Tree++: Truncated Tree Based Graph Kernels

Wei Ye, Zhen Wang, Rachel Redberg, and Ambuj Singh

Abstract—Graph-structured data arise ubiquitously in many application domains. A fundamental problem is to quantify their similarities. Graph kernels are often used for this purpose, which decompose graphs into substructures and compare these substructures. However, most of the existing graph kernels do not have the property of scale-adaptivity, i.e., they cannot compare graphs at multiple levels of granularities. Many real-world graphs such as molecules exhibit structure at varying levels of granularities. To tackle this problem, we propose a new graph kernel called Tree++ in this paper. At the heart of Tree++ is a graph kernel called the path-pattern graph kernel. The path-pattern graph kernel first builds a truncated BFS tree rooted at each vertex and then uses paths from the root to every vertex in the truncated BFS tree as features to represent graphs. The path-pattern graph kernel can only capture graph similarity at fine granularities. In order to capture graph similarity at coarse granularities, we incorporate a new concept called super path into it. The super path contains truncated BFS trees rooted at the vertices in a path. Our evaluation on a variety of real-world graphs demonstrates that Tree++ achieves the best classification accuracy compared with previous graph kernels.

Index Terms—Graph kernel, graph classification, truncated tree, path, path pattern, super path, BFS

1 Introduction

Structured data are ubiquitous in many application domains. Examples include proteins or molecules in bioinformatics, communities in social networks, text documents in natural language processing, and images annotated with semantics in computer vision. Graphs are naturally used to represent these structured data. One fundamental problem with graph-structured data is to quantify their similarities which can be used for downstream tasks such as classification. For example, chemical compounds can be represented as graphs, where vertices represent atoms, edges represent chemical bonds, and vertex labels represent the types of atoms. We compute their similarities for classifying them into different classes. In the pharmaceutical industry, the molecule-based drug discovery needs to find similar molecules with increased efficacy and safety against a specific disease.

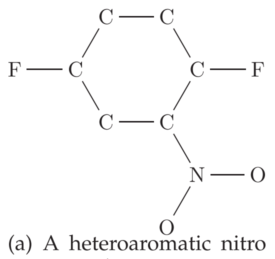
Fig. 1 shows two chemical compounds from the MUTAG [1], [2] dataset which has 188 chemical compounds and can be divided into two classes. We can observe from Figs. 1a and 1b that the numbers of the atoms C (Carbon), N (Nitrogen), F (Fluorine), O (Oxygen), and Cl (Chlorine), and their varying combinations make the functions of these two chemical compounds different. We can also observe that chemical compounds (graphs) can be of arbitrary size and shape, which makes most of the machine learning methods not applicable to graphs because most of them can only handle objects of a fixed size. Tsitsulin et al. in their paper NetLSD [3] argued that an ideal method for graph comparison should fulfill three desiderata. The first one is permutation-invariance which means the method should be invariant to the ordering of vertices; The second one is scale-adaptivity which means the method should have different levels of granularities for comparing graphs. The last one is size-invariance which means the method can compare graphs of different sizes.

Graph kernels have been developed and widely used to measure the similarities between graph-structured data. Graph kernels are instances of the family of R-convolution kernels [4]. The key idea is to recursively decompose graphs into their substructures such as graphlets [5], trees [6], [7], walks [8], paths [9], and then compare these substructures from two graphs. A typical definition for graph kernels is $\mathcal{K}(\mathsf{G}_1, \mathsf{G}_2) = \sum_{S\in\mathcal{S}} \psi(\mathsf{G}_1, S)\psi(\mathsf{G}_2, S)$, where $\mathcal{S}$ contains all unique substructures from two graphs, and $\psi(\mathsf{G}_i, S)$ represents the number of occurrences of the unique substructure S in the graph $\mathsf{G}_i, (i = 1, 2)$.

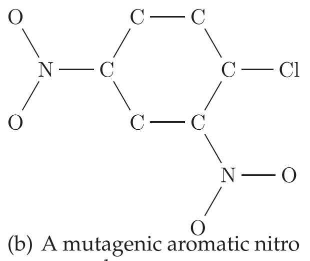
In the real world, many graphs such as molecules have structures at multiple levels of granularities. Graph kernels should not only capture the overall shape of graphs (whether they are more like a chain, a ring, a chain that branches, etc.), but also small structures of graphs such as chemical bonds and functional groups. For example, a graph kernel should capture that the chemical bond C—F in the heteroaromatic nitro compound (Fig. 1a) is different from the chemical bond C—Cl in the mutagenic aromatic nitro compound (Fig. 1b). In addition, a graph kernel should capture that the functional groups (as shown in Fig. 2) in the two chemical compounds (as shown in Fig. 1) are different. Most of the existing graph kernels only have two properties, i.e., permutation-invariance and size-invariance. They cannot capture graph similarity at multiple levels of granularities. For instance, the very popular Weisfeiler-Lehman subtree kernel (WL) [6], [7] builds a subtree of height h at each vertex and then counts the

• W. Ye, R. Redberg, and A. Singh are with the Department of Computer Science, University of California, Santa Barbara, CA 93106 USA. E-mail: {weiye, rredberg, ambuj}@cs.ucsb.edu.
• Z. Wang is with the Department of Electrical Engineering, Columbia University, New York, NY 10027 USA. E-mail: zw2501@columbia.edu.

Manuscript received 20 May 2019; revised 15 Aug. 2019; accepted 2 Oct. 2019. Date of publication 8 Oct. 2019; date of current version 5 Mar. 2021. (Corresponding author: Wei Ye.)
Recommended for acceptance by X. Li.
Digital Object Identifier no. 10.1109/TKDE.2019.2946149

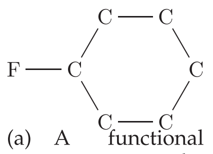


(a) A heteroaromatic nitro compound.

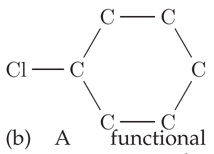


(b) A mutagenic aromatic nitro compound.

Fig. 1. Two chemical compounds from the MUTAG [1], [2] dataset. Explicit hydrogen atoms have been removed from the original dataset. Edges represent four chemical bond types, i.e., single, double, triple, or aromatic. (We do not show the edge type here for brevity.) The labels of vertices represent the types of atoms.



(a) A functional group in the heteroaromatic nitro compound.



(b) A functional group in the mutagenic aromatic nitro compound.

Fig. 2. Functional groups in the two chemical compounds from the MUTAG dataset.

occurrences of each kind of subtree in the graph. WL can only capture the graph similarity at coarse granularities, because subtrees can only consider neighborhood structures of vertices. The shortest-path graph kernel [9] counts the number of pairs of shortest paths which have the same source and sink labels and the same length in two graphs. It can only capture the graph similarity at fine granularities, because shortest-paths do not consider neighborhood structures. The Multiscale Laplacian Graph Kernel (MLG) [10] is the first graph kernel that can handle substructures at multiple levels of granularities, by building a hierarchy of nested subgraphs. However, MLG needs to invert the graph Laplacian matrix and thus its running time is very high as can be seen from Table 3 in Section 5.

In this paper, we propose a new graph kernel TREE++ that can compare graphs at multiple levels of granularities. To this end, we first develop a base kernel called the path-pattern graph kernel that decomposes a graph into paths. For each vertex in a graph, we build a truncated BFS tree of depth d rooted at it, and lists all the paths from the root to every vertex in the truncated BFS tree. Then, we compare two graphs by counting the number of occurrences of each unique path in them. We prove that the path-pattern graph kernel is positive semi-definite. The path-pattern graph kernel can only compare graphs at fine granularities. To compare graphs at coarse granularities, we extend the definition of a path in a graph and define a new concept *super path*. Each element in a path is a distinct vertex while each element in a super path is a truncated BFS tree of depth k rooted at the distinct vertices in a path. k could be zero, and in this case, a super path degenerates to a path. Incorporated with the concept of super path, the path-pattern graph kernel can capture graph similarity at different levels of granularities, from the atomic substructure *path* to the community substructure *structural identity*.

Our contributions in this paper are summarized as follows:

- We propose the path-pattern graph kernel that can capture graph similarity at fine granularities.
- We propose a new concept of super path whose elements can be trees. After incorporating the concept of super path into the path-pattern graph kernel, it can capture graph similarity at coarse granularities.
- We call our final graph kernel TREE++ as it employs truncated BFS trees for comparing graphs both at fine granularities and coarse granularities. TREE++ runs very fast and scales up easily to graphs with thousands of vertices.
- TREE++ achieves the best classification accuracy on most of the real-world datasets.

The paper is organized as follows: Section 2 discusses related work. Section 3 covers the core ideas and theory behind our approach, including the path-pattern graph kernel, the concept of super path, and the TREE++ graph kernel. Using real-world datasets, Sections 4 and 5 compare TREE++ with related techniques. Section 6 makes some discussions and Section 7 concludes the paper.

2 RELATED WORK

The first family of graph kernels is based on walks and paths, which first decompose a graph into random walks [8], [11], [12], [13], [14] or paths [9], and then compute the number of matching pairs of them. Gärtner et al. [11] investigate two approaches to compute graph kernels: one uses the length of all walks between each pair of vertices to define the graph similarity; the other defines one feature for every possible label sequence and then counts the number of walks in the direct product graph of two graphs matching the label sequence, of which the time complexity is $\mathcal{O}(n^6)$. If using some advanced approximation methods [8], the time complexity could be decreased to $\mathcal{O}(n^3)$. Kashima et al. [12] use random walks to generate label paths. The graph kernel is defined as the inner product of the count vector averaged over all possible label paths. Propagation kernels [15] leverage early-stage distributions of random walks to capture structural information hidden in vertex neighborhood. RetGK [14] introduces a structural role descriptor for vertices, i.e., the return probabilities features (RPF) generated by random walks. The RPF is then embedded into the Hilbert space where the corresponding graph kernels are derived. Borgwardt et al. [9] propose graph kernels based on shortest paths in graphs. It counts the number of pairs of shortest paths which have the same source and sink labels and the same length in two graphs. If the original graph is fully connected, the pairwise comparison of all edges in both graphs will cost $\mathcal{O}(n^4)$.

The second family of graph kernels is based on subgraphs, which include these kernels [5], [10], [16], [17] that decompose a graph into small subgraph patterns of size k nodes, where $k \in \{3, 4, 5\}$. And graphs are represented as the number of all types of subgraph patterns. The subgraph patterns are called graphlets [18]. Exhaustive enumeration of all graphlets are prohibitively expensive ($\mathcal{O}(n^k)$). Thus, Shervashidze et al. [5] propose two theoretically grounded acceleration schemes. The first one uses the method of random sampling, which is motivated by the idea that the more sufficient number of random samples is drawn, the closer the empirical distribution to the actual distribution of graphlets in a graph. The second one exploits the algorithms for efficiently counting graphlets in graphs of low degree. Costa et al. [16] propose a novel graph kernel called the Neighborhood Subgraph Pairwise Distance Kernel (NSPDK) to decompose a graph into all pairs of neighborhood subgraphs of small radium at increasing distances. The authors first compute a fast graph invariant string encoding for the pairs of neighborhood subgraphs via a label function that assigns labels from a finite alphabet Σ to the vertices in the pairs of neighborhood subgraphs. Then a hash function is used to transform the strings to natural numbers. MLG [10] is developed for capturing graph structures at a range of different scales, by building a hierarchy of nested subgraphs.

The third family of graph kernels is based on subtree patterns, which decompose graphs into subtree patterns and then count the number of common subtree patterns in two graphs. Ramon et al. [19] construct a graph kernel considering the subtree patterns which are rooted subgraphs at vertices. Every subtree pattern has a tree-structured signature. For each possible subtree pattern signature, the paper associates a feature of which the value is the number of times that a subtree of that signature occurs in graphs. For all pairs of vertices from two graphs, the subtree-pattern kernel counts all pairs of matching subtrees of the same signature of height less than or equal to d. Mahé et al. [20] revisit and extend the subtree-pattern kernel proposed in [19] by introducing a parameter to control the complexity of the subtrees, varying from common walks to large common subtrees. Weisfeiler-Lehman subtree kernel (WL) [6], [7] is based on the Weisfeiler-Lehman test of isomorphism [21] for graphs. The Weisfeiler-Lehman test of isomorphism belongs to the family of color refinement algorithms that iteratively update vertex colors (labels) until reaching the fixed number of iterations, or the vertex label sets of two graphs differ. In each iteration, the Weisfeiler-Lehman test of isomorphism algorithm augments vertex labels by concatenating their neighbors' labels and then compressing the augmented labels into new labels. The compressed labels correspond to subtree patterns. WL counts common original and compressed labels in two graphs.

Recently, some research works [22], [23] focus on augmenting the existing graph kernels. DGK [22] deals with the problem of diagonal dominance in graph kernels. The diagonal dominance means that a graph is more similar to itself than to any other graphs in the dataset because of the sparsity of common substructures across different graphs. DGK leverages techniques from natural language processing to learn latent representations for substructures. Then the similarity matrix between substructures is computed and integrated into graph kernels. If the number of substructures is high, it will cost a lot of time and memory to compute the similarity matrix. OA [23] develops some base kernels that generate hierarchies from which the optimal assignment kernels are computed. The optimal assignment kernels can provide a more valid notion of graph similarity. The authors finally integrate the optimal assignment kernels into the Weisfeiler-Lehman subtree kernel. In addition to the above-described literature, there are also some literature [3], [24], [25], [26], [27], [28] for graph classification that are related to our work.

The graph kernels elaborated above are only for graphs with discrete vertex labels (attributes) or no vertex labels. Recently, researchers begin to focus on the developments of graph kernels on graphs with continuous attributes. GraphHopper [29] is an extention of the shortest-path kernel. Instead of comparing paths based on the products of kernels on their lengths and endpoints, GraphHopper compares paths through kernels on the encountered vertices while hopping along shortest paths. The discriptor matching (DM) kernel [30] maps every graph into a set of vectors (descriptors) which integrate both the attribute and neighborhood information of vertices, and then uses a set-of-vector matching kernel [31] to measure graph similarity. HGK [32] is a general framework to extend graph kernels from discrete attributes to continuous attributes. The main idea is to iteratively map continuous attributes to discrete labels by randomized hash functions. Then HGK compares these discrete labeled graphs by an arbitrary graph kernel such as the Weisfeiler-Lehman subtree kernel or the shortest-path kernel. GIK [33] proposes graph invariant kernels that exploit a vertex invariant kernel (spectral coloring kernel) to combine both the similarities of vertex labels and vertex structural roles.

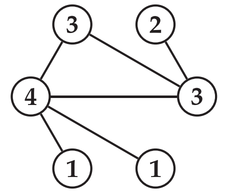
3 The Model

In this section, we introduce a new graph kernel called Tree++, which is based on the base kernel called the path-pattern graph kernel. The path-pattern graph kernel employs the truncated BFS (Breadth-First Search) trees rooted at each vertex of graphs. It uses the paths from the root to any other vertex in the truncated BFS trees of depth d as features to represent graphs. The path-pattern graph kernel can only capture graph similarity at fine granularities. To capture graph similarity at coarse granularities, i.e., structural identities of vertices, we first propose a new concept called super path whose elements can be trees. Then, we incorporate the concept of super path into the path-pattern graph kernel.

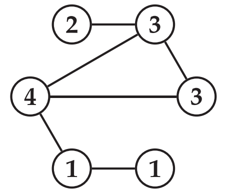
3.1 Notations

We first give notations used in this paper to make it self-contained. In this work, we use lower-case Roman letters (e.g. a, b) to denote scalars. We denote vectors (row) by boldface lower case letters (e.g. $\mathbf{x}$) and denote its ith element by $\mathbf{x}(i)$. Matrices are denoted by boldface upper case letters (e.g. $\mathbf{X}$). We denote entries in a matrix as $\mathbf{X}(i,j)$. We use $\mathbf{x} = [x_1, \ldots, x_n]$ to denote creating a vector by stacking scalar x_i along the columns. Similarly, we use $\mathbf{X} = [\mathbf{x}_1; \ldots; \mathbf{x}_n]$ to denote creating a matrix by stacking the vector $\mathbf{x}_i$ along the rows. Consider an undirected labeled graph $\mathsf{G} = (\mathcal{V}, \mathcal{E}, l)$, where $\mathcal{V}$ is a set of graph vertices with number $|\mathcal{V}|$ of vertices, $\mathcal{E}$ is a set of graph edges with number $|\mathcal{E}|$ of edges, and $l : \mathcal{V} \rightarrow \Sigma$ is a function that assigns labels from a set of positive integers Σ to vertices. Without loss of generality, $|\Sigma| \leq |\mathcal{V}|$.

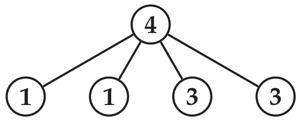
An edge e is denoted by two vertices uv that are connected to it. In graph theory [34], a walk is defined as a sequence of vertices, e.g., $(v_1, v_2, \ldots)$, where consecutive vertices are connected by an edge. A trail is a walk that consists



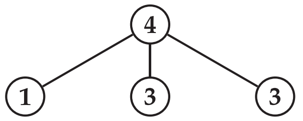
(a) An undirected labeled graph G_1.



(b) An undirected labeled graph G_2.



(c) A truncated BFS tree of depth one rooted at the vertex with label 4 in the graph G_1



(d) A truncated BFS tree of depth one rooted at the vertex with label 4 in the graph G_2

Fig. 3. Illustration of the path patterns in graphs. $\Sigma = \{1, 2, 3, 4\}$.

of all distinct edges. A path is a trail that consists of all distinct vertices and edges. A spanning tree ST of a graph G is a subgraph that includes all of the vertices of G, with the minimum possible number of edges. We extend this definition to the truncated spanning tree. A truncated spanning tree T is a subtree of the spanning tree ST, with the same root and of the depth d. The depth of a subtree is the maximum length of paths between the root and any other vertex in the subtree. Two undirected labeled graphs $\mathsf{G}_1 = (\mathcal{V}_1, \mathcal{E}_1, l_1)$ and $\mathsf{G}_2 = (\mathcal{V}_2, \mathcal{E}_2, l_2)$ are isomorphic (denoted by $\mathsf{G}_1 \simeq \mathsf{G}_2$) if there is a bijection $\varphi : \mathcal{V}_1 \rightarrow \mathcal{V}_2$, (1) such that for any two vertices $u, v \in \mathcal{V}_1$, there is an edge uv if and only if there is an edge $\varphi(u)\varphi(v)$ in G_2; (2) and such that $l_2(\varphi(v)) = l_1(v)$.

Let $\mathcal{X}$ be a non-empty set and let $\mathcal{K} : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ be a function on the set $\mathcal{X}$. Then $\mathcal{K}$ is a kernel on $\mathcal{X}$ if there is a real Hilbert space $\mathcal{H}$ and a mapping $\phi : \mathcal{X} \rightarrow \mathcal{H}$ such that $\mathcal{K}(x, y) = \langle \phi(x), \phi(y) \rangle$ for all x, y in $\mathcal{X}$, where $\langle \cdot, \cdot \rangle$ denotes the inner product of $\mathcal{H}$, ϕ is called a feature map and $\mathcal{H}$ is called a feature space. $\mathcal{K}$ is symmetric and positive-semidefinite. In the case of graphs, let $\phi(\mathsf{G})$ denote a mapping from graph to vector which contains the counts of atomic substructures in graph G. Then, the kernel on two graphs G_1 and G_2 is defined as $\mathcal{K}(\mathsf{G}_1, \mathsf{G}_2) = \langle \phi(\mathsf{G}_1), \phi(\mathsf{G}_2) \rangle$.

3.2 The Path-Pattern Graph Kernel

We first define the path pattern as follows:

Definition 1 (Path Pattern). *Given an undirected labeled graph $\mathsf{G} = (\mathcal{V}, \mathcal{E}, l)$, we build a truncated BFS tree $\mathsf{T} = (\mathcal{V}', \mathcal{E}', l)$ ($\mathcal{V}' \subseteq \mathcal{V}$ and $\mathcal{E}' \subseteq \mathcal{E}$) of depth d rooted at each vertex $v \in \mathcal{V}$. The vertex v is called root. For each vertex $v' \in \mathcal{V}'$, there is a path $P = (v, v_1, v_2, \ldots, v')$ from the root v to v' consisting of distinct vertices and edges. The concatenated labels $l(P) = (l(v), l(v_1), l(v_2), \ldots, l(v'))$ is called a path pattern of G.*

Note from this definition that a path pattern could only contain the root vertex. Figs. 3a and 3b show two example undirected labeled graph G_1 and G_2. Figs. 3c and 3d show two truncated BFS trees of depth $d = 1$ on the vertices with label 4 in G_1 and G_2, respectively. To build unique BFS trees, the child vertices of each parent vertex in the BFS tree are sorted in ascending order according to their label values. If two vertices have the same label values, we sort them again in ascending order by their eigenvector centrality [35] values. We use eigenvector centrality to measure the importance of a vertex. A vertex has high eigenvector centrality value if it is linked to by other vertices that also have high eigenvector centrality values, without implying that this vertex is highly linked. All of the path patterns of the root of the BFS tree in Fig. 3c are as follows: $(4), (4, 1), (4, 1), (4, 3), (4, 3)$. All of the path patterns of the root of the BFS tree in Fig. 3d are as follows: $(4), (4, 1), (4, 3), (4, 3)$. On each vertex in the graph G_1 and G_2, we first build a truncated BFS tree of depth d, and then generate all of its corresponding path patterns. The multiset[1] $\mathcal{M}$ of the graph G is a set that contains all the path patterns extracted from BFS trees of depth d rooted at each vertex of the graph.

Let $\mathcal{M}_1$ and $\mathcal{M}_2$ be two multisets corresponding to the two graphs G_1 and G_2. Let the union of $\mathcal{M}_1$ and $\mathcal{M}_2$ be $\mathcal{U} = \mathcal{M}_1 \cup \mathcal{M}_2 = \{l(P_1), l(P_2), \ldots, l(P_{|\mathcal{U}|})\}$. Define a map $\psi : \{\mathsf{G}_1, \mathsf{G}_2\} \times \Sigma \rightarrow \mathbb{N}$ such that $\psi(\mathsf{G}, l(P_i))$ is the number of occurrences of the path pattern $l(P_i)$ in the graph G. The definition of the path-pattern graph kernel is given as follows:

$$\mathcal{K}_{pp}(\mathsf{G}_1, \mathsf{G}_2) = \sum_{l(P_i) \in \mathcal{U}} \psi(\mathsf{G}_1, l(P_i)) \psi(\mathsf{G}_2, l(P_i)). \tag{1}$$

Theorem 1. *The path-pattern graph kernel $\mathcal{K}_{pp}$ is positive semidefinite.*

Proof. The path-pattern graph kernel $\mathcal{K}_{pp}$ in Equation (1) can also be written as follows:

$$\mathcal{K}_{pp}(\mathsf{G}_1, \mathsf{G}_2) = \langle \phi(\mathsf{G}_1), \phi(\mathsf{G}_2) \rangle$$

where $\phi(\mathsf{G}_1) = [\psi(\mathsf{G}_1, l(P_1)), \psi(\mathsf{G}_1, l(P_2)), \ldots, \psi(\mathsf{G}_1, l(P_{|\mathcal{U}|}))]$ and $\phi(\mathsf{G}_2) = [\psi(\mathsf{G}_2, l(P_1)), \psi(\mathsf{G}_2, l(P_2)), \ldots, \psi(\mathsf{G}_2, l(P_{|\mathcal{U}|}))]$.

Inspired by earlier works on graph kernels, we can readily verify that for any vector $\mathbf{x} \in \mathbb{R}^n$, we have

$$\begin{aligned}\mathbf{x}\mathcal{K}_{pp}\mathbf{x} &= \sum_{i,j=1}^{n} x_i x_j \mathcal{K}_{pp}(\mathsf{G}_i, \mathsf{G}_j) \\ &= \sum_{i,j=1}^{n} x_i x_j \langle \phi(\mathsf{G}_i), \phi(\mathsf{G}_j) \rangle \\ &= \left\langle \sum_{i=1}^{n} x_i \phi(\mathsf{G}_i), \sum_{j=1}^{n} x_j \phi(\mathsf{G}_j) \right\rangle \\ &= \left\| \sum_{i=1}^{n} x_i \phi(\mathsf{G}_i) \right\| \geq 0.\end{aligned} \tag{2}$$

□

For example, if the depth of BFS tree is set to one, the multisets $\mathcal{M}_1$ and $\mathcal{M}_2$ are as follows:

$$\begin{aligned}\mathcal{M}_1 = \{&(1), (1, 4), (1), (1, 4), (4), (4, 1), (4, 1), (4, 3), (4, 3), \\ &(3), (3, 3), (3, 4), (2), (2, 3), (3), (3, 2), (3, 3), (3, 4)\} \\ \mathcal{M}_2 = \{&(1), (1, 1), (1), (1, 1), (1, 4), (4), (4, 1), (4, 3), (4, 3), \\ &(2), (2, 3), (3), (3, 2), (3, 3), (3, 4), (3), (3, 3), (3, 4)\}.\end{aligned}$$

The union of $\mathcal{M}_1$ and $\mathcal{M}_2$ is $\mathcal{U} = \mathcal{M}_1 \cup \mathcal{M}_2$ which is a normal set containing unique elements. The elements are sorted lexicographically.

1. A set that can contain the same element multiple times.

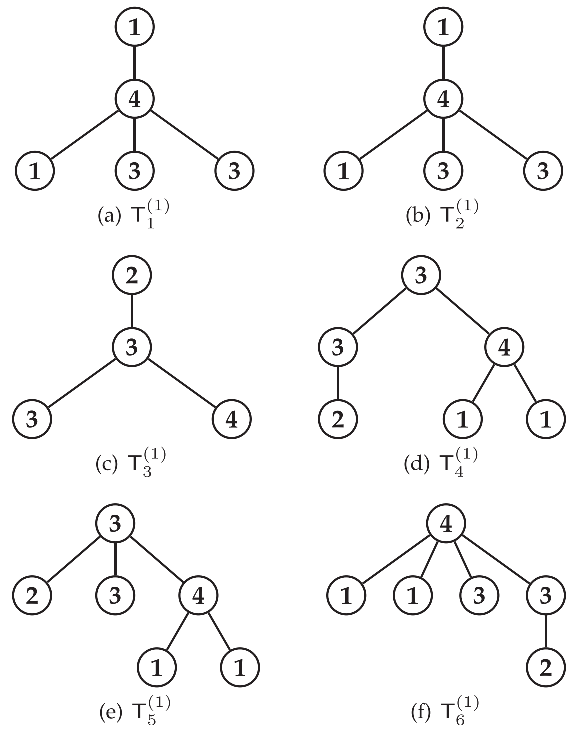


Fig. 4. A truncated BFS tree of depth two rooted at each vertex in the undirected labeled graph G_1.

$$\begin{aligned}\mathcal{U} = \{&(1),(1,1),(1,4),(2),(2,3),\\ &(3),(3,2),(3,3),(3,4),(4),(4,1),(4,3)\}.\end{aligned}$$

Considering that a path uv is equivalent to its reversed one vu in undirected graphs, we remove the repetitive path patterns in $\mathcal{U}$ and finally we have:

$$\mathcal{U} = \{(1),(1,1),(1,4),(2),(2,3),(3),(3,3),(3,4),(4)\}$$

For each path pattern in the set $\mathcal{U}$, we count its occurrences in G_1 and G_2 and have the following:

$$\begin{aligned}\phi(\mathsf{G}_1) &= [\psi(\mathsf{G}_1,(1)),\psi(\mathsf{G}_1,(1,1)),\ldots,\psi(\mathsf{G}_1,(4))]\\ &= [2,0,4,1,2,2,2,4,1]\\ \phi(\mathsf{G}_2) &= [\psi(\mathsf{G}_2,(1)),\psi(\mathsf{G}_2,(1,1)),\ldots,\psi(\mathsf{G}_2,(4))]\\ &= [2,2,2,1,2,2,2,4,1].\end{aligned}$$

Thus $\mathcal{K}_{pp}(\mathsf{G}_1,\mathsf{G}_2) = \langle\phi(\mathsf{G}_1),\phi(\mathsf{G}_2)\rangle = 42$

The path-pattern graph kernel will be used as the base kernel for our final TREE++ graph kernel. We can see that the path-pattern graph kernel decomposes a graph into its substructures, i.e., paths. However, paths cannot reveal the structural or topological information of vertices. Thus, the path-pattern graph kernel can only capture graph similarity at fine granularities. Likewise, most of the graph kernels that belong to the family of R-convolution framework [4] face the same problem colloquially stated as losing sight of the forest for the trees. To capture graph similarity at coarse granularities, we need to zoom out our perspectives on graphs and focus on the structural identities.

3.3 Incorporating Structural Identity

Structural identity is a concept to define the class of vertices in a graph by considering the graph structure and their relations to other vertices. In graphs, vertices are often associated with some functions that determine their roles in the graph. For example, each of the proteins in a protein-protein interaction (PPI) network has a specific function, such as enzyme, antibody, messenger, transport/storage, and structural component. Although such functions may also depend on the vertex and edge attributes, in this paper, we only consider their relations to the graph structures. Explicitly considering the structural identities of vertices in graphs for the design of graph kernels has been missing from the literature except the WeisfeilerLehman subtree kernel [6], [7], Propagation kernel [15], MLG [10], and RetGK [14].

To incorporate the structural identity information into graph kernels, in this paper, we extend the definition of path in graphs and define super path as follows:

Definition 2 (Super Path). *Given an undirected labeled graph $\mathsf{G} = (\mathcal{V},\mathcal{E},l)$, we build a truncated BFS tree $\mathsf{T} = (\mathcal{V}',\mathcal{E}',l)$ ($\mathcal{V}' \subseteq \mathcal{V}$ and $\mathcal{E}' \subseteq \mathcal{E}$) of depth d rooted at each vertex $v \in \mathcal{V}$. The vertex v is called root. For each vertex $v' \in \mathcal{V}'$, there is a path $P = (v, v_1, v_2, \ldots, v')$ from the root v to v' consisting of distinct vertices and edges. For each vertex in P, we build a truncated BFS tree of depth k rooted at it. The sequence of trees $S = \left(\mathsf{T}_v, \mathsf{T}_{v_1}, \mathsf{T}_{v_2}, \ldots, \mathsf{T}_{v'}\right)$ is called a super path.*

We can see that the definition of super path includes the definition of path in graphs. Path is a special case of super path when the truncated BFS tree on each distinct vertex in a path is of depth 0.

The problem now is that what is the path pattern corresponding to the super path? In other words, what is the label of each truncated BFS tree in the super path? In this case, we also need to extend the definition of the label function l described in Section 3.1 as follows: $l : \mathsf{T} \rightarrow \Sigma$ (Σ here is different from above. We abuse the notation.) is a function that assigns labels from a set of positive integers Σ to trees. Thus, the definition of the path pattern for super paths is: the concatenated labels $l(S) = \left(l(\mathsf{T}_v), l(\mathsf{T}_{v_1}), l(\mathsf{T}_{v_2}), \ldots, l(\mathsf{T}'_v)\right)$ is called a path pattern.

For each truncated BFS tree, we need to hash it to a value which is used for its label. In this paper, we just use the concatenation of the labels of its vertices as a hash method. Note that the child vertices of each parent vertex in the BFS trees are sorted by their label and eigenvector centrality values, from low to high. Thus, each truncated BFS tree is uniquely hashed to a string of vertex labels. For example, in Fig. 4, $\mathsf{T}_1^{(1)}$ can be denoted as (1,4,1,3,3), and $\mathsf{T}_4^{(1)}$ can be denoted as (3,3,4,2,1,1). Now, the label function $l : \mathsf{T} \rightarrow \Sigma$ can assign the same positive integers to the same trees (the same sequences of vertex labels). In our implementation, we use a set to store BFS trees of depth k rooted at each vertex in a dataset of graphs. In this case, the set will only contain unique BFS trees. For BFS trees shown in Figs. 4 and 5, the set will contain $\mathsf{T}_1^{(2)}$, $\mathsf{T}_2^{(2)}$, $\mathsf{T}_1^{(1)}$, $\mathsf{T}_3^{(1)}$, $\mathsf{T}_4^{(2)}$, $\mathsf{T}_5^{(1)}$, $\mathsf{T}_5^{(2)}$, $\mathsf{T}_4^{(1)}$, $\mathsf{T}_6^{(1)}$, and $\mathsf{T}_6^{(2)}$. Note that the truncated BFS trees in the set are sorted lexicographically. We can use the index of each truncated BFS tree in the set as its label. For instance, $l : \mathsf{T}_1^{(2)} \rightarrow 1$, $l : \mathsf{T}_2^{(2)} \rightarrow 2$, $l : \mathsf{T}_1^{(1)} \rightarrow 3$, $l : \mathsf{T}_3^{(1)} \rightarrow 4$, $l : \mathsf{T}_4^{(2)} \rightarrow 5$, $l : \mathsf{T}_5^{(1)} \rightarrow 6$, $l : \mathsf{T}_5^{(2)} \rightarrow 7$, $l : \mathsf{T}_4^{(1)} \rightarrow 8$, $l : \mathsf{T}_6^{(1)} \rightarrow 9$, and $l : \mathsf{T}_6^{(2)} \rightarrow 10$. If we use the labels of these truncated BFS trees to relabel their

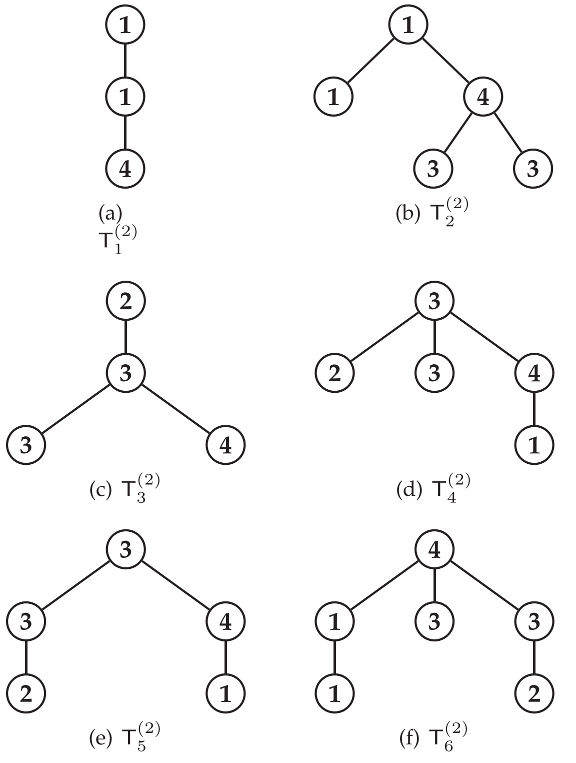


Fig. 5. A truncated BFS tree of depth two rooted at each vertex in the undirected labeled graph G_2.

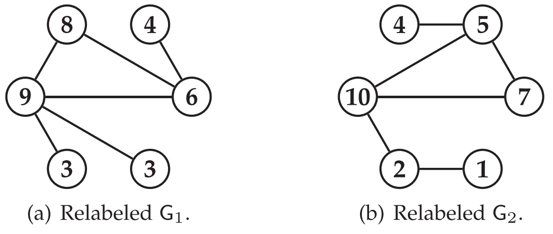


Fig. 6. Relabel graphs. $\Sigma = \{1,2,3,4,5,6,7,8,9,10\}$.

root vertices, graphs G_1 and G_2 in Figs. 3a and 3b become graphs shown in Figs. 6a and 6b.

One observation is that if two vertices have the same structural identities, their corresponding truncated BFS trees are the same and thus they will have the same new labels. For example, Figs. 4a and 4b show two truncated BFS trees on the two vertices with the same label 1 in Fig. 3a. The two trees are identical, and thus these two vertices' structural identities are identical, and their new labels in Fig. 6a are also the same. This phenomenon also happens across graphs, e.g., the vertices with label 2 in Figs. 3a and 3b also have the same labels in Figs. 6a and 6b (vertices with the label 4). Figs. 4d and 4e show another two truncated BFS trees on the two vertices with label 3 in Fig. 3a. We can see that they have different structural identities. Thus, by integrating structural identities into path patterns, we can distinguish path patterns at different levels of granularities. If we build truncated BFS trees of depth 0 rooted at each vertex for super paths, the two path patterns (1,4,3) in Fig. 3a and (1,4,3) in Fig. 3b (The starting vertex is the left-bottom corner vertex with label 1, and the end vertex is the right-most vertex with label 3.) are identical. However, if we build super paths using truncated BFS trees of depth two (as shown in Figs. 4 and 5), the two path patterns become the two new path patterns (3,9,6) and (2,10,7). They are totally different.

3.4 Tree++

Definition 3 (Graph Similarity at the k-level of Granularity). *Given two undirected labeled graphs G_1 and G_2, we build truncated BFS trees of depth d rooted at each vertex in these two graphs. All paths in all of these BFS trees are contained in a set $\mathcal{P}$. For each path in $\mathcal{P}$, we build a truncated BFS tree of depth $k (k \geq 0)$ rooted at each vertex in the path. All super paths are contained in a set $\mathcal{S}^{(k)}$ ($\mathcal{S}^{(k)} = \mathcal{P}$, if $k = 0$). The graph similarity at the k-level of granularity is defined as follows:*

$$\mathcal{K}_{pp}^{(k)}(\mathsf{G}_1, \mathsf{G}_2) = \sum_{\substack{S_i^{(k)} \in \mathcal{S}^{(k)} \\ l(S_i^{(k)}) \in \mathcal{U}^{(k)}}} \psi\Big(\mathsf{G}_1, l(S_i^{(k)})\Big) \psi\Big(\mathsf{G}_2, l(S_i^{(k)})\Big), \quad (3)$$

where $\mathcal{U}^{(k)}$ is a set that contains all of unique path patterns at the k-level of granularity.

To make our path pattern graph kernel capture graph similarity at multiple levels of granularities, we formalize the following:

$$\mathcal{K}_{Tree++}(\mathsf{G}_1, \mathsf{G}_2) = \sum_{i=0}^{k} \mathcal{K}_{pp}^{(i)}(\mathsf{G}_1, \mathsf{G}_2). \quad (4)$$

We call the above formulation as TREE++. Note that TREE++ is positive semi-definite because a sum of positive semi-definite kernels is also positive semi-definite.

In the following, we give the algorithmic details of our TREE++ graph kernel in Algorithm 1. Lines 2–8 generate paths for each vertex in each graph. For each vertex v, we build a truncated BFS tree of depth d rooted at it. The time complexity of BFS traversal of a graph is $\mathcal{O}(|\mathcal{V}| + |\mathcal{E}|)$, where $|\mathcal{V}|$ is the number of vertices, and $|\mathcal{E}|$ is the number of edges in a graph. For convenience, we assume that $|\mathcal{E}| > |\mathcal{V}|$ and all the n graphs have the same number of vertices and edges. The worst time complexity of our path generation for all the n graphs is $\mathcal{O}(n \cdot |\mathcal{V}| \cdot (|\mathcal{E}| + |\mathcal{V}|))$. Lines 11–18 generate super paths from paths. The number of paths generated in lines 2–8 is $n \cdot |\mathcal{V}| \cdot (|\mathcal{E}| + |\mathcal{V}|)$. For each path, it at most contains $|\mathcal{V}|$ vertices. For each vertex in the path, we need to construct a BFS tree of depth k, which costs $\mathcal{O}(|\mathcal{E}|)$. Thus, the worst time complexity of generating super paths for n graphs is $\mathcal{O}(n \cdot |\mathcal{V}|^2 \cdot |\mathcal{E}|^2 + n \cdot |\mathcal{V}|^3 \cdot |\mathcal{E}|)$. Line 19 sorts the elements in $\mathcal{U}$ lexicographically, of which the time complexity is bounded by $\mathcal{O}(|\mathcal{E}|)$ [7]. Lines 20–21 count the occurrences of each unique path pattern in graphs. For each unique path pattern in $\mathcal{U}^{(i)}$, we need to count its occurrences in each graph. The time complexity for counting is bounded by $\mathcal{O}(q \cdot m)$, where q is the maximum length of all $\mathcal{U}^{(i)} (0 \leq i \leq k)$, and m is the maximum length of AllSuperPaths[j] $(1 \leq j \leq n)$. Thus, the time complexity of lines 10–21 is $\mathcal{O}(n \cdot |\mathcal{V}|^2 \cdot |\mathcal{E}|^2 + n \cdot |\mathcal{V}|^3 \cdot |\mathcal{E}| + n \cdot q \cdot m)$. The time complexity for line 23 is bounded by $\mathcal{O}(n^2 \cdot q)$. The worst time complexity of our TREE++ graph kernel for n graphs with the depth of k truncated BFS trees for super paths is $\mathcal{O}(k \cdot n \cdot |\mathcal{V}|^2 \cdot |\mathcal{E}|^2 + k \cdot n \cdot |\mathcal{V}|^3 \cdot |\mathcal{E}| + k \cdot n^2 \cdot q + k \cdot n \cdot q \cdot m)$.

TABLE 1
Statistics of the Real-World Datasets Used in the Experiments

Dataset	Size	Class #	Avg. Node#	Avg. Edge#	Label #
BZR	405	2	35.75	38.36	10
BZR_MD	306	2	21.30	225.06	8
COX2	467	2	41.22	43.45	8
COX2_MD	303	2	26.28	335.12	7
DHFR	467	2	42.43	44.54	9
DHFR_MD	393	2	23.87	283.01	7
NCI1	4110	2	29.87	32.30	37
PROTEINS	1113	2	39.06	72.82	3
Mutagenicity	4337	2	30.32	30.77	14
PTC_MM	336	2	13.97	14.32	20
PTC_FR	351	2	14.56	15.00	19
KKI	83	2	26.96	48.42	190

Algorithm 1. Tree++

```
Input: A set of graphs $\mathcal{G} = \{\mathsf{G}_1, \mathsf{G}_2, \ldots, \mathsf{G}_n\}$ and their vertex
       label functions $\mathcal{L} = \{l_1, l_2, \ldots, l_n\}$, $d$, $k$
Output: The computed kernel matrix $\mathbf{K} \in \mathbb{N}^{n\times n}$
1  $\mathbf{K} \leftarrow$ zeros($(n, n)$), AllPaths$\leftarrow${};
   /* Path generation                                        */
2  for $i \leftarrow 1$ to $n$ do
3      Paths$\leftarrow$[];                              /* list   */
4      foreach vertex $v \in \mathsf{G}_i$ do
5          Build a truncated BFS tree $\mathsf{T}$ of depth $d$ rooted at the
             vertex $v$;
6          foreach vertex $v' \in \mathsf{T}$ do
7              Paths.append($P$);
                 /* $P = (v, v_1, v_2, \ldots, v')$                  */
8      AllPaths[$i$] $\leftarrow$Paths;
   /* Compute Tree++ .                                       */
9  for $i \leftarrow 0$ to $k$ do
     /* Super path generation.                               */
10     $\mathcal{U}^{(i)}$ $\leftarrow$set( ), AllSuperPaths$\leftarrow${};
11     for $j \leftarrow 1$ to $n$ do
12         SuperPaths$\leftarrow$ [], Paths$\leftarrow$AllPaths[$j$];
13         foreach $P \in$ Paths do
14             foreach vertex $v$ in $P$ do
15                 Build a truncated BFS tree $\mathsf{T}_v$ of depth $i$ rooted at
                     the vertex $v$ in graph $\mathsf{G}_j$;
16                     SuperPaths.append($l(S^{(i)})$);    /*    $S^{(i)} = (\mathsf{T}_{v_1},$
                     $\mathsf{T}_{v_2}, \ldots), P = (v_1, v_2, \ldots)$     */
17             $\mathcal{U}^{(i)}$.add($l(S^{(i)})$);
18         AllSuperPaths[$j$] $\leftarrow$SuperPaths;
             /* contains all the super paths in graph
             $\mathsf{G}_j$                                          */
19     $\mathcal{U}^{(i)}$ $\leftarrow$sort($\mathcal{U}^{(i)}$); /* lexicographically          */
20     for $j \leftarrow 1$ to $n$ do
21         $\phi(\mathsf{G}_j) \leftarrow \left[\psi(\mathsf{G}_j, l(S_1^{(j)})), \psi(\mathsf{G}_j, l(S_2^{(j)}), \ldots, \psi(\mathsf{G}_j, l(S_{|\mathcal{U}^{(i)}|}^{(j)})\right]$
           /* count the number $\psi(\mathsf{G}_j, l(S^{(j)}))$ of the occur-
           rences of each path pattern stored in AllSu-
           perPaths[$j$].
           $l(S_1^{(j)}), l(S_2^{(j)}), \ldots, l\left(S_{|\mathcal{U}^{(i)}|}^{(j)}\right) \in \mathcal{U}^{(i)}$          */
22     $\mathbf{\Phi} \leftarrow [\phi(\mathsf{G}_1); \phi(\mathsf{G}_2); \ldots; \phi(\mathsf{G}_n)]$;
23     $\mathbf{K} \leftarrow \mathbf{K} + \mathbf{\Phi} \cdot \mathbf{\Phi}^\top$;
24 return K ;
```

4 Experimental Setup

We run all the experiments on a desktop with an Intel Core i7-8700 3.20 GHz CPU, 32 GB memory, and Ubuntu 18.04.1 LTS operating system, Python version 2.7. Tree++ is written in Python. We make our code publicly available at Github.[2]

We compare Tree++ with seven state-of-the-art graph kernels, i.e., MLG [10], DGK [22], RetGK [14], Propa [15], PM [26], SP [9], and WL [7]. We also compare Tree++ with one state-of-the-art graph classification method FGSD [28] which learns features from graphs and then directly feed them into classifiers. We set the parameters for our Tree++ graph kernel as follows: The depth of the truncated BFS tree rooted at each vertex is set as $d = 6$, and the depth k of the truncated BFS tree in the super path is chosen from {0, 1, 2, . . . , 7} through cross-validation. The parameters for the comparison methods are set according to their original papers. We use the implementations of Propa, PM, SP, and WL from the GraKeL [36] library. The implementations of other methods are obtained from their corresponding websites. A short description for each comparison method is given as follows:

- MLG [10] is a graph kernel that builds a hierarchy of nested subgraphs to capture graph structures at a range of different scales.
- DGK [22] uses techniques from natural language processing to learn latent representations for substructures extracted by graph kernels such as SP [9], and WL [7]. Then the similarity matrix between substructures is computed and integrated into the computation of the graph kernel matrices.
- RetGK [14] introduces a structural role descriptor for vertices, i.e., the return probabilities features (RPF) generated by random walks. The RPF is then embedded into the Hilbert space where the corresponding graph kernels are derived.
- Propa [15] leverages early-stage distributions of random walks to capture structural information hidden in vertex neighborhood.
- PM [26] embeds graph vertices into vectors and use the Pyramid Match kernel to compute the similarity between the sets of vectors of two graphs.
- SP [9] counts the number of pairs of shortest paths which have the same source and sink labels and the same length in two graphs.
- WL [7] is based on the Weisfeiler-Lehman test of isomorphism [21] for graphs. It counts the number of occurrences of each subtrees in graphs.
- FGSD [28] discovers family of graph spectral distances and their based graph feature representations to classify graphs.

All graph kernel matrices are normalized according to the method proposed in [29]. For each entry $\mathbf{K}(i, j)$, it will be normalized as $\mathbf{K}(i, j)/\sqrt{\mathbf{K}(i, i)\mathbf{K}(j, j)}$. All diagonal entries will be 1. We use 10-fold cross-validation with a binary C-SVM [37] to test classification performance of each graph kernel. The parameter C for each fold is independently tuned from $\{1, 10, 10^2, 10^3\}$ using the training data from that fold. We repeat the experiments ten times and report the average

2. https://github.com/yeweiysh/TreePlusPlus

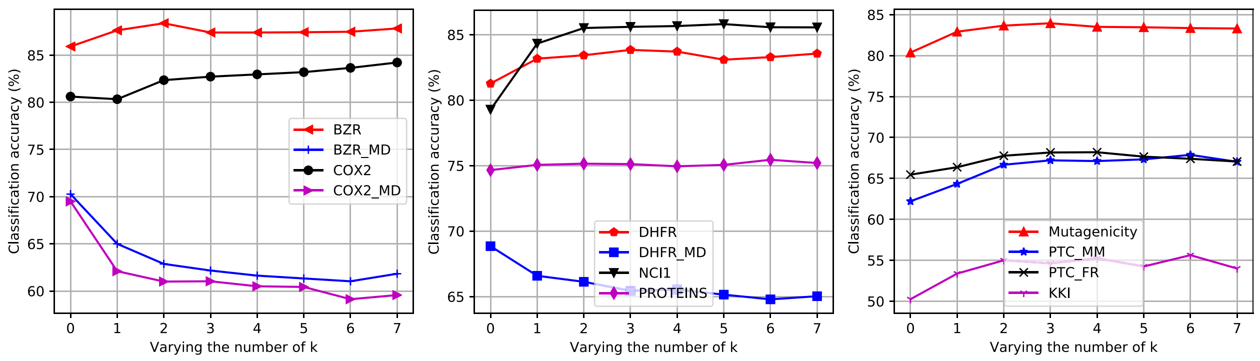


Fig. 7. The classification accuracy of Tree++ on each real-world dataset when varying the number of k (the depth of the truncated BFS tree in the super path).

classification accuracies and standard deviations. We also test the running time of each method on each real-world dataset.

In order to test the efficacy of our graph kernel Tree++, we adopt twelve real-word datasets whose statistics are given in Table 1. Fig. 9 shows the distributions of vertex number, edge number and degree in these twelve real-world datasets.

Chemical Compound Datasets. The chemical compound datasets BZR, BZR_MD, COX2, COX2_MD, DHFR, and DHFR_MD are from the paper [38]. Chemical compounds or molecules are represented by graphs. Edges represent the chemical bond type, i.e., single, double, triple or aromatic. Vertices represent atoms. Vertex labels represent atom types. BZR is a dataset of 405 ligands for the benzodiazepine receptor. COX2 is a dataset of 467 cyclooxygenase-2 inhibitors. DHFR is a dataset of 756 inhibitors of dihydrofolate reductase. BZR_MD, COX2_MD, and DHFR_MD are derived from BZR, COX2, and DHFR respectively, by removing explicit hydrogen atoms. The chemical compounds in the datasets BZR_MD, COX2_MD, and DHFR_MD are represented as complete graphs, where edges are attributed with distances and labeled with the chemical bond type. NCI1 [39] is a balanced dataset of chemical compounds screened for the ability to suppress the growth of human non-small cell lung cancer.

Molecular Compound Datasets. The dataset PROTEINS is from [40]. Each protein is represented by a graph. Vertices represent secondary structure elements. Edges represent that two vertices are neighbors along the amino acid sequence or three-nearest neighbors to each other in space. Mutagenicity [41] is a dataset of 4337 molecular compounds which can be divided into two classes mutagen and non-mutagen. The PTC [1] dataset consists of compounds labeled according to carcinogenicity on rodents divided into male mice (MM), male rats (MR), female mice (FM) and female rats (FR).

Brain Network Dataset. KKI [42] is a brain network constructed from the whole brain functional resonance image (fMRI) atlas. Each vertex corresponds to a region of interest (ROI), and each edge indicates correlations between two ROIs. KKI is constructed for the task of Attention Deficit Hyperactivity Disorder (ADHD) classification.

5 Experimental Results

In this section, we first evaluate Tree++ with differing parameters on each real-world dataset, then compare Tree++ with eight baselines on classification accuracy and runtime.

5.1 Parameter Sensitivity

In this section, we test the performance of our graph kernel Tree++ on each real-world dataset when varying its two parameters, i.e., the depth k of the truncated BFS tree in the super path, and the depth d of the truncated BFS tree rooted at each vertex to extract path patterns. We vary the number of k and d both from zero to seven. When varying the number of k, we fix $d = 7$. When varying the number of d, we fix $k = 1$.

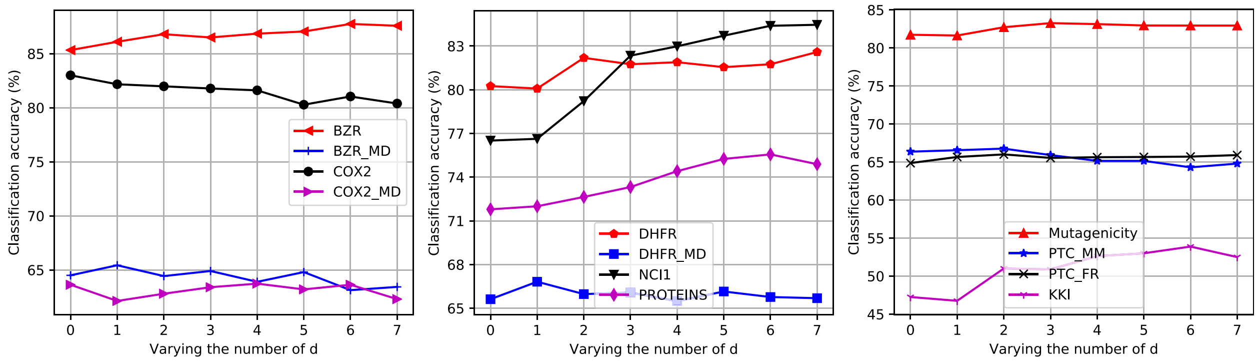


Fig. 8. The classification accuracy of Tree++ on each real-world dataset when varying the number of d (the depth of the truncated BFS tree rooted at each vertex to extract path patterns).

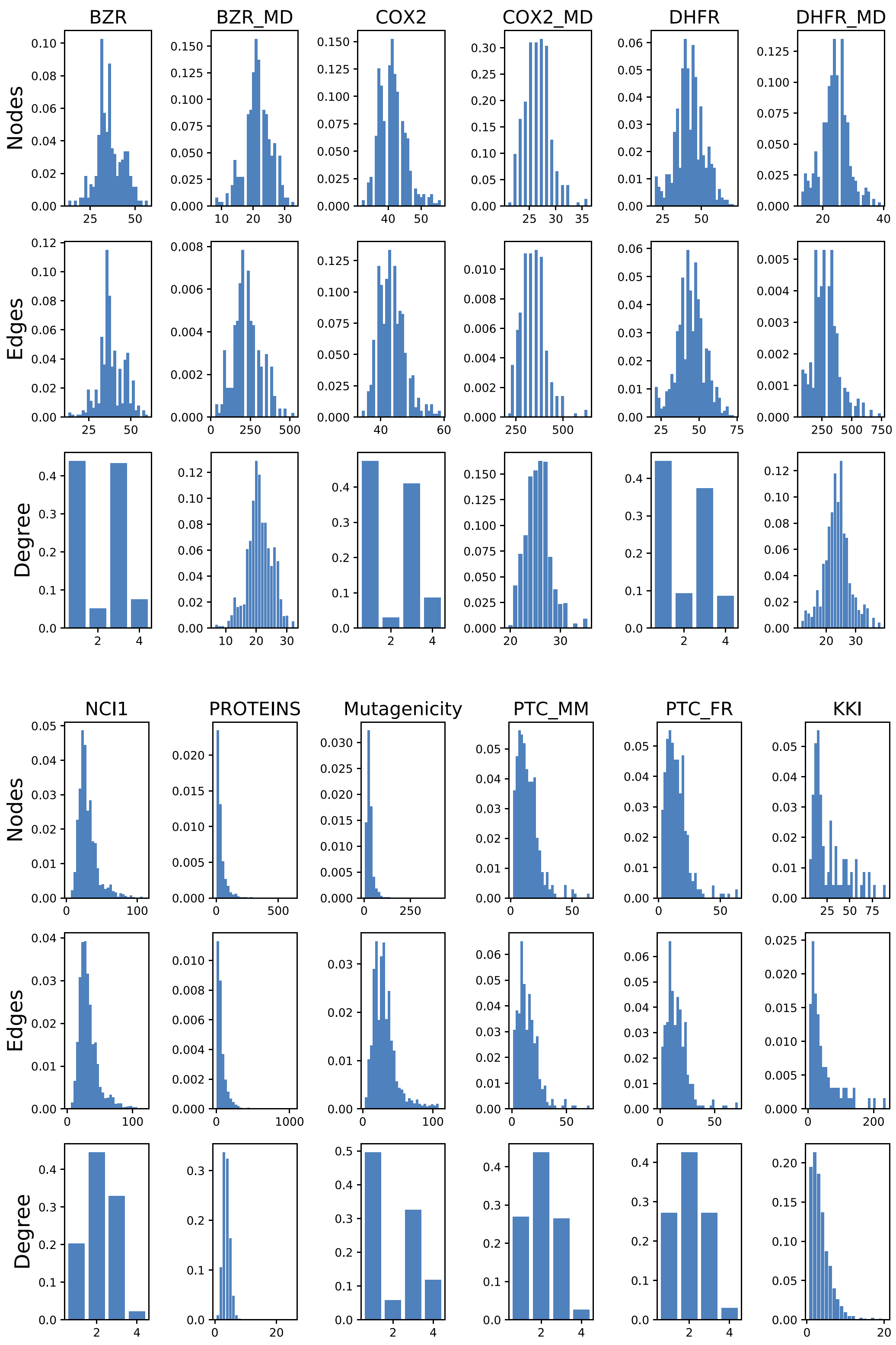


Fig. 9. The rows illustrate the distributions of node number, edge number, and degree in the datasets used in the paper.

Fig. 7 shows the classification accuracy of Tree++ on each real-world dataset when varying the number of k. On the original chemical compound datasets, we can see that Tree++ tends to reach better classification accuracy with increasing values of k. The tendency is reversed on the derived chemical compound datasets where explicit hydrogen atoms are removed. We can see from Fig. 9 that compared with the original datasets BZR, COX2, and DHFR, the derived datasets BZR_MD, COX2_MD, and DHFR_MD have more diverse edge and degree distributions, i.e., the edge number and degree vary more than those of the original datasets. In addition, their degree distributions do not follow the power law. For graphs with many high degree vertices, the concatenation of vertex labels as a hash method for BFS trees in super paths can hurt the performance. For example, two BFS trees in super paths may just have one

TABLE 2
Comparison of Classification Accuracy (± Standard Deviation) of Tree++ and Its Competitors on the Real-World Datasets

Dataset	Tree++	MLG	DGK	RetGK	Propa	PM	SP	WL	FGSD
BZR	**87.88** ± 1.00	86.28 ± 0.59	83.08 ± 0.53	86.30 ± 0.71	85.95 ± 0.85	82.35 ± 0.47	85.65 ± 1.02	87.25 ± 0.77	85.38 ± 0.85
BZR_MD	**69.47** ± 1.14	48.87 ± 2.44	58.50 ± 1.52	62.77 ± 1.69	61.53 ± 1.27	68.20 ± 1.24	68.60 ± 1.94	59.67 ± 1.47	61.00 ± 1.35
COX2	**84.28** ± 0.85	76.91 ± 1.14	78.30 ± 0.29	81.85 ± 0.83	81.33 ± 1.36	77.34 ± 0.82	80.87 ± 1.20	81.20 ± 1.05	78.30 ± 1.03
COX2_MD	**69.20** ± 1.69	48.17 ± 2.43	51.57 ± 1.71	59.47 ± 1.66	55.33 ± 1.70	63.60 ± 0.87	65.70 ± 1.66	56.30 ± 1.55	48.97 ± 1.90
DHFR	**83.68** ± 0.59	79.61 ± 0.50	64.13 ± 0.89	82.33 ± 0.66	80.67 ± 0.52	64.59 ± 1.25	77.80 ± 0.98	82.39 ± 0.90	78.13 ± 0.58
DHFR_MD	**68.87** ± 0.91	67.87 ± 0.12	67.90 ± 0.26	64.44 ± 0.98	64.18 ± 0.97	66.21 ± 1.01	68.00 ± 0.36	64.00 ± 0.47	66.62 ± 0.78
NCI1	**85.77** ± 0.12	78.20 ± 0.32	66.72 ± 0.29	84.28 ± 0.25	79.71 ± 0.39	63.55 ± 0.44	73.12 ± 0.29	84.79 ± 0.22	75.99 ± 0.51
PROTEINS	75.46 ± 0.47	72.01 ± 0.83	72.59 ± 0.51	75.77 ± 0.66	72.71 ± 0.83	73.66 ± 0.67	**76.00** ± 0.29	75.32 ± 0.20	70.14 ± 0.67
Mutagenicity	**83.64** ± 0.27	76.85 ± 0.38	66.80 ± 0.15	82.89 ± 0.18	81.47 ± 0.34	69.06 ± 0.14	77.52 ± 0.13	83.51 ± 0.27	70.71 ± 0.39
PTC_MM	**68.03** ± 0.61	61.21 ± 1.08	67.09 ± 0.49	65.79 ± 1.76	64.12 ± 1.43	62.27 ± 1.51	62.18 ± 2.22	67.18 ± 1.61	57.88 ± 1.97
PTC_FR	**68.71** ± 1.29	64.31 ± 2.00	67.66 ± 0.32	66.77 ± 0.99	65.14 ± 2.04	64.86 ± 0.88	66.91 ± 1.46	66.17 ± 1.02	63.80 ± 1.51
KKI	**55.63** ± 1.69	48.00 ± 3.64	51.25 ± 4.17	48.50 ± 2.99	50.88 ± 4.17	52.25 ± 2.49	50.13 ± 3.46	50.38 ± 2.77	49.25 ± 4.76

different vertex, which can lead to different hashing. Thus, with increasing values of k, two graphs with many high degree vertices tend to be more dissimilar, which is a reason for the decreasing of classification accuracy in datasets BZR_MD, COX2_MD, and DHFR_MD. Since the degree distribution of the brain network dataset KKI follow the power law, we can observe a tendency of Tree++ to reach better classification accuracy with increasing values of k. On all the molecular compound datasets whose degree distribution also follow the power law, we also observe a tendency of Tree++ to reach better classification accuracy with increasing values of k. Another observation is that the classification accuracy of Tree++ first increase and then remain stable with increasing values of k. One explaination is that if smaller values of k can distinguish the structure identities of vertices, larger values of k will not benefit much to the increase of classification accuracy.

Fig. 8 shows the classification accuracy of Tree++ on each real-world dataset when varying the number of d. On all the chemical compound datasets except COX2, and on all the molecular compound datasets and brain network dataset, Tree++ tends to become better with increasing values of d. The phenomena are obvious because deep BFS trees can capture more path patterns around a vertex.

5.2 Classification Results

Table 2 shows the classification accuracy of our graph kernel Tree++ and its competitors on the twelve real-world datasets. Tree++ is superior to all of the competitors on eleven real-world datasets. On the dataset COX2_MD, the classification accuracy of Tree++ has a gain of 5.3 percent over that of the second best method SP, and has a gain of 43.7 percent over that of the worst method MLG. On the dataset KKI, the classification accuracy of Tree++ has a gain of 6.5 percent over that of the second best method PM, and has a gain of 15.9 percent over that of the worst method MLG. On the datasets DHFR_MD, Mutagenicity, and PTC_MM, Tree++ is slightly better than WL. On the dataset PROTEINS, SP achieves the best classification accuracy. Tree++ achieves the second best classification accuracy. However, the classification accuracy of SP only has a gain of 0.7 percent over that of Tree++. To summarize, our Tree++ kernel achieves the highest accuracy on eleven datasets and is comparable to SP on the dataset PROTEINS.

5.3 Runtime

Table 3 demonstrates the running time of every method on the real-world datasets. Tree++ scales up easily to graphs with thousands of vertices. On the dataset Mutagenicity, Tree++ finishes its computation in about one minute. It costs RetGK about twelve minutes to finish. It even costs MLG about one hour to finish. On the dataset NCI1, Tree++ finishes its computation in about one minute, while RetGK uses about twelve minutes and MLG uses about one hour. On the other datasets, Tree++ is comparable to SP and WL.

TABLE 3
Comparison of Runtime (in seconds) of Tree++ and Its Competitors on the Real-World Datasets

Dataset	Tree++	MLG	DGK	RetGK	Propa	PM	SP	WL	FGSD
BZR	11.29	142.80	1.60	13.70	11.76	16.80	12.37	1.73	0.73
BZR_MD	4.73	89.93	1.23	4.22	4.89	17.15	17.81	7.72	0.07
COX2	14.57	78.29	2.26	15.73	7.28	6.48	4.81	0.92	0.14
COX2_MD	7.83	4.42	1.10	5.62	1.71	4.67	2.67	1.14	0.07
DHFR	26.24	200.05	4.44	48.95	14.17	16.01	12.07	1.95	0.22
DHFR_MD	8.10	19.03	1.12	10.02	3.01	6.82	4.65	1.49	0.08
NCI1	81.68	3315.42	39.35	761.45	221.84	326.43	22.89	101.68	1.39
PROTEINS	59.56	3332.31	48.83	49.07	27.72	32.79	36.38	38.66	0.49
Mutagenicity	87.09	4088.53	24.67	735.48	526.96	672.33	28.03	94.05	1.56
PTC_MM	1.99	152.69	1.08	2.84	2.44	10.00	1.83	0.95	0.08
PTC_FR	2.19	170.05	1.14	3.16	5.73	10.01	2.48	1.77	0.09
KKI	2.00	67.58	0.65	0.40	0.57	1.25	1.27	0.25	0.02

6 Discussion

Differing from the Weisfeiler-Lehman subtree kernel (WL) which uses subtrees (each vertex can appear repeatedly) to extract features from graphs, we use BFS trees to extract features from graphs. In this case, every vertex will appear only once in a BFS tree. Another different aspect is that we count the number of occurrences of each path pattern while WL counts the number of occurrences of each subtree pattern. If the BFS trees used in the construction of path patterns and super paths are of depth zero, Tree++ is equivalent to WL using subtree patterns of depth zero; If the BFS trees used in the construction of path patterns are of depth zero, and of super paths are of depth one, Tree++ is equivalent to WL using subtree patterns of depth one. In other cases, Tree++ and the Weisfeiler-Lehman subtree kernel deviate from each other. Tree++ is also related to the shortest-path graph kernel (SP) in the sense that both of them use the shortest paths in graphs. SP counts the number of pairs of shortest paths which have the same source and sink labels and the same length in two graphs. Each shortest-path used in SP is represented as a tuple in the form of "(source, sink, length)" which does not explicitly consider the intermediate vertices. However, Tree++ explicitly considers the intermediate vertices. If two shortest-paths with the same source and sink labels and the same length but with different intermediate vertices, SP cannot distinguish them whereas Tree++ can. Thus compared with SP, Tree++ has higher discrimination power.

As discussed in Section 1, WL can only capture the graph similarity at coarse granularities, and SP can only capture the graph similarity at fine granularities. By inheriting merits both from trees and shortest-paths, our method Tree++ can capture the graph similarity at multiple levels of granularities. Although MLG can also capture the graph similarity at multiple levels of granularities, it needs to invert the graph Laplacian matrix, which costs a lot of time. Tree++ is scalable to large graphs. Tree++ is built on the truncated BFS trees rooted at each vertex in a graph. One main problem is that the truncated BFS trees are not unique. To solve this problem, we build BFS trees considering the label and eigenvector centrality values of each vertex. Alternatively, we can also use other centrality metrics such as closeness centrality [43] and betweenness centrality [44] to order the vertices in the BFS trees. An interesting research topic in the future is to investigate the effects of using different centrality metrics to construct BFS trees on the performance of Tree++.

As stated in Section 2, hash functions have been integrated into the design of graph kernels. But they are just adopted for hashing continuous attributes to discrete ones. Conventionally, hash functions are developed for efficient nearest neighbor search in databases. Usually, people first construct a similarity graph from data and then learn a hash function to embed data points into a low-dimensional space where neighbors in the input space are mapped to similar codes [45]. For two graphs, we can first use hash functions such as Spectral Hashing (SH) [46] or graph embedding methods such as DeepWalk [47] to embed each vertex in a graph into a vector space. Each graph is represented as a set of vectors. Then, following RetGK [14], we can use the Maximum Mean Discrepancy (MMD) [48] to compute the similairty between two sets of vectors. Finally, we have a kernel matrix for graphs. This research direction is worth exploring in the future. Tree++ is designed for graphs with discrete vertex labels. Another research direction in the future is to extend Tree++ to graphs with both discrete and continuous attributes.

7 Conclusion

In this paper, we have presented two novel graph kernels: (1) The path-pattern graph kernel that uses the paths from the root to every other vertex in a truncated BFS tree as features to represent a graph; (2) The Tree++ graph kernel that incorporates a new concept of super path into the path-pattern graph kernel and can compare the graph similarity at multiple levels of granularities. Tree++ can capture topological relations between not only individual vertices, but also subgraphs, by adjusting the depths of truncated BFS trees in the super paths. Empirical studies demonstrate that Tree++ is superior to other well-known graph kernels in the literature regarding classification accuracy and runtime.

Acknowledgments

The authors would like to thank anonymous reviewers for their constructive and helpful comments. This work was supported partially by the National Science Foundation (grant # IIS-1817046) and by the U. S. Army Research Laboratory and the U. S. Army Research Office (grant # W911NF-15-1-0577).

References

[1] N. Kriege and P. Mutzel, "Subgraph matching kernels for attributed graphs," in *Proc. 29th Int. Conf. Mach. Learn.*, 2012, pp. 291–298.

[2] 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," *J. Medicinal Chemistry*, vol. 34, no. 2, pp. 786–797, 1991.

[3] A. Tsitsulin, D. Mottin, P. Karras, A. Bronstein, and E. Müller, "Netlsd: Hearing the shape of a graph," in *Proc. 24th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2018, pp. 2347–2356.

[4] D. Haussler, "Convolution kernels on discrete structures," Dept. Comput. Sci., Univ. California at Santa Cruz, Santa Cruz, CA, Tech. Rep. UCSC-CRL-99–10, 1999.

[5] N. Shervashidze, S. Vishwanathan, T. Petri, K. Mehlhorn, and K. Borgwardt, "Efficient graphlet kernels for large graph comparison," in *Proc. 12th Int. Conf. Artif. Intell. Statist.*, 2009, pp. 488–495.

[6] N. Shervashidze and K. M. Borgwardt, "Fast subtree kernels on graphs," in *Proc. 22nd Int. Conf. Neural Inf. Process. Syst.*, 2009, pp. 1660–1668.

[7] N. Shervashidze, P. Schweitzer, E. J. V. Leeuwen, K. Mehlhorn, and K. M. Borgwardt, "Weisfeiler-lehman graph kernels," *J. Mach. Learn. Res.*, vol. 12, pp. 2539–2561, 2011.

[8] S. V. N. Vishwanathan, N. N. Schraudolph, R. Kondor, and K. M. Borgwardt, "Graph kernels," *J. Mach. Learn. Res.*, vol. 11, pp. 1201–1242, 2010.

[9] K. M. Borgwardt and H.-P. Kriegel, "Shortest-path kernels on graphs," in *Proc. 5th IEEE Int. Conf. Data Mining*, 2005, pp. 8–pp.

[10] R. Kondor and H. Pan, "The multiscale laplacian graph kernel," in *Proc. 30th Int. Conf. Neural Inf. Process. Syst.*, 2016, pp. 2990–2998.

[11] T. Gärtner, P. Flach, and S. Wrobel, "On graph kernels: Hardness results and efficient alternatives," in *Learning theory and kernel machines*. Berlin, Germany: Springer, 2003, pp. 129–143.

[12] H. Kashima, K. Tsuda, and A. Inokuchi, "Marginalized kernels between labeled graphs," in *Proc. 20th Int. Conf. Int. Conf. Mach. Learn.*, 2003, pp. 321–328.

[13] M. Neumann, N. Patricia, R. Garnett, and K. Kersting, "Efficient graph kernels by randomization," in *Proc. Joint Eur. Conf. Mach. Learn. Knowl. Discovery Databases*, 2012, pp. 378–393.

[14] Z. Zhang, M. Wang, Y. Xiang, Y. Huang, and A. Nehorai, "RetGK: Graph kernels based on return probabilities of random walks," in *Proc. 32nd Int. Conf. Neural Inf. Process. Syst.*, 2018, pp. 3964–3974.

[15] M. Neumann, R. Garnett, C. Bauckhage, and K. Kersting, "Propagation kernels: Efficient graph kernels from propagated information," *Mach. Learn.*, vol. 102, no. 2, pp. 209–245, 2016.

[16] F. Costa and K. D. Grave, "Fast neighborhood subgraph pairwise distance kernel," in *Proc. 27th Int. Conf. Int. Conf. Mach. Learn.*, 2010, pp. 255–262.

[17] T. Horváth, T. Gärtner, and S. Wrobel, "Cyclic pattern kernels for predictive graph mining," in *Proc. 10th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2004, pp. 158–167.

[18] N. Pržulj, D. G. Corneil, and I. Jurisica, "Modeling interactome: scale-free or geometric?" *Bioinf.*, vol. 20, no. 18, pp. 3508–3515, 2004.

[19] J. Ramon and T. Gärtner, "Expressivity versus efficiency of graph kernels," in *Proc. 1st Int. Workshop Mining Graphs Trees Sequences*, 2003, pp. 65–74.

[20] P. Mahé and J.-P. Vert, "Graph kernels based on tree patterns for molecules," *Mach. Learn.*, vol. 75, no. 1, pp. 3–35, 2009.

[21] B. Weisfeiler and A. Lehman, "A reduction of a graph to a canonical form and an algebra arising during this reduction," *Nauchno-Technicheskaya Informatsia*, vol. 2, no. 9, pp. 12–16, 1968.

[22] P. Yanardag and S. Vishwanathan, "Deep graph kernels," in *Proc. 21th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2015, pp. 1365–1374.

[23] N. M. Kriege, P.-L. Giscard, and R. Wilson, "On valid optimal assignment kernels and applications to graph classification," in *Proc. 30th Int. Conf. Neural Inf. Process. Syst.*, 2016, pp. 1623–1631.

[24] X. Kong, W. Fan, and P. S. Yu, "Dual active feature and sample selection for graph classification," in *Proc. 17th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2011, pp. 654–662.

[25] J. B. Lee, R. Rossi, and X. Kong, "Graph classification using structural attention," in *Proc. 24th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2018, pp. 1666–1674.

[26] G. Nikolentzos, P. Meladianos, and M. Vazirgiannis, "Matching node embeddings for graph similarity," in *Proc. 31st AAAI Conf. Artif. Intell.*, 2017, pp. 2429–2435.

[27] M. Niepert, M. Ahmed, and K. Kutzkov, "Learning convolutional neural networks for graphs," in *Proc. 33rd Int. Conf. Int. Conf. Mach. Learn.*, 2016, pp. 2014–2023.

[28] S. Verma and Z.-L. Zhang, "Hunt for the unique, stable, sparse and fast feature learning on graphs," in *Proc. Advances Neural Inf. Process. Syst.*, 2017, pp. 88–98.

[29] A. Feragen, N. Kasenburg, J. Petersen, M. de Bruijne, and K. Borgwardt, "Scalable kernels for graphs with continuous attributes," in *Proc. Advances Neural Inf. Process. Syst.*, 2013, pp. 216–224.

[30] Y. Su, F. Han, R. E. Harang, and X. Yan, "A fast kernel for attributed graphs," in *Proc. SIAM Int. Conf. Data Mining*, 2016, pp. 486–494.

[31] K. Grauman and T. Darrell, "Approximate correspondences in high dimensions," in *Proc. Advances Neural Inf. Process. Syst.*, 2007, pp. 505–512.

[32] C. Morris, N. M. Kriege, K. Kersting, and P. Mutzel, "Faster kernels for graphs with continuous attributes via hashing," in *Proc. IEEE 16th Int. Conf. Data Mining*, 2016, pp. 1095–1100.

[33] F. Orsini, P. Frasconi, and L. De Raedt, "Graph invariant kernels," in *Proc. 24th Int. Joint Conf. Artif. Intell.*, 2015, pp. 3756–3762.

[34] F. Harary, "Graph theory," Addison Wesley series in mathematics, Addison-Wesley, 1971.

[35] P. Bonacich, "Power and centrality: A family of measures," *Amer. J. Sociology*, vol. 92, no. 5, pp. 1170–1182, 1987.

[36] G. Siglidis, G. Nikolentzos, S. Limnios, C. Giatsidis, K. Skianis, and M. Vazirgiannis, "Grakel: A graph kernel library in python," *arXiv:1806.02193*, 2018.

[37] C.-C. Chang and C.-J. Lin, "LIBSVM: A library for support vector machines," *ACM Trans. Intell. Syst. Technol.*, vol. 2, no. 3, 2011, Art. no. 27.

[38] 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," *J. Chemical Inf. Comput. Sci.*, vol. 43, no. 6, pp. 1906–1915, 2003.

[39] N. Wale, I. A. Watson, and G. Karypis, "Comparison of descriptor spaces for chemical compound retrieval and classification," *Knowl. Inf. Syst.*, vol. 14, no. 3, pp. 347–375, 2008.

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

[41] K. Riesen and H. Bunke, "Iam graph database repository for graph based pattern recognition and machine learning," in *SPR and SSPR*. Berlin, Germany: Springer, 2008, pp. 287–297.

[42] S. Pan, J. Wu, X. Zhu, G. Long, and C. Zhang, "Task sensitive feature exploration and learning for multitask graph classification," *IEEE Trans. Cybernetics*, vol. 47, no. 3, pp. 744–758, 2017.

[43] G. Sabidussi, "The centrality index of a graph," *Psychometrika*, vol. 31, no. 4, pp. 581–603, 1966.

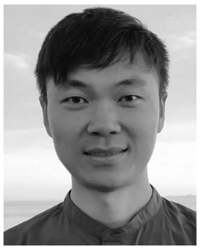
[44] L. C. Freeman, "A set of measures of centrality based on betweenness," *Sociometry*, vol. 40, no. 1, pp. 35–41, 1977.

[45] W. Liu, J. Wang, S. Kumar, and S.-F. Chang, "Hashing with graphs," in *Proc. 28th Int. Conf. Mach. Learn.*, pp. 1–8, 2011.

[46] Y. Weiss, A. Torralba, and R. Fergus, "Spectral hashing," in *Proc. 21st Int. Conf. Neural Inf. Process. Syst.*, 2009, pp. 1753–1760.

[47] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *Proc. 20th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2014, pp. 701–710.

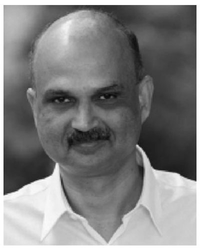
[48] A. Gretton, K. M. Borgwardt, M. J. Rasch, B. Schölkopf, and A. Smola, "A kernel two-sample test," *J. Mach. Learn. Res.*, vol. 13, no. Mar, pp. 723–773, 2012.



Wei Ye received the PhD degree in computer science from Institut für Informatik, Ludwig-Maximilians-Universität München, Munich, Germany, in 2018. He is currently a postdoctoral researcher with the DYNAMO lab at the University of California, Santa Barbara. Before joining the DYNAMO lab, he worked as a researcher in the Department of AI Platform, Tencent, China. His research interests include graph-based machine learning and their applications, network interactions, and dynamic networks.

Zhen Wang received the BEng degree in electronic information engineering from the University of Electronic Science and Technology of China, Chengdu, China, and the BEng degree with first class honour in electronics and electrical engineering from the University of Glasgow, Scotland, United Kingdom, both in 2018. He is currently working towards the MSc degree at Columbia University. His research interests include graph mining and statistical machine learning.

Rachel Redberg received the BA degree in applied mathematics from the University of California, Berkeley, in 2015. She is currently working toward the PhD degree in computer science at the University of California, Santa Barbara. Her research interests include network analysis and applications of network science to biological systems.



Ambuj Singh is a professor of computer science at the University of California, Santa Barbara (UCSB). He joined UCSB's Computer Science Department after his PhD degree from the University of Texas at Austin, in 1989. He has written more than 180 technical papers in the areas of distributed computing, databases, and bioinformatics. He is currently on the editorial boards of three journals, and has served on program committees of several conferences, workshops and international meetings. His current research interests include network science, data mining, machine learning, bioinformatics, graph querying, and mining.