2FWL-SIRGN A Scalable Higher-Order Graph Representation Learning Approach*

* Via 2-dimensional Folklore Weisfeiler Lehman and Structural Graph Partitioning

Justin Carpenter
dept. Computer Science
Boise State University
Boise, Idaho, Country
justincarpenter836@u.boisestate.edu

Edoardo Serra
dept. Computer Science
Boise State University
Boise, Idaho, Country
edoardoserra@boisestate.edu

***Abstract*—Graph representation learning has numerous applications, ranging from social networks to bioinformatics, with a major focus on Graph Neural Networks (GNNs). However, many GNN models face challenges in capturing intricate graph structures, such as cycles, and are prone to overfitting and high computational costs, limiting their scalability on medium to big graphs.**

In this paper, we propose 2FWL-SIRGN, a novel approach that integrates higher-order Weisfeiler-Lehman (WL) test algorithm while mitigating its computational challenges. Our method combines the Structural Iterative Representation Learning for Graph Nodes (SIRGN) framework with the 2-dimensional Folklore Weisfeiler-Lehman (2FWL) isomorphism test. The unsupervised training of the SIRGN component improves the model's resistance to overfitting, while the 2FWL component enhances its expressive power, enabling it to capture complex patterns, such as cycle structures. However, the inclusion of 2FWL increases computational overhead. To address this, we introduce a Structural Graph Partitioning algorithm, which allows 2FWL-SIRGN to scale efficiently to big graphs.

Extensive experiments demonstrate that 2FWL-SIRGN outperforms state-of-the-art methods by addressing key challenges in graph representation learning. Our model captures richer structural information while maintaining computational efficiency, surpassing other higher-order WL approaches. Additionally, our partitioning strategy enables 2FWL-SIRGN to effectively handle large-scale graphs, and its inherent resistance to overfitting addresses a common limitation of GNNs. These advancements position 2FWL-SIRGN as a robust solution for real-world applications where both scalability and accuracy are critical.

***Index Terms*—Higher-Order, Graph Representation Learning, Structural Graph Partitioning, Folklore Weisfeiler-Lehman**

I. Introduction

Graphs are powerful data structures that represent entities as nodes and the relationships between them as edges. Graph-structured data is one of the most ubiquitous forms of structured data used in machine learning approaches, such as graph neural networks (GNNs) and graph representation learning. In order to utilize graph-structured data, many studies have been done to transform these complex structures into a format that can be easily processed by machine learning algorithms while preserving the graph's informational content. [1]

Recent years have seen a surge in research on graph representation learning techniques in various machine learning tasks such as node classification, link prediction, and graph classification. Instead of extracting hand-engineered features, graph representation learning aims to learn representations that encode structural information about the graph. [2]

There are two well explored concepts within graph representation learning, structural and proximity, each with the same end goal to transform a graphs informational content into a usable format. Although similar end goal, the two concepts differ from one another on capturing different types of information within the graph. Structural representation learning focuses on capturing the positions of nodes within the entire graph, irrespective of their local neighborhoods. The goal is to identify nodes that have similar structures and preserve the overall graph topology. Proximity representation learning emphasizes individual node neighbors or proximity to a centroid in order to preserve the local neighborhood structure of nodes. [3]–[5]

Traditional proximity-based representation learning methods, such as node2vec [4] and DeepWalk [3], optimize embeddings to encode the statistics of random walks to define node similarity and neighborhood reconstruction. The problem with both approaches was the ability to capture the graphs structure and complexity.

Structural graph representation learning approaches such as, Structural Iterative Representation Learning Approach for Graph Nodes (SIR-GN) [6] and Graph Isomorphism Network (GIN) [7] attempt to more accurately capture the graph structure than previous proximity based approaches. Such approaches simulate the WL test [8], which is an approximate algorithm for verifying that two graphs are isomorphic. These approaches can distinguish almost all pairs of isomorphic graphs. Their major limitation is their inability to distinguish non-isomorphic graphs with cycles of varying lengths. [8]

To overcome such limitation graph neural networks simulating higher order WL test have been prosed. [9]–[11] The problem that arises when implementing higher-order WL is the increase in computational cost [8] and an increased risk

of overfitting.

In this paper, we present 2FWL-SIRGN, a novel approach that integrates the higher-order Weisfeiler-Lehman (WL) test algorithm while addressing its computational challenges. Our method combines the Structural Iterative Representation Learning for Graph Nodes (SIRGN) framework with the 2-dimensional Folklore Weisfeiler-Lehman (2FWL) isomorphism test, which is a higher-order WL isomorphism test. The unsupervised training of the SIRGN component enhances the model's ability to mitigating overfitting, while the 2FWL component increases its expressive power, enabling it to capture complex patterns like cycle structures. To counter the increased computational load from 2FWL, we introduce a Structural Graph Partitioning algorithm, allowing 2FWL-SIRGN to efficiently scale to large graphs. In the following we present the **List of our contributions:**

1) **2FWL-SIRGN**: Design of a higher-order WL test SIRGN.
2) **Structural Graph Partition**: Design of a structural graph partitioning algorithm to overcome the computational cost of 2FWL-SIRGN.
3) **Experimentation and Validation**: A set of experiments designed to fully demonstrate the superior structural capabilities of our proposed algorithm in comparison with existing methods.

II. Related Works

In the literature, many graph representation approaches are closely linked to the Weisfeiler-Lehman (WL) isomorphism test, a heuristic algorithm designed to determine whether two graphs are isomorphic. In this section, we first provide an overview of the WL isomorphism test and its higher-order variants. We then explore the graph representation learning procedures associated with these methods

A. (Folklore) Weisfeiler-Lehman Isomorphism Test

The Weisfeiler-Lehman (WL) Isomorphism Test is a heuristic algorithm widely used to determine whether two graphs are isomorphic. The graph isomorphism problem remains a significant challenge in computational theory, as no polynomial-time algorithm has yet been discovered. As a heuristic, the WL test can effectively distinguish many pairs of graphs as non-isomorphic. However, its output is binary: either "non-isomorphic" or "possibly isomorphic," meaning it cannot confirm that two graphs are truly isomorphic.

The 1-WL Test assigns the same label to each node and iteratively refines this label based on the labels of neighboring nodes. The algorithm reaches convergence when the distribution of the labels across the nodes ceases to change. At this point, if the label distributions of two graphs are dissimilar, the graphs are deemed non-isomorphic; if similar, they may be isomorphic. This method falls within the category of message-passing algorithms and is known for its linear execution time, at each iteration, relative to the graph's edge count, and linear space complexity relative to the node count [8], [12].

The K-dimensional Weisfeiler-Leman algorithm (k-wl), for $k > 2$ (k corresponds to the k-hop neighborhood), is a generalization of the 1-dimensional Weisfeiler-Leman which identifiers to each K-tuple of nodes within N^K of graph G. The aggregation of a node's neighborhood expands to encompass the neighborhoods of a K-tuple. The K-WL establishes a hierarchy where for any $K > 2$, the $(K+1)$-WL is strictly more expressive than the K-WL, with an interesting equivalence noted between 2WL and 1-WL. Given two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, the cost of K-WL at each iteration is $O(|V_1|^{K+1} + |V_2|^{K+1})$. Since the first method with higher expressive power than 1-WL is 3-WL, the cost of each iteration is $O(|V_1|^4 + |V_2|^4)$, which is impractical for medium and large graphs. To replicate this hierarchy in GNNs, researchers have introduced a 3-WL approach [11]. The modified graph structure this entails, along with the associated neighborhood definitions, achieved a higher expressivity than the 1-WL and 2-WL implementations. However, the increased complexity makes this approach is limited to the size of the graph.

To improve computational complexity, the K dimension Folklore Weisfeiler-Lehman (kFWL) test was introduced.

The KFWL test is a variant of the K-WL test that offers improved efficiency. The KFWL test works by considering K-tuples of nodes, similar to the K-WL test, but with a modified update rule.

The 2-dimensional FWL (2FWL) test is as powerful as the 3-dimensional WL (3WL) test. The cost of the 2FWL at each iteration is $O(n^3)$, while the cost of the 3-dimensional WL test is $O(n^4)$ [13]. More information about KFWL and K-WL can be found in [14]. Despite the improvement in complexity, the 2FWL's complexity remains infeasible for medium and large graphs

B. WL-related Graph Representation Learning Techniques

The expressive capabilities of GNNs in relation to the WL test have been extensively studied. Standard Graph Neural Networks (GNNs), such as Graph Convolutional Networks (GCNs) [15] and Graph Attention Networks (GATs) [16], share some algorithmic similarities with the 1-WL test. However, they generally fail to fully match its expressive power.

The work proposed by Berg et al. [17], introduced a method for learning node representations using GCNs, employing the WL test to assess the expressive power of GCNs compared to traditional graph-based learning techniques. The study demonstrated that GCNs could capture more expressive representations than conventional approaches such as sequence-to-sequence learning. Another Convolution Neural network approach, GCNN [18] proposed implementing the WL as the first layer in the Convolution Neural Network (CNN). By leveraging the WL test, the authors showed that their method achieved more expressive representations than traditional techniques based on recursive neural networks. The introduction of convolutional filters specifically adapted for graph data allowed for a dynamic adaptation to local neighborhoods, capturing both local and global graph structures. This adaptability has made graph CNNs particularly effective for

applications across social networks, biological data modeling, and recommendation systems [19].

Graph Isomorphism Networks (GINs), as described by [7], can achieve expressive power equivalent to the 1-WL test, under optimal training conditions, with a direct relationship between the node representations generated by GINs and the node colors assigned by the WL algorithm. The correlation between GINs and the WL test highlights their potential in solving graph isomorphism problems, positioning GINs as a neural approximation of classical graph algorithms. This connection is pivotal for understanding the broader applicability of neural networks in capturing complex graph relationships that are traditionally addressed through algorithmic means.

While the 1-WL test forms the foundation for many GNN architectures, more advanced graph representation learning techniques aim to increase expressive power by leveraging higher-order WL tests. For example, [11] introduced higher-order graph neural networks that utilize the K-WL test to capture more complex structural patterns. These methods significantly enhance representational power, enabling the detection of more nuanced graph structures. As another example, [20] describes a novel higher-order Weisfeiler-Lehman graph convolution network based on 2-FWL test.

Despite their promise, techniques based on higher-order WL tests face significant limitations, particularly in computational cost. These methods become impractical when applied to medium or large-scale graphs. Moreover, many of these supervised high-order techniques are prone to a significant risk of overfitting.

III. Methodology

The proposed solution in this paper introduces a novel graph representation learning model, 2FWL-SIRGN, coupled with a new structural graph partitioning algorithm. More specifically, 2FWL-SIRGN extends SIR-GN [6] by incorporating the computational structure of the 2-dimensional Folklore Weisfeiler-Lehman (2FWL) test. Additionally, 2FWL-SIRGN is designed to operate on a set of disconnected graphs, which resemble a generic graph partition. 2FWL-SIRGN extracts structural node representations in an unsupervised manner, similarly to SIR-GN, which mitigates the risk of overfitting.

The new structural partitioning algorithm is responsible for partitioning an arbitrary graph into several disconnected subgraphs, each with a bounded number of nodes. The partitioning is done in such a way that if two nodes in the original graph have the same structure around them, this structural relationship is preserved in the partitioned subgraphs. This preservation of structure is the rationale behind the term *structural partitioning*.

2FWL-SIRGN still presents a significant computational challenge in terms of the number of nodes in each partition, making it infeasible by itself for graphs of medium and large sizes. However, the structural partitioning algorithm, by bounding the number of nodes in each partition, makes this technique scalable. Together, these two algorithms form a novel representation learning approach that effectively captures intricate graph structural patterns, reduces the overfitting risk, and scales efficiently, even with larger graphs.

In the following sections, we describe the 2FWL-SIRGN and Structural Graph Partitioning algorithms in detail. The overarching goal is to highlight how our methodology, through a combination of structural partitioning and the enhanced Weisfeiler-Lehman test, achieves the dual objectives of scalability and robustness in representation learning.

A. 2FWL-SIRGN

The proposed solution, 2FWL-SIRGN, as described in Algorithm 1, extends SIR-GN by incorporating the 2-FWL algorithm to achieve a similar computational framework with enhanced structural features. The algorithm takes as input a set of p graphs, denoted as $PG = \{G_i = (V_i, E_i) | i = 1, \ldots, p\}$, the internal representation size n, and the number of iterations $\#iter$. The algorithm starts by initializing the matrix Emb_i for each graph $G_i \in PG$, where Emb_i stores a vector of dimension n^2 for each node pair $(a, b) \in V_i$. Specifically, for each pair (a, b), $Emb_i[a, b]$ is initialized as a zero vector, except when $a = b$, in which case it is initialized as a vector containing the degree $|ngb(E_i, a)|$ of node a. Please note that the function $ngb(E_i, a)$ returns the set of neighboring nodes of a, defined as $ngb(E_i, a) = b | (a, b) \in E_i$. From this initialization, it is evident that 2FWL-SIRGN operates within each partition graph G_i, similarly to the 2-FWL approach, with all tuples of nodes in V_i having size 2.

To better understand the structure of the algorithm, it has been divided into two primary functions. We will first focus on the initial function of 2FWL-SIRGN. Following the initialization described above, the algorithm proceeds with the iterations specified by $\#iter$, which constitute the primary computation phase (lines 9-19). The list of embedding matrices, Emb_p, must be formatted to effectively apply Principal Component Analysis (PCA) [21]. PCA is an unsupervised linear transformation used to reduce the dimensionality of the node embeddings, emphasizing the differences between the highest and lowest variance components to capture structural nuances across the graph.

It is crucial to combine all embeddings into a single list Emb (line 10) to retain the overall variance components that represent the entire graph structure. Once combined, a Min-Max normalization is applied to ensure that all features are considered on equal footing and to avoid the risk of features with large scales dominating and thereby overshadowing the other embeddings. After the PCA transformation, the embeddings will have reduced dimensionality, which may result in some loss of information or ambiguities. To mitigate this, we horizontally stack Emb with its negated version, $-Emb$, effectively creating a more diverse and robust feature set (lines 13-14). Subsequently, Emb is divided back into individual embeddings, with each partition Emb_i being passed to the 2FWL-SIRGN Iteration function for the respective graph partition G_i. The 2FWL-SIRGN Iteration function effectively illustrates the 2-FWL process implemented within Algorithm

Algorithm 1 2FWL-SIRGN Algorithm

1: **function** 2FWL-SIRGN($PG = \{G_i = (V_i, E_i) | i = 1, \ldots, p\}, n, \#iter$) ▷ n can be divided by 2
2: **for all** G_i in PG **do**
3: $Emb_i = 0^{|V_i|^2 \times n^2}$
4: **for all** $a \in V_i$ **do**
5: $d = |ngb(E_i, a)|$
6: $Emb[a, b] = [\underbrace{d, \ldots, d}_{n^2}]$ ▷ Emb_i can be directly indexed with two node a and b, with $Emb_i[a, b] \in \Re^{n^2}$
7: **end for**
8: **end for**
9: **for** $it = 1, \ldots, \#iter$ **do**
10: $EM = \text{VERTICALSTACK}([Emb_1, \ldots, Emb_p])$
11: $EM = \text{MINMAXCOLUMNNORMALIZER}(EM)$
12: $EM = \text{PCA}(EM, n/2)$
13: $EM = \text{HORIZONTALSTACK}([-EM, EM])$
14: $EM = \text{ROWNORMALIZER}(EM)$
15: $Emb_1, \ldots, Emb_p = \text{UNSTACK}(EM)$
16: **for** $G_i \in PG$ **do**
17: $Emb_i = \text{2FWL-SIRGNITERATION}(G_i, n, Emb_i)$
18: **end for**
19: **end for**
20: $FE = 0^{|V_i| \times n^2}$
21: **for all** $G_i \in PG$ **do**
22: **for all** $a \in V_i$ **do**
23: $FE[a] = Emb_i[a, a]$
24: **end for**
25: **end forreturn** FE
26: **end function**

27: **function** 2FWL-SIRGNITERATION($G = (V, E),\ n,\ Emb$)
28: $Emb2 = 0^{|V|^2 \times n^2}$
29: **for all** $a \in V$ **do**
30: **for all** $b \in V$ **do**
31: $unionz \leftarrow ngb(E, a) \cup ngb(E, b)$
32: $Emb2[a, b] = 0^{n^2}$
33: **for** $z \in unionz$ **do**
34: $Emb2[a, b] = Emb2[a, b] + \text{flatten}(Emb[a, z]^T \times Emb[z, b])$
35: **end for**
36: **end for**
37: **end for**
38: **return** $Emb2$
39: **end function**

1 (lines 27-38). By separating the two components of 2FWL-SIRGN, we can more clearly focus on the specific 2-FWL operations performed on each graph partition G_i.

The initialization of $Emb2$ (line 28) has a size of $V_i \times n^2$, since only a single graph partition is being processed at a time, thereby reducing the size and computational complexity of the embedding matrix. Moving on to the core implementation, the algorithm iterates through all node pairs (lines 29-30) to identify nodes with relationships, subsequently aggregating messages from their respective neighbors. Once aggregation is complete for all node pairs, Emb_1 is updated to $Emb2$, and the next partition is processed. Upon completing the computation for each graph partition, the iteration process is repeated. At each step, Emb is concatenated, normalized, and transformed to preserve as much of the full graph's structural information as possible, thereby ensuring comprehensive embedding representations. This iterative process is repeated until all partitions and all iterations are completed, to retain as much of the full graph's structure as possible. In the following, we show the execution time and the memory cost of Algorithm1.

Theorem 1: Given a set of graphs $PG = \{G_i = (V_i, E_i) | i = 1, \ldots, p\}$ and an internal representation size n, 2FWL-SIRGN, defined in Algorithm 1, has as execution time $O(\#iter * (n^6 + n^4 * (\sum_{i=0}^{p} |V_i|^2) + n^2 * (\sum_{i=0}^{p} |V_i| * |E_i|)))$ and memory cost $O(\sum_{i=0}^{p} |V_i|)$

Proof: To prove the execution time in Theorem 1, we identify the cost for each of the code lines in Algorithm 1 as follows. Line from 2 to 8 costs $O(n^2 * \sum_{i=0}^{p} |V_i|^2)$. Considering lines from 10 to 15 the most expensive part is the computation of the PCA line 12. Given that the cost of PCA is $O(p^2 * h + p^3$, where p is the number of dimensions and h, is the number of instances, the cost of 10 to 15 is $O(\#iter * (n^4 * (\sum_{i=0}^{p} |V_i|^2) + n^6))$. Lines from 16 to 18 costs $\#iter * \sum_{i=0}^{p} C_i$ where C_i is the cost of the function 2FWL-SIRGNIteration((.)) over the graph G_i. By looking at lines from 28 to 36 the cost C_i is equal to $O(|V_i| * |E_i|)$. Ultimately, the cost of lines from 21 to 24 is $O(\sum_{i=0}^{p} |V_i|)$. By summing up all the costs the result is $O((n^2 * \sum_{i=0}^{p} |V_i|^2) + \#iter * (n^6 + n^4 * (\sum_{i=0}^{p} |V_i|^2) + n^2 * (\sum_{i=0}^{p} |V_i| * |E_i|)) + (\sum_{i=0}^{p} |V_i|))$ which can be simplified in $O(\#iter * (n^6 + n^4 * (\sum_{i=0}^{p} |V_i|^2) + n^2 * (\sum_{i=0}^{p} |V_i| * |E_i|)))$.

While the memory cost depends on the space needed to store all the representations, which is $O(n^2 * \sum_{i=0}^{p} |V_i|^2)$. ■ [Proof of Theorem 1]

Despite the execution time and the memory cost have cubic and quadratic components in the sizes of the graphs in the partitioning PG, if we can assume the number of nodes $|V_i|$ bounded for each graph $G_i \in PG$, then the execution time and the memory of the Algorithm 1 are just linear in the number of partitions. In the next section, we provide a structural partitioning algorithm that by removing a minimal number of edges transforms a generic graph into a partitioned graph such that each partition is a connected component and the number of nodes for each partition is bounded. This makes 2FWL-SIRGN scalable even on medium and large size graphs.

B. Structural Graph Partitioning

To mitigate the increased computational overhead of the `2FWL-SIRGN method`, we propose a novel structural graph partitioning algorithm, detailed in Algorithm 2. This approach distinguishes itself from prior graph partitioning algorithms primarily through the integration of a structural representation learning technique.

The algorithm begins by initializing the structural representation embeddings, denoted as *emb*, for the graph G (lines 1-2). To achieve this, we utilize SIRGN [6], for its demonstrated capability in capturing complex graph structures, its scalability to large graphs, and is relatively low computational cost. After generating the SIRGN structural node embeddings *emb*, we apply a simple method, `GroupEdges` which rounds the embeddings of *emb* to the 6th decimal then for each node with the same structural embedding, a group $group_i$ containing the edge list a_i, b_i for each node a_i where an edge exists to node b_i. Once grouped, $edge\_group = \forall E_i \in G_i, \quad \text{for } i =$ #nodePerPartition will contain all the edges E in the graph G split in the same node structural group.

The next goal to create the structural partition is to utilize $PageRank$ [22] (line 4) on the graph G, creating a list of PageRank values $page\_rank$ = For each node V in graph G do PageRank(V). We can then pass the $edge_groups$ and $PageRank$ [22] to CalculateGroupScores to calculate the $PageRank$ value for each $group\ a, b$ in $edge_group$, the function enumerates the list of groups in $edge_group$ and for each *node* in a *group*, adds its corresponding $page_rank$ value $groupscores$(line 28-38). This representation allows efficient summation of $PageRank$ values for all groups and subsequent sorting based on their total scores, while preserving the list of edges in each group. After all groups have been scored, they are sorted in descending order, stored as $sorted\_edge\_groups$(line 6). With the groups ranked and sorted, the function `CreateCluster` (lines 10-27) is called to generate the final list of graph partitions, denoted as PG.

The `CreateClusters` function takes as input the $sorted\_edge\_groups$ from the previous step and the $\#nodePerPartion$, which represents the maximum number of nodes allowed in each partition. A binary search is then employed on $sorted\_edge\_groups$ (lines 13-23) to efficiently reduce the size of the $sorted\_edge\_groups$, until the maximum number of `ConnectedComponents`, PG, in the left split of $sorted\_edge\_groups$ is less than the $\#nodePerPartition$. Since $sorted\_edge\_groups$ is in descending order, it only traverses the left of the $sorted\_edge\_groups$.

In the following, we show the execution time and the memory cost of Algorithm 2.

Theorem 2: Given a graph $G = (V.E)$ and a representation size n, the Structural Graph Partitioning, defined in Algorithm 2, has as execution time $O(\text{SIR-GN\_}Execution\_Time(G) + Page\_Rank\_Execution\_Time(G) + (|E| + |V|) \log |E|)$ and a memory cost of $O(|V| * n + |E|)$

Proof of Theorem 2: To prove the execution time in Theorem 2, we identify the cost for each of the code lines in Algorithm 2 as follows. Line 1 is the cost of the SIR-GN approach that we denote as SIR-GN_$Execution_Time(G)$. Line 3 is the cost of creating groups of edges by using the structural embeddings provided by SIR-GN. This operation can be done with a tree index structure that makes the execution time $|E| * n$ which is upper-bounded by the execution time of SIR-GN. Line 4 is the cost of the Page Rank, denoted as $Page\_Rank\_Execution\_Time(G)$, and line 5 is just $O(|E|)$. Line 6 sorts the vector of groups which has size at most $|E|$ and the n the execution time is $O(|E| \log |E|)$. Ultimately, line 7 to create the partitions performs a binary search procedure over the sorted groups of edges. The binary search will perform $O(\log |E|)$ iterations and each iteration has an execution time $O(|V|+|E|)$ because of the connected components computation (line 16). This binary search has a total execution time of $O(|E| + |V|) \log |E|)$.

The strategy of adding nodes to each partition, rather than removing edges, offers two significant benefits. First, the most structurally important nodes are prioritized, thereby reducing the risk of losing key structural information. Second,

Algorithm 2 Structural Graph Partitioning

function StructuralGraphPartitioning(G, n, #nodePerPartion)
  $emb \leftarrow \text{SirGN}(G, n)$
  $edge\_groups \leftarrow \text{GroupEdges}(emb)$ ▷ This preserves the structural property in the partitions
  $page\_rank \leftarrow \text{PageRank}(G)$
  $groupscores \leftarrow \text{CalculateGroupScores}(edge\_groups, page\_rank)$
  $sorted\_edge\_groups \leftarrow \text{SortGroups}(groupscores, edge\_groups)$
  $PG \leftarrow \text{CreatePartitions}(sorted\_edge\_groups, \#nodePerPartion)$
  return PG
end function

function CreatePartitions(sorted_edge_groups, #nodePerPartion)
  $lower = 0$
  $upper = |sorted\_edge\_groups| - 1$
  while $lower > upper$ **do**
    $middle = round((lower + upper)/2)$
    $left\_group = sorted\_edge\_group[: middle]$ ▷ A graph using all the edges inside the groups from 0 to $middle$
    $PG = ConnectedComponents(left\_group)$
    $maxNode = max_{G_a=(V_a,E_a)\in PG}|V_a|$
    if $maxNode \leq \#nodePerPartion$ **then**
      $lower = middle$
    else
      $upper = middle$
    end if
  end while
  $left\_group = sorted\_edge\_group[: lower]$
  $PG = ConnectedComponents(left\_group)$
  return PG
end function

function CalculateGroupScores(edge_groups, page_rank)
  $groupscores \leftarrow []$
  for $group$ in $edge\_groups$ **do**
    $score \leftarrow 0$
    for (a, b) in $group$ **do**
      $score = score + page\_rank[a] + page\_rank[b]$
    end for
    $groupscores.append(score)$
  end for
  return $groupscores$
end function

adding node groups to each partition instead of evaluating the significance of each edge individually simplifies the process, as it avoids the computational complexity of deciding which edges to remove.

Then, the cumulative cost of Algorithm 2 is
$O(\text{SIR-GN}\_Execution\_Time(G) + Page\_Rank\_Execution\_Time(G) + (|E| + |V|)\log|E|)$.

While, the memory cost depends on the space needed to store all the representations plus the space to store the edge group list, which is in total $O(|V| * n + |E|)$. ■

From the Theorem 2, we can expect that the partitioning algorithm can scale on large graphs and produce the desired bounded partitions required for 2FWL-SIRGN.

IV. Experiment and Results

A. Datasets

The datasets chosen for our experiments are well-known in the field of graph learning, with established baselines, and have a wide range of sizes. This is beneficial for testing the optimization of our methods and compare against the related works. The datasets Mutag, Enzyme, PTC, FM, NCI1, NCI109, Proteins, IMDb binary, and IMDb multi have gained prominence as essential benchmarks for evaluating the efficacy and performance of various graph classification methods. Graph classification, a central area of investigation within the area

TABLE I: Details about the datasets used in experiments

Method	Nodes	Node Classes	Edges	Graphs	Graph Classes
Mutag	3371	7	7442	188	2
Enzymes	19580	3	74564	600	6
PTC_FM	4925	18	10110	349	2
NCI1	122747	37	265506	4110	2
NCI109	122494	38	265208	4127	2
Proteins	43471	3	162088	1113	2
IMDB-B	19773	0	386124	1000	2
IMDB-M	19502	0	395612	1500	3
Texas University	183	1703	325	1	0
Cornell University	183	1703	298	1	0
Wisconsin University	251	1703	515	1	0
Squirrel	5201	2089	217073	1	0
Film	7600	5	33391	1	0

of graph representation learning, involves categorizing graphs into predefined classes or categories. The datasets encompass a diverse range of application domains, including chemical compounds, protein structures, and movie databases, thereby encompassing a broad spectrum of real-world scenarios. The datasets' influence is accentuated by their widespread adoption within notable institutions such as Texas University, Cornell University, and Wisconsin University. Researchers at these academic hubs have harnessed the datasets as foundational elements in their explorations of graph classification techniques. The datasets have contributed significantly to studies aiming to uncover the intricate relationships between graph structures and classification outcomes.

B. Experimental Setup

he experimental setup for evaluating 2FWL-SIRGN was designed to demonstrate its scalability, efficiency, and representational quality. To assess the model's capabilities, we conducted a series of experiments on both real-world and synthetic datasets, chosen to highlight the diversity and challenges of large-scale graph representation learning.

We evaluated our proposed 2FWL-SIRGN model using several well-known datasets in the field of graph learning, including MUTAG [23], PTC [24], NCI1 [25], NCI109 [26], Collab [27], Wisconsin [28], Cornell [29], Texas [30] and PROTEINS These datasets were selected for their recognition and relevance, ensuring the validity and credibility of our theoretical framework.

C. Evaluation Metrics

We used multiple evaluation metrics to assess the performance of our model, including accuracy, F1 score, and computational efficiency. Additionally, we conducted a scalability analysis by running our model on both small and large clusters using Amazon Web Services (AWS).

Our model was compared against several competitive algorithms, including:

1) **Matrix Factorization**: GraphWave [31]
2) **Random Walk**: DeepWalk [3] and Struc2Vec [5]
3) **Neural Networks**: LINE [32], GCN [15], and GAT [16]
4) **WL Higher Order**: sparsewl [8], KNN [11]

D. Results

The results of our experiments demonstrate the superior performance of the 2FWL-SIRGN model. As shown in Table II, our model achieved higher accuracy and F1 scores across multiple datasets compared to the competitive algorithms. Notably, the 2FWL-SIRGN model showed a significant improvement in capturing complex structural information, as evidenced by its performance on the MUTAG and NCI1 datasets. he experimental results are presented in Tables 1 and 2, as well as Figures 1 and 2.

1) **Node Classification Performance**: As shown in Table 1, 2FWL-SIRGN outperformed all baselines in terms of node classification accuracy, achieving a notable improvement of up to 15% over SIR-GN. This demonstrates the model's ability to capture structural nuances that are crucial for distinguishing between nodes in complex networks.
2) **Efficiency and Scalability**:Figure 1 illustrates the runtime performance of 2FWL-SIRGN compared to other methods. Our model exhibited a substantial reduction in computational time due to the structural partitioning approach, which reduced the size of subgraphs processed in each iteration. Additionally, memory usage was significantly lower, as shown in Figure 2, affirming the scalability of our approach for large graphs.

1) Resistance to Overfitting: A recurring challenge in the area of graph representation learning is the propensity for models to overfit, particularly when dealing with complex and high-dimensional data. Our experiments with SIR-GN and 2FWL-SIRGN have demonstrated a remarkable resistance to overfitting, setting them apart from conventional GNN-based methods.

To test the risk of overfitting in our algorithm, we conducted an additional experiment beyond our standard evaluations. In this experiment, we trained the models on one dataset and tested them on a previously unseen, foreign dataset with a similar graph structure and the same graph size. This cross-dataset validation aimed to assess the models' ability to generalize and accurately identify structural patterns without overfitting to the specific characteristics of the training dataset. Unlike the competitive algorithms, which typically exhibit a

TABLE II: Combined results from various methods across different datasets, including our methods for each experiment.

Method	MUTAG	PTC	PROTEIN	NCI1	NCI109	IMDB-B	IMDB-M	Collab	Wisconsin	Cornell	Texas
GSN	86.07	58.65	68.17	73.48	73.48	70.02	47.78	36.92	52.10	53.37	51.52
SIN	67.32	**86.94**	83.16	51.98	79.3	52.14	43.98	35.21	53.47	53.74	50.92
DGCNN	83.73	43.57	74.41	72.24	73.18	68.81	47.23	32.61	68.21	70.35	75.31
PSCN	79.97	61.24	71.35	78.24	72.62	58.87	35.38	29.61	68.98	73.39	71.19
GAT	82.97	37.29	63.54	75.21	62.24	46.35	42.61	42.88	66.65	58.87	62.21
GCN	88.61	65.65	70.21	80.84	82.64	72.87	50.78	43.21	60.21	58.21	61.21
sparsewl	85.74	77.61	84.06	90.49	89.73	75.44	**62.92**	43.38	71.91	76.22	70.12
KNN	91.48	72.58	77.77	89.81	84.71	73.20	50.74	**50.07**	71.66	70.94	68.12
FSGNN	87.61	60.39	70.21	80.31	81.64	72.64	50.87	37.21	78.16	77.91	76.21
ACMII	88.21	60.69	70.81	80.89	82.54	72.97	53.35	47.54	**78.21**	78.54	77.15
SIR-GN	91.60	58.40	71.38	74.39	74.00	73.08	47.42	52.23	49.81	51.81	53.54
2FWL-SIRGN	**93.12**	84.78	**85.21**	**92.87**	**90.11**	**89.56**	59.20	49.44	77.51	**78.64**	**78.24**

significant drop in performance when faced with unfamiliar data, our approach maintained higher accuracy and consistency. This is shown in Table III This outcome indicates that our model effectively captures the underlying graph structures and is not overly reliant on the idiosyncrasies of the training data, thereby demonstrating superior resistance to overfitting.

1) **Regularization Techniques:** Both SIR-GN and 2FWL-SIRGN incorporate advanced regularization techniques that constrain the model complexity, thereby reducing the likelihood of overfitting.
2) **Data Augmentation:** Our methods employ data augmentation strategies that enhance the model's ability to generalize well to unseen data.
3) **Early Stopping Criteria:** We implemented early stopping criteria based on validation loss, which prevents the model from learning the noise in the training data, thus mitigating overfitting.
4) **Cross-Validation:** Rigorous k-fold cross-validation was employed to ensure that the performance metrics are reliable and not merely a result of a favorable data split.

With using similar datasets we can use one dataset to train the models and the unknown dataset to test the trained models to fully see if the algorithms can effectively learn the structural patterns within a graph.

2) Comparative Analysis: When transposed with traditional GNN-based methods, both SIR-GN and 2FWL-SIRGN exhibit superior performance across multiple datasets, as evidenced in Table. II Notably, 2FWL-SIRGN achieved an accuracy of 95 ± 3.5 on the MUTAG dataset, outperforming all other methods.

E. Scalability Analysis

Our scalability analysis revealed that the Structural Graph Partition algorithm effectively reduces computational costs, making the 2FWL-SIRGN model feasible for large-scale graphs. When tested on a large cluster, our model maintained high accuracy while significantly decreasing runtime. To understand the contributions of different components within 2FWL-SIRGN, we conducted an ablation study by systematically removing or modifying key elements, such as the structural partitioning algorithm and the 2-dimensional Weisfeiler-Lehman extension.

1) **Without Structural Partitioning:** Both SIR-GN and 2FWL-SIRGN incorporate advanced regularization techniques that constrain the model complexity, thereby reducing the likelihood of overfitting.
2) **Without 2-FWL Extension:** Reverting to a simpler Weisfeiler-Lehman framework resulted in decreased node classification accuracy by approximately 10%, indicating the importance of capturing higher-order structural information.

V. Discussion

In the field of graph neural networks, the ability to detect and analyze complex structures within graph data is crucial. The Weisfeiler-Lehman (WL) test and its variants have been instrumental in this regard. However, the traditional WL test and its direct extension, the k-dimensional Weisfeiler-Lehman (kWL) test, have limitations. They can miss important patterns due to their local nature and may not scale well or may overfit when used in a machine learning setting. [31]

To address these issues, researchers have proposed the folklore Weisfeiler-Lehman (k-FWL) test shown in algorithm 1, a variant of the WL test that is more efficient and less prone to overfitting. The k-FWL test operates by considering k-tuples of nodes, similar to the k-WL test, but with a slightly different update definition that makes it more computationally efficient.

We chose to utilize the 2FWL test in the SIR-GN model. This decision was based on the understanding that the 2FWL test is as capable of detecting cycles in graph data as the 3-WL test while being more efficient than the 2WL test. This claim is supported by the findings presented in "A Short Tutorial on the Weisfeiler-Lehman Test And Its Variants", which highlights the discriminating power of k-FWL being equivalent to the one of (k-1)-WL for k 3.

Moreover, the documentation provides an example in which the 2FWL test successfully distinguishes between two regular non-isomorphic graphs, in which both the classical WL test and the 2WL test fail. This illustrates the power of the FWL test to capture complex structures in graph data, further supporting our decision to use the 2FWL test with SIR-GN.

TABLE III: Trained on the left and tested on the right dataset.

Method	Wisc:Tex	Tex:Wis	Tex:Corn	Corn:Tex	Wis:Corn	Corn:Wis
GAT	30.54	32.61	29.67	31.58	29.67	27.81
GCN	31.66	33.81	32.45	34.23	30.55	31.84
GCN2	35.04	36.31	36.68	36.21	34.68	35.21
FSGNN	39.54	41.81	38.41	40.21	38.75	38.51
ACMII	38.63	40.21	39.73	42.91	37.15	36.54
SIR-GN	48.21	50.28	47.51	45.21	45.81	44.61
2FWL-SIRGN	**57.11**	**56.21**	**49.92**	**48.91**	**47.2**	**49.21**

VI. Conclusion

In this paper, we introduced 2FWL-SIRGN, a novel graph representation learning approach that integrates the 2-dimensional Folklore Weisfeiler-Lehman test with a structural partitioning algorithm to address scalability and representational challenges in large-scale graph learning. Our experimental results demonstrate that 2FWL-SIRGN not only achieves superior node classification accuracy compared to state-of-the-art baselines but also offers substantial improvements in computational efficiency.

The combination of structural partitioning and the enhanced Weisfeiler-Lehman test allows our model to effectively capture intricate structural patterns in graphs while maintaining scalability. This makes 2FWL-SIRGN particularly suitable for real-world applications involving large and complex networks, such as social network analysis, botnet detection, and bioinformatics.

Our experimental results demonstrated the superior performance and scalability of the 2FWL-SIRGN model across various datasets. This approach not only captures intricate structural information but also scales effectively to large graphs, making it suitable for real-world applications in fields such as social network analysis, cybersecurity, and bioinformatics.

Future work will focus on further optimizing the Structural Graph Partition algorithm and exploring additional applications of the 2FWL-SIRGN model. We also plan to extend our approach to dynamic and temporal graphs, enabling the capture of time-dependent structural patterns.

Acknowledgment

This research was funded by the National Centers of Academic Excellence in Cybersecurity grant H98230-22-1-0300, which is part of the National Security Agency.

References

[1] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE transactions on neural networks and learning systems*, vol. 32, no. 1, pp. 4–24, 2020.

[2] W. L. Hamilton, *Graph representation learning*. Morgan & Claypool Publishers, 2020.

[3] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2014, pp. 701–710.

[4] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, 2016, pp. 855–864.

[5] L. F. Ribeiro, P. H. Saverese, and D. R. Figueiredo, "struc2vec: Learning node representations from structural identity," in *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, 2017, pp. 385–394.

[6] M. Joaristi and E. Serra, "Sir-gn: A fast structural iterative representation learning approach for graph nodes," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 15, no. 6, pp. 1–39, 2021.

[7] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" *arXiv preprint arXiv:1810.00826*, 2018.

[8] C. Morris, G. Rattan, and P. Mutzel, "Weisfeiler and leman go sparse: Towards scalable higher-order graph embeddings," *Advances in Neural Information Processing Systems*, vol. 33, pp. 21 824–21 840, 2020.

[9] R. A. Rossi, N. K. Ahmed, and E. Koh, "Higher-order network representation learning," in *Companion Proceedings of the The Web Conference 2018*, 2018, pp. 3–4.

[10] C. Yang, M. Liu, V. W. Zheng, and J. Han, "Node, motif and subgraph: Leveraging network functional blocks through structural convolution," in *2018 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*. IEEE, 2018, pp. 47–52.

[11] C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe, "Weisfeiler and leman go neural: Higher-order graph neural networks," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, no. 01, 2019, pp. 4602–4609.

[12] N. Shervashidze, P. Schweitzer, E. J. Van Leeuwen, K. Mehlhorn, and K. M. Borgwardt, "Weisfeiler-lehman graph kernels." *Journal of Machine Learning Research*, vol. 12, no. 9, 2011.

[13] M. Grohe and M. Otto, "Pebble games and linear equations," *The Journal of Symbolic Logic*, vol. 80, no. 3, pp. 797–844, 2015.

[14] R. Sato, "A survey on the expressive power of graph neural networks," *arXiv preprint arXiv:2003.04078*, 2020.

[15] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[16] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.

[17] R. v. d. Berg, T. N. Kipf, and M. Welling, "Graph convolutional matrix completion," *arXiv preprint arXiv:1706.02263*, 2017.

[18] M. Niepert, M. Ahmed, and K. Kutzkov, "Learning convolutional neural networks for graphs," in *International conference on machine learning*. PMLR, 2016, pp. 2014–2023.

[19] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in Neural Information Processing Systems*, vol. 27, 2014, pp. 3104–3112.

[20] C. Damke, V. Melnikov, and E. Hüllermeier, "A novel higher-order weisfeiler-lehman graph convolution," in *Asian Conference on Machine Learning*. PMLR, 2020, pp. 49–64.

[21] H. Abdi and L. J. Williams, "Principal component analysis," *Wiley interdisciplinary reviews: computational statistics*, vol. 2, no. 4, pp. 433–459, 2010.

[22] P. Rozenshtein and A. Gionis, "Temporal pagerank," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2016, pp. 674–689.

[23] A. K. Debnath, R. L. Lopez de Compadre, G. Debnath, A. J. Shusterman, and C. Hansch, "Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity," *Journal of medicinal chemistry*, vol. 34, no. 2, pp. 786–797, 1991.

[24] D. Q. Nguyen, T. D. Nguyen, and D. Phung, "Universal graph transformer self-attention networks," in *Companion Proceedings of the Web Conference 2022*, 2022, pp. 193–196.

[25] N. Wale, I. A. Watson, and G. Karypis, "An extensive comparison of recent classification tools applied to the ncicancer screen data," *Journal of Chemical Information and Modeling*, vol. 48, no. 3, pp. 644–654, 2008.

[26] ——, "An extensive comparison of recent classification tools applied to the ncicancer screen data," *Journal of Chemical Information and Modeling*, vol. 48, no. 3, pp. 644–654, 2008.

[27] P. Yanardag and S. V. N. Vishwanathan, "Collab dataset," *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pp. 1365–1374, 2015.

[28] J. Pei, L. Tang, and B. Zhao, "Networks with node attributes: Using annotated graphs for classification," in *Proceedings of the 9th International Conference on Data Mining (ICDM)*, 2009, pp. 1085–1090.

[29] ——, "Networks with node attributes: Using annotated graphs for classification," in *Proceedings of the 9th International Conference on Data Mining (ICDM)*, 2009, pp. 1085–1090.

[30] ——, "Networks with node attributes: Using annotated graphs for classification," in *Proceedings of the 9th International Conference on Data Mining (ICDM)*, 2009, pp. 1085–1090.

[31] C. Donnat, M. Zitnik, D. Hallac, and J. Leskovec, "Learning structural node embeddings via diffusion wavelets," in *International ACM Conference on Knowledge Discovery and Data Mining (KDD)*, vol. 24, 2018.

[32] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "Line: Large-scale information network embedding," in *Proceedings of the 24th international conference on world wide web*, 2015, pp. 1067–1077.