STRUCTURAL OPTIMIZATION MAKES GRAPH CLASSIFICATION SIMPLER AND BETTER

Anonymous authors
Paper under double-blind review

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. Here, based on an optimization method, we investigate the feasibility of improving graph classification performance while simplifying the model learning process. Inspired by progress in structural information assessment, we optimize the given data sample from graphs to encoding trees. In particular, we minimize the structural entropy of the transformed encoding tree to decode the key structure underlying a graph. This transformation is denoted as structural optimization. Furthermore, we propose a novel feature combination scheme, termed hierarchical reporting, for encoding trees. In this scheme, features are transferred from leaf nodes to root nodes by following the hierarchical structures of encoding trees. We then present an implementation of the scheme in a tree kernel and a convolutional network to perform graph classification. The tree kernel follows label propagation in the Weisfeiler-Lehman (WL) subtree kernel, but it has a lower runtime complexity $O(n)$. The convolutional network is a special implementation of our tree kernel in the deep learning field and is called Encoding Tree Learning (ETL). We empirically validate our tree kernel and convolutional network with several graph classification benchmarks and demonstrate that our methods 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 hard 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 models; 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)). 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. Graph neural networks (GNNs) have emerged and have recently been ubiquitous within deep learning for graphs because of their ability to model structural information (Hamilton et al., 2017; Kipf & Welling, 2017; Zhang et al., 2018; Xu et al., 2019). In GNNs, each node recursively updates its feature vector through a message passing (or neighborhood aggregation) scheme, in which the feature vectors from neighbors are aggregated to compute a new node feature vector (Gilmer et al., 2017; Xu et al., 2018). For tasks involving the overall characteristics, an entire graph representation can be obtained through a pooling operation (Xu et al., 2019; Ying et al., 2018). To improve the performance of basic GNNs, various more complex models have been developed. Considering the differences among graph nodes, attention mechanisms have been adopted in GNNs' message passing schemes to focus on more informative neighbors (Veličković et al., 2018; Zhang et al., 2019). Regarding the pooling process, in addition to the basic sum or average pooling methods, more complicated pooling operations have been proposed for better learning with respect to entire graphs, such as SORTPOOL (Zhang et al., 2018), DIFF-POOL (Ying et al., 2018), STRUCTPOOL (Yuan & Ji, 2020) and SOPOOL (Wang & Ji, 2020). In

this context, the improvement in performance comes with the price of model complexity, similar to the routine in deep learning. We argue that there is a way to boost performance while reducing the complexity of basic models.

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 but is very likely to result in information loss and poor performance. Therefore, *structural optimization*, which transforms the original structure of data into a simplified form while maintaining crucial features, is proposed. Different from the classic optimization, which optimizes the parameters of a given target (e.g., the parameters in a neural network are searched by minimizing the loss function), this optimization aims to search the simplified structure of data. In addition, these retained features are expected to not only keep the key information of datasets but more importantly, many other features that negatively influence the given task can also be excluded. Consequently, this method could possibly yield a better result with even higher efficiency due to the simplified structure.

This work is inspired by structural entropy (Li & Pan, 2016; Li, 2021), a metric designed to assess the structural information of a graph. Structural entropy can also be used to decode the key structure of a graph as a measure of the complexity of its hierarchical structure. In this paper, we realize structural optimization by transforming the given graph into a corresponding encoding tree that reflects the hierarchical organization of data. The crucial structural information underlying a graph can be kept in the encoding tree with minimal structural entropy. Knowing that the essence of deep learning success is its superior feature characterization ability (Bengio et al., 2021), our core view is that an encoding tree obtained after structural optimization is a much simpler data structure that also preserves the main features of the input graph.

Based on simplified encoding 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 encoding trees. We then present an implementation of the scheme in a tree kernel and a convolutional network, denoted as WL Encoding Tree (WL-ET) kernel and Encoding Tree Learning (ETL), to perform graph classification. The tree kernel follows the label propagation in the WL subtree kernel but has a lower runtime complexity $O(n)$. ETL is a special implementation of our tree kernel in the deep learning field. Finally, we empirically validate our tree kernel and ETL 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 ETL approach also achieves competitive performance relative to that of baselines.

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 an optimization method.

- Based on structural optimization, we optimize the given data sample from graphs to encoding trees, which are much simpler data structures that have optimized feature characterization abilities.

- We develop a novel tree kernel (WL-ET) and a convolutional network (ETL) 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 & 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. Graph classification involves identifying the characteristics of an entire graph and is universal in a variety of domains, such as urban computing (Bao et al., 2017), social network analysis (Backstrom & Leskovec, 2011), chemoinformatics (Duvenaud et al., 2015), bioinformatics (Borgwardt et al., 2005), and even code analysis (Thost & Chen, 2020). In addition to previous techniques such as graph kernels (Shervashidze et al., 2011), recently, GNNs emerged and have become a popular way to handle graph 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 GIN (Xu et al., 2019) was presented in theoretical analyses regarding the expressive power of GNNs in terms of graph structure capture. 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., 2018). 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; Yuan & Ji, 2020; Wang & Ji, 2020). All these pooling methods help models achieve state-of-the-art results but increase the model complexity and the volume of computations.

Structural entropy. Structural entropy is a measure of the complexity of the hierarchical structure of a graph (Li & Pan, 2016; Li, 2021). The structural entropy of a graph is defined as the average length of the codewords obtained under a specific encoding scheme for a random walk. That is, when a random walk takes one step from u to v and we use v's codeword to label this step, the codeword of the longest common ancestor of u and v on the encoding 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, and it is why we call it structural entropy. The encoding tree, which is considered to be the essential hierarchical structure of the given graph, is achieved when the structural entropy is minimized. Structural optimization is the task of searching for this optimum. For more information on structural entropy, please refer to (Li & Pan, 2016). 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.

Very recently, a novel hierarchical encoding algorithm based on structural entropy optimization was proposed (Pan et al., 2021). This algorithm stratifies a given graph into multiple levels by minimizing structural entropy, during which the sparsest levels of a graph are differentiated recursively. This provides an efficient approach to approximate the optimal hierarchical structure. In this paper, we apply this algorithm to structural optimization.

3 METHODOLOGY

Based on structural optimization, we analyze the feature characterization of a graph by optimizing its structure and carry out graph classification on the new simplified structure. Specifically, we analyze the hierarchical structure of a graph that an encoding tree incorporates. A tree is a much simpler data structure than its original graph, while a high-quality encoding tree retains the structural information of the graph. Next, we first present the structural optimization process that transforms a graph into its encoding tree for simplified feature characterization. Based on the optimized encoding trees, we propose a tree kernel and a corresponding implementation in a deep learning model for graph classification. We elaborate on them below.

3.1 STRUCTURAL OPTIMIZATION

Given a weighted graph $G = (V, E, w)$ and an encoding tree T for G, the *structural entropy of G on T* is defined as

$$\mathcal{H}^T(G) = -\sum_{\alpha \in T} \frac{g_\alpha}{\text{vol}(V)} \log \frac{\text{vol}(\alpha)}{\text{vol}(\alpha^-)}. \tag{1}$$

We define the *structural entropy of G* to be the minimum entropy among all encoding trees, and it is denoted by $\mathcal{H}(G) = \min_T \{\mathcal{H}^T(G)\}$. $\mathcal{H}^T(G)$ is essentially the optimal hierarchical structure in the sense that the average length of the codewords obtained with a random walk under the aforementioned encoding scheme is minimized.

To formulate a natural encoding tree with a certain height, we define the *k-dimensional structural entropy of G* for any positive integer k to be the minimum value among all encoding trees with heights of at most k:

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

The algorithm proposed by (Pan et al., 2021) is devoted to computing a k-dimensional encoding tree with minimum structural entropy. We use this algorithm for structural optimization, which yields an encoding tree T with a height of at most k for G, where $T = (V_T, E_T)$, $V_T = (V_T^0, \ldots, V_T^k)$ and $V_T^0 = V$. We denote $n = |V|$ and $m = |E|$. It is worth noting that although the time complexity of this algorithm is $O(m \log m + n^2)$ in the worst case, if we denote by $h_{\max}$ the maximum height among the binary trees that appear during the construction of T, the time complexity reduces to $O(m \log m + h_{\max} n)$. Since the minimized structural entropy tends to generate balanced encoding trees (Pan et al., 2021), $h_{\max}$ is usually of order $O(\log n)$, which reduces the time complexity of structural optimization to $O(m \log m)$ (almost linear in the number of edges). In this paper, we compare the real running times of structural optimization on all test datasets in Appendix E with other parts of our methods, and the results show that the time required for structural optimization is much less than that of WL-ET and ETL and only accounts for 0.002% to 4% of the ETL time requirement.

3.2 TREE KERNEL FOR GRAPH CLASSIFICATION

Following the construction of the WL subtree kernel (Shervashidze et al., 2011), we propose a novel tree kernel that measures the similarity between encoding trees, named the WL-ET 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 according to the hierarchical structures of encoding trees. Finally, our tree kernel also adopts the counts of the node labels at different heights of an encoding tree as the feature vector of the original graph.

Hierarchical Reporting. The key idea of this scheme is to assign labels to non-leaf 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 encoding tree. See Figure 1 in Appendix A, for an illustration of this scheme.

Definition 1. *Let T_1 and T_2 be any two encoding trees with the same height h. There exists a set of letters $\Sigma^i \in \Sigma$, which are node labels appearing at the i-th ($i < h$) height of T_1 or T_2 (i.e., the nodes at the i-th height 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 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 the letter b_j^i in the encoding tree T_1.*

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

$$k_{EncodingTree}(T_1, T_2) = <\varphi_{EncodingTree}(T_1), \varphi_{EncodingTree}(T_2)>, \tag{3}$$

where

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

and

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

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 encoding trees. An illustration of this kernel is shown in Figure 1.

Theorem 1. *WL-ET kernel on two encoding trees T_1 and T_2 with the same height h 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 simplest method in graph classification to the best of our knowledge (Wu et al., 2020).*

Proof. Given a graph G, the biggest encoding tree is a binary encoding tree. Thus, the complexity of WL-ET kernel on the binary encoding tree is the worst case (i.e., $O_{EncodingTree} \leq$

$O_{BinaryEncodingTree}$). The complexity on the binary encoding tree is calculated as:

$$\begin{aligned} O_{BinaryEncodingTree} &= O(|V_T|) \\ &= O(|V_T^0| + |V_T^1|, \ldots, +|V_T^k|) \\ &\leq O(2n) \\ &= O(n) \end{aligned}$$

$\square$

3.2.1 Computing the tree kernel on many encoding trees

In addition to the tree kernel designed for measuring the similarity between encoding trees, we aim to propose an algorithm to compute the feature vectors formed on N encoding trees for classification. As shown in Algorithm 1, we present the process for one iteration of our tree kernel computation process on N encoding trees. This algorithm consists of the same 4 steps as those in the WL test, including multiset-label determination, multiset sorting, label compression and relabeling. The core difference is in step 1, where the multiset-label $L^i(v)$ consists of labels from the child nodes of v rather than its neighbors. Consistent with the WL test, Σ is sufficiently large to make f injective. In a case with N encoding trees, a Σ of size $2nN$ suffices.

Algorithm 1 One iteration of the tree kernel computation process on N encoding trees

1: Multiset-label determination
   - For $i = 0$, set $L^i(v) = b^0(v)$, where $b^0(v)$ is the initial leaf node label.
   - For $i > 0$, each node v with height i in an encoding tree T is assigned a multiset label $L^i(v) = \{b^{i-1}(u)|u \in \mathcal{C}(v)\}$, where $\mathcal{C}(v)$ denotes the children of node v in T.
2: Multiset sorting
   - The elements in $L^i(v)$ are sorted in ascending order and then united into a string $s^i(v)$.
3: Label compression
   - An injective function $f : \Sigma^* \to \Sigma$ is employed to compress each string $s^i(v)$ into a new and short label.
4: Relabeling
   - $b^i(v) := f(s^i(v))$ is set for all nodes with height i in T.

Theorem 2. *For N encoding trees with height h, WL-ET kernel on all pairs of these encoding trees can be computed in $O(nN^2)$ [1], which is much simpler than the WL subtree kernel formed on N graphs ($O(Nhm + N^2hn)$) (Shervashidze et al., 2011) and is the simplest graph classification method.*

Proof. The runtime complexity of WL-ET kernel with a naive application derived by computing an $N \times N$ kernel matrix is $O(nN^2)$. $\square$

3.3 ETL for graph classification

Based on our tree kernel, we develop a novel deep learning architecture, *Encoding Tree Learning (ETL)*, that generalizes the hierarchical reporting scheme to update the hidden features of non-leaf nodes for graph classification. ETL uses the tree structure and leaf node features X_v to learn the representation vector of the entire graph r_T. ETL follows the proposed hierarchical reporting scheme, where the representation of a non-leaf node is updated by aggregating the hidden features of its children. Formally, the i-th layer of ETL is

$$r_v^i = \text{MLP}^i \left(\sum\nolimits_{u \in \mathcal{C}(v)} r_u^{(i-1)} \right). \tag{4}$$

where r_v^i is the feature vector of node v with height i in the encoding tree. We initialize $r_v^0 = X_v$, and $\mathcal{C}(v)$ is a set of child nodes of v. As shown in Equation 4, we employ summation and

---

[1] Again, we compute the runtime complexity with the form of binary encoding tree, which is the upper bound of our normal encoding tree.

multilayer perceptrons (MLPs) to perform hierarchical reporting in ETL. 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_v^h can be naively employed as the representation of the entire encoding 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 encoding tree with layer representations concatenated across all heights/layers of the ETL structure:

$$r_T = \text{CONCAT}\left(\text{LAYERPOOL}(\{r_v^i | v \in T^i\}) | i = 0, 1, \ldots, h\right), \tag{5}$$

where r_v^i is the feature vector of node v with height i in encoding tree T and h is the height of T. In ETL, LAYERPOOL in Equation 5 can be replaced with the summation or averaging of all node vectors within the same iterations as in GIN, which provably generalizes our tree kernel.

4 COMPARISONS TO RELATED WORK

We now discuss why the structural optimization is capable of making graph classification simpler and better.

The simplicity is mainly reflected in three aspects. First, we employ structural optimization to transform the given data sample from a graph into an encoding tree, which is a simpler data structure for representation. Second, based on the transformed encoding trees, our proposed tree kernel is much simpler than the most popular graph kernel, the WL subtree kernel (Shervashidze et al., 2011), for graph classification in terms of runtime complexity. As seen in Theorem 1, our tree kernel for the similarity measurement of a pair of encoding trees can be computed in time $O(n)$, which is naturally less than the $O(hm)$ required for running the WL subtree kernel. Finally, regarding the use of deep learning models for graph classification, our proposed ETL also achieves a lower complexity than previous GNNs, such as the graph convolutional network (GCN) (Kipf & Welling, 2017) and GIN (Xu et al., 2019). Considering that ETL is generalized from our tree kernel, the runtime complexity of ETL for learning an encoding tree is $O(n)$, while at least $O(hm)$ is required for previous GNNs to learn a graph (Wu et al., 2020).

The great success of deep networks is attributed to their excellent feature characterization power; put differently, "they exploit a particular form of compositionality in which features in one layer are combined in many different ways to create more abstract features in the next layer" (Bengio et al., 2021). Regarding graph representation learning, the expressive power of GNNs has also been theoretically proven (Xu et al., 2019). To achieve better graph classification, we optimize the feature characterization ability, including feature extraction and combination, of kernel-based and deep learning-based methods. In particular, through structural optimization, we transform a graph into a powerful data structure, an encoding tree, which is simple but helpful for feature extraction and combination. Specifically, we first optimize the feature extraction process. The encoding tree optimized from a graph has the minimum structural entropy and decodes the key structure underlying the original graph. Following the hierarchical structure of the encoding tree, the features extracted for each tree node are hierarchical, while GNNs are designed to learn representations of the original graph nodes in a flat space. For feature combination, we optimize the form of compositionality in different iterations/layers. With the hierarchical reporting scheme, the features are combined from bottom to top according to the hierarchical relationships among tree layers, which is obviously different from the form of message passing in GNNs.

5 EXPERIMENTS

We validate the effectiveness of structural optimization by comparing the experimental results of our tree kernel and ETL with those of the most popular kernel-based methods and GNNs on graph classification tasks [2].

---

[2]The code of WL-ET kernel and ETL will be public after review.

5.1 DATASETS

We conduct graph classification on 5 benchmarks: 3 social network datasets (IMDB-BINARY, IMDB-MULTI, and COLLAB) and 2 bioinformatics datasets (MUTAG and PTC) (Xu et al., 2019) [3]. There is a difference between the data of bioinformatic datasets and social network datasets; that is, 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; the combination of node degrees and node categorical labels are taken for bioinformatic graphs. Correspondingly, the initial node features of the ETL 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 5 employed datasets, and detailed data descriptions are shown in Appendix C.

5.2 CONFIGURATIONS

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

For configuration of ETL, the number of ETL iterations is consistent with the heights of the associated encoding trees, which are also $\in [2, 3, 4, 5]$. All MLPs have 2 layers, as in the setting of the GIN (Xu et al., 2019). 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 learning rate is decayed by half every 50 epochs. Other tuned hyperparameters for ETL include the number of hidden dimensions $\in \{16, 32, 64\}$; the minibatch size $\in \{32, 128\}$; the dropout ratio $\in \{0, 0.5\}$ after the final output; 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 ETL; specifically, sum pooling is conducted on the bioinformatics datasets, and mean pooling is conducted on the social datasets due to better test performance.

5.3 BASELINES

We compare our tree kernel and ETL model 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 Anonymous Walk Embeddings (AWE) (Ivanov & Burnaev, 2018); (2) state-of-the-art deep learning methods, i.e., Diffusion-Convolutional Neural Network (DCNN) (Atwood & Towsley, 2016), PATCHY-SAN (Niepert et al., 2016), Deep Graph CNN (DGCNN) (Zhang et al., 2018) and GIN (Xu et al., 2019). 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.

6 RESULTS

The results of validating our tree kernel and ETL model 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 our tree kernel exceed those of other kernel-based methods on 4 out of 5 benchmarks. For the only failed dataset, MUTAG, our tree kernel still achieves very competitive performance. Notably, our tree 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 an optimization method rather than a deep learning method.

---

[3]Considering the limitations of encoding trees on disconnected graphs, we utilize additional configurations for the other 4 datasets that contain such graphs. The results are presented in Appendix B.

Table 1: **Classification accuracies on 5 benchmarks (%).** The best results are highlighted in boldface. On datasets where WL-ET and ETL 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 the results of the baselines, we highlight those that are significantly higher than those of all other methods.

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-ET	**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
ETL	**76.7±4.5**	**53.1±4.5**	**81.8±1.2**	**90.6±6.8**	**66.3±4.3**

In the lower panel containing the deep learning methods, we can observe that the results of ETL are naturally superior to the accuracies of the tree kernel, which further confirms the outperformance of deep learning in terms of feature characterization. In addition, ETL also yields the best performance on 4 out of 5 datasets, while competitive performance can still be observed on the other dataset (i.e., MUTAG). These results indicate that optimization methods can not only coexist with but also further boost deep learning methods. We also compare the volumes of computations of ETL and GIN-0 in Appendix D, and the results show that ETL requires only 22% of the volume of computations used in the GIN on average.

7 CONCLUSION AND FUTURE WORK

In this paper, to boost the performance of basic models while simplifying its learning process, we propose structural optimization, which is a structural transformation from original datasets to a simplified new structure that preserves the key features of the input data. Utilizing a recently developed structural entropy minimization algorithm, we improve upon the graph classification performance by simplifying the corresponding kernel method and deep learning method. In particular, our proposed tree kernel and ETL make graph classification simpler and better with optimized encoding trees. In addition to the excellent graph classification performance, the ETL that derived from structural optimization even possess powerful interpretability with respect to node importance and feature combination paths because of the hierarchical structure of the constructed encoding tree [4]. Thus, an interesting direction for future work is to interpret the power of ETL. Furthermore, despite the superior performance of our proposed methods in graph classification, they are not fit for another important task in graph realm, i.e., node classification. Hence, how structural optimization makes node classification simpler and better may be another underlying direction for future work.

---

[4]An illustration of the underlying interpretability of ETL is shown in Appendix F.

REFERENCES

James Atwood and Don Towsley. Diffusion-convolutional neural networks. In *Advances in neural information processing systems*, pp. 1993–2001, 2016.

Lars Backstrom and Jure Leskovec. Supervised random walks: predicting and recommending links in social networks. In *Proceedings of the fourth ACM International Conference on Web Search and Data Mining*, pp. 635–644, 2011.

Jie Bao, Tianfu He, Sijie Ruan, Yanhua Li, and Yu Zheng. Planning bike lanes based on sharing-bikes' trajectories. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 1377–1386, 2017.

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

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.

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 *Advances in Neural Information Processing Systems*, volume 33, pp. 1877–1901, 2020.

Chih-Chung Chang and Chih-Jen Lin. Libsvm: a library for support vector machines. *ACM transactions on intelligent systems and technology (TIST)*, 2(3):1–27, 2011.

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 *Proceedings of the 28th International Conference on Neural Information Processing Systems*, pp. 2224–2232, 2015.

Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International Conference on Machine Learning*, pp. 1263–1272. PMLR, 2017.

William L Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pp. 1025–1035, 2017.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2016.

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

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

Sergey Ivanov and Evgeny Burnaev. Anonymous walk embeddings. In *International conference on machine learning*, pp. 2186–2195. PMLR, 2018.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2017.

Angsheng Li. Mathematical principles of information science. *Manuscript in preparation*, 2021.

Angsheng Li and Yicheng Pan. Structural information and dynamical complexity of networks. *IEEE Transactions on Information Theory*, 62(6):3290–3339, 2016.

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, 2016a.

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, 2016b.

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*, 9(1):3265, 2018.

Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. In *International conference on machine learning*, pp. 2014–2023. PMLR, 2016.

Yicheng Pan, Feng Zheng, and Bingchen Fan. An information-theoretic perspective of hierarchical clustering. *arXiv preprint arXiv:2108.06036*, 2021.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 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.

Veronika Thost and Jie Chen. Directed acyclic graph neural networks. In *International Conference on Learning Representations*, 2020.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pp. 5998–6008, 2017.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018.

Zhengyang Wang and Shuiwang Ji. Second-order pooling for graph neural networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24, 2020.

Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. In *International Conference on Machine Learning*, pp. 5453–5462. PMLR, 2018.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019.

Rex Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William L Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pp. 4805–4815, 2018.

Hao Yuan and Shuiwang Ji. Structpool: Structured graph pooling via conditional random fields. In *Proceedings of the 8th International Conference on Learning Representations*, 2020.

Kai Zhang, Yaokang Zhu, Jun Wang, and Jie Zhang. Adaptive structural fingerprints for graph attention networks. In *International Conference on Learning Representations*, 2019.

Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. *Advances in Neural Information Processing Systems*, 31:5165–5175, 2018.

Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An end-to-end deep learning architecture for graph classification. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

A   ILLUSTRATION OF WL ENCODING TREE KERNEL
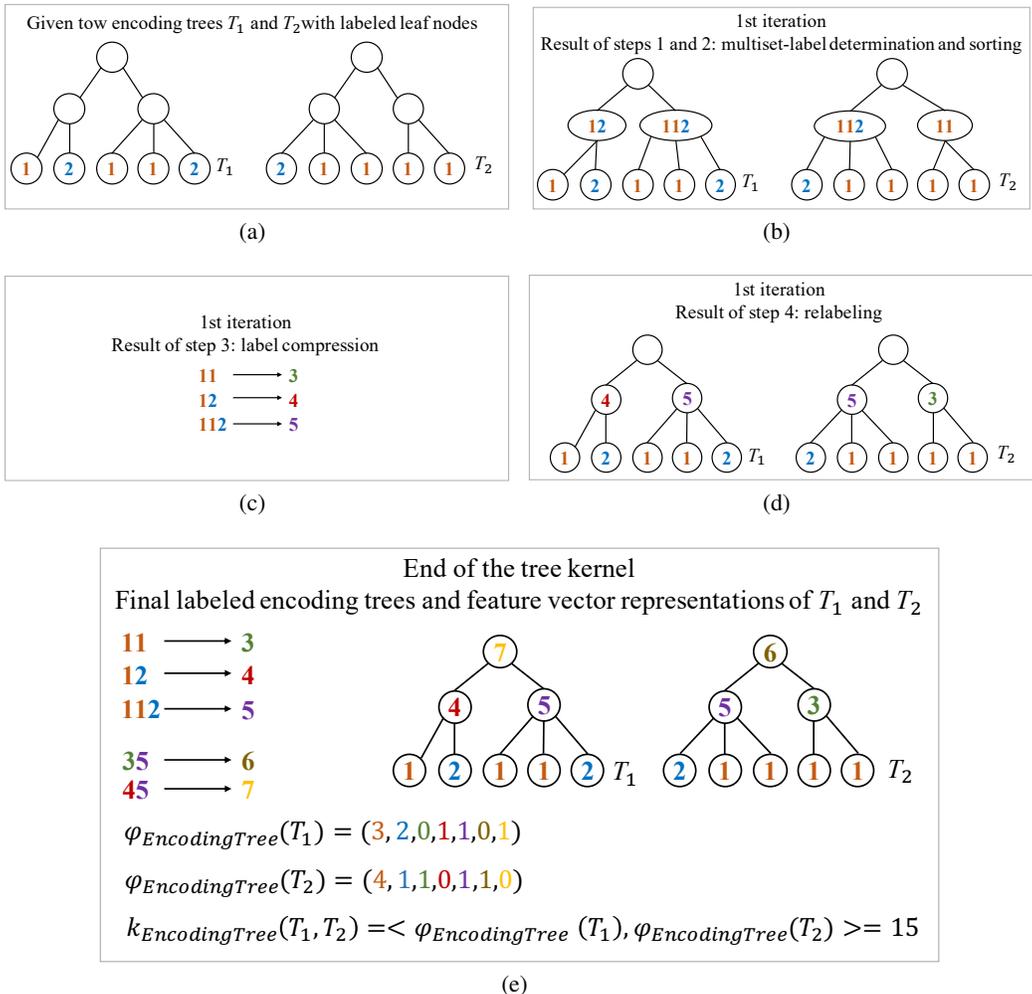
Figure 1: Illustration of computing the tree kernel on two encoding trees with height=2. Here, $\{1, 2, \ldots, 7\} \in \Sigma$ are considered letters.

B   EXPERIMENTS ON DATASETS WITH DISCONNECTED GRAPHS

Considering the limitation of encoding tree on disconnected graphs, we take additional configurations for the other 4 datasets that contain disconnected graphs.

B.1   DATASETS

There are 4 other well-known graph classification benchmarks with disconnected graphs: 2 bioinformatics datasets (PROTEINS and NCI1) and 2 social network datasets (REDDIT-BINARY and REDDIT-MULTI5K). The input features for these 4 datasets are consistent with the feature handling approach in the main text. Notably, we take additional configurations for the disconnected graphs contained in these 4 datasets. (1) We transform each connected component from a disconnected graph sample separately into corresponding encoding trees, (2) and then, we combine these separate encoding trees into one tree through naive root node merging. Table 2 summarizes the data statistics of the adopted benchmarks, and detailed data descriptions are shown in Appendix C.

B.2 Models and configurations

Following the setting in the main text, 10-fold cross-validation is conducted to make a fair comparison, and we present the average accuracies obtained to validate the performance of our methods on graph classification tasks. Regarding the configuration of our tree kernel, we also tune the hyperparameter C of the SVM and the height of the encoding tree $\in [2, 3, 4, 5]$. We set the other hyperparameter γ as *auto* for REDDIT-BINARY and REDDIT-MULTI5K and as *scale* for PROTEINS and NCI1. The configuration of ETL on these 4 datasets is consistent with that in the main text.

B.3 Results

We compare our methods with the same baselines and report the results in Table 2. Our methods only achieve superior performance on one dataset (PROTEINS), where disconnected graphs occupy a 4% proportion of the data. For the other three datasets, WL has the best performance among all GNN-based models on NCI1, and GIN-0 obtains the highest accuracies on the REDDIT datasets. One explanation for this phenomenon is that the structural information underlying the disconnected graphs can hardly be captured by the methods that are based on structural optimization.

Table 2: Classification accuracies on datasets with disconnected graphs (%).

Dataset	RDT-B	RDT-M5K	PROTEINS	NCI1
# Graphs	2000	5000	1113	4110
# Disconnected Graphs	1022	3630	46	580
# Classes	2	5	2	2
Avg. # Nodes	429.6	508.5	39.1	29.8
Kernel-based methods				
WL	81.0±3.1	52.5±2.1	75.0±3.1	**86.0±1.8**
AWE	87.9±2.5	54.7±2.9		
WL-ET	**86.9±6.1**	**53.3±2.4**	**76.2±3.3**	76.5±3.3
Deep learning methods				
DCNN			61.3	62.6
PATCHSAN'	86.3±1.6	49.1±0.7	75.9±2.8	78.6±1.9
DGCNN			75.5	74.4
GIN-0	**92.4±2.5**	**57.5±1.5**	76.2±2.8	82.7±1.7
ETL	86.8±1.9	51.9±2.6	**76.5±2.5**	79.3±1.4

C Details of the datasets

Here, we present detailed descriptions of the 9 benchmarks utilized in this paper.

Social network datasets. IMDB-BINARY and IMDB-MULTI are movie collaboration datasets. Each graph corresponds to an ego network for each actor/actress, where the nodes correspond to actors/actresses and an edge is drawn between two actors/actresses if they appear in the same movie. Each graph is derived from a prespecified genre of movies, and the task is to classify the genre from which each graph is derived. REDDIT-BINARY and REDDIT-MULTI5K are balanced datasets, where each graph corresponds to an online discussion thread and nodes correspond to users. An edge is drawn between two nodes if at least one of them responds to another's comment. The task is to classify each graph into the community or subreddit to which it belongs. COLLAB is a scientific collaboration dataset derived from 3 public collaboration datasets, namely, High Energy Physics, Condensed Matter Physics and Astro Physics. Each graph corresponds to an ego network of a different researcher from each field. The task is to classify each graph into a field to which the corresponding researcher belongs.

Bioinformatics datasets. MUTAG is a dataset containing 188 mutagenic aromatic and heteroaromatic nitro compounds with 7 discrete labels. PROTEINS is a dataset where the nodes are secondary

structure elements (SSEs), and there is an edge between two nodes if they are neighbors in the given amino acid sequence or in 3D space. The dataset has 3 discrete labels, representing helixes, sheets or turns. PTC is a dataset containing 344 chemical compounds that reports the carcinogenicity of male and female rats and has 19 discrete labels. NCI1 is a dataset made publicly available by the National Cancer Institute (NCI) and is a subset of balanced datasets containing chemical compounds screened for their ability to suppress or inhibit the growth of a panel of human tumor cell lines; this dataset possesses 37 discrete labels.
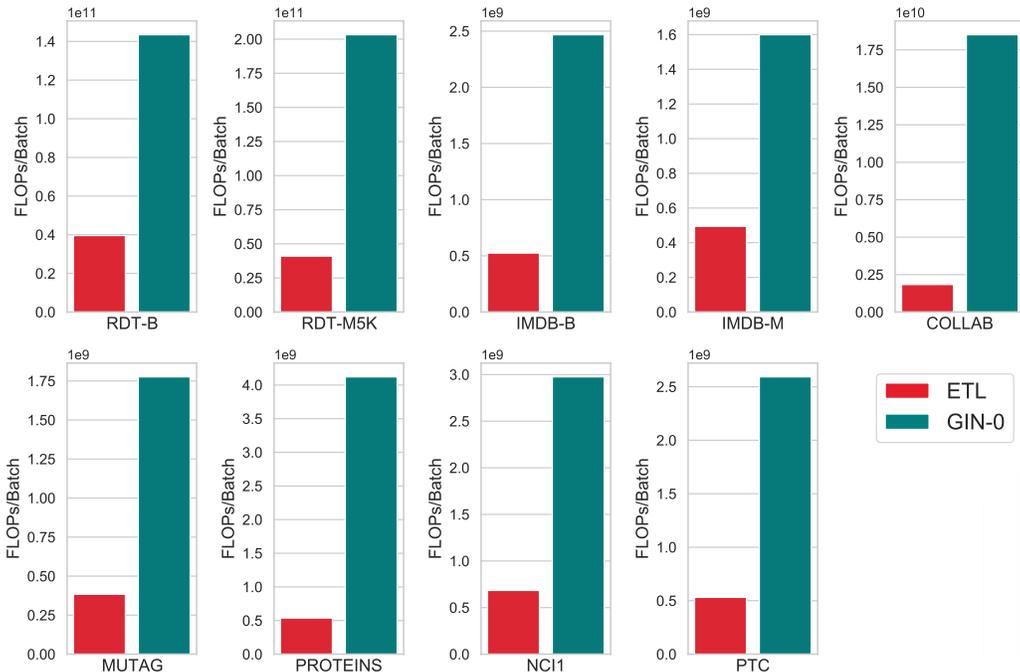
Figure 2: Comparison regarding the required volume of computations.

D   ANALYSIS OF THE EFFICIENCY OF ETL

In addition to the lower runtime complexity of our ETL approach than that of state-of-the-art GNNs ($O(n) < O(hm)$), we also compare the volumes of computations, i.e., the numbers of floating-point operations per second (FLOPs), required by ETL and GIN-0 under the same parameter settings on all datasets. Specifically, we fix the number of iterations to 4 (the 5 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 Figure 2. One can see that the volume of computations required by our ETL method is consistently smaller than that of GIN-0. More concretely, ETL needs only 22% of the volume of computations needed by the GIN on average.

E   THE RUNNING TIME OF STRUCTURAL OPTIMIZATION

The total time required for generating a classifier includes the time of structural optimization and the training time of ETL, in which the structural optimization only needs to run once, while ETL needs hundreds of epochs of training before its testing even with fixed hyperparameters. Thus, we compare the real running times of structural optimization (SO), WL-ET [5] and ETL [6] on all datasets with the fixed hyperparameters described in Section D. The results are shown in Figure 3, and we can see that the time required for structural optimization is much less than the time needed by WL-ET and ETL and only accounts for 0.002% to 4% of the time required by ETL.

---

[5]The running time of WL-ET is the time required for performing 10-fold cross-validation with C-SVM.

[6]We calculate the actual running time of ETL with fixed hyperparameters under 300 epochs training.
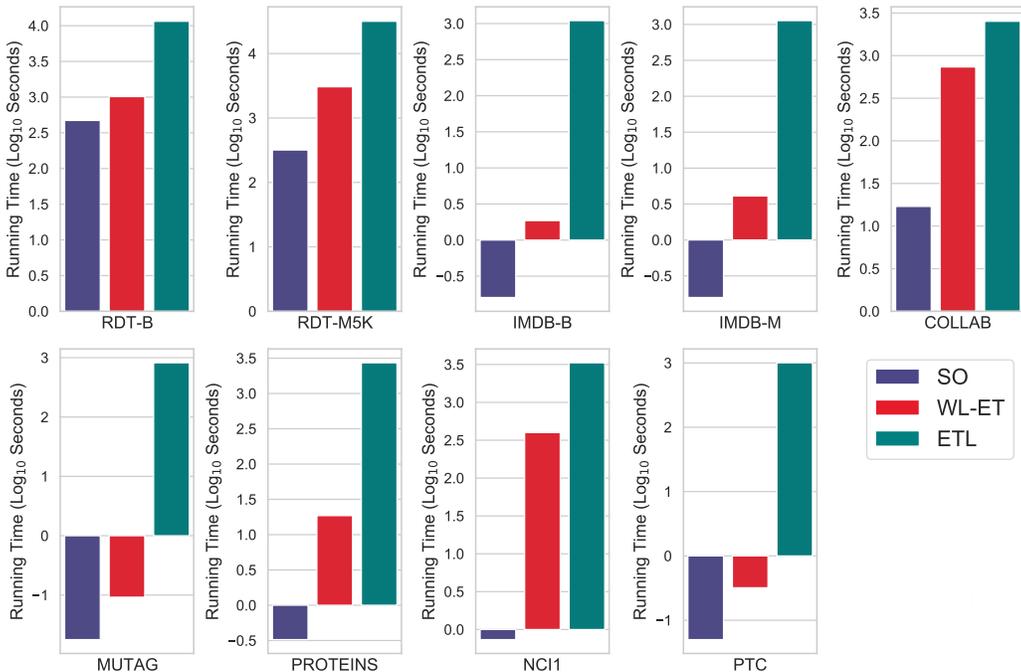
Figure 3: **Running time comparison**. Structural optimization and WL-ET are conducted on a machine with an AMD Ryzen 3900x and 64 GB of RAM. ETL is trained with a Tesla V100 GPU. The Y axis is the log with a base of 10 seconds because of the very large distinction in scale.

F   UNDERLYING INTERPRETABILITY ADVANTAGE

Compared with GNNs, our ETL is more interpretable regarding its node importance levels and feature combination paths. Based on the optimized encoding trees, the features propagating in ETL follow a single direction rather than the complicated data loops found in graphs, which makes it hard for GNNs to interpret the entire information propagation path.

Following the decomposition methods used in GNNs, we can not only measure the importance levels of nodes with different heights but also identify the most important path for feature combination by combining the importance scores of the nodes in the path. A toy example to demonstrate the advantage of ETL in terms of interpretability can be found in Figure 4. As in the decomposition methods of GNNs. The intuition of this idea is to build score decomposition rules to distribute the prediction scores from the output space to the input space. Starting from the output layer, the model's prediction is treated as the initial target score. Then, the score is decomposed and distributed to the neurons in the previous layer following the decomposition rules. By repeating such procedures until covering the input space, the importance scores for node features can be obtained, which can also be combined to represent path importance.
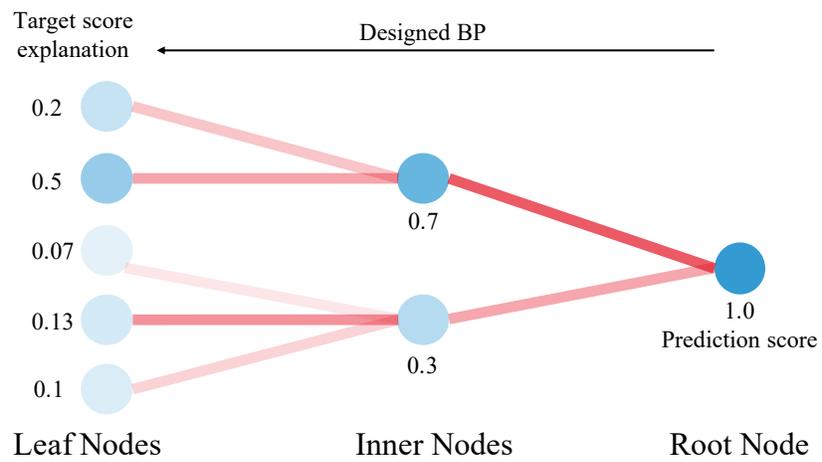
Figure 4: An example of the interpretability of ETL with a decomposition method.