A Simple yet Effective Method for Graph Classification

Junran Wu[1*], **Shangzhe Li**[1,2*], **Jianhao Li**[1], **Yicheng Pan**[1†] and **Ke Xu**[1†]

[1]State Key Lab of Software Development Environment, Beihang University, Beijing, 100191, China
[2]School of Mathematical Science, Beihang University, Beijing 100191, China
{wu_junran, shangzheli, lijianhao, yichengp, kexu}@buaa.edu.cn

Abstract

In deep neural networks, better results can often be obtained by increasing the complexity of previously developed basic models. However, it is unclear whether there is a way to boost performance by decreasing the complexity of such models. Intuitively, given a problem, a simpler data structure comes with a simpler algorithm. Here, we investigate the feasibility of improving graph classification performance while simplifying the learning process. Inspired by structural entropy on graphs, we transform the data sample from graphs to coding trees, which is a simpler but essential structure for graph data. Furthermore, we propose a novel message passing scheme, termed hierarchical reporting, in which features are transferred from leaf nodes to root nodes by following the hierarchical structure of coding trees. We then present a tree kernel and a convolutional network to implement our scheme for graph classification. With the designed message passing scheme, the tree kernel and convolutional network have a lower runtime complexity of $O(n)$ than Weisfeiler-Lehman subtree kernel and other graph neural networks of at least $O(hm)$. We empirically validate our methods with several graph classification benchmarks and demonstrate that they achieve better performance and lower computational consumption than competing approaches.

1 Introduction

Over the years, deep learning has achieved great success in perception tasks, such as recognizing objects or understanding language, which are difficult for traditional machine learning methods [Bengio *et al.*, 2021]. To further enhance performance, research efforts have generally been devoted to designing more complex models based on previously developed basic ones; such improvements include increasing model depths (e.g., ResNet [He *et al.*, 2016]), integrating more complicated components (e.g., Transformer [Vaswani *et al.*, 2017]) or even both (e.g., GPT3 [Brown *et al.*, 2020] and NAS [Liu *et al.*, 2020]). However, little work has focused on the research direction of boosting performance through simplifying the basic model learning process.

Similarly, there are many interesting tasks involving graphs that are hard to learn with normal deep learning models, which prefer data with a grid-like structure. Thus, graph neural networks (GNNs) have recently been ubiquitous within deep learning for graphs, and achieve great success in various domains because of their ability to model structural information [Hamilton *et al.*, 2017; Zhang *et al.*, 2018; Xu *et al.*, 2019]. In order to pursuit more superior performance, various more complex components have also been developed and integrated into basic GNNs [Veličković *et al.*, 2018; Ying *et al.*, 2018; Zhang *et al.*, 2018]. In this context, the improvement in performance comes at the price of model complexity, analogous to routines in deep learning. Here, we investigate the feasibility of improving graph classification performance by simplifying model learning. Generally, given a problem, a simpler data structure comes with a simpler algorithm. Motivated by structural entropy [Li and Pan, 2016], a metric designed to assess the structural information of a graph, the essential structure of a graph can be decoded by this metric as a measure of the complexity of its hierarchical structure. Therefore, as shown in Fig. 1, we propose an algorithm to simplify the given data sample of a graph by transforming it into a corresponding coding tree that reflects the hierarchical organization of data, in which the crucial structural information underlying the graph can be retained in the coding tree with minimal structural entropy.

Based on the simplified coding trees, we propose a novel feature combination scheme for graph classification, termed hierarchical reporting. In this scheme, we transfer features from leaf nodes to root nodes based on the hierarchical structures of the associated coding trees. For two kinds of basic learning algorithms (i.e., kernel-based methods and GNNs), we propose two corresponding simplified learning algorithms; put differently, we present an implementation of our scheme in a tree kernel and a convolutional network, denoted the Weisfeiler-Lehman coding tree (WL-CT) kernel and hierarchical reporting network (HRN), to perform graph classification. The tree kernel follows the label propagation in the Weisfeiler-Lehman (WL) subtree kernel but has a lower runtime complexity of $O(n)$. HRN is an implementation of

*Equal Contribuation.
†Correspondence to: Yicheng Pan, Ke Xu.

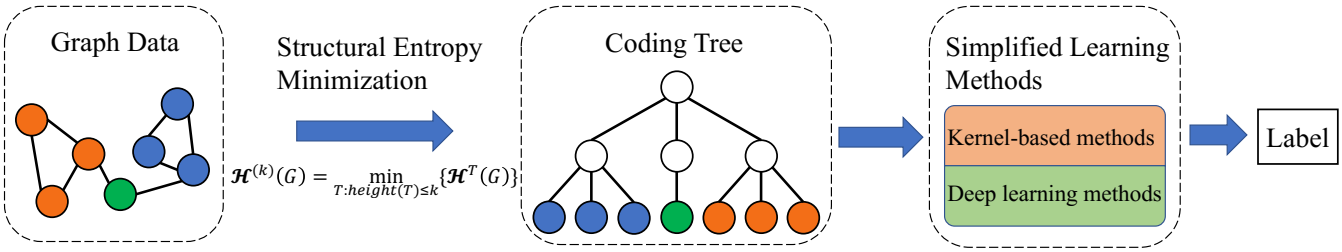

Figure 1: **Overview of graph simplification via structural entropy minimization.** Structural entropy guides the simplification for graphs to decode the underlying structure, and several learning algorithms can benefit from it.

our tree kernel in the deep learning field. Finally, we empirically validate our WL-CT kernel and HRN on various graph classification datasets. Our tree kernel surpasses the state-of-the-art kernel-based methods and even outperforms GNNs on several benchmarks. Our HRN also achieves state-of-the-art performance on most benchmarks.

We list our contributions in this work as follows:

- We present a novel direction to boost the performance of learning models while reducing complexity.
- With structural entropy minimization, we optimize the given data sample from graphs to coding trees, which are much simpler data structures that also retain the crucial structural information of graphs.
- We develop two efficient learning algorithms, i.e., WL-CT and HRN, and empirically present their discriminative power on many graph classification benchmarks.

2 Related Work

GNNs have achieved state-of-the-art results on various tasks with graphs, such as node classification [Veličković *et al.*, 2018], link prediction [Zhang and Chen, 2018] and graph classification [Hamilton *et al.*, 2017; Zhang *et al.*, 2018; Xu *et al.*, 2019]. In this work, we devote our attention to graph classification scenarios.

Graph classification involves identifying the characteristics of an entire graph and is ubiquitous in a variety of domains, such as social network analysis [Backstrom and Leskovec, 2011], chemoinformatics [Duvenaud *et al.*, 2015], and bioinformatics [Borgwardt *et al.*, 2005]. In addition to previous techniques such as graph kernels [Shervashidze *et al.*, 2011], GNNs have recently emerged and become a popular way to handle graph-related tasks due to their effective and automatic extraction of graph structural information [Hamilton *et al.*, 2017; Zhang *et al.*, 2018; Xu *et al.*, 2019]. To address the limitations of various GNN architectures, the graph isomorphism network (GIN) [Xu *et al.*, 2019] was presented in theoretical analyses regarding the expressive power of GNNs in terms of capturing graph structures. All GNNs are broadly based on a recursive message passing (or neighborhood aggregation) scheme, where each node recursively updates its feature vector with the "message" propagated from neighbors [Gilmer *et al.*, 2017; Xu *et al.*, 2019]. The feature vector representing an entire graph for graph classification can be obtained by a graph pooling scheme [Ying *et al.*, 2018], such as the summation of all node feature vectors of the graph. Accordingly, much effort has been devoted to exploiting graph pooling schemes, which are applied before the final classification step [Zhang *et al.*, 2018; Ying *et al.*, 2018]. All these pooling methods help models achieve state-of-the-art results but increase the model complexity and the volume of computations.

Structural entropy is a measure of the complexity of the hierarchical structure of a graph [Li and Pan, 2016]. The structural entropy of a graph is defined as the average length of the codewords obtained under a specific coding scheme for a random walk. That is, when a random walk takes one step from node u to node v, the codeword of the longest common ancestor of u and v on the coding tree, which is also their longest common prefix, is omitted. This shortens the average codeword length. Equivalently, the uncertainty of a random walk is characterized by this value, which is the origin of the term structural entropy. The coding tree of a graph that achieves the minimum structural entropy indicates the optimal hierarchical structure. Furthermore, two- and three-dimensional structural entropy, which measure the complexity of two- and three-level hierarchical structures, respectively, have been applied in bioinformatics [Li *et al.*, 2018], medicine [Li *et al.*, 2016b], the structural robustness and security of networks [Li *et al.*, 2016a], etc. For tasks regarding graphs, structural entropy can be used to decode the essential structure as a measure of the complexity of its hierarchical structure.

Thus far, there has been little attention paid to the model promotion and efficiency rising through data sample simplification. In this paper, with the guidance of structural entropy, we investigate the feasibility of improving graph classification performance with lower model complexity.

3 Methodology

In this section, we first introduce an algorithm for graph simplification in which we decode the essential structure of an input graph to the corresponding coding tree by minimizing its structural entropy. Based on the coding trees, we propose a tree kernel and a corresponding implementation in a deep learning model for graph classification. We elaborate on them in the following.

3.1 Graph Simplification via Structural Entropy Minimization

According to the definition in [Li and Pan, 2016], given a graph $G = (V, E)$ and a coding tree T for G, where $|V| = n$

and $|E| = m$, the structural entropy of G on T is defined as

$$\mathcal{H}^T(G) = -\sum_{v_t \in T} \frac{g_{v_t}}{vol(V)} \log \frac{vol(v_t)}{vol(v_t^+)}, \tag{1}$$

where v_t is a nonroot node in T that also represents a node subset $\subset V$, v_t^+ refers to the parent node of v_t, g_{v_t} is the number of edges with exactly one endpoint in v_t, and $vol(V)$ and $vol(v_t)$ are the sums of degrees of nodes in V and v_t, respectively. The structural entropy of G is defined to be the minimum entropy among all coding trees, and it is denoted by $\mathcal{H}(G) = \min_T\{\mathcal{H}^T(G)\}$.

To formulate a natural coding tree with a certain height, the k-dimensional structural entropy of G for any positive integer k is the minimum value among all coding trees with heights of at most k:

$$\mathcal{H}^{(k)}(G) = \min_{T:\text{height}(T)\leq k} \{\mathcal{H}^T(G)\}. \tag{2}$$

In this paper, to realize the simplification of given graphs with a certain k-dimensional structural entropy, we propose a greedy algorithm to compute the k-dimensional coding tree with minimum structural entropy in Algorithm 1. Specifically, we first build a full-height binary coding tree from an initial coding tree. In this phase, we iteratively merge two nonroot nodes, which can minimize the structural entropy at each step. Then, to form the k-dimensional coding tree T for G, we repeatedly delete one edge from T, which can minimize the structural entropy restoration at each step. Finally, a coding tree T with a certain height k for G can be obtained, where $T = (V_T, E_T)$, $V_T = (V_T^0, \ldots, V_T^k)$ and $V_T^0 = V$.

The time complexity of k-dimensional coding tree is $O(h_{max}(m \log n + n))$, where h_{max} is the maximum height of coding tree T during the generation of the full-height binary coding tree. Since the minimized structural entropy tends to construct balanced coding trees, h_{max} is generally of order $O(\log n)$. Furthermore, the number of edges of graph is generally greater than the number of nodes, thus the runtime of Algorithm 1 almost scales linearly in the number of edges.

Algorithm 1 k-dimensional coding tree on structural entropy

Input: a graph $G = (V, E)$, a positive integer $k > 1$
Output: a k-dimensional coding tree T

Initialize T with one root node v_r and all nodes in V as leaves;
Define the function $merge(v_i, v_j)$ for T to insert a new node between v_r and (v_i, v_j);
Define the function $delete(v_p, v_c)$ for T to delete v_c from T and link $v_c.children$ to v_p, where $v_c.parent = v_p$;
// Generate the full-height binary coding tree T for G;
while $|v_r.children| > 2$ **do**
    Select v_i and v_j from $v_r.children$, which make $argmax_{(v_i,v_j)}\{\mathcal{H}^T(G) - \mathcal{H}^{T_{merge(v_i,v_j)}}(G)\}$;
    Execute $merge(v_i, v_j)$;
end while
if Height(T) $\leq k$ **then**
    return T;
end if
// Compress T to height k;
while Height(T) $> k$ **do**
    Select v_p and v_c from T, which make $argmin_{(v_p,v_c)}\{\mathcal{H}^{T_{delete(v_p,v_c)}}(G) - \mathcal{H}^T(G)\}$;
    Execute $delete(v_p, v_c)$;
end while
return T;

3.2 A Tree Kernel for Graph Classification

To perform classification on coding trees, following the construction of the WL subtree kernel [Shervashidze *et al.*, 2011], we propose a novel tree kernel that measures the similarity between coding trees named the WL-CT kernel. The key difference between the two kernels is the label propagation scheme, where we develop a *hierarchical reporting* scheme to propagate labels from child nodes to their parents based on the hierarchical structures of coding trees. Finally, our tree kernel adopts the counts of node labels of the entire coding tree as the feature vector of the original graph.

Hierarchical reporting. The key idea of this scheme is to assign labels to nonleaf nodes by aggregating and sorting the labels from their child nodes and then to compress these sorted label sets into new and short labels. Labels from the leaf nodes are iteratively propagated to the root node, which means that the iteration time of this scheme is determined by the height of the coding tree, and naturally avoids the convergence problem of WL subtree kernel.

Definition 1. *Let T_1 and T_2 be any two coding trees with the same height k. There exists a set of letters $\Sigma^i \in \Sigma$, which are node labels appearing at the i-th ($i < k$) layer of T_1 or T_2 (i.e., the nodes with height i are assigned labels with hierarchical reporting). Σ^0 is the set of leaf node labels of T_1 and T_2. Assume that any two Σ^i are disjoint, and every $\Sigma^i = \{b_1^i, \ldots, b_{|\Sigma^i|}^i\}$ is ordered without a loss of generality. We define a function $c^i : \{T_1, T_2\} \times \Sigma^i \to \mathbb{B}$ such that $c^i(T_1, b_j^i)$ counts the number of letter b_j^i in coding tree T_1.*

The tree kernel on the two trees (T_1 and T_2) with height k after the root nodes are assigned labels is defined as:

$$WL\text{-}CT(T_1, T_2) = < \varphi_{CT}(T_1), \varphi_{CT}(T_2) >, \tag{3}$$

where

$$\varphi_{CT}(T_1) = (c^0(T_1, b_1^0), \ldots, c^0(T_1, b_{|\Sigma^0|}^0), \ldots, c^k(T_1, b_1^k), \ldots, c^k(T_1, b_{|\Sigma^k|}^k))$$

and

$$\varphi_{CT}(T_2) = (c^0(T_2, b_1^0), \ldots, c^0(T_2, b_{|\Sigma^0|}^0), \ldots, c^k(T_2, b_1^k), \ldots, c^k(T_2, b_{|\Sigma^k|}^k)).$$

Following the label counting process in the WL subtree kernel, our tree kernel is also designed to count the number of common labels in two coding trees.

Theorem 1. *The WL-CT kernel on two coding trees T_1 and T_2 with the same height k can be computed in time $O(n)$, which is much simpler than the WL subtree kernel ($O(hm)$) with h iterations on m edges [Shervashidze* et al.*, 2011] and is the method in graph classification with the lowest time complexity to the best of our knowledge [Wu* et al.*, 2020].*

Proof. Given a graph G, the runtime of the WL subtree kernel is $O(hm)$ for h iteration, as there are $O(m)$ elements in the multisets of a graph in each WL test iteration. Correspondingly, given a coding tree T for G, the time complexity of the WL-CT kernel is also the total number of elements in the multiset of the coding tree due to the similar label propagation scheme. In addition, the number of elements in the multiset is determined by the number of times that label propagation occurs, which is the edge number $|E| = m$. Thus, the time complexity of the WL-CT kernel is determined by the number of edges in T. As shown in Algorithm 1, the coding tree T with the most edges is the full-height binary coding tree T_B, i.e., $O(T) \leq O(T_B)$. The complexity on the full-height binary coding tree is calculated as:

$$\begin{aligned} O(T_B) &= O(|E_{T_B}|) \\ &= O(|V_{T_B}| - 1) \\ &= O(|V^0_{T_B}| + |V^1_{T_B}|, \ldots, +|V^{h_{max}}_{T_B}| - 1) \\ &< O(2|V^0_{T_B}|) \\ &= O(2|V|) \\ &= O(n). \end{aligned}$$

□

3.3 A Convolutional Network for Graph Classification

Based on our tree kernel, we develop a novel graph convolutional network, *HRN*, that generalizes the hierarchical reporting scheme to update the hidden features of nonleaf nodes for graph classification. HRN uses the hierarchical structure of coding tree and leaf node features X_v to learn the representation vector of the entire coding tree r_T. HRN follows the proposed hierarchical reporting scheme, where the representation of a nonleaf node is updated by aggregating the hidden features of its children. Formally, the i-th layer of HRN is

$$r^i_v = \text{MLP}^i \left(\sum\nolimits_{u \in \mathcal{C}(v)} r^{(i-1)}_u \right), \tag{4}$$

where r^i_v is the feature vector of node v with height i in the coding tree, $r^0_v = x_v$, and $\mathcal{C}(v)$ is a set of child nodes of v. As shown in Equation 4, we employ summation and multilayer perceptrons (MLPs) to perform hierarchical reporting in HRN. Sum aggregators have been theoretically proven to be injective over multisets and more powerful than mean and max aggregators [Xu *et al.*, 2019]. MLPs are capable of representing the compositions of functions because of the universal approximation theorem [Hornik *et al.*, 1989; Hornik, 1991].

For graph classification, the root node representation r^k_v can be naively employed as the representation of the entire coding tree r_T. However, as discussed in [Xu *et al.*, 2019], better results could be obtained from features in earlier iterations. To cover all hierarchical information, we employ hidden features from each height/iteration of the model. This is achieved by an architecture similar to the GIN [Xu *et al.*, 2019], where we model the entire coding tree with layer representations concatenated across all heights/layers of the HRN structure:

$$\begin{aligned} r_T = \text{CONCAT}(&\text{LAYERPOOL}(\{r^i_v | v \in V^i_T\}) \\ &| i = 0, 1, \ldots, k), \end{aligned} \tag{5}$$

where r^i_v is the feature vector of node v with height i in coding tree T and k is the height of T. In HRN, LAYERPOOL in Equation 5 can be replaced with the summation or averaging of all node vectors within the same iterations, which generalizes our tree kernel.

4 Experiments

We validate the feasibility of performance improving while simplifying model complexity by comparing the experimental results of the WL-CT kernel and HRN with those of the most popular kernel-based methods and GNNs on graph classification tasks [1].

Datasets. We conduct graph classification on five benchmarks: three social network datasets (IMDB-BINARY, IMDB-MULTI, and COLLAB) and two bioinformatics datasets (MUTAG and PTC) [Morris *et al.*, 2020] [2]. The data of the bioinformatic datasets and social network datasets differ in that the nodes in bioinformatics graphs have categorical labels that do not exist in social networks. Thus, the initial node labels for the tree kernel are organized as follows: the node degrees are taken as node labels for social networks, whereas the node degrees and node categorical labels are taken according to each bioinformatic dataset. Correspondingly, the initial node features of the HRN inputs are set to one-hot encodings of the node degrees for social networks and a combination of the one-hot encodings of the degrees and categorical labels for bioinformatic graphs. Table 1 summarizes the characteristics of the five employed datasets.

Configurations. Following [Xu *et al.*, 2019], 10-fold cross-validation is conducted, and we present the average accuracies achieved to validate the performance of our methods in graph classification [3] Regarding the configuration of our tree kernel, we adopt the C-support vector machine (C-SVM) [Chang and Lin, 2011] as the classifier and tune the hyperparameter C of the SVM and the height of the coding tree $\in [2, 3, 4, 5]$. We implement the classification program with an SVM from Scikit-learn, where we set another hyperparameter γ as *auto* for IMDB-BINARY and IMDB-MULTI and as *scale* for COLLAB, MUTAG and PTC, and we set the other hyperparameters as their default values.

For configuration of HRN, the number of HRN iterations is consistent with the height of the associated coding trees, which is also $\in [2, 3, 4, 5]$. All MLPs have 2 layers, as in the setting of the GIN. For each layer, batch normalization is applied to prevent overfitting. We utilize the Adam optimizer and set its initial learning rate to 0.01. For a better fit, the

[1] The code of the WL-CT kernel and HRN can be found at https://github.com/Wu-Junran/HierarchicalReporting.

[2] Considering the limitations of coding trees on disconnected graphs, we only adopt datasets that do not contain such graphs.

[3] Currently, there have been three kinds of configurations for experimental results; we elaborate on them in Appendix https://github.com/Wu-Junran/HierarchicalReporting/blob/master/appendix.pdf.

learning rate decays by half every 50 epochs. Other tuned hyperparameters for HRN include the number of hidden dimensions $\in \{16, 32, 64\}$, the minibatch size $\in \{32, 128\}$, and the dropout ratio $\in \{0, 0.5\}$ after LAYERPOOL. The number of epochs for each dataset is selected based on the best accuracy within cross-validation results. We apply the same layer-level pooling approach (LAYERPOOL in Eq. 5) for HRN; specifically, sum pooling is conducted on the bioinformatics datasets, and average pooling is conducted on the social datasets to ensure a direct comparison with GIN-0.

Baselines. It is worth noting that recent research efforts have been devoted to the adoption of attention or graph pooling mechanisms, which also sheds light on the path of our future work. However, little attention has been paid to the core part of graph classification models. Therefore, in this paper, we employ models without an attention mechanism or well-designed graph pooling for a fair comparison. We compare our WL-CT kernel and HRN configured above with several state-of-the-art baselines for graph classification: (1) kernel-based methods, i.e., the WL subtree kernel [Shervashidze *et al.*, 2011] and AWE [Ivanov and Burnaev, 2018]; (2) state-of-the-art deep learning methods, i.e., DCNN [Atwood and Towsley, 2016], PATCHY-SAN [Niepert *et al.*, 2016], DGCNN [Zhang *et al.*, 2018], GIN-0 [Xu *et al.*, 2019] and LP-GNN [Tiezzi *et al.*, 2021]. The accuracies of the WL subtree kernel are derived from [Xu *et al.*, 2019]. For AWE and the deep learning baselines, we utilize the accuracies contained in their original papers.

5 Results

The results of validating the WL-CT kernel and HRN on graph classification tasks are presented in Table 1. Our methods are shown in boldface. In the panel of kernel-based methods, we can observe that the accuracies of the WL-CT kernel exceed those of other kernel-based methods on four out of five benchmarks. For the only failed dataset, MUTAG, the WL-CT kernel still achieves very competitive performance. Notably, the WL-CT kernel even outperforms the state-of-the-art deep learning method (i.e., GIN-0) on IMDB-MULTI, COLLAB and MUTAG, which implies that superior performance can sometimes be obtained through a traditional machine learning methods rather than a deep learning model.

In the lower panel containing the deep learning methods, we can observe that the results of HRN are naturally superior to the accuracies of the WL-CT kernel, which further confirms the excellent performance of neural networks. In addition, HRN also yields the best performance on three out of five datasets, while competitive performance can still be observed on the other two datasets. In particular, the best performance of HRN is achieved on three social network datasets, which means social networks are more redundant than bioinformatic networks and that the key structure decoded from social networks is more conducive to model learning. In the light of the runtime complexity and experimental accuracies presented above, we can conclude that there is a way to boost the performance with simplified learning algorithms.

Dataset	IMDB-B	IMDB-M	COLLAB	MUTAG	PTC
# Graphs	1000	1500	5000	188	344
# Classes	2	3	3	2	2
Avg. # Nodes	19.8	13.0	74.5	17.9	25.5
Kernel-based methods					
WL	73.8±3.9	50.9±3.8	78.9±1.9	90.4±5.7	59.9±4.3
AWE	74.5±5.9	51.5±3.6	73.9±1.9	87.9±9.8	
WL-CT	**74.7±3.5**	**52.4±4.5**	**81.5±1.2**	**89.5±6.1**	**63.7±4.7**
Deep learning methods					
DCNN	49.1	33.5	52.1	67.0	56.6
PATCHY-SAN	71.0±2.2	45.2±2.8	72.6±2.2	**92.6±4.2**	60.0±4.8
DGCNN	70.0	47.8	73.7	85.8	58.6
GIN-0	75.1±5.1	52.3±2.8	80.2±1.9	89.4±5.6	64.6±7.0
LP-GNN-Single	65.3±4.7	46.6±3.7		90.5±7.0	64.4±5.9
LP-GNN-Multi	76.2±3.2	51.1±2.1		92.2±5.6	**67.9±7.2**
HRN	**77.5±4.3**	**52.8±2.7**	**81.8±1.2**	**90.4±8.9**	**65.7±6.4**

Table 1: **Classification accuracies on five benchmarks (%).** The best results are highlighted in boldface. On datasets where WL-CT and HRN are not strictly the highest-scoring models among the baselines, our methods still achieve competitive results; thus, their accuracies are still highlighted in boldface. For baselines, we highlight those that are significantly higher than those of all other methods.

Variants of HRN with different LAYERPOOL approaches. In the experimental setup, we employ sum pooling for the bioinformatics datasets and average pooling for social network datasets to ensure a fair comparison with GIN-0. However, as described in the paper of GIN, sum pooling on bioinformatics datasets and average pooling on social datasets are adopted due to their better test performance [Xu *et al.*, 2019]. To explore the upper limits of HRN, we adopt various basic pooling approaches for the five benchmarks. In particular, in addition to the adopted sum and average pooling, we also select the hidden feature of the root node as the representation of the entire coding tree, denoted ROOT. The classification accuracies of HRN with three kinds of LAYERPOOL approaches for coding trees under different heights are shown in Table 2. With the exception of PTC and IMDB-MULTI, HRN does not achieve the optimal performance for the other three benchmarks with the LAYERPOOL selection of GIN-0. Specifically, HRN obtains the highest accuracies of 77.6% for IMDB-BINARY with sum pooling, 82.2% for COLLAB with ROOT, and 91.4% for MUTAG with average pooling. In particular, the best result of HRN on COLLAB is realized with the root node representation rather than the sum or average pooling designed with skip connections for better performance. The reason for this phenomenon may be that the model overfits the detailed information in the shal-

LAYERPOOL	$\mathbb{H}$	IMDB-B	IMDB-M	COLLAB	MUTAG	PTC
ROOT	2	75.7±5.3	52.5±3.0	**82.2±1.4**	89.5±8.4	62.4±9.9
	3	75.3±5.9	52.5±2.5	81.6±1.3	88.4±6.0	59.8±5.3
	4	75.3±5.6	52.2±3.6	80.5±1.3	88.8±8.6	63.6±5.8
	5	75.2±5.3	51.5±3.8	79.9±1.1	89.5±6.5	62.6±7.9
SUM	2	76.7±3.3	52.3±3.1	82.1±0.9	89.3±6.4	64.0±6.3
	3	**77.6±4.1**	52.3±2.3	81.0±0.8	90.4±8.9	62.2±5.6
	4	76.6±3.5	52.7±1.4	80.0±1.4	90.0±8.6	**65.7±6.4**
	5	75.0±2.5	51.7±2.1	79.2±0.9	87.2±4.4	63.7±7.1
AVERAGE	2	76.9±2.9	52.6±3.3	81.8±1.2	**91.4±6.1**	61.6±8.9
	3	77.5±4.3	**52.8±2.7**	81.2±1.3	88.8±5.6	61.0±7.9
	4	76.4±3.0	52.6±2.1	80.5±1.6	87.2±7.2	64.4±9.9
	5	75.5±2.5	52.2±2.1	79.6±0.9	87.8±5.8	63.1±7.4
HRN-SOTA		77.6±4.1	52.8±2.7	82.2±1.4	91.4±6.1	65.7±6.4

Table 2: **Graph classification accuracies of HRN with different LAYERPOOL approaches (%).** The best results of each benchmark are highlighted in boldface. $\mathbb{H}$ denotes the coding tree height.

Dataset	IMDB-B	IMDB-M	COLLAB	MUTAG	PTC
WL-CT-RBBT	74.8±4.6	51.3±3.0	80.6±1.5	88.9±6.0	60.8±9.9
WL-CT-H2	74.1±4.7	52.4±4.5	80.8±1.4	89.5±6.1	61.5±7.3
HRN-RBBT	73.0±2.4	52.0±2.4	78.7±1.0	87.2±6.9	63.1±4.2
HRN-H2	76.9±2.9	52.6±3.3	81.8±1.2	89.3±6.4	64.0±6.3

Table 3: Classification accuracies of the RBBT and coding tree with a height of two on five benchmarks (%).

low layers, which leads to misjudgment of the global property of graphs, especially considering that the average number of nodes in COLLAB is the largest (see Table 1); specifically, more detailed information is encountered. In summary, these results of HRN with different pooling methods indicate the great potential of HRN in graph representation learning.

The guidance of structural entropy. Despite the state-of-the-art performance of our methods, we further evaluate the effectiveness of our k-dimensional coding tree algorithm. In particular, we do not form coding trees with the structural entropy minimization but use only a random coding tree, i.e., a randomly balanced binary tree (RBBT) with a height of two. For each graph within datasets, we produce the corresponding RBBT with graph nodes as leaves and employ our designed WL-CT kernel and HRN to perform graph classification (i.e., WL-CT-RBBT and HRN-RBBT). To ensure a fair comparison, we also fix the height of the coding tree to two (i.e., WL-CT-H2 and HRN-H2). The results are shown in Table 3. As can be seen, the structural entropy-guided coding tree surpasses the random coding tree not only on kernel-based methods but also on deep learning methods. The RBBT obtains higher accuracy only on the IMDB-BINARY dataset with kernel-based methods.

The effectiveness of essential structure within graphs. Following the label propagation scheme within the WL subtree kernel [Shervashidze *et al.*, 2011], we present a WL-CT kernel to take full advantage of the hierarchical structure of coding trees and validate its superior performance on five benchmarks. Here, we delve deeper into the effectiveness of the essential structure within graphs. Specifically, we employ those classical graph kernels, including the Core Framework kernel, Neighborhood Hash kernel, ODD-STh kernel, Propagation kernel, Pyramid Match kernel, and Shortest Path kernel, rather than the WL-CT kernel to perform classification. Note that the nonleaf nodes within coding trees are virtual nodes without initial labels; thus, we assign all nonleaf nodes in coding trees with the same label 0. As shown in Table 4, graphs and coding trees have distinct advantages under different kernels, while coding trees can still achieve superior performance even with these kernels designed for graphs with the exception of the core framework kernel, which further confirms that the coding trees decoded by structural entropy minimization are key structures of given graphs.

Graph Kernel	Data Type	IMDB-BINARY	IMDB-MULTI	COLLAB	MUTAG	PTC	Higher
Core Framework	Graph	73.7±5.3	53.2±5.0	81.0±1.8	86.2±4.8	63.1±4.1	**3/5**
	Coding Tree	74.6±5.4	52.5±4.2	80.2±1.6	89.9±6.0	60.4±7.5	2/5
Neighborhood Hash	Graph	68.9±4.6	49.1±4.5	71.3±1.6	87.3±5.8	62.8±7.3	1/5
	Coding Tree	76.4±4.5	51.8±3.8	81.2±1.2	87.8±7.5	61.3±9.2	**4/5**
ODD-STh	Graph	72.8±3.4	50.9±4.4	79.9±1.9	83.4±9.1	63.4±8.5	1/5
	Coding Tree	74.0±4.7	51.5±4.9	80.1±1.9	87.3±5.8	62.8±5.8	**4/5**
Propagation	Graph	73.1±4.8	51.5±4.0	79.1±1.4	84.1±6.6	60.8±5.5	2/5
	Coding Tree	75.3±5.2	50.9±4.3	78.3±1.0	86.2±4.8	62.5±6.8	**3/5**
Pyramid Match	Graph	75.7±4.5	51.4±4.5	81.9±1.2	88.9±7.9	61.6±8.0	1/5
	Coding Tree	74.5±4.9	51.8±4.8	82.2±1.7	89.9±7.7	65.1±8.5	**4/5**
Shortest Path	Graph	74.4±4.0	53.2±5.2	80.0±1.9	86.7±4.3	62.2±4.2	2/5
	Coding Tree	74.6±5.4	52.5±4.2	80.6±1.8	89.9±6.0	60.4±7.5	**3/5**

Table 4: Classification accuracies of coding trees and original graphs within various graph kernels on 5 benchmarks (%).

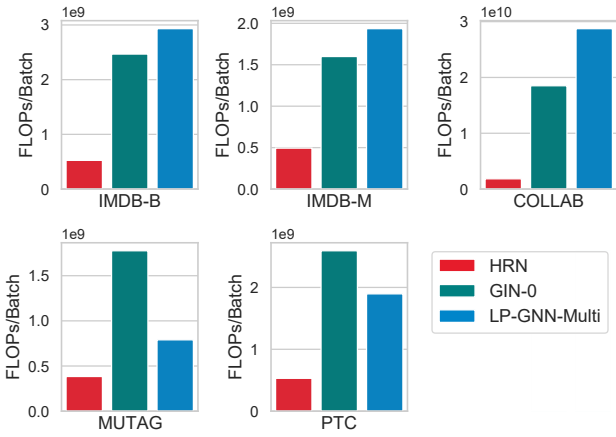

Figure 2: Comparison regarding the volume of computations.

Computational efficiency. In addition to the lower runtime complexity of our HRN than that of state-of-the-art GNNs ($O(n) < O(hm)$), we further compare the volumes of computations, i.e., the numbers of floating-point operations per second (FLOPs), required by HRN, GIN-0 and LP-GNN-Multi under the same parameter settings on all datasets. Specifically, we fix the number of iterations to four (the five GNN layers of GIN-0 include the input layer), the number of hidden units to 32, the batch size to 128 and the final dropout ratio to 0. The results are shown in Fig. 2. One can see that the volume of computations required by HRN is consistently smaller than that of GIN-0 and LP-GNN-Multi. More concretely, HRN needs only 20% (25%) of the volume of computations needed by the GIN-0 (LP-GNN-Multi) on average.

6 Conclusion

In this paper, we investigate the feasibility of improving graph classification performance while simplifying the learning process. In particular, through structural entropy minimizing, we decode the essential structure of given graphs to a simpler data form, i.e., coding trees. With this optimized new structure, we improve upon the graph classification performance by simplifying the corresponding kernel method and deep learning method. In summary, our proposed WL-CT kernel and HRN are not only theoretically and experimentally much more efficient but also capable of achieving state-of-the-art performance on most benchmarks. In addition to the excellent graph classification performance, out methods are not fit for another important task in the graph realm, i.e., node classification. Hence, determining how to improve the performance of node classification with a simpler learning model may be another underlying direction for future work.

Acknowledgments

This research was supported by NSFC (Grant No. 61932002).

References

[Atwood and Towsley, 2016] James Atwood and Don Towsley. Diffusion-convolutional neural networks. In *NeurIPS*, pages 1993–2001, 2016.

[Backstrom and Leskovec, 2011] Lars Backstrom and Jure Leskovec. Supervised random walks: predicting and recommending links in social networks. In *WSDM*, pages 635–644, 2011.

[Bengio *et al.*, 2021] Yoshua Bengio, Yann Lecun, and Geoffrey Hinton. Deep learning for ai. *Communications of the ACM*, 64(7):58–65, 2021.

[Borgwardt *et al.*, 2005] Karsten M Borgwardt, Cheng Soon Ong, Stefan Schönauer, SVN Vishwanathan, Alex J Smola, and Hans-Peter Kriegel. Protein function prediction via graph kernels. *Bioinformatics*, 21(suppl_1):i47–i56, 2005.

[Brown *et al.*, 2020] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. In *NeurIPS*, volume 33, pages 1877–1901, 2020.

[Chang and Lin, 2011] Chih-Chung Chang and Chih-Jen Lin. Libsvm: a library for support vector machines. *TIST*, 2(3):1–27, 2011.

[Duvenaud *et al.*, 2015] David Duvenaud, Dougal Maclaurin, Jorge Aguilera-Iparraguirre, Rafael Gómez-Bombarelli, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. In *NeurIPS*, pages 2224–2232, 2015.

[Gilmer *et al.*, 2017] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *ICML*, pages 1263–1272. PMLR, 2017.

[Hamilton *et al.*, 2017] William L Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NeurIPS*, pages 1025–1035, 2017.

[He *et al.*, 2016] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016.

[Hornik *et al.*, 1989] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[Hornik, 1991] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257, 1991.

[Ivanov and Burnaev, 2018] Sergey Ivanov and Evgeny Burnaev. Anonymous walk embeddings. In *ICML*, pages 2186–2195. PMLR, 2018.

[Li and Pan, 2016] Angsheng Li and Yicheng Pan. Structural information and dynamical complexity of networks. *IEEE TIT*, 62(6):3290–3339, 2016.

[Li *et al.*, 2016a] Angsheng Li, Qifu Hu, Jun Liu, and Yicheng Pan. Resistance and security index of networks: structural information perspective of network security. *Scientific Reports*, 6:26810, 2016.

[Li *et al.*, 2016b] Angsheng Li, Xianchen Yin, and Yicheng Pan. Three-dimensional gene maps of cancer cell types: Structural entropy minimisation principle for defning tumour subtypes. *Scientific Reports*, 6:20412, 2016.

[Li *et al.*, 2018] Angsheng Li Li, Xianchen Yin, Bingxiang Xu, Danyang Wang, Jimin Han, Yi Wei, Yun Deng, Ying Xiong, and Zhihua Zhang. Decoding topologically associating domains with ultra-low resolution hi-c data by graph structural entropy. *Nature Communications*, 2018.

[Liu *et al.*, 2020] Jiaheng Liu, Shunfeng Zhou, Yichao Wu, Ken Chen, Wanli Ouyang, and Dong Xu. Block proposal neural architecture search. *IEEE TIP*, 30:15–25, 2020.

[Morris *et al.*, 2020] Christopher Morris, Nils M. Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. Tudataset: A collection of benchmark datasets for learning with graphs. In *ICML 2020 Workshop on Graph Representation Learning and Beyond*, 2020.

[Niepert *et al.*, 2016] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. In *ICML*. PMLR, 2016.

[Shervashidze *et al.*, 2011] Nino Shervashidze, Pascal Schweitzer, Erik Jan Van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(9), 2011.

[Tiezzi *et al.*, 2021] Matteo Tiezzi, Giuseppe Marra, Stefano Melacci, and Marco Maggini. Deep constraint-based propagation in graph neural networks. *IEEE TPAMI*, 2021.

[Vaswani *et al.*, 2017] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, pages 5998–6008, 2017.

[Veličković *et al.*, 2018] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *ICLR*, 2018.

[Wu *et al.*, 2020] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE TNNLS*, 32(1):4–24, 2020.

[Xu *et al.*, 2019] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *ICLR*, 2019.

[Ying *et al.*, 2018] Rex Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William L Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. In *NeurIPS*, pages 4805–4815, 2018.

[Zhang and Chen, 2018] Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. *NeurIPS*, 31:5165–5175, 2018.

[Zhang *et al.*, 2018] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An end-to-end deep learning architecture for graph classification. In *AAAI*, 2018.