Hierarchical Abstracting Graph Kernel

Runze Yang, Hao Peng, Angsheng Li, Peng Li, Chunyang Liu, Philip S. Yu *Fellow, IEEE*

Abstract—Graph kernels have been regarded as a successful tool for handling a variety of graph applications since they were proposed. However, most of the proposed graph kernels are based on the R-convolution framework, which decomposes graphs into a set of substructures at the same abstraction level and compares all substructure pairs equally; these methods inherently overlook the utility of the hierarchical structural information embedded in graphs. In this paper, we propose **H**ierarchical **A**bstracting **G**raph **K**ernels (HAGK), a novel set of graph kernels that compare graphs' hierarchical substructures to capture and utilize the latent hierarchical structural information fully. Instead of generating non-structural substructures, we reveal each graph's hierarchical substructures by constructing its *hierarchical abstracting*, specifically, the hierarchically organized nested node sets adhering to the principle of structural entropy minimization. To compare a pair of hierarchical abstractings, we propose two novel substructure matching approaches, *Local Optimal Matching* (LOM) and *Priority Ordering Matching* (POM), to find appropriate matching between the substructures by different strategies recursively. Extensive experiments demonstrate that the proposed kernels are highly competitive with the existing state-of-the-art graph kernels, and verify that the hierarchical abstracting plays a significant role in the improvement of the kernel performance.

Index Terms—Graph Classification, Graph Kernels, Structural Entropy.

1 Introduction

Graph kernels [1] are one of the most potent approaches to handling graph-related tasks in various domains. Roughly speaking, the graph kernel methods implicitly map the discrete graph structures into a high-dimensional feature space and measure the similarities between graph pairs. These similarities can then be fed to various kernel-based machine learning algorithms, e.g., SVM [2] and PCA [3], to perform different tasks. For example, in the Web context, graph kernels [4] can help classify graph-represented web pages or documents into categories or topics, aiding in content recommendation or search engine optimization. In addition to leveraging feature information conveyed through nodes and edges, previous works [5], [6], [7], [8], [9] have focused on exploiting graph structural information by comparing various substructures extracted from the whole graph structure. For instance, the random walk graph kernel [6] decomposes graphs into random walks and compares the walk pairs to compute kernel values. Utilizing structural information in this manner can significantly enhance the expressive power of graph kernels, i.e., the ability to distinguish between two graphs, since the difference between substructures from different graphs can be easily detected by comparison.

In the real world, graphs are often hierarchically organized, as illustrated in Fig. 1(a): the substructures consist of sub-substructures. The hierarchical substructures contain

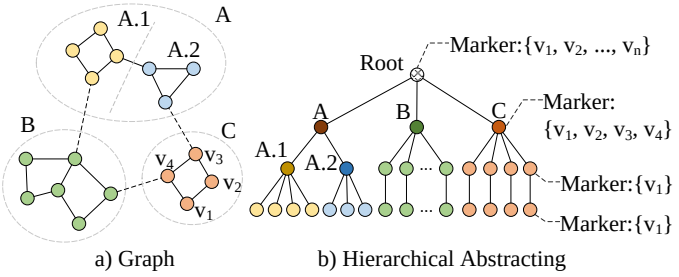
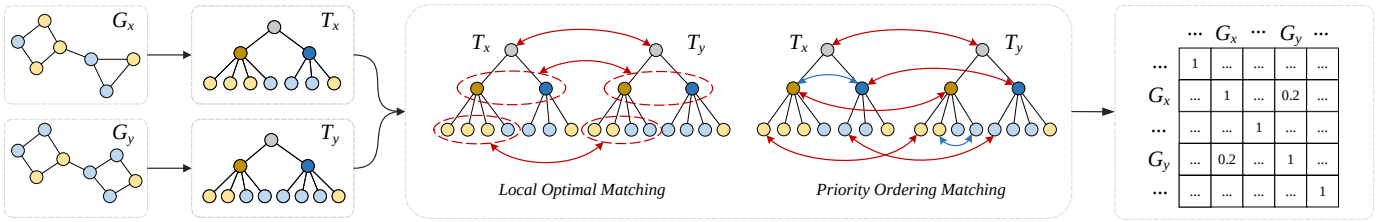


Fig. 1: a) An example of a graph with three 2-level substructures, "A", "B", and "C", where "A" consists of two 3-level substructures "A.1" and "A.2". b) The graph's 3-dimensional hierarchical abstracting. Each tree node has a marker denoting the node set of its corresponding substructure.

rich hierarchical structural information that is potential in graph-related tasks. Intuitively speaking, discovering the hierarchical substructures of graphs can contribute to the expressive power of graph kernels because the difference between the sub-substructures may help distinguish the initially similar substructures based on the "Similarity by Composition" principle [10]. We introduce this principle as follows: If two objects are similar, then each of their composing components must have a distinct similar matching component on the other object. If an object is a tree, then two trees are similar if each subtree from the root has a distinct matching or similar subtree on the other tree. Similarly, for two graphs to be similar, their subgraphs must have matching or similar counterparts. This principle can be applied recursively to determine whether two objects are similar. However, few existing graph kernels [5], [6], [7], [8], [9] follow this principle. Most graph kernels decompose substructures at the same level without hierarchy, thereby ignoring the structural information embedded in their sub-substructure comparisons. For example, the walk substructures decomposed by the random walk graph kernel are not hierarchical and any walk can be compared with any other walk, which violates the "Similarity by Composition"

- *Runze Yang, Hao Peng, and Angsheng Li are with the Guangdong Laboratory of Artificial Intelligence and Digital Economy (SZ), Shenzhen 518132, China, and the State Key Laboratory of Software Development Environment, Beihang University, Beijing 1000191, China. E-mail: {yangrunze, penghao, angsheng}@buaa.edu.cn;*
- *Peng Li is with the Academy of Military Sciences, Beijing 100091, China. E-mail: lp_academy@163.com;*
- *Chunyang Liu is with Didi Chuxing, Beijing 100193, China. E-mail: liuchunyang@didiglobal.com;*
- *Philip S. Yu is with the Department of Computer Science, University of Illinois at Chicago, Chicago, IL 60607, USA. E-mail: psyu@uic.edu.*

Manuscript received March 2024, revised September 2024, accepted November 2024. (Corresponding author: Hao Peng.)



a) Hierarchical abstracting construction b) Hierarchical abstracting comparison by node-matching approaches c) Kernel matrix calculation

Fig. 2: The overview of the proposed HAGK method.

principle [10]. Therefore, a current challenge is to develop a graph kernel that enables hierarchical substructure comparison following the "Similarity by Composition" principle and fully utilizes the hierarchical information embedded in graphs.

To tackle this challenge, we propose a new family of **H**ierarchical **A**bstracting **G**raph **K**ernels (HAGK) to fully capture the hierarchical structural information by decomposing graphs into multi-level nested substructures and comparing the hierarchically organized substructures from different graphs. Rather than decomposing graphs into hierarchically equivalent substructures, we use a tree structure—namely *hierarchical abstractings* (see Fig. 1(b))—to encode the hierarchical structural information within the graphs. The nodes of a hierarchical abstracting represent the substructures in the form of nested node sets extracted from a graph following the *principle of structural entropy minimization* [11], which is widely used to discover a graph's natural hierarchical structure. The graph substructure comparison problem can be converted into a tree comparison problem since each substructure is mapped into a hierarchical abstracting node.

Nevertheless, the hierarchical abstracting comparison raises two key issues: (1) *How to measure the similarity between two hierarchical abstracting nodes?* (2) *How to align or match the hierarchical abstracting nodes at different levels?* For the first issue, one of the most effective ways to measure the similarity is to count the number of "common substructures" between the two comparison objects. Since the objects are non-structural node sets and we mainly focus on node-labeled graphs in this paper, an intuitive way is to calculate the number of common node labels. Therefore, we choose the histogram intersection function [12], which counts common labels of all categories as the inner kernel to compare two hierarchical abstracting nodes. For the second issue, we have two options: one is to follow the R-convolution framework [5], i.e., compare all pairs of hierarchical nodes between two graphs and add the similarities up; the other is to leverage the concept of optimal assignment [12], [13], [14] to find the optimal match between hierarchical abstracting nodes which maximizes the total similarities. We choose the latter for the following two reasons: the time consumption will be too high if we compare all pairs of nodes when the tree height goes deep; the tree structure is naturally suitable to recursively "one-to-one" node matching from root to leaves. To be more specific, we introduce a new node-matching method named *Local Optimal Matching* (LOM), which first matches the roots, and then recursively finds the local optimal matching between the direct successors of two matched nodes, thereby maximizing their total similarity. This node-matching approach keeps all comparisons at the same level and avoids meaningless comparisons, e.g. root and leaf nodes are intuitively not comparable. However, the optimal assignment-based LOM has two inherent disadvantages: the high time complexity and no theoretical guarantee for the positive semi-definite property which is critical in kernel validity [15]. The former is due to the extra time consumption of the Jonker-Volgenant algorithm [16] from the optimal matching process while the latter is mainly because the order of substructure alignment between different graphs is not transitive making the induced similarity measure not satisfy the triangle inequality. To address these issues, we introduce another node-matching approach *Priority Ordering Matching* (POM) which first orders the sibling nodes of hierarchical abstractings and then matches the arranged nodes according to their relative positions. Its hierarchical ordering process not only reduces the matching time but also keeps the substructure alignment order to guarantee the positive semi-definite property. Finally, LOM and POM can induce corresponding HAGK kernels by adding up the similarities between all matched nodes and the kernel matrix can then be calculated for each pair of graphs in a dataset preparing for downstream tasks like graph classification.

The overview of the proposed HAGK method is shown in Fig. 2. First, the hierarchical abstractings of graphs are constructed (Fig. 2(a)). Second, the hierarchical abstractings are compared pairwise by LOM or POM to compute kernel values (Fig. 2(b)). Third, the kernel matrix is formed by collecting the kernel values of all pairs of hierarchical abstractings and then given to SVM [2] to perform classification (Fig. 2(c)). To summarize, our contributions are three-fold.

• **Problem.** We design a new set of HAGK kernels to fully leverage hierarchical structural information by comparing graphs' hierarchical substructures. To our best knowledge, this is the *first* attempt to compare hierarchically nested structures decomposed from graphs for kernel computation.

• **Algorithm.** We propose two novel matching approaches, LOM and POM, which find appropriate matching between the hierarchical substructures by different strategies.

• **Evaluation.** We have conducted extensive experiments to validate the effectiveness and analyze the properties of the HAGK kernels via thorough comparisons with state-of-the-art graph kernels.

This paper is organized as follows. Section 2 and Section 3 present the related work and preliminary, respectively. Section 4 outlines our proposed graph kernels. Section 5 shows the results and analysis of the experiments. Section 6 concludes the paper.

2 Related Work

Below we summarize research on graph kernels, structural entropy, and similarity of hierarchical domain structures.

2.1 Graph Kernels

One of the most popular methods to handle graph tasks is graph neural network models (GNN) [17], [18], [19]. The advantages of GNN methods include their ability to automatically learn feature representations of graphs, handle large-scale graph data, and capture the topological structure and relationships between nodes [17], [20]. However, they may suffer from overparameterization and overfitting on small graphs [21], [22]. Graph kernel methods [1] are another type of traditional approach for computing graph similarities with a certain theoretical basis. Some graph kernels, like WLSK [23], can consider the distribution of node labels and capture the distribution of subtree patterns using the Wasserstein distance. Nevertheless, some graph kernels may have high time complexity when dealing with large-scale graph data.

Most graph kernels are developed based on the R-convolution framework [5], which decomposes graphs into substructures and compares graphs regarding the pairwise similarity between all their substructures. The types of the substructures include walks [6], paths [7], cycles [8] and subtrees [9], [23], etc. Besides the R-convolution kernels, another family of graph kernels, the optimal assignment kernels [13], [14], [12], has also received a lot of attention. These kernels compare graphs by finding the optimal matching between the substructures, which maximizes the overall similarity. However, most R-convolution kernels and optimal assignment kernels compare substructures at the same level, lacking the ability to utilize the hierarchical structural information embedded in graphs, which our method focuses on.

The subgraph matching kernel [24] counts the number of matchings between subgraphs up to a fixed size and therefore has polynomial runtime. The Wasserstein Weisfeiler-Lehman kernel [25] extends WLSK to continuously attributed graphs by treating each graph as node embeddings and using the Wasserstein distance to compare them. The isolation graph kernel [26] employs a distributional kernel in the framework of kernel mean embedding, avoiding the costly computation of the Wasserstein distance. The hierarchical transitive-aligned graph kernel (HTAK) [27] transitively aligns vertices between pairs of graphs by means of a family of hierarchical prototype representations. The multi-scale Wasserstein shortest-path graph kernel (MWSP) [28] employs the Wasserstein distance to compute the similarity between the multi-scale shortest-path node feature maps of two graphs, capturing the distributions of shortest paths. To conclude, [24], [25] and [26] support attributed graphs while [27] and [28] mainly focus on unattributed graphs. The graph neural tangent kernel (GNTK) [21] is inspired by the connections between overparameterized neural networks and kernel methods. It is equivalent to an infinitely wide GNN trained by gradient descent and has been proven to learn a class of smooth functions on graphs. GraphQNTK [22] extends GNTK by incorporating the attention mechanism and quantum computing into GNTK's structure and computation. The faster graph neural tangent kernel [29] accelerates GNTK by decoupling the Kronecker-vector product in GNTK construction into matrix multiplications and a newly designed iterative sketching algorithm. GNTK and its variants are powerful due to the effectiveness of the GNN's representation. Nevertheless, they may be overparameterized and overfitting on small graphs.

2.2 Structural Entropy

In recent years, Li and Pan [11] have proposed *structural entropy* for measuring structural information embedded in graph data. By minimizing the structural entropy, the natural hierarchical structures of a graph can be revealed. This information measurement has been used extensively in the fields of biological information [30], [31], information security [32], [33], and graph neural networks [34], [35], [36], [37]. In this paper, we construct hierarchical abstractings following the principle of structural entropy minimization for utilizing hierarchical structural knowledge.

2.3 Similarity of Hierarchical Domain Structures

Recently, some work has been done on the comparison of hierarchical structures. For instance, Ganesan et al. [38] have developed a method to compare hierarchical domain structures extracted from the data in the field of information retrieval. Wu et al. [39] improve the Weisfeiler-Lehman graph kernel by propagating labels from child nodes to their parents based on the hierarchical structures of the encoding trees [11]. To the best of our knowledge, the HAGK kernels firstly give a method to compare hierarchically nested substructures decomposed from graphs in an optimal assignment scheme.

3 Preliminary

In this paper, our main focus is on node-labeled, undirected, and unweighted graphs. Given a graph dataset $\mathcal{G} = \{G_1, G_2, ..., G_{|\mathcal{G}|}\}$, each graph in $\mathcal{G}$ is denoted as $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ and $\mathcal{E}$ represent the set of nodes and the set of edges in the graph, respectively. Assume that each node has a label $l \in \mathcal{L}$, where $\mathcal{L} = \{l_1, l_2, ..., l_{|\mathcal{L}|}\}$ denotes the label set of $\mathcal{G}$. In the following, we introduce the basic definitions and foundations used throughout the paper.

To represent the hierarchical substructures, we use a novel tree structure, namely *hierarchical abstracting* (Definition 1), by adapting the concept of the partitioning tree proposed by Li and Pan [11]. Different from the partitioning tree in [11], the height of leaf nodes in the hierarchical abstracting is designed to be the same for the convenience of comparison. The height of the root of a hierarchical abstracting is set as 1 and $(\text{leaf height} - 1)$ is defined as a dimension of the hierarchical abstracting to keep consistent with [11]. In this representation, each i-level substructure is denoted by an i-height tree node of the graph's hierarchical abstracting. Each i-height non-leaf tree node has several $(i+1)$-height direct successors, denoting that each i-level substructure is decomposed into several $(i+1)$-level non-overlap substructures, forming a partition of the i-level substructure. The node sets of all substructures on the same level constitute a partition of the graph node set. Fig. 1 gives an example of a graph and its 3-dimensional hierarchical abstracting.

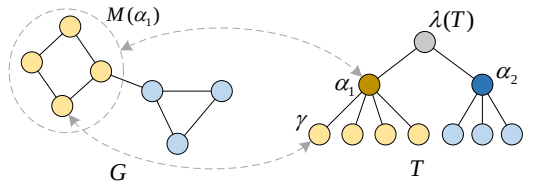


Fig. 3: An illustration of the hierarchical abstracting and structural entropy calculation. Tree node α_1 corresponds to the node set $M(\alpha_1)$. Leaf node γ corresponds to a yellow graph node. The node structural entropy calculation of α_1 and γ is listed below.

To find the best hierarchical abstracting, we generalize the principle of the structural entropy minimization (Definition 2) [11]. Fig. 3 illustrates the calculation of structural entropy. Following this principle, we construct the hierarchical abstracting for each graph to reveal the graph's essential hierarchical structure by minimizing the structural entropy.

Definition 1 (Hierarchical Abstracting)**.** *The hierarchical abstracting T of an undirected unweighted graph $G = (\mathcal{V}, \mathcal{E})$ is a directed tree with a root node $\lambda(T)$. Let $V(T)$ be the node set of T. Let the marker of a node $\alpha \in V(T)$, denoted by $M(\alpha)$, be a subset of $\mathcal{V}$. Let $M(\lambda(T)) = \mathcal{V}$. Let α^+ be the set of the direct successors of a non-leaf node α. Let α_i^+ be the i-th direct successor of α from left to right. For each non-leaf node α, $M(\alpha_1^+), M(\alpha_2^+), ..., M(\alpha_{|\alpha^+|}^+)$ constitute a partition of $M(\alpha)$. The marker of a leaf node γ is a single node set, i.e., $|M(\gamma)| = 1$.*

Definition 2 (Structural Entropy Minimization Principle)**.** *The optimal hierarchical abstracting T^* of $G = (\mathcal{V}, \mathcal{E})$ minimizes the structural entropy of G, i.e.,*

$$T^* = \min_T H^T(G). \tag{1}$$

$H^T(G)$ denotes the structural entropy of G by a hierarchical abstracting T:

$$H^T(G) = \sum_{\alpha \in V(T) - \{\lambda(T)\}} H^T(G, \alpha), \tag{2}$$

where $H^T(G, \alpha)$ is the node structural entropy of α:

$$H^T(G, \alpha) = -\frac{\mathrm{cut}(\alpha)}{2|\mathcal{E}|} \log_2 \frac{\mathrm{vol}(\alpha)}{\mathrm{vol}(\alpha^-)}, \tag{3}$$

where $\mathrm{cut}(\alpha)$ represents the number of cut edges of $M(\alpha)$; $\mathrm{vol}(\alpha)$ denotes the volume of $M(\alpha)$, i.e., the sum of the degrees of all nodes in $M(\alpha)$; α^- is the parent of α.

4 The Proposed HAGK

In this section, we first present the two novel node-matching approaches, LOM and POM (Section 4.1), which compare two hierarchical abstractings. After that, we introduce the HAGK kernels based on the two approaches (Section 4.2).

4.1 The Node-Matching Approaches

Before we introduce the two node-matching approaches, we first describe the inner kernel, i.e., the histogram intersection function [12], which measures the similarity between two matched hierarchical abstracting nodes. Let T_x denote the hierarchical abstracting of $G_x \in \mathcal{G}$ and $\lambda(T_x)$ denote the root of T_x. Given two nodes α and β from two different hierarchical abstractings, the histogram intersection function between two hierarchical abstracting nodes is defined as:

$$s(\alpha, \beta) = \sum_{i=1}^{|\mathcal{L}|} \min\{h(M(\alpha), i), h(M(\beta), i)\}, \tag{4}$$

where $h(M(\alpha), i)$ denotes the number of the graph nodes with label l_i in $M(\alpha)$. Let $\alpha \bowtie \beta$ denote that α and β are matched. In the following, we describe how hierarchical abstracting nodes are matched in LOM and POM, respectively.

4.1.1 Local Optimal Matching (LOM)

We provide the definition of LOM in Definition 3, accompanied by an illustration in Fig. 4. As shown in this figure, LOM first matches the roots of a pair of hierarchical abstracting. Next, it matches the roots' direct successors to maximize the sum of the pairwise similarities. Finally, the direct successors of the newly matched nodes are matched recursively.

Definition 3 (Local Optimal Matching)**.** *Given two hierarchical abstractings T_x and T_y, their roots are first matched, i.e., $\lambda(T_x) \bowtie \lambda(T_y)$. For two nodes $\alpha \in V(T_x)$ and $\beta \in V(T_y)$, if $\alpha \bowtie \beta$, then $\alpha_i^+ \bowtie \beta_{q^*(i)}^+$ (w.l.o.g. $|\alpha^+| \leq |\beta^+|$), where q^* is an injection between the subscripts of the direct successors of α and β and satisfies:*

$$q^* = \arg\max_{q \in Q} \sum_{i=1}^{|\alpha^+|} s(\alpha_i^+, \beta_{q(i)}^+), \tag{5}$$

where Q is the set of all injections from $\{1, 2, ..., |\alpha^+|\}$ to $\{1, 2, ..., |\beta^+|\}$.

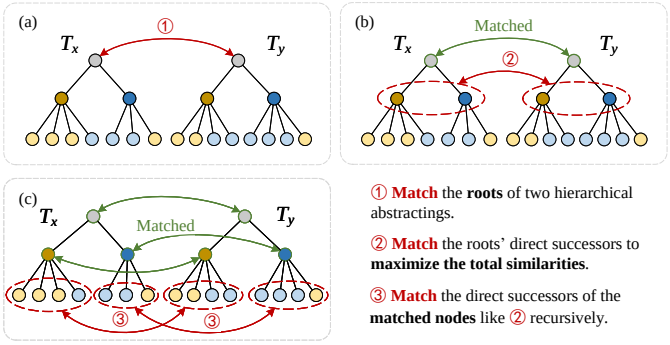


Fig. 4: An illustration of LOM.

4.1.2 Priority Ordering Matching (POM)

Node Order Relation. We begin by introducing the concept of node order relation, which defines the priority of the nodes of a hierarchical abstracting. Roughly speaking, we choose the *node structural entropy* defined in Eq. (3) as the priority indicator. That is, given two nodes α_1 and α_2 from a hierarchical abstracting T, we define that α_1 has priority over α_2, denoted by $\alpha_1 \succcurlyeq \alpha_2$, if $H^T(G, \alpha_1) \geq H^T(G, \alpha_2)$. One of the main reasons why we use node structural entropy is that it approximately represents the probability of random walking from the whole graph to a particular hierarchical substructure [11]. When we compare two hierarchical abstractings ordered by node structural entropy, the matched substructures usually have similar graph invariants (cut edge

and volume) and the same rank of random walk probability, which indicates that this measure does have the capability for characterizing and matching appropriate substructures. Another reason is that the node structural entropy can be easily obtained as the intermediate of the hierarchical abstracting construction process, which requires no extra time complexity.

We name $\boldsymbol{h}(M(\alpha)) = [h(M(\alpha),1), ..., h(M(\alpha),|\mathcal{L}|)]$ as the label histogram of α. Given two hierarchical abstracting nodes α and β, we define $\boldsymbol{h}(M(\alpha)) > \boldsymbol{h}(M(\beta))$ as $\exists N \in \{1,2,...,|\mathcal{L}|\}, \forall i \in \{1,2,...,N-1\}, h(M(\alpha),i) = h(M(\beta),i)$ and $h(M(\alpha),N) > h(M(\beta)\ ,N)$. The strict definition of node order relation is shown in Definition 4. In this definition, we first recursively define that two nodes are equal when their node structural entropy values, label histograms, and direct successors are equal, respectively. Secondly, we recursively define that a node is greater than or equal ("$\succcurlyeq$") to another node, roughly speaking, when the former's node structural entropy, label histogram, or direct successors is greater than or equal ("$\succcurlyeq$") to the latter's. Note that only the first condition of "$\succcurlyeq$" definition (the node order relation indicator) is used most of the time and the others are mainly used to keep the uniqueness of the ordered hierarchical abstracting to guarantee the positive semi-define property.

Definition 4 (Node Order Relation)**.** *Given two hierarchical abstracting nodes $\alpha \in V(T)$ and $\beta \in V(T')$ where T and T' are both constructed from G. Define $\alpha = \beta$ if and only if (1) $H^T(G,\alpha) = H^{T'}(G,\beta)$ (if α and β are non-root nodes) and $\boldsymbol{h}(M(\alpha)) = \boldsymbol{h}(M(\beta))$; and (2) $|\alpha^+| = |\beta^+| = 0$, or $|\alpha^+| = |\beta^+| \neq 0$, $\forall i \in \{1,2,...,|\alpha^+|\}$, $\alpha_i^+ = \beta_i^+$. Define $\alpha \succcurlyeq \beta$ if and only if α and β satisfy one of the following conditions:*

1) $H^T(G,\alpha) > H^{T'}(G,\beta)$;
2) $H^T(G,\alpha) = H^{T'}(G,\beta)$ *and* $\boldsymbol{h}(M(\alpha)) > \boldsymbol{h}(M(\beta))$;
3) $H^T(G,\alpha) = H^{T'}(G,\beta)$ *and* $\boldsymbol{h}(M(\alpha)) = \boldsymbol{h}(M(\beta))$; $\exists N \in \{1,2,...,\min(|\alpha^+|,|\beta^+|)\}$, $\forall i \in \{1,2,...,N\}$, $\alpha_i^+ = \beta_i^+$ *and* $\alpha_{N+1}^+ \succcurlyeq \beta_{N+1}^+$;
4) $H^T(G,\alpha) = H^{T'}(G,\beta)$ *and* $\boldsymbol{h}(M(\alpha)) = \boldsymbol{h}(M(\beta))$; $\forall i \in \{1,2,...,\min(|\alpha^+|,\ |\beta^+|)\}$, $\alpha_i^+ = \beta_i^+$ *and* $|\alpha^+| \geq |\beta^+|$;
5) $\alpha = \beta$.

Ordered Hierarchical Abstracting. We first give the definition of the equivalence of hierarchical abstractings (Definition 5). Briefly speaking, if we can convert T into T' only by swapping sibling nodes of T without changing the corresponding decomposed substructures, T and T' are equivalent. In other words, the hierarchical substructures represented by T and T' are the same. Next, we describe the final ordered hierarchical abstracting (Definition 6): for each node, all its direct successors are arranged in descending order of structural entropy. This ensures the same alignment order when comparing substructures between different graphs, which makes the kernel transitive and thus positive semi-definite. In addition, we prove that any hierarchical abstracting can only be converted into a unique ordered hierarchical abstracting (Theorem 1), which also aims to guarantee the positive semi-definite property.

Definition 5 (Equivalence of Hierarchical Abstractings)**.** *Given two hierarchical abstractings T and T' constructed from G, we define that T equals to T', denoted by $T = T'$, as $\lambda(T) = \lambda(T')$. We say that T are equivalent to T', denoted by $T \cong T'$, if and only if T can be converted into T' only by swapping the position of its sibling nodes.*

Definition 6 (Ordered Hierarchical Abstracting)**.** *An ordered hierarchical abstracting $\vec{T}$ is a hierarchical abstracting where each non-leaf node α's r direct successors satisfy $\alpha_1^+ \succcurlyeq \alpha_2^+ \succcurlyeq ... \succcurlyeq \alpha_r^+$.*

Lemma 1. *For any hierarchical abstracting T, there must exist an ordered hierarchical abstracting $\vec{T}$ such that $\vec{T} \cong T$.*

Proof. Let T' be a copy of T. Execute Order($\lambda(T')$) (Algorithm 4). Then for each non-leaf node $\alpha \in V(T')$, its r direct successors satisfy $\alpha_1^+ \preccurlyeq \alpha_2^+ \preccurlyeq ... \preccurlyeq \alpha_r^+$. So T' is an ordered hierarchical abstracting. Since Algorithm 4 sorts the nodes only by swapping their position, we have $T' \cong T$. Therefore Lemma 1 is proved. □

Lemma 2. *Let T be an arbitrary hierarchical abstracting. Let $\vec{T}$ be an ordered hierarchical abstracting and $\vec{T} \cong T$. Then $\vec{T}$ is unique.*

Proof. Suppose that there exists two ordered hierarchical abstractings $\vec{T}_1$ and $\vec{T}_2$ where $\vec{T}_1 \cong T$, $\vec{T}_2 \cong T$ and $\vec{T}_1 \neq \vec{T}_2$. Obviously we have $\vec{T}_1 \cong \vec{T}_2$. According to Definition 5, $\lambda(\vec{T}_1) \neq \lambda(\vec{T}_2)$ since $\vec{T}_1 \neq \vec{T}_2$. Due to $\boldsymbol{h}(M(\lambda(\vec{T}_1))) = \boldsymbol{h}(M(\lambda(\vec{T}_2)))$ and $|\lambda(\vec{T}_1)^+| = |\lambda(\vec{T}_2)^+|$, we have $\exists i \in \{1,2,...,|\lambda(\vec{T}_1)^+|\}$, $\lambda(\vec{T}_1)_i^+ \neq \lambda(\vec{T}_2)_i^+$ (Definition 6). Let α and β denote a pair of unequal nodes at the same position in $\lambda(\vec{T}_1)^+$ and $\lambda(\vec{T}_2)^+$. If $H^{\vec{T}_1}(G,\alpha) \neq H^{\vec{T}_2}(G,\beta)$ or $\boldsymbol{h}(M(\alpha)) \neq \boldsymbol{h}(M(\beta))$ or $|\alpha^+| \neq |\beta^+|$, the order of the nodes in $\vec{T}_1$ and $\vec{T}_2$ must be different. Hence, $\vec{T}_1$ and $\vec{T}_2$ cannot be ordered at the same time (contradiction). If $|\alpha^+| = |\beta^+|$, we have $\exists i \in \{1,2,...,|\alpha^+|\}$, $\alpha_i^+ \neq \beta_i^+$. We then let α' and β' denote a pair of unequal nodes at the same position in α'^+ and β'^+, and the above process can be executed recursively. When α' and β' are leaf nodes, $|\alpha'^+| = |\beta'^+| = 0$, and thus $\alpha' = \beta'$ (contradiction). Thus $\vec{T}_1 = \vec{T}_2$ and Lemma 2 is proved. □

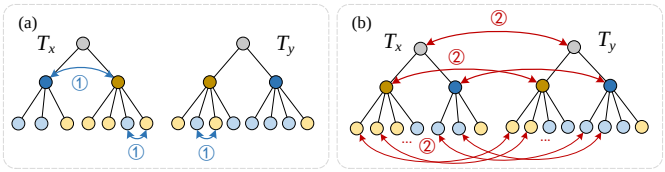
Theorem 1. *Given an arbitrary hierarchical abstracting T, it must be equivalent to a unique ordered hierarchical abstracting $\vec{T}$.*

Proof. Theorem 1 holds since Lemma 1 and 2 hold. □

Definition of POM. Finally, we give the definition of POM in Definition 7 and an illustration in Fig. 5. As shown in this figure, POM first swaps the siblings to get the ordered hierarchical abstractings based on the node order relation and then matches the nodes at the same relative position of the ordered hierarchical abstractings.

Definition 7 (Priority Ordering Matching)**.** *Given two ordered hierarchical abstractings $\vec{T}_x$ and $\vec{T}_y$, their roots are first matched, i.e., $\lambda(\vec{T}_x) \bowtie \lambda(\vec{T}_y)$. For two nodes $\alpha \in V(\vec{T}_x)$ and $\beta \in V(\vec{T}_y)$, if $\alpha \bowtie \beta$, then $\alpha_i^+ \bowtie \beta_i^+$, $i = 1,2,...,\min\{|\alpha^+|,|\beta^+|\}$.*

Discussion. If two hierarchical abstractings, T_x and T_y, are closely matched, the POM generally will produce the optimal matching on substructures as the corresponding substructures will follow the same priority ordering. That is, matching pairs will not be missed under POM. However, if two hierarchical abstractings, T_x and T_y, are not closely matched, e.g., one high-priority substructure in T_x is not present in T_y, the POM may result in a less optimal matching



① **Swap** the siblings to get **ordered hierarchical abstractings** according to the node order relation. ② **Match** the nodes at the same relative position of the ordered hierarchical abstractings.

Fig. 5: An illustration of POM.

of their substructures and a lower kernel value. This should not be a problem since the goal is to identify the pair as a non-match. In Section 5.3 the effectiveness of POM is validated.

4.2 The Hierarchical Abstracting Graph Kernels

In this subsection, we describe the HAGK kernels based on LOM and POM, namely HAGK-LOM and HAGK-POM (Section 4.2.1). They add up the similarity of the matched nodes between a pair of hierarchical abstractings to compute the kernel value. We also prove that the HAGK-POM kernel is positive semi-definite, enabling it to implicitly map data to higher-dimensional spaces and capture complex relationships. In addition, we analyze the computational complexity of the hierarchical abstracting construction and the HAGK kernel value computation algorithms (Section 4.2.2).

4.2.1 HAGK-LOM and HAGK-POM

Definition 8 (HAGK-LOM and HAGK-POM)**.** *For a pair of graphs G_x and G_y, T_x and T_y are two hierarchical abstractings constructed from G_x and G_y by a deterministic algorithm $\mathcal{S}$. The HAGK-LOM kernel k_{LOM} between G_x and G_y is defined as:*

$$k_{LOM}(G_x, G_y; \mathcal{S}) = \sum_{\alpha \in V(T_x)} \sum_{\beta \in V(T_y)} \mathbb{I}_{LOM}(\alpha \bowtie \beta) s(\alpha, \beta), \quad (6)$$

where $\mathbb{I}_{LOM}(\alpha \bowtie \beta) = 1$ if α and β are matched by LOM, otherwise 0. Let $\vec{T}_x$ and $\vec{T}_y$ be two ordered hierarchical abstractings where $\vec{T}_x \cong T_x$ and $\vec{T}_y \cong T_y$. The HAGK-POM kernel k_{POM} between G_x and G_y is defined as:

$$k_{POM}(G_x, G_y; \mathcal{S}) = \sum_{\alpha \in V(\vec{T}_x)} \sum_{\beta \in V(\vec{T}_y)} \mathbb{I}_{POM}(\alpha \bowtie \beta) s(\alpha, \beta), \quad (7)$$

where $\mathbb{I}_{POM}(\alpha \bowtie \beta) = 1$ if α and β are matched by POM, otherwise 0.

Theorem 2. *HAGK-POM is positive semi-definite.*

Proof. Given a deterministic hierarchical abstracting construction algorithm $\mathcal{S}$, any graph $G = (\mathcal{V}, \mathcal{E}) \in \mathcal{G}$ must correspond to a unique hierarchical abstracting T. According to Theorem 1, there exists a unique ordered hierarchical abstracting $\vec{T} \cong T$. We use a coordinate vector $\vec{a} = [a_1, a_2, ..., a_k]$ to denote a certain k-height node of $\vec{T}$, where a_i is an integer and $1 \le a_i \le |\mathcal{V}|$. $[a_1]$ denotes the a_1-th direct successor of $\lambda(\vec{T})$, and $[a_1, a_2, ..., a_i]$ represents the a_i-th direct successor of the node at $[a_1, a_2, ..., a_{i-1}]$. At this time, all nodes in $V(\vec{T})$ are mapped into a complete η-ary tree T_C, in which each non-leaf node has η direct successors, where η is the maximum node number of all graphs in $\mathcal{G}$. The node denoted by $[a_1, a_2, ..., a_k]$ is mapped to the n-th node of all k-height nodes in T_C where

$$n = 1 + \sum_{i=1}^{k} (a_i - 1)\eta^{k-i}. \quad (8)$$

Thus, we can encode the markers and positions of all nodes of $\vec{T}$ with an ∞-dimensional vector θ by a hierarchical traversal of T_C:

$$\theta_j = \begin{cases} \xi(\vec{a}), j \in \{1 + \sum_{i=1}^{k} a_i \eta^{k-i} | \vec{a} \in V(\vec{T})\}; \\ 0, \text{others}, \end{cases} \quad (9)$$

where θ_j represents the j-th node of the hierarchical traversal on T_C, and $\xi(\vec{a}) = [\xi_1(\vec{a}), \xi_2(\vec{a}), ..., \xi_D(\vec{a})]$ represents the encoding vector of the marker of $\vec{a}$ where each part $\xi_i(\vec{a})$ is a η-dimensional 0-1 vector in which

$$\xi_{it}(\vec{a}) = \begin{cases} 1, 1 \le t \le h(M(\vec{a}), i); \\ 0, \text{others}. \end{cases} \quad (10)$$

Therefore, each graph G corresponds to a unique encoding vector, denoted by $\theta(G)$. For a pair of graphs G_1 and G_2, the HAGK-POM kernel is equal to the inner product of their encoding vectors, i.e.,

$$k_{\text{POM}}(G_1, G_2; \mathcal{S}) = < \theta(G_1), \theta(G_2) > . \quad (11)$$

Therefore, HAGK-POM maps a graph into a ∞-dimensional vector space, thus Theorem 2 is proved. Actually, whether HAGK-POM is positive semi-definite depends on whether the inner kernel is positive semi-definite. Obviously, $s(\cdot, \cdot)$ is positive semi-definite since

$$s(\vec{a}, \vec{b}) = < \xi(\vec{a}), \xi(\vec{b}) > . \quad (12)$$

□

Fig. 6 shows a brief illustration of Theorem 2. First, the sample graph is converted to a unique hierarchical abstracting using $\mathcal{S}$ (Fig. 6(a)-(b)). Second, the hierarchical abstracting is ordered by Algorithm 4 (Fig. 6(c)). Third, each node of the ordered hierarchical abstracting is mapped with the node at the same position of a complete η-ary tree (Fig. 6(d)). Finally, the mapped complete η-ary tree is encoded into an encoding vector (Fig. 6(e)). Thus, the graph kernel can then be regarded as the inner product of two encoding vectors and is obviously positive semi-definite. However, there exists a strong assumption that $\mathcal{S}$ must be a deterministic algorithm that any two same input graphs must be converted into a unique hierarchical abstracting. That is, the input order of the edge sequences must not affect the output. The hierarchical abstracting construction algorithm used in our experiments is sensitive to the input order of data, and developing an input-insensitive construction algorithm is rather difficult.

4.2.2 Algorithms and Computational Ananlysis

Hierarchical Abstracting Construction. It is a cost-effective variant of the structural entropy minimization algorithm proposed by Li et al. [30] which converts a graph into its 3-dimensional hierarchical abstracting. For a complete graph classification evaluation process, we need to compare all

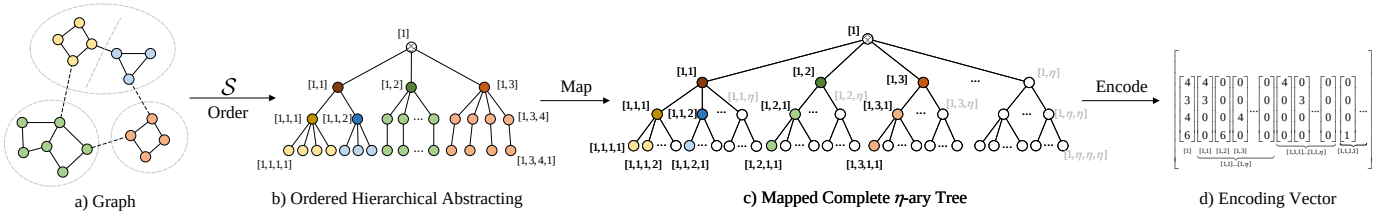


Fig. 6: Illustration of the proof of Theorem 2.

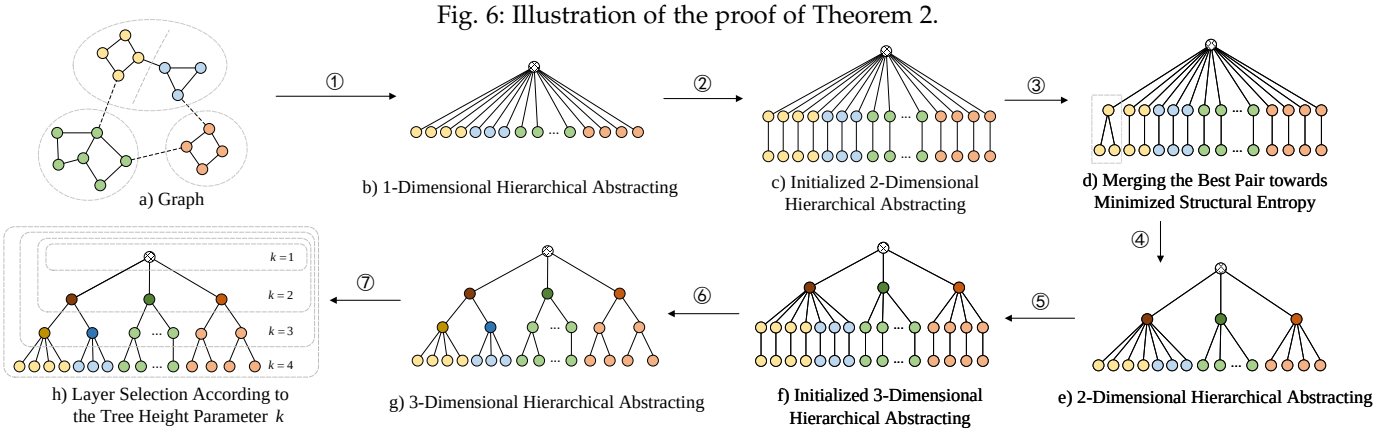


Fig. 7: Illustration of the hierarchical abstracting construction. The graph is first initialized to an initialized k-height ($(k-1)$-dimensional) hierarchical abstracting, and then nodes are merged greedily to get a k-height hierarchical abstracting.

pairs of graphs in a dataset. Therefore we pre-construct the hierarchical abstractings for all graphs and save the node structural entropy for the POM process. The total time complexity of the pre-constructing process is $O(n^3)$ for each 3-dimensional hierarchical abstracting, where n denotes the maximum number of nodes in each graph.

Algorithm 1 shows the pseudo-code of the 3-dimensional hierarchical abstracting construction following Li et al. [30]. First, an initial 2-dimensional hierarchical abstracting is constructed (line 1-3). It has $|\mathcal{V}|$ 2-height nodes and $|\mathcal{V}|$ 3-height nodes. Each of the 2- and 3-height nodes have a single node set marker. Second, we use Algorithm 2 to repeatedly merge the optimal pair of the direct successors of the root, which minimizes the structural entropy after merging (line 4). Third, we add $|\mathcal{V}|$ 4-height nodes to the hierarchical abstracting and execute Algorithm 2 for each 3-height node to get the 3-dimensional hierarchical abstracting (line 5-9).

Fig. 7 gives an illustration of the hierarchical abstracting construction and the layer selection according to the tree height k. In this instance, the sample graph (Fig. 7(a)) is first converted into a 1-dimensional hierarchical abstracting (Fig. 7(b)). Each leaf node corresponds to a node in the graph. Second, each leaf node is appended with a child that corresponds to the same node of the graph, forming the initialized 2-dimensional hierarchical abstracting (Fig. 7(c)). Third, the best pair of the 2-height nodes are merged towards the minimized structural entropy (Fig. 7(d)). The merged nodes correspond to all nodes its children correspond to. Fourth, the merging operation is repeatedly executed until no candidate pair is able to be merged to decrease the structural entropy (Fig. 7(e)). Fifth, like the second step, the initialized 3-dimensional hierarchical abstracting is formed (Fig. 7(f)). Sixth, the children of each 2-height node are repeatedly merged to get the final 3-dimensional hierarchical abstracting (Fig. 7(g)). At last, we choose the top k layer to compare with other hierarchical abstracting according to the tree height parameter (Fig. 7(h)).

Kernel Value Computation. For the HAGK-LOM kernel, we use $\mathsf{LOM}(\lambda(T_x), \lambda(T_y))$ (Algorithm 3) to recursively match the nodes and add up the node pair similarities of T_x and T_y. Let n_i be the total number of the i-height nodes of a hierarchical abstracting, and let $n_{i,j}$ denote the number of the direct successors of the j-th $(i-1)$-height node. In this algorithm, the key step is matching nodes between two given node sets (line 6), in which we first calculate all the pairwise similarities to form a matrix ($O(n_{i,j}^2|\mathcal{L}|)$) and then use the Jonker-Volgenant algorithm [16] to find the optimal assignment ($O(n_{i,j}^3)$). Hence, the time complexity of HAGK-LOM is $O(n_2^2|\mathcal{L}| + n_2^3 + \sum_j(n_{3,j}^2|\mathcal{L}| + n_{3,j}^3) + \sum_z(n_{4,z}^2|\mathcal{L}| + n_{4,z}^3))$ for 3-dimensional hierarchical abstracting comparison. Since $n \geq \sum_z n_{4,z} \geq \sum_j n_{3,j} \geq n_2$, the total time complexity is $O(n^2(|\mathcal{L}|+n))$. For the HAGK-POM kernel, we first order the hierarchical abstractings using the hierarchical abstracting ordering algorithm. Specifically, this algorithm receives a node of an arbitrary hierarchical abstracting and orders all successors of the given node recursively.

Algorithm 4 shows the pseudo-code of the hierarchical abstracting ordering algorithm, and Fig. 8 gives an example of execute $\mathsf{Order}(\lambda(T))$ on the sample hierarchical abstracting T. In this example, we suppose that nodes with lighter colors should be further to the left. Fig. 8(a) gives a 3-dimensional hierarchical abstracting that is not yet ordered. We then execute Algorithm 4 with the input of its root node. Sorting will be done from bottom to top, from left to right (Fig. 8(a)-(e)). Finally, we get the ordered sample hierarchical abstracting shown as Fig. 8(f). Then we compute the kernel value using $\mathsf{POM}(\lambda(T_x), \lambda(T_y))$ (Algorithm 5). For the hierarchical abstracting ordering algorithm, we recursively sort the direct successors of each non-leaf

node. In the worst case, all the sorting direct successors are equal according to Definition 4. All their successors need to be traversed, and all labels need to be compared once while comparing. Then the worst time complexity is $O(\sum_z n_{4,z} \log n_{4,z} |\mathcal{L}| + \frac{n_4}{n_3} \sum_j n_{3,j} \log n_{3,j} |\mathcal{L}| + (\frac{n_3}{n_2} + \frac{n_4}{n_2}) n_2 \log n_2 |\mathcal{L}|) = O(n_4 \log \frac{n_4}{n_2} |\mathcal{L}| + (n_3 + n_4) \log n_2 |\mathcal{L}|) = O(n \log n |\mathcal{L}|)$. For Algorithm 5, nodes are directly matched according to their positions, and their similarities are added up to get the kernel value, which requires $O(n_2|\mathcal{L}| + n_3|\mathcal{L}| + n_4|\mathcal{L}|) = O(n|\mathcal{L}|)$. In summary, the total time complexity of HAGK-POM is $O(n \log n |\mathcal{L}|)$. Overall, the time complexity of the above is summarized in Table 1.

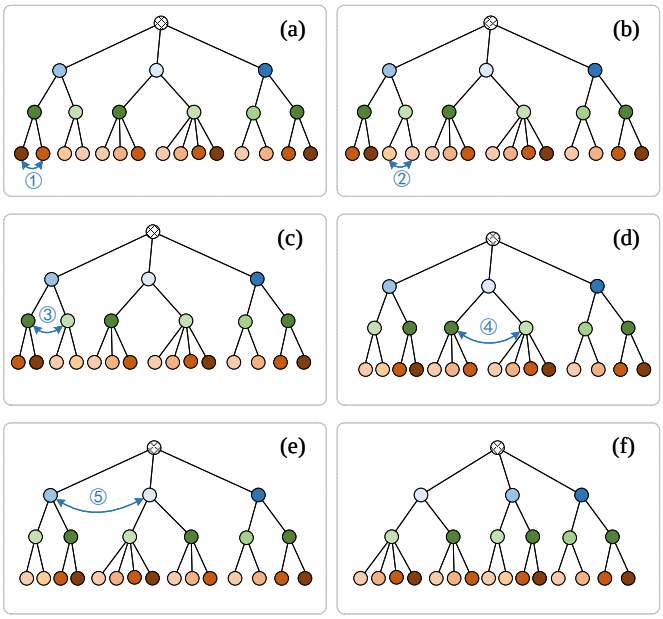


Fig. 8: An illustration of the hierarchical abstracting ordering algorithm. The steps are briefly listed as follows. (a-b) The leaf nodes (4-height nodes) are ordered. (c-d) The 3-height nodes are ordered. (e) The 2-height nodes are ordered. The final ordered hierarchical abstracting is shown in (f).

Algorithm 1: Hierarchical Abstracting Construction

Input: A graph $G = (\mathcal{V}, \mathcal{E})$.
Output: 3-dimensional hierarchical abstracting T^3 of G.

Let T be a hierarchical abstracting with only a root node $\lambda(T)$ where $M(\lambda(T)) = \mathcal{V}$;
Let the direct successors of $\lambda(T)$ be $|\mathcal{V}|$ new 2-height nodes $\gamma_1^{(2)}, \gamma_2^{(2)}, ..., \gamma_{|\mathcal{V}|}^{(2)}$ where $M(\gamma_i^{(2)}) = \{v_i\}$ and $v_i \in \mathcal{V}$;
Let the direct successor of $\gamma_i^{(2)}$ be a new 3-height node $\gamma_i^{(3)}$ where $M(\gamma_i^{(3)}) = \{v_i\}$, $i = 1, 2, ..., |\mathcal{V}|$;
RepeatMerge$(T, \lambda(T))$; // See Algorithm 2
Let the direct successor of $\gamma_i^{(3)}$ be a new 4-height node $\gamma_i^{(4)}$ where $M(\gamma_i^{(4)}) = \{v_i\}$, $i = 1, 2, ..., |\mathcal{V}|$;
for all node $\alpha \in \lambda(T)^+$ **do**
  RepeatMerge(T, α);
end for
$T^3 \leftarrow T$.

Algorithm 2: Repeat Merging: RepeatMerge(T, α)

Input: A hierarchical abstracting T of graph G, a hierarchical abstracting node $\alpha \in V(T)$.
Output: A new hierarchical abstracting T' of G.

$H_{\text{min}} \leftarrow H^T(G)$;
repeat
  $T_{\text{temp}} \leftarrow T$;
  $i_{\text{min}} \leftarrow 0, j_{\text{min}} \leftarrow 0$;
  for all node $\beta_i \in \alpha^+$ $(i = 1, 2, ..., |\alpha^+|)$ in T_{temp} **do**
    for all node $\beta_j \in \alpha^+$ $(j = 1, 2, ..., |\alpha^+|$ and $j \neq i)$ **do**
      $M(\beta_i) \leftarrow M(\beta_i) \cup M(\beta_j)$; $\beta_i^+ \leftarrow \beta_i^+ \cup \beta_j^+$;
      Remove β_j;
      if $H^{T_{\text{temp}}}(G) < H_{\text{min}}$ **then**
        $i_{\text{min}} \leftarrow i, j_{\text{min}} \leftarrow j$; $H_{\text{min}} \leftarrow H^{T_{\text{temp}}}(G)$;
      end if
      $T_{\text{temp}} \leftarrow T$;
    end for
  end for
  if $i_{\text{min}} \neq 0$ **and** $j_{\text{min}} \neq 0$ **then**
    $M(\beta_{i_{\text{min}}}) \leftarrow M(\beta_{i_{\text{min}}}) \cup M(\beta_{j_{\text{min}}})$;
    $\beta_{i_{\text{min}}}^+ \leftarrow \beta_{i_{\text{min}}}^+ \cup \beta_{j_{\text{min}}}^+$;
    Remove $\beta_{j_{\text{min}}}$;
  end if
until $i_{\text{min}} = 0$ **or** $j_{\text{min}} = 0$;
$T' \leftarrow T_{\text{temp}}$.

Algorithm 3: LOM(α, β)

Input: Two nodes α and β from hierarchical abstractings T_x and T_y, respectively.
Output: The function value l.

if $|\alpha^+| > |\beta^+|$ **then**
  $l \leftarrow$ LOM(β, α);
end if
$l \leftarrow s(\alpha, \beta)$;
if $\alpha^+ \neq \emptyset$ **and** $\beta^+ \neq \emptyset$ **then**
  $q^* \leftarrow \arg\max_{q \in Q} \sum_{i=1}^{|\alpha^+|} s(\alpha_i^+, \beta_{q(i)}^+)$;
  for all node $\alpha_i^+ \in \alpha^+$ **do**
    $l \leftarrow l +$ LOM$(\alpha_i^+, \beta_{q^*(i)}^+)$;
  end for
end if

5 Experiments

In this section, we conduct extensive experiments to demonstrate the effectiveness of our proposed HAGK kernels. Through the experiments, we aim to answer the following five research questions: **RQ1:** How effective are the HAGK kernels in graph classification tasks? **RQ2:** To what extent can hierarchical abstracting with different depths and different node-matching approaches boost the performance of the inner kernel (Eq. (4))? **RQ3:** How much time do the HAGK kernels take during the hierarchical abstracting construction and the graph classification process? **RQ4:** How do substructures match at each abstraction level in different comparison cases? **RQ5:** To what extent do the matched substructure pairs at each abstraction level contribute to the total similarity?

Algorithm 4: Hierarchical Abstracting Ordering: $\mathsf{Order}(\alpha)$

Input: A hierarchical abstracting node α of T.

1: **for all** non-leaf nodes $\beta \in \alpha^+$ **do**
2: $\quad\mathsf{Order}(\beta)$;
3: **end for**
4: Sort the nodes in α^+ to ensure $\alpha_1^+ \succcurlyeq ... \succcurlyeq \alpha_{|\alpha^+|}^+$.

Algorithm 5: $\mathsf{POM}(\alpha, \beta)$

Input: Two nodes α and β from two ordered hierarchical abstractings $\vec{T}_x$ and $\vec{T}_y$, respectively.
Output: The function value p.

1: $p \leftarrow s(\alpha, \beta)$;
2: **if** $\alpha^+ \neq \emptyset$ **and** $\beta^+ \neq \emptyset$ **then**
3: $\quad$**for** $i = 1, 2, ..., \min\{|\alpha^+|, |\beta^+|\}$ **do**
4: $\quad\quad p \leftarrow p + \mathsf{POM}(\alpha_i^+, \beta_i^+)$;
5: $\quad$**end for**
6: **end if**

5.1 Experimental Setups

Evaluation Process. Firstly, we pre-construct 3-dimensional hierarchical abstractings for all graphs in each dataset and report the time consumed. Then we use the graph classification task to compare the performance between the HAGK kernels and the existing state-of-the-art kernels. For the graph classification process, we perform a 10-fold cross-validation using the C-SVM from LIBSVM library [43] to compute the classification accuracy. For each kernel and each dataset, we find the optimal parameters for the C-SVM and the kernel by an inner 10-fold cross-validation on each training split of the outer 10-fold. The search ranges of the parameters follow the setting of Siglidis et al. [44]. We repeat the whole process for 10 times and report the mean and the standard deviation of the classification accuracy. We conduct the experiments using a graph kernel python library, namely "GraKeL" [44]. The hardware we adopted is Intel(R) Xeon(R) Silver 4214R CPU @ 2.40GHz.

Datasets. We choose various datasets from different domains, including MUTAG, BZR_MD, COX2_MD, ER_MD, PTC_FM (small molecules), MSRC_9, MSRC_21C (computer vision), PROTEINS (bioinformatics), SYNTHIE (synthetic data), and IMDB-M (social networks)[1]. The statistics of the datasets are shown in Table 2. More detailed descriptions of the used datasets are listed below.

• **MUTAG** [46]. MUTAG consists of 188 chemical compounds divided into two classes according to their mutagenic effect on a bacterium.

• **MSRC_9 and MSRC_21C** [47]. The MSRC datasets are state-of-the-art datasets in semantic image processing originally introduced by Winn et al. [48]. Each image is represented by a conditional Markov random field graph. The nodes of each graph are derived by over-segmenting the images using the quick shift algorithm, resulting in one graph among the superpixels of each image. Nodes are connected if the superpixels are adjacent, and each node can further be annotated with a semantic label.

• **BZR_MD, COX2_MD, and ER_MD** [49]. These datasets are chemical compound datasets that come with 3D coordinates and were used to study the pharmacophore kernel.

• **PTC_FM** [50]. The PTC dataset contains compounds labeled according to carcinogenicity on rodents divided into male mice (MM), male rats (MR), female mice (FM), and female rats (FR). We use PTC_FM in this paper.

• **SYNTHIE** [51]. The SYNTHIE dataset is generated using Erdős-Rényi graphs [52] with structural and attribute-based class divisions through random edge modifications and attribute assignments.

• **PROTEINS** [53]. PROTEINS is a dataset of proteins that are classified as enzymes or non-enzymes. Nodes represent the amino acids and two nodes are connected by an edge if they are less than 6 Angstroms apart.

• **IMDB-M** [54]. "IMDB" is a set of relational datasets that consists of a network of 1000 actors or actresses who played roles in movies in IMDB. A node represents an actor or actress, and an edge connects two nodes when they appear in the same movie. In IMDB-M, the edges are collected from three different genres: Comedy, Romance, and Sci-Fi.

HAGK Hyper-parameter Settings. The HAGK kernels have two parameters: the node-matching approach $A \in \{\text{LOM}, \text{POM}\}$ and the tree height (the abstraction level) $k \in \{1, 2, 3, 4\}$. A decides which node-matching approach is chosen and k denotes that only the first k layers of the 3-dimensional hierarchical abstractings are used while the rest is ignored.

Baselines. For the competing graph kernels, we choose (1) the Weisfeiler-Lehman subtree kernel (WLSK) [23], (2) the Weisfeiler-Lehman optimal assignment kernel (WLOA) [14], (3) the pyramid match graph kernel (PM) [12], (4) the shortest path kernel (SP) [7], (5) the core variants of the SP kernel (CORE-SP) [40], (6) the graphlet sampling kernel (GS) [41], (7) the ordered decomposition DAGs kernel (ODD) [42], (8) the multi-scale Wasserstein shortest-path graph kernel (MWSP) [28], and (9) the hierarchical transitive-aligned graph kernel (HTAK) [27]. Among them, WLOA and PM are optimal assignment kernels while others are all R-convolutional kernels.

Codes. The codes for all baseline models and HAGK, along with all datasets, are publicly accessible on GitHub[2].

TABLE 1: Time complexity of HAGK algorithms.

Hierarchical Abstracting Construct (3d)	$O(n^3)$
HAGK-LOM	$O(n^2(\|\mathcal{L}\| + n))$
HAGK-POM	$O(n \log n \|\mathcal{L}\|)$

5.2 Performance Comparison (RQ1)

Table 3 shows the performance of HAGK and the existing graph kernels on graph classification. According to the results, HAGK is highly competitive with the current methods. More specifically, on 7 of the 10 datasets, MSRC_9, MSRC_21C, BZR_MD, COX2_MD, ER_MD, PTC_FM, and SYNTHIE, the classification accuracy of HAGK is higher than

1. All the datasets are available on http://graphkernels.cs.tu-dortmund.de [45].

2. https://github.com/SELGroup/HAGK

TABLE 2: The statistics of the used 10 datasets in the experiments.

	MUTAG	MSRC_9	MSRC_21C	BZR_MD	COX2_MD	ER_MD	PTC_FM	SYNTHIE	PROTEINS	IMDB-M
Mean # nodes	17.93	40.58	40.28	21.30	26.28	21.33	14.11	95.00	30.96	13.00
Mean # edges	19.79	97.94	96.60	225.06	336.12	234.85	14.48	172.93	72.82	65.94
# graphs	188	221	209	306	303	446	349	400	1113	1500
# classes	2	8	20	2	2	2	2	4	2	3

TABLE 3: Classification accuracy (%± standard error) of HAGK and the existing kernels. The highest and second highest results are highlighted with **boldface** and underline, respectively.

	MUTAG	MSRC_9	MSRC_21C	BZR_MD	COX2_MD	ER_MD	PTC_FM	SYNTHIE	PROTEINS	IMDB-M
WLSK [23]	82.05±0.36	89.65±0.98	80.43±1.31	59.46±1.79	57.19±1.95	65.92±1.20	62.58±2.25	50.42±1.96	74.56±1.03	48.49±0.53
WLOA [14]	84.05±1.70	90.93±1.44	82.35±1.32	63.42±1.65	60.08±1.32	65.45±0.98	62.78±2.11	50.62±1.92	**76.40±0.40**	**49.36±0.68**
PM [12]	**86.67±0.60**	90.55±0.82	84.25±1.40	65.83±1.37	61.18±1.66	69.77±1.58	57.53±2.83	51.25±1.92	72.54±1.13	46.23±0.44
SP [7]	80.88±1.49	90.64±1.38	81.49±1.75	67.32±1.69	63.74±0.68	60.41±1.91	60.72±1.63	50.00±1.68	75.63±0.75	42.35±0.98
CORE-SP [40]	82.05±0.36	90.21±1.58	82.45±1.13	65.62±1.75	60.29±2.24	68.19±2.34	59.95±2.62	52.92±1.60	75.89±1.62	48.36±0.43
GS [41]	78.03±0.35	13.00±2.16	11.72±1.65	48.77±2.45	48.28±1.62	59.42±0.01	61.19±1.32	44.62±1.94	71.13±0.59	35.38±0.39
ODD [42]	77.40±1.28	88.44±2.07	80.68±1.71	67.31±1.47	61.20±1.59	63.08±2.47	63.06±1.83	51.85±1.57	71.41±1.26	42.35±0.55
MWSP [28]	84.50±9.70	91.49±6.24	83.68±5.47	63.05±7.62	57.05±4.97	69.83±4.25	62.75±3.40	53.05±1.63	OOM*	47.93±3.86
HTAK [27]	82.82±0.85	88.65±2.33	80.56±0.89	67.36±0.99	60.25±0.16	68.88±0.57	61.20±0.25	52.98±1.76	72.25±1.10	47.54±0.53
HAGK	84.59±0.75	**92.30±0.49**	**84.27±0.64**	**68.44±1.25**	**63.84±0.82**	**70.15±0.95**	**63.22±1.08**	**53.45±1.40**	75.94±0.86	48.68±0.51

* "OOM" denotes that the case is out of memory.

all other baselines. On three other datasets, i.e. MUTAG, PROTEINS, and IMDB-M, HAGK achieves the second highest accuracy. The main reason for the outstanding effectiveness of HAGK is that it fully extracts and captures the latent hierarchical structural information of graphs by constructing and comparing their hierarchical abstractings under the guidance of the principle of structural entropy minimization. In contrast, WLSK, WLOA, SP, GS, and ODD decompose the graph into substructures at the same abstraction level, missing a large amount of structural information embedded in deep hierarchy. Although WLSK and WLOA can indeed obtain some structural information in the form of logical subtrees through multiple iterations [14], compared with HAGK, their neighbor-aggregation-based strategy is still local and lacks theoretical guidance in terms of partitioning. Their substructures are partitioned using the intuitive aggregation method, while we partition substructures using structural entropy [11]. MWSP uses the Wasserstein distance to compute the similarity between the multi-scale shortest-path node feature maps of two graphs and captures the distributions of shortest paths. However, MWSP is not positive semi-definite, which may impact the graph kernel's effectiveness. The HTAK kernel is another existing kernel that utilizes the hierarchical method for classifying graphs. There are two main differences between our HAGK and HTAK. First, HTAK relies on graph embedding methods to map graph nodes into vector spaces, which loses a lot of topology information. Second, in HTAK, the k-means method used to gather nodes at the same level to form hierarchies has no theoretical guarantee, while our HAGK uses the structural entropy minimization principle proposed by [11] and reveals the natural structures of graphs. CORE-SP uses the k-core decomposition [55] to discover topological and hierarchical properties of graphs. However, the k-core decomposition is designed only based on node degree and lacks sufficient theoretical support to utilize the natural structure of graphs. Similar to HAGK, PM also adopts hierarchical structures to partition graph nodes. However, in PM the graph nodes must first be embedded into a vector space, which may aggravate the loss of rich topological information. In summary, the above results and analysis demonstrate that HAGK has achieved state-of-the-art performance compared to current mainstream methods.

5.3 Comparison between LOM & POM and between Different Tree Heights (RQ2)

Accuracies with Different Hyper-parameters. The classification accuracies of the HAGK kernels under LOM and POM with different tree heights are shown in Fig. 9. For the node-matching approach A, POM performs better than LOM in 7 of the 10 datasets. Specifically speaking, on MSRC_21C, ER_MD, and SYNTHIE, LOM can enhance the classification effectiveness while POM cannot. On the contrary, on MUTAG, BZR_MD, COX2_MD, and PROTEINS, only POM could boost the performance. In addition, on MSRC_9, PTC_FM, and IMDB-M, both LOM and POM are effective though POM is better. The main reason for this may be that HAGK-POM is a positive semi-define kernel which could theoretically allow for better optimization of the SVM. For the tree height k, HAGK performs best on 4, 5, and 1 datasets when $k = 2$, 3, and 4, respectively. The accuracy boosts of the best tree height k compared with the inner kernel ($k = 1$) are annotated in Fig. 9, ranging from 2.11% on MSRC_21C and 8.86% on IMDB-M. These results demonstrate that the hierarchical abstractings take full advantage of the graph structural information and effectively improve the inner kernel with various optional depths.

Kernel Matrix Visualization. To better illustrate the enhancement of our hierarchical abstracting, we visualize the kernel matrices of HAGK-LOM ($k = 4$), HAGK-POM ($k = 4$), and HAGK ($k = 1$) on IMDB-B [54] in Fig. 10. IMDB-B is a dataset similar to IMDB-M but contains only two classes: Action and Romance. The darker color in each matrix grid denotes the higher normalized kernel value (similarity) between two graphs, i.e., $k_N(G_x, G_y) = \frac{k(G_x,G_y)}{\sqrt{k(G_x,G_x)}\sqrt{k(G_y,G_y)}}$. As we can see from the two left matrices, the intra-class similarities are significantly higher than the inter-class similarities both in HAGK-LOM ($k = 4$) and HAGK-POM ($k = 4$). In contrast, HAGK ($k = 1$) (the 3rd matrix) struggles to distinguish between the two classes without the help of hierarchical abstractings. Additionally, more visualization

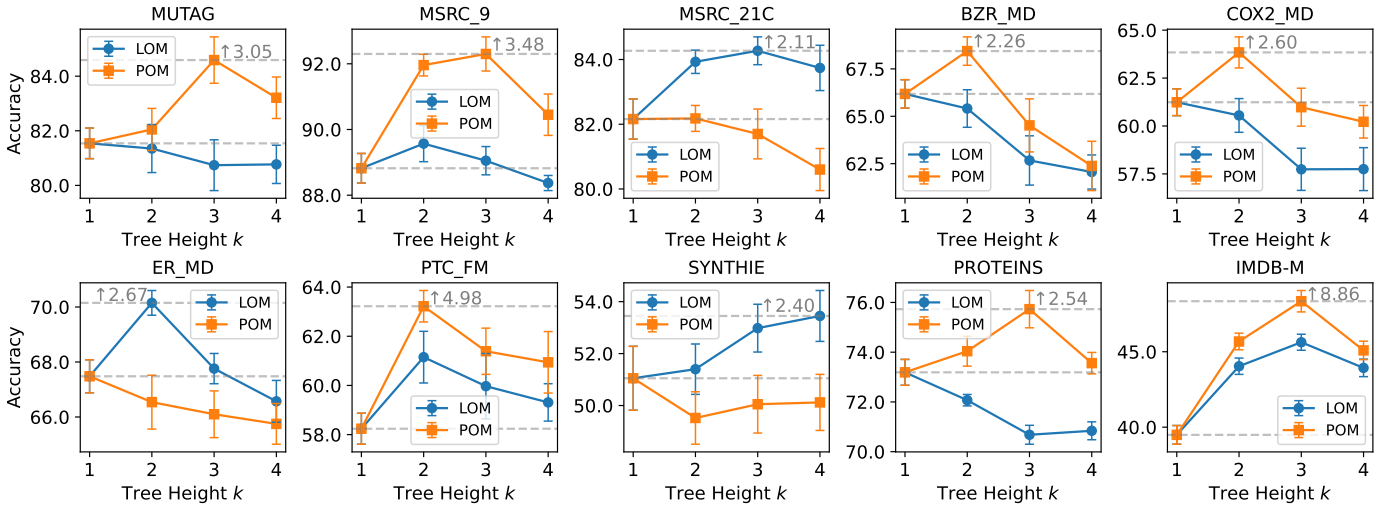


Fig. 9: The accuracy of the HAGK kernels with different node-matching approaches A and different tree height k. When $k = 1$, the HAGK kernels will degenerate to the histogram intersection kernel (Eq. (4)) with no hierarchical abstractings.

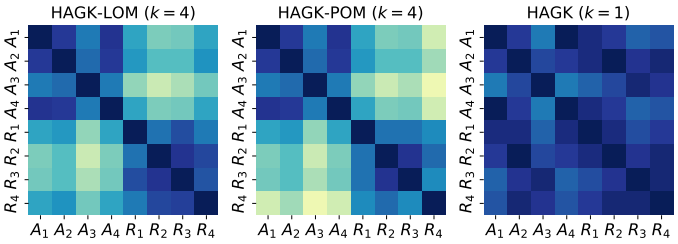


Fig. 10: The visualization of the HAGK kernel matrices of 8 example graphs from 2 different classes of the IMDB-B dataset. Graph A_1-A_4 are of class "Action" while graph R_1-R_4 are of class "Romance".

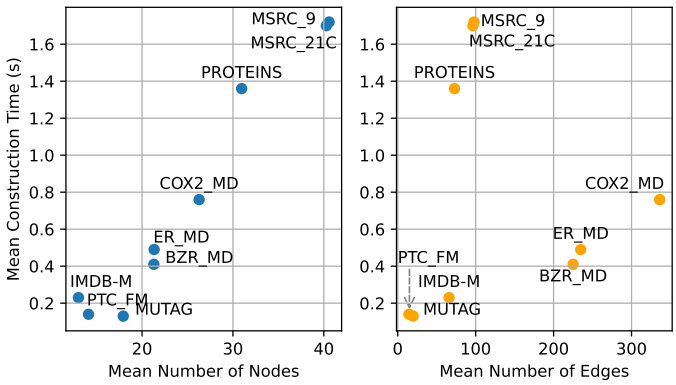


Fig. 11: Mean hierarchical abstracting construction time for each graph on each dataset except SYNTHIE. The x-axis indicates the mean number of graph nodes (left) or edges (right). For SYNTHIE, the mean time is 22.50s, with mean node number 95.00 and mean edge number 172.93.

results, including partial kernel matrices with numerical represented accuracy and whole kernel matrices, can be seen in Appendix B.

5.4 Computational Time Analysis (RQ3)

Time Consumption of Hierarchical Abstracting Construction. Fig. 11 shows the mean time consumption for converting each graph into its hierarchical abstracting on each dataset. In Fig. 11, the two subplots (a) and (b) demonstrate the relationship between mean construction time and mean number of nodes and edges. From the results, we can conclude that the mean time consumption grows nearly linearly with the mean node number yet shows no significant correlation with the number of edges. Besides, the mean time for constructing a hierarchical abstracting is always less than 2s, and the total construction time for each dataset is no more than 1513s (except the large dataset SYNTHIE, with 22.50s per graph and 9001s in total). This indicates that the time cost is practically acceptable when we pre-construct the hierarchical abstractings in advance. More details of the time consumption analysis of hierarchical abstracting construction can be seen in Appendix A.

Time Consumption of HAGK and the Optimal Assignment Competitors. We report the time consumption on the graph classification task on each dataset of all optimal assignment kernels, including our proposed HAGK, as well as the two best performing baselines, WLOA, and PM, in Fig. 12.

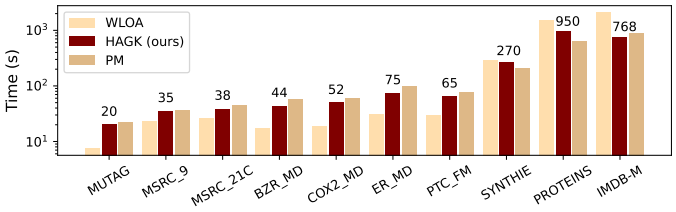
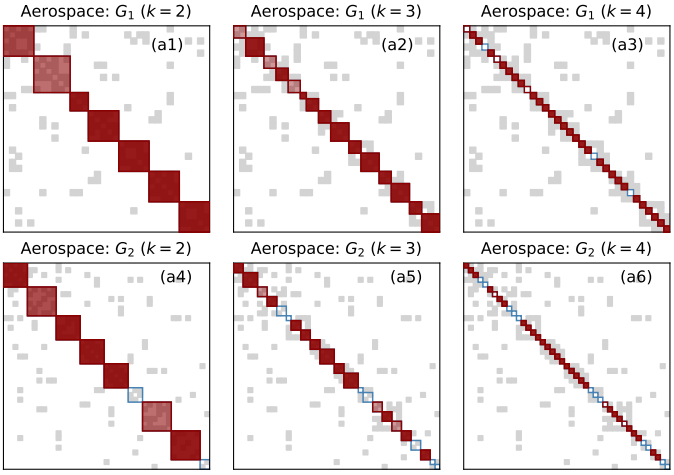
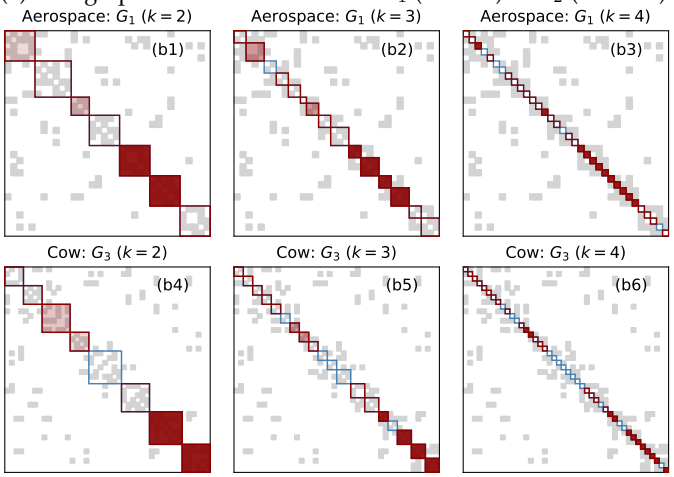


Fig. 12: Time consumption for graph classification evaluation of the optimal assignment kernels (HAGK, WLOA, and PM) compared in Table 3. The times of HAGK are annotated.

Overall, there is no significant difference in the order of magnitude for the three methods. More specifically, the time consumption of HAGK is larger than that of WLOA and less than (and close to) that of PM on all datasets except SYNTHIE, PROTEINS (WLOA>HAGK>PM) and IMDB-M (WLOA>PM>HAGK). Combining the accuracy results shown in Table 3, we can find that HAGK can beat other 2 optimal assignment kernels while maintaining small differences in time consumption. The detail time consumption analysis of all HAGK variants and other competitors are



(a) Two graphs from the same class: G_1 (1st row) & G_2 (2nd row).



(b) Two graphs from different classes: G_1 (1st row) & G_3 (2nd row).

Fig. 13: Matched substructures by LOM at each abstraction level ($k = 2, 3, 4$) of two different graph pairs from MSRC_9 (shown in (a) and (b), respectively). Graphs are represented by adjacency matrices and the grid (i, j) is filled with grey when there is an edge between node i and j of the graph. Red-edged blocks represent the matched substructures while blue-edged blocks denote the mismatched ones. Each red-edged block in an upper graph is matched with another one in the lower graph at the same level. Blocks with darker colors indicate higher similarity with their matched blocks.

shown in Appendix A.

5.5 Case Study: Matched Substructures (RQ4, 5)

To further explore and demonstrate how the mechanism of hierarchical abstracting matching can distinguish two graphs, we visualize the matched substructures under views of different abstraction levels between graphs from the same class and graphs from different classes in Fig. 13. To better display the hierarchical structure of each graph, we first arrange graph nodes by clustering them according to the hierarchical abstracting and rank the nested clusters in descending order of structural entropy before we show their adjacency matrices.

As we can see from Fig. 13(a), for graphs from the same class, most of the substructures are matched at each abstraction level. Furthermore, the substructure similarities are quite high (most of the blocks are dark red), representing the two graphs are very similar. For example, when $k = 2$, there are 7 substructures in G_1 (Fig. 13(a1)) matched with another 7 of the 9 substructures in G_2 (Fig. 13(a4)), and most of them are quite similar. When $k = 3$ or $k = 4$, there are only a few substructures mismatched. On the contrary, when comparing graphs from different classes (Fig. 13(b)), the mismatched substructures are significantly increasing (especially evident in Fig. 13(b6)). Moreover, the similarity of the matched ones is apparently lower than those of Fig. 13(a). For instance, there are only two high-similarity matched blocks between G_1 and G_3 when $k = 2$ (Fig. 13(b1) and (b4)) and four when $k = 3$ (Fig. 13(b2) and (b5)). To summarize, this case study visually demonstrates our method's ability to distinguish between two graphs from both the same class and different classes. More case studies on the dataset MSRC_9 can be found in Appendix C.

5.6 Ablation Study

Table 4 shows the ablation study of the HAGK kernels under different hierarchical abstracting construction strategies ("Louvain", "Random" and "SEM") and different kernel design schemes ("OA" and "R-conv"). Specifically, "Louvain" denotes that we recursively use Louvain [56], one of the most commonly used community discovery algorithms, to construct 3-dimensional hierarchical abstractings. "Random" denotes that we randomly partition the graph node set to construct hierarchical abstractings. "SEM" denotes that the hierarchical abstracting is constructed following the principle of structural entropy minimization [11], which is used in our proposed method. "OA" denotes the "one-to-one" optimal assignment scheme and "R-conv" denotes the "one-to-all" R-convolution scheme (summing up the similarities of all pairs of substructures). As we can see from the results, our method achieves the best accuracy in all 5 chosen datasets, indicating that the "SEM"+"OA" combination can extract and utilize the hierarchical structural information best.

TABLE 4: Test accuracy of HAGK under different setups in 5 representative datasets.

	MUTAG	MSRC_9	BZR_MD	PTC_FM	IMDB-M
Louvain-OA	82.30	89.63	64.25	60.23	47.21
Random-OA	77.56	83.02	59.66	55.08	41.36
SEM-Rconv	80.25	91.16	66.67	61.03	44.50
SEM-OA (ours)	**84.59**	**92.30**	**68.44**	**63.22**	**48.68**

5.7 Robustness Analysis

To evaluate HAGK's robustness against adversarial graphs, we randomly remove edges from or add edges to the original graph structure of the MUTAG dataset and validate the performance on the corrupted graphs. We change the ratios of modified edges from 0 to 0.5 to simulate different attack intensities. We compared our method to WLOA [14] (the strongest optimal assignment baseline) and WLSK [23] (the most widely used graph kernel). As we can see in Fig. 14, HAGK consistently achieves better or comparable results in both settings. When the edge deletion rates become larger, our method shows more significant performance gains, indicating that HAGK is more robust against serious structural attacks.

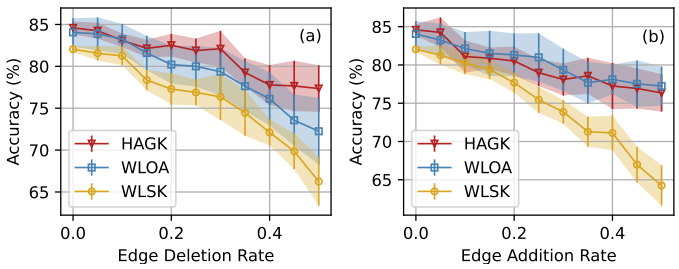


Fig. 14: Test accuracy in the scenarios where graphs are perturbed by edge deletion or addition.

5.8 Discussions

Between HAGK-LOM and [38]. [38] provided a genetic way to compare two hierarchies, which is similar to LOM. They both begin by comparing the roots and then comparing the matched children. However, the node of our defined hierarchical abstracting corresponds to a set of graph nodes, which is different from the definition of the hierarchy mentioned in [38].

Limitations. HAGK is based on node-labeled graphs, so it cannot utilize the node attribute information. This makes it perform less satisfactorily in specific datasets with rich attribute information, such as ENZYMES [53]. Therefore, for the sake of fairness in comparison, we choose HTAK [27] and MWSP [28] which also focus on non-attributed graphs as recent baselines.

6 Conclusion

In this paper, we propose a family of HAGK kernels, measuring the similarity of the hierarchical abstractings to fully extract and capture the hierarchical structural information contained in graphs. To compare hierarchical abstractings, we design two novel node-matching approaches, LOM and POM. The former is based on the optimal assignment idea and the latter is based on node arrangement and alignment matching. The proposed method outperforms the existing competitors and is verified to be capable of boosting the inner histogram intersection kernel considerably. In the future, we will extend the proposed kernels to other downstream tasks.

Acknowledgments

This work is supported by the National Key R&D Program of China through grant 2022YFB3104703, NSFC through grants 61932002, 62322202, and 62432006, Guangdong Basic and Applied Basic Research Foundation through grant 2023B1515120020, Open Research Fund from Guangdong Laboratory of Artificial Intelligence and Digital Economy (SZ) under Grant No. GML-KF-24-08, and CCF-DiDi GAIA Collaborative Research Funds for Young Scholars.

References

[1] N. M. Kriege, F. D. Johansson, and C. Morris, "A survey on graph kernels," *Applied Network Science*, vol. 5, no. 1, pp. 1–42, 2020.
[2] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers," in *Proceedings of the COLT*, 1992, pp. 144–152.
[3] B. Schölkopf, A. Smola, and K.-R. Müller, "Nonlinear component analysis as a kernel eigenvalue problem," *Neural Computation*, vol. 10, no. 5, pp. 1299–1319, 1998.
[4] F. Rousseau, E. Kiagias, and M. Vazirgiannis, "Text categorization as a graph classification problem," in *Proceedings of the ACL*, 2015, pp. 1702–1712.
[5] D. Haussler *et al.*, "Convolution kernels on discrete structures," Tech. Rep., 1999.
[6] H. Kashima, K. Tsuda, and A. Inokuchi, "Marginalized kernels between labeled graphs," in *Proceedings of the ICML*, 2003, pp. 321–328.
[7] K. M. Borgwardt and H.-P. Kriegel, "Shortest-path kernels on graphs," in *Proceedings of the ICDM*, 2005, pp. 74–81.
[8] T. Horváth, T. Gärtner, and S. Wrobel, "Cyclic pattern kernels for predictive graph mining," in *Proceedings of the SIGKDD*, 2004, pp. 158–167.
[9] J. Ramon and T. Gärtner, "Expressivity versus efficiency of graph kernels," in *Proceedings of the First International Workshop on Mining Graphs, Trees and Sequences*, 2003, pp. 65–74.
[10] O. Boiman and M. Irani, "Similarity by composition," *Proceedings of the NIPS*, vol. 19, 2006.
[11] A. Li and Y. Pan, "Structural information and dynamical complexity of networks," *IEEE Transactions on Information Theory*, vol. 62, no. 6, pp. 3290–3339, 2016.
[12] G. Nikolentzos, P. Meladianos, and M. Vazirgiannis, "Matching node embeddings for graph similarity," in *Proceedings of the AAAI*, 2017, pp. 2429–2435.
[13] H. Fröhlich, J. K. Wegner, F. Sieker, and A. Zell, "Optimal assignment kernels for attributed molecular graphs," in *Proceedings of the ICML*, 2005, pp. 225–232.
[14] N. M. Kriege, P.-L. Giscard, and R. Wilson, "On valid optimal assignment kernels and applications to graph classification," *in Proceedings of the NeurIPS*, pp. 1615–1623, 2016.
[15] J.-P. Vert, "The optimal assignment kernel is not positive definite," *arXiv preprint arXiv:0801.4061*, 2008.
[16] R. Jonker and A. Volgenant, "A shortest augmenting path algorithm for dense and sparse linear assignment problems," *Computing*, vol. 38, no. 4, pp. 325–340, 1987.
[17] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proceedings of the ICLR*, 2017.
[18] G. Corso, H. Stark, S. Jegelka, T. Jaakkola, and R. Barzilay, "Graph neural networks," *Nature Reviews Methods Primers*, vol. 4, no. 1, p. 17, 2024.
[19] B. Khemani, S. Patil, K. Kotecha, and S. Tanwar, "A review of graph neural networks: concepts, architectures, techniques, challenges, datasets, applications, and future directions," *Journal of Big Data*, vol. 11, no. 1, p. 18, 2024.
[20] M. Li, Z. Ma, Y. G. Wang, and X. Zhuang, "Fast haar transforms for graph neural networks," *Neural Networks*, vol. 128, pp. 188–198, 2020.
[21] S. S. Du, K. Hou, R. R. Salakhutdinov, B. Poczos, R. Wang, and K. Xu, "Graph neural tangent kernel: Fusing graph neural networks with graph kernels," *in Proceedings of the NeurIPS*, vol. 32, 2019.
[22] Y. Tang and J. Yan, "Graphqntk: quantum neural tangent kernel for graph data," *in Proceedings of the NeurIPS*, vol. 35, pp. 6104–6118, 2022.
[23] 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.
[24] N. Kriege and P. Mutzel, "Subgraph matching kernels for attributed graphs," in *Proceedings of the ICML*, 2012, p. 291–298.
[25] M. Togninalli, E. Ghisu, F. Llinares-López, B. Rieck, and K. Borgwardt, "Wasserstein weisfeiler-lehman graph kernels," *Proceedings of the NeurIPS*, pp. 6436–6446, 2019.
[26] B.-C. Xu, K. M. Ting, and Y. Jiang, "Isolation graph kernel," in *Proceedings of the AAAI*, vol. 35, no. 12, 2021, pp. 10 487–10 495.
[27] L. Bai, L. Cui, and H. Edwin, "A hierarchical transitive-aligned graph kernel for un-attributed graphs," in *Proceedings of the ICML*, 2022, pp. 1327–1336.
[28] W. Ye, H. Tian, and Q. Chen, "Multi-scale wasserstein shortest-path graph kernels for graph classification," *IEEE Transactions on Artificial Intelligence*, 2023.
[29] S. Jiang, Y. Man, Z. Song, Z. Yu, and D. Zhuo, "Fast graph neural tangent kernel via kronecker sketching," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 6, 2022, pp. 7033–7041.
[30] A. Li, X. Yin, B. Xu, D. Wang, J. Han, Y. Wei, Y. Deng, Y. Xiong, and Z. Zhang, "Decoding topologically associating domains with ultra-low resolution hi-c data by graph structural entropy," *Nature Communications*, vol. 9, no. 1, pp. 1–12, 2018.

[31] Y. W. Zhang, M. B. Wang, and S. C. Li, "Supertad: robust detection of hierarchical topologically associated domains with optimized structural information," *Genome Biology*, vol. 22, no. 1, pp. 1–20, 2021.

[32] A. Li, X. Zhang, and Y. Pan, "Resistance maximization principle for defending networks against virus attack," *Physica A: Statistical Mechanics and its Applications*, vol. 466, pp. 211–223, 2017.

[33] Y. Liu, J. Liu, Z. Zhang, L. Zhu, and A. Li, "Rem: From structural entropy to community structure deception," *Proceedings of the NeurIPS*, pp. 12 918–12 928, 2019.

[34] J. Wu, X. Chen, K. Xu, and S. Li, "Structural entropy guided graph hierarchical pooling," in *Proceedings of the ICML*, 2022, pp. 24 017–24 030.

[35] C. Zhang, H. Zhu, X. Peng, J. Wu, and K. Xu, "Hierarchical information matters: Text classification via tree based graph neural network," in *Proceedings of the COLING*, 2022, pp. 950–959.

[36] D. Zou, H. Peng, X. Huang, R. Yang, J. Li, J. Wu, C. Liu, and P. S. Yu, "Se-gsl: A general and effective graph structure learning framework through structural entropy optimization," in *Proceedings of the ACM Web Conference 2023*, 2023, pp. 499–510.

[37] Z. Yang, G. Zhang, J. Wu, J. Yang, Q. Z. Sheng, H. Peng, A. Li, S. Xue, and J. Su, "Minimum entropy principle guided graph neural networks," in *Proceedings of the ACM WSDM*, 2023, pp. 114–122.

[38] P. Ganesan, H. Garcia-Molina, and J. Widom, "Exploiting hierarchical domain structure to compute similarity," *ACM TOIS*, vol. 21, no. 1, pp. 64–93, 2003.

[39] J. Wu, S. Li, J. Li, Y. Pan, and K. Xu, "A simple yet effective method for graph classification," in *Proceedings of the IJCAI*, 2022, pp. 3580–3586.

[40] G. Nikolentzos, P. Meladianos, S. Limnios, and M. Vazirgiannis, "A degeneracy framework for graph similarity." in *IJCAI*, 2018, pp. 2595–2601.

[41] N. Shervashidze, S. Vishwanathan, T. Petri, K. Mehlhorn, and K. Borgwardt, "Efficient graphlet kernels for large graph comparison," in *Artificial Intelligence and Statistics*, 2009, pp. 488–495.

[42] G. Da San Martino, N. Navarin, and A. Sperduti, "A tree-based kernel for graphs," in *Proceedings of the SDM*, 2012, pp. 975–986.

[43] C.-C. Chang and C.-J. Lin, "Libsvm: a library for support vector machines," *ACM TIST*, vol. 2, no. 3, pp. 1–27, 2011.

[44] G. Siglidis, G. Nikolentzos, S. Limnios, C. Giatsidis, K. Skianis, and M. Vazirgiannis, "Grakel: A graph kernel library in python." *Journal of Machine Learning Research*, vol. 21, no. 54, pp. 1–5, 2020.

[45] C. Morris, N. M. Kriege, F. Bause, K. Kersting, P. Mutzel, and M. Neumann, "Tudataset: A collection of benchmark datasets for learning with graphs," *arXiv preprint arXiv:2007.08663*, 2020.

[46] 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.

[47] M. Neumann, R. Garnett, C. Bauckhage, and K. Kersting, "Propagation kernels: efficient graph kernels from propagated information," *Machine Learning*, vol. 102, pp. 209–245, 2016.

[48] J. Winn, A. Criminisi, and T. Minka, "Object categorization by learned universal visual dictionary," in *Proceedings of the ICCV*, vol. 2. IEEE, 2005, pp. 1800–1807.

[49] J. J. Sutherland, L. A. O'brien, and D. F. Weaver, "Spline-fitting with a genetic algorithm: A method for developing classification structure- activity relationships," *Journal of chemical information and computer sciences*, vol. 43, no. 6, pp. 1906–1915, 2003.

[50] C. Helma, R. D. King, S. Kramer, and A. Srinivasan, "The predictive toxicology challenge 2000–2001," *Bioinformatics*, vol. 17, no. 1, pp. 107–108, 2001.

[51] C. Morris, N. M. Kriege, K. Kersting, and P. Mutzel, "Faster kernels for graphs with continuous attributes via hashing," in *2016 IEEE 16th International Conference on Data Mining (ICDM)*. IEEE, 2016, pp. 1095–1100.

[52] P. ERDdS and A. R&wi, "On random graphs i," *Publ. math. debrecen*, vol. 6, no. 290-297, p. 18, 1959.

[53] K. M. Borgwardt, C. S. Ong, S. Schönauer, S. Vishwanathan, A. J. Smola, and H.-P. Kriegel, "Protein function prediction via graph kernels," *Bioinformatics*, vol. 21, no. suppl_1, pp. i47–i56, 2005.

[54] P. Yanardag and S. Vishwanathan, "Deep graph kernels," in *Proceedings of the 21th ACM SIGKDD*, 2015, pp. 1365–1374.

[55] S. B. Seidman, "Network structure and minimum degree," *Social networks*, vol. 5, no. 3, pp. 269–287, 1983.

[56] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of statistical mechanics: theory and experiment*, vol. 2008, no. 10, p. P10008, 2008.

Runze Yang is currently a Master's degree candidate in the School of Computer Science and Engineering at Beihang University. His research interests include machine learning and data mining.

Hao Peng is currently a Professor at the School of Cyber Science and Technology in Beihang University. His current research interests include machine learning, deep learning, and reinforcement learning. He is the Associate Editor of the International Journal of Machine Learning and Cybernetics (IJMLC) and Neural Networks.

Angsheng Li is a professor at the School of Computer Science, Beihang University. His research areas include Computability Theory, Computational Theory, and Information Science. His current interests focus on mathematical principles of the information world and structural information principles of machine learning and intelligent machinery.

Peng Li is currently a researcher at the Academy of Military Sciences. His research interests include machine learning and information systems.

Chunyang Liu is currently a researcher at Didi Chuxing Technology Co., Ltd. His research interests include machine learning and structural data mining.

Philip S. Yu is a Distinguished Professor and the Wexler Chair in Information Technology at the Department of Computer Science, University of Illinois Chicago. He is a Fellow of the ACM and IEEE. Dr. Yu has published more than 1,200 referred conference and journal papers cited more than 190,000 times with an H-index of 196. He has applied for more than 300 patents. Dr. Yu was the Editor-in-Chief of ACM TKDD (2011-2017) and IEEE TKDE (2001-2004).