric. The formula for accuracy is:

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}} \tag{11}$$

For this dataset, we assigned the parameter θ a value of 0.2 across all tests, as extensive experimentation showed that variations in this parameter did not significantly impact the results. This lack of sensitivity to θ is attributed to the specific topology of the graphs in these datasets, which suggests that their structural characteristics do not heavily influence the effectiveness of θ in pruning decisions.

The NCP algorithm exhibits promising results for this dataset, particularly as the parameter β increases, which indicates a stricter criterion for node importance. Although the accuracy slightly declines as β increases from 3 to 6, this suggests a trade-off between the intensity of pruning and classification performance. Our algorithm maintains consistent performance and completes the task in a relatively short time, demonstrating its efficiency in pruning, while preserving essential graph features.

The WIS method, which prunes edges at varying percentages ranging from 20% to 100%, demonstrates a distinct trend: higher pruning rates lead to reduced accuracy. This consistent decline aligns with the removal of edges, potentially eliminating critical structural information necessary for GCN's performance. The time taken increases significantly with higher pruning percentages, suggesting that while WIS can drastically reduce graph size, it does so at the expense of both performance and efficiency.

Figure 2 shows the distribution of accuracy values for both NCP and WIS: NCP demonstrates stable and high accuracy across various parameter settings compared to WIS, which exhibits more variability in accuracy as its parameters change. The baseline accuracy (without pruning) is consistently at 0.71, serving as a reference point for evaluation. These boxes also represent the average accuracy of graph classification over 10 runs with random seeds for each algorithm.

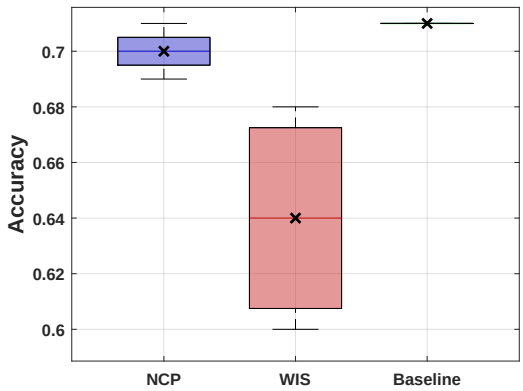

Figure 2. Distribution of obtained accuracy through NCP, WIS, and baseline.

Figure 3 illustrates the overall average number of nodes and edges pruned as the parameter β varies for the PROTEINS experiment using the NCP algorithm. Moreover, Table 2 compares the average runtime of the NCP and WIS algorithms, measured in seconds. The NCP algorithm shows the shortest runtime, with an average of 16.32 s,

demonstrating its computational efficiency. In contrast, the WIS algorithm has the longer runtime (25.59 s). The variability in runtime, indicated by the standard deviation, is smaller for the NCP algorithm, suggesting more consistent measures across different runs, while the WIS algorithm exhibits higher variability.

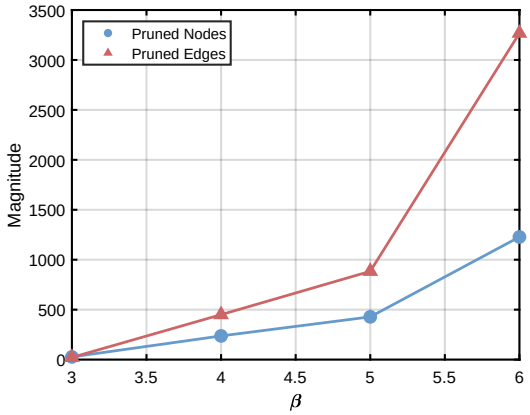

Figure 3. Pruned nodes and edges by NCP through varying β for the PROTEINS experiment.

Table 2. Average runtime of NCP and WIS over PROTEINS experiment.

Algorithm	NCP	WIS	Baseline
Time (Sec)	**16.32** $\pm$ **0.09**	25.59 $\pm$ 5	17.66

4.2. Experiment on the CFG Dataset

Our experiment on the CFG dataset evaluates the efficacy of the NCP algorithm in managing the complexity of control flow graphs from both benign and malicious software samples. The CFG dataset presents a unique challenge due to its intricate and variable structure. Figure 4 illustrates a benign sample from the CFG dataset after processing with the NCP algorithm. The removed nodes are also displayed to indicate their respective categories. In this visualization, the green nodes represent Nexus Nodes, the blue nodes are the retained Connector Nodes, the yellow nodes indicate pruned Connector Nodes, and the red nodes correspond to Sparse Nodes.

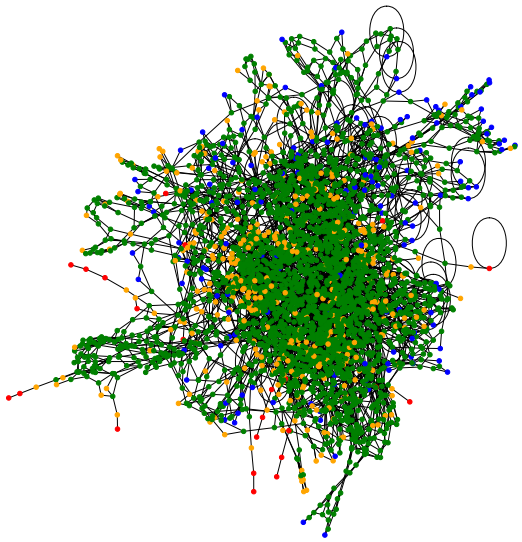

Figure 4. A benign graph sample from the CFG dataset processed using the NCP algorithm. Node categories are color-coded as follows: green for Nexus Nodes, blue for retained Connector Nodes, yellow for pruned Connector Nodes, and red for Sparse Nodes.

In this experiment, we particularly focus on analyzing how different settings of the θ parameter influence classification accuracy when applying NCP, with β values ranging from 3 to 10. The results, as illustrated in Figure 5, show that as θ varies from 0.05 to 0.5, the best accuracy peaks at $\theta = 0.4$ with an accuracy of 0.89. This demonstrates an optimal balance between node reduction and the preservation of critical graph structure, highlighting the effectiveness of the NCP algorithm in managing graph complexity for graph classification.

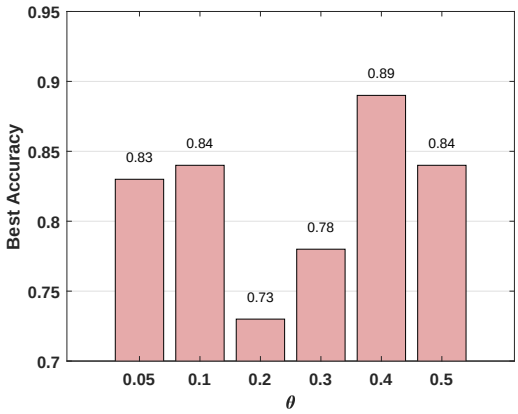

Figure 5. Best obtained accuracy across varying θ for the CFG dataset using the NCP algorithm, with β values ranging from 3 to 10.

Table 3 provides a comparison of runtime and accuracy between the NCP method (achieving its best result through hyperparameter tuning) and the WIS method in the CFG experiment. It demonstrates that the NCP algorithm not only achieves higher accuracy but also significantly reduces computation time compared to the WIS method, which shows a substantial increase in computation time even with only 20% of edges pruned. This efficiency is crucial for real-world applications, where time and computational resources are often limited.

Table 3. Runtime and performance analysis of NCP and WIS.

Algorithm	NCP	WIS	Baseline
Accuracy	**0.89**	0.70	0.85
Time (Sec)	**75.16**	64,842.94	81.12

Overall, the NCP algorithm offers a robust solution for graph reduction for the CFG experiment by effectively balancing the trade-offs between node pruning and accuracy retention. Its adaptability across different configurations makes it a valuable tool in graph-based machine learning tasks, especially for datasets with complex and heterogeneous structures.

5. Conclusions

In this paper, we introduced the NCP algorithm, an effective graph reduction technique aimed at optimizing GNNs by simplifying graph structures, while retaining essential topological properties. Our method strategically prunes less significant nodes, enhancing the manageability and computational efficiency of GNNs, which is particularly beneficial for large-scale graph applications. The experiments conducted on both the PROTEINS and CFG datasets demonstrate that NCP outperforms WIS in terms of accuracy, runtime, and preservation of graph integrity. This makes NCP a valuable tool for applications requiring efficient graph processing.

Looking ahead, we plan to extend our work to include edge prediction tasks, which are vital for network analysis, bioinformatics, and social network analysis. Adapting our pruning techniques to optimize edge prediction will potentially broaden the applicability

of GNNs to handle even larger graphs efficiently. Further research will also explore the integration of NCP with various GNN architectures to enhance model generalization and performance across diverse applications, aiming to forge more robust and adaptable GNN models for complex tasks.

Author Contributions: Conceptualization, R.R.-F.; methodology, H.S. and R.R.-F.; software, G.H.; validation, G.H.; formal analysis, H.S.; investigation, H.S.; resources, G.H.; data curation, G.H.; writing—original draft preparation, H.S.; writing—review and editing, R.R.-F. and A.A.G.; visualization, H.S.; supervision, R.R.-F. and A.A.G.; project administration, A.A.G. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The study did not involve the creation of new datasets. All datasets used in this research are publicly available and have been referenced in the manuscript.

Conflicts of Interest: The authors declare no conflicts of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

References

1. Gao, X.; Yu, J.; Jiang, W.; Chen, T.; Zhang, W.; Yin, H. Graph Condensation: A Survey. *arXiv* **2024**, arXiv:2401.11720.
2. Tian, Y.; Hankins, R.A.; Patel, J.M. Efficient aggregation for graph summarization. In Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, Vancouver, BC, Canada, 10–12 June 2008; pp. 567–580.
3. Amiri, S.E.; Adhikari, B.; Bharadwaj, A.; Prakash, B.A. NetGist: Learning to Generate Task-Based Network Summaries. In Proceedings of the 2018 IEEE International Conference on Data Mining (ICDM), Singapore, 17–20 November 2018; pp. 857–862.
4. Loukas, A.; Vandergheynst, P. Spectrally approximating large graphs with smaller graphs. *arXiv* **2018**, arXiv:1802.07510.
5. Loukas, A. Graph reduction with spectral and cut guarantees. *arXiv* **2018**, arXiv:1808.10650.
6. Huang, Z.; Zhang, S.; Xi, C.; Liu, T.; Zhou, M. Scaling Up Graph Neural Networks via Graph Coarsening. *arXiv* **2021**, arXiv:2106.05150.
7. Kumar, M.; Sharma, A.; Saxena, S.; Kumar, S. Featured Graph Coarsening with Similarity Guarantees. In Proceedings of the International Conference on Machine Learning, Honolulu, HI, USA, 23–29 July 2023; pp. 17953–17975.
8. Jin, W.; Zhao, L.; Zhang, S.; Liu, Y.; Tang, J.; Shah, N. Graph Condensation for Graph Neural Networks. *arXiv* **2022**, arXiv:2110.07580.
9. Gao, J.; Wu, J. Multiple sparse graphs condensation. *Knowl.-Based Syst.* **2023**, *278*, 110904. [CrossRef]
10. Yang, B.; Wang, K.; Sun, Q.; Ji, C.; Fu, X.; Tang, H.; You, Y.; Li, J. Does Graph Distillation See Like Vision Dataset Counterpart? *arXiv* **2023**, arXiv:2310.09192.
11. Feng, Q.; Jiang, Z.; Li, R.; Wang, Y.; Zou, N.; Bian, J.; Hu, X. Fair Graph Distillation. *Adv. Neural Inf. Process. Syst.* **2023**, *36*, 80644–80660.
12. Mao, R.; Fan, W.; Li, Q. GCARe: Mitigating Subgroup Unfairness in Graph Condensation Through Adversarial Regularization. *Appl. Sci.* **2023**, *13*, 9166. [CrossRef]
13. Li, X.; Wang, K.; Deng, H.; Liang, Y.; Wu, D. Attend Who is Weak: Enhancing Graph Condensation via Cross-Free Adversarial Training. *arXiv* **2023**, arXiv:2311.15772.
14. Gao, X.; Chen, T.; Zang, Y.; Zhang, W.; Nguyen, Q.V.H.; Zheng, K.; Yin, H. Graph Condensation for Inductive Node Representation Learning. *arXiv* **2023**, arXiv:2307.15967.
15. Liu, Y.; Qiu, R.; Tang, Y.; Yin, H.; Huang, Z. PUMA: Efficient Continual Graph Learning with Graph Condensation. *arXiv* **2023**, arXiv:2312.14439.
16. Zhang, T.; Zhang, Y.; Wang, K.; Yang, B.; Zhang, K.; Shao, W.; Liu, P.; Zhou, J.T.; You, Y. Two Trades is not Baffled: Condensing Graph via Crafting Rational Gradient Matching. *arXiv* **2024**, arXiv:2402.04924.
17. Fang, J.; Li, X.; Sui, Y.; Gao, Y.; Zhang, G.; Wang, K.; Wang, X.; He, X. EXGC: Bridging Efficiency and Explainability in Graph Condensation. *arXiv* **2024**, arXiv:2402.05962.
18. Wang, L.; Fan, W.; Li, J.; Ma, Y.; Li, Q. Fast Graph Condensation with Structure-based Neural Tangent Kernel. *arXiv* **2024**, arXiv:2310.11046.
19. Xiao, Z.; Liu, S.; Wang, Y.; Zheng, T.; Song, M. Disentangled Condensation for Large-scale Graphs. *arXiv* **2024**, arXiv:2401.12231.
20. Zhang, Y.; Zhang, T.; Wang, K.; Guo, Z.; Liang, Y.; Bresson, X.; Jin, W.; You, Y. Navigating Complexity: Toward Lossless Graph Condensation via Expanding Window Matching. *arXiv* **2024**, arXiv:2402.05011.
21. Althöfer, I.; Das, G.; Dobkin, D.; Joseph, D.; Soares, J. On sparse spanners of weighted graphs. *Discret. Comput. Geom.* **1993**, *9*, 81–100. [CrossRef]
22. Batson, J.D.; Spielman, D.A.; Srivastava, N. Twice-Ramanujan Sparsifiers. *arXiv* **2009**, arXiv:0808.0163.

23. Chen, T.; Sui, Y.; Chen, X.; Zhang, A.; Wang, Z. A Unified Lottery Ticket Hypothesis for Graph Neural Networks. *arXiv* **2021**, arXiv:2102.06790.
24. Wickman, R.; Zhang, X.; Li, W. A Generic Graph Sparsification Framework using Deep Reinforcement Learning. *arXiv* **2023**, arXiv:2112.01565.
25. Razin, N.; Verbin, T.; Cohen, N. On the Ability of Graph Neural Networks to Model Interactions Between Vertices. *arXiv* **2023**, arXiv:2211.16494.
26. Daneshfar, F.; Soleymanbaigi, S.; Yamini, P.; Amini, M.S. A survey on semi-supervised graph clustering. *Eng. Appl. Artif. Intell.* **2024**, *133*, 108215. [CrossRef]
27. Wu, T.; You, X.; Xian, X.; Pu, X.; Qiao, S.; Wang, C. Towards deep understanding of graph convolutional networks for relation extraction. *Data Knowl. Eng.* **2024**, *149*, 102265. [CrossRef]
28. Practical Security Analytics-Pe-Malware-Machine-Learning-Dataset. Available online: https://practicalsecurityanalytics.com/pe-malware-machine-learning-dataset/ (accessed on 26 October 2024).
29. GitHub-Iosifache/DikeDataset: DikeDataset. Available online: https://github.com/iosifache/DikeDataset (accessed on 26 October 2024).
30. Borgwardt, K.M.; Ong, C.S.; Schoenauer, S.; Vishwanathan, S.V.N.; Smola, A.J.; Kriegel, H.-P. Protein function prediction via graph kernels. *Bioinformatics* **2005**, *21* (Suppl. S1), i47–i56. [CrossRef]
31. Dobson, P.D.; Doig, A.J. Distinguishing enzyme structures from non-enzymes without alignments. *J. Mol. Biol.* **2003**, *330*, 771–783. [CrossRef]