Multiscale Wasserstein Shortest-Path Graph Kernels for Graph Classification

Wei Ye, Hao Tian, and Qijun Chen, Senior Member, IEEE

Abstract—Graph kernels are conventional methods for computing graph similarities. However, the existing R-convolution graph kernels cannot resolve both of the two challenges: 1) comparing graphs at multiple different scales; and 2) considering the distributions of substructures when computing the kernel matrix. These two challenges limit their performances. To mitigate both of the two challenges, we propose a novel graph kernel called the multiscale Wasserstein shortest-path graph kernel (MWSP), at the heart of which is the multiscale shortest-path node feature map, of which each element denotes the number of occurrences of the shortest path around a node. The shortest path is represented by the concatenation of all the labels of nodes in it. Since the shortest-path node feature map can only compare graphs at local scales, we incorporate into it the multiple different scales of the graph structure, which are captured by the truncated BFS trees of different depths rooted at each node in a graph. We use the Wasserstein distance to compute the similarity between the multiscale shortest-path node feature maps of two graphs, considering the distributions of shortest paths. We empirically validate MWSP on various benchmark graph datasets and demonstrate that it achieves state-of-the-art performance on most datasets.

Impact Statement—Many real-world graphs exhibit the small-world property, which means the average shortest-path length is small. The shortest-path graph kernel (SP) is suitable for computing the similarities between such graphs. However, it represents the shortest path as a triplet that contains the labels of the source and sink nodes and the length of this shortest path as components. This simple representation does not take the labels of the other constituent nodes in the shortest path into account and thus loses information. In addition, SP compares graphs without considering the multiple different scales of the graph structure which is common in real-world graphs. Moreover, the sum aggregation of the shortest path similarities in SP ignores valuable information such as the distribution of the shortest paths. The MWSP introduced in this article overcomes these challenges and generates improved results on various graph benchmark datasets, making it quite competitive on the graph classification task.

Index Terms—Graph kernels, multiple different scales, shortest path, Wasserstein distance.

I. INTRODUCTION

GRAPH-STRUCTURED data are ubiquitous in many scenarios, such as protein–protein interaction networks, social networks, citation networks, and cyber-physical systems. One interesting and fundamental problem with graph-structured data is quantifying their similarities, which can be measured by graph kernels. Most of the graph kernels belong to the family of the R-convolution kernels [1] that decompose graphs into substructures and compare them. There are many kinds of substructures used in graph kernels, such as walks [2], [3], [4], paths [5], [6], subgraphs [7], [8], [9], [10], and subtree patterns [11], [12], [13], [14], [15], [16].

The existing R-convolution graph kernels cannot resolve both of the two challenges. First, when computing the similarity between two graphs, many of them count the number of occurrences of each unique substructure in each graph and aggregate the occurrences. Simple aggregation operation such as sum aggregation neglects the distributions of substructures in each graph, which might restrict their ability to capture complex relationships between substructures. Second, many of them cannot compare graphs at multiple different scales. For example, Weisfeiler–Lehman subtree kernel (WL) [13], [14] computes graph similarity at coarse scales, because subtrees only consider the whole neighborhood structures of nodes rather than single edges between two nodes; shortest-path graph kernel (SP) [5] computes the graph similarity at fine scales because shortest paths do not take neighborhood structures into account. In many real-world scenarios, graphs such as social networks have structures at multiple different scales. Fig. 1 shows an online discussion thread on Reddit. Nodes represent users and edges represent that at least one of the connected two users responds to the other’s comment. We can see that this graph contains structures at multiple different scales, such as chains, rings, and stars. Graph kernels should not only differentiate the overall shape of graphs but also their small substructures.

Recently, some graph kernels [10], [16] have been proposed, only solving one of these two challenges. For example, Wasserstein Weisfeiler-Lehman (WWL) [16] adopts the Weisfeiler–Lehman test of graph isomorphism [17] to relabel each node. The new label of a node corresponds to the subtree pattern

1https://www.reddit.com/r/AskReddit

Manuscript received 17 July 2023; revised 6 October 2023; accepted 13 November 2023. Date of publication 20 November 2023; date of current version 21 June 2024. This work was supported in part by the National Natural Science Foundation of China under Grant 61796184, in part by the National Key Research and Development Program of China under Grant 2020AA0108100, and in part by the Fundamental Research Funds for the Central Universities of China. This article was recommended for publication by Associate Editor Melike Sah Direkoglu upon evaluation of the reviewers’ comments. (Corresponding author: Wei Ye.)

The authors are with the College of Electronic and Information Engineering, Tongji University, Shanghai 201804, China (e-mail: yew@tongji.edu.cn; 213306@tongji.edu.cn; qjchen@tongji.edu.cn).

This article has supplementary downloadable material available at https://doi.org/10.1109/TAL.2023.3333830, provided by the authors. Digital Object Identifier 10.1109/TAL.2023.3333830.
of depth $d$. To mitigate the two challenges, we propose a multiscale shortest-path graph/node feature map, capturing the multiple different scales of the graph structure. The shortest-path graph/node feature map is constructed by counting the number of occurrences of shortest paths in a graph or around a node. As aforementioned, SP cannot compare graphs at multiple different scales. To integrate the multiple different scales of the graph structure into the shortest-path graph/node feature map, we allow components in the shortest path to be BFS trees of different depths. To represent such kind of shortest paths, we project each BFS tree into an integer label by an injective hash method, i.e., hashing the same BFS trees to the same integer label and different BFS trees to different integer labels. Thus, the representation of such kind of shortest paths is just the concatenation of the integer labels of all BFS trees in this shortest path. We concatenate the multiple different scales of shortest-path graph/node feature maps and adopt the Wasserstein distance to compute graph similarity, capturing the distributions of shortest paths.

Our main contributions can be summarized as follows:

1) We build a truncated BFS tree of limited depth rooted at each node to restrict the maximal length of the shortest paths in a graph, guided by the small-world property. Each shortest path is represented by the concatenation of all its node labels rather than the simple triplet representation, considering all information (node labels) in this shortest path.

2) We propose the multiscale shortest-path graph/node feature map that incorporates the different scales of the graph structure into the shortest-path graph/node feature map and can compare graphs at multiple different scales. The different scales of the graph structure are captured by the truncated BFS trees of varying depths rooted at each node.

3) We develop a graph kernel called multiscale Wasserstein shortest-path graph kernel (MWSP) that employs the Wasserstein distance to compute the similarity between the multiscale shortest-path node feature maps of two graphs, capturing the distributions of shortest paths.

4) We show in the experiments that our graph kernel MWSP achieves the best classification accuracy on most benchmark datasets.

The rest of the article is organized as follows: We discuss related work in the following section. Section III describes the preliminaries. Section IV introduces our approach, including the shortest-path graph/node feature map, multiscale shortest-path graph/node feature map, and our graph kernel MWSP. Section V compares MWSP with other graph kernels for graph classification on the benchmark datasets. Section VI concludes the article.

II. RELATED WORK

Graph kernels can be categorized by the substructures they use, e.g., graph kernels based on random walks and paths, graph kernels based on subgraphs, and graph kernels based on subtree patterns.

Random walk graph kernels [2], [3] decompose graphs into their substructures random walks and compute the number of matching random walks from two graphs, which can be
transformed to perform random walks on the direct product graph of two input graphs. Gärtner et al. [2] introduce the concept of direct product graph and derive two closed-form random walk graph kernels, which are called the geometric kernel and the exponential kernel, respectively. The former kernel is based on the personalized PageRank diffusion [20] while the latter kernel is based on the heat kernel diffusion [21]. The random walk has closed-form expressions but its time complexity is at least $O(|V|^3)$ [3], where $|V|$ is the number of nodes. RetGK [4] proposes a graph kernel that is applicable to both attributed and nonattributed graphs. Each graph is represented by a multidimensional tensor that is composed of the return probability of short random walks starting from and ending in the same node, and the vectors of random walks starting from and ending in different nodes. The multidimensional tensor is embedded into the Hilbert space by maximum mean discrepancy (MMD) [22] and the random walk graph kernels are computed. However, RetGK may face the halting problem [23], i.e., the return probability of short random walks dominates that of the long random walks, which leads to that graph similarity depends on the short random walk similarity. Borgwardt and Kriegel [5] propose the SP that decomposes graphs into shortest paths between each pair of nodes in graphs. SP counts the number of pairs of matching shortest paths that have the same triplet representation in two graphs. Just as described in Section I, the triplet representation of the shortest path used in SP may lose information and SP cannot compare graphs at multiple different scales.

The graphlet kernel (GK) [7] decomposes graphs into subgraphs called graphlets [24]. Nodes in graphlet may be not fully connected. The isolated nodes function as noises, which decreases GK’s performance. The number of nodes a graphlet can have is restricted to five for the efficiency of enumeration. Even by this restriction, exhaustive enumeration of all graphlets in a graph is still prohibitively expensive. Thus, Shervashidze et al. [7] propose to speed up the enumeration by using the method of random sampling. Neighborhood subgraph pairwise distance kernel (NSPDK) [8] decomposes a graph into all pairs of neighborhood subgraphs of small radius at increasing distances. To speed up the kernel matrix computation, NSPDK uses a fast graph invariant string encoding for the pairs of neighborhood subgraphs. This graph invariant string encoding does not manage the hash collision problem and thus is not injective. Two nodes with different labels and local graph structures may be hashed to the same new label, reducing the accuracy of graph similarity. MLG [10] can compare graphs at multiple different scales of the graph structure. To solve the graph invariance problem, the FLG kernel is developed. Then, each node is associated with a subgraph centered around it, and the FLG kernel between each pair of these subgraphs is computed. To compute graph similarity at multiple different scales, the size of subgraphs is increased and the FLG kernel is recursively applied to these larger subgraphs. However, MLG does not consider the distributions of subgraphs when computing graph kernels.

WL [13], [14] augments each node using the first-order Weisfeiler–Lehman test of graph isomorphism [17]. It concatenates the labels of a node and all its neighboring nodes into a string and hashes the concatenated labels into new labels, aggregating neighborhood information around a node. The node label augmentation in WL is referred to as color refinement. Colors (labels) diverge too fast, which causes graph similarity too coarse. Gradual Weisfeiler-Leman (GWL) [25] is proposed to slow the color divergence, using node neighborhood clustering for augmentation. However, this augmentation method is not injective. Weisfeiler-Leman graph alignment kernel (GAWL) [26] first aligns nodes in different graphs if these two nodes have the same labels assigned by the first-order Weisfeiler–Lehman test of graph isomorphism. The adjacency matrix of each graph is then reordered by the permutation matrix generated from the alignment. Graph similarity is computed by applying a linear kernel on the aligned adjacency matrices. WWL [16] extends WL to graphs with continuous attributes by considering each graph as a set of node embeddings and using the Wasserstein distance to compute the similarity between the two sets of node embeddings. Each node is associated with two kinds of features, one of which is the subtree feature map generated by WL and the other of which is continuous attributes. The continuous attributes of each node are propagated to its neighboring nodes by an average aggregation operator, which is similar to the message-passing mechanism of graph neural networks [27], [28], [29], [30]. Optimal assignment kernel (OA) [31] first generates graph hierarchies and then computes the optimal assignment kernels, which are guaranteed positive semidefinite. The optimal assignment kernels are integrated into the WL to improve its performance. Weisfeiler-Lehman Filter kernel (FWL) [32] is based on a filtration kernel that compares Weisfeiler–Lehman subtree feature occurrence distributions (filtration histograms) over sequences of nested subgraphs, which are generated by sequentially considering all the edges whose weights are less than a changing threshold. This strategy not only provides graph features on different levels of granularities but also considers the life span of graph features. FWL’s expressiveness is more powerful than the ordinary WL. Since the subtree pattern used in these six graph kernels is not aware of single edges, all of them cannot compare graphs at fine scales.

Recently, researchers have tried to go beyond the R-convolution graph kernels. DGK utilizes word2vec [33] to learn latent representations for substructures, such as graphlets, shortest paths, and subtree patterns. The similarity matrix between substructures is computed and integrated into the kernel matrix. If the number of substructures is large, the time complexity of computing the pairwise substructure similarity is high. Nikolentzos et al. [34] use the eigenvectors of the graph adjacency matrices as graph embeddings and propose two graph kernels to compute the similarity between two graph embeddings. The first graph kernel uses the Earth Mover’s Distance [35] to compute the distance between two graph embeddings; and the second one uses the Pyramid Match kernel [36], [37] to find an approximate correspondence between two graph embeddings. However, eigenvectors cannot capture the multiple different scales of the graph structure. HTAK [38] transitively aligns the vertices between graphs through a family of hierarchical prototype graphs, which are generated by the k-means clustering method. The number of cluster centroids needs to be preassigned and it is hard to set properly for different graphs.
(GNTK) [39] is inspired by the connections between overparameterized neural networks and kernel methods [40], [41]. It is equivalent to an infinitely wide GNN trained by gradient descent. GNTK is proven to learn a class of smooth functions on graphs. GraphQNTK [42] is an extension of GNTK by introducing the attention mechanism and quantum computing into GNTK’s structure and computation. Both GNTK and GraphQNTK are more powerful than R-convolution graph kernels because of the high representation power of neural networks. Nevertheless, they may be overparameterized and overfitting on small graphs.

III. PRELIMINARIES

A. Notations

In this work, the row vector is denoted by boldface lowercase letter (e.g., \( \mathbf{x} = [x_1, \ldots, x_n] \)). We consider an undirected labeled graph \( G = (\mathcal{V}, \mathcal{E}, \ell) \), where \( \mathcal{V} \) is a set of nodes of size \( |\mathcal{V}| \), \( \mathcal{E} \) is a set of graph edges of size \( |\mathcal{E}| \), and \( \ell : \mathcal{V} \to \Sigma \) is a function that assigns labels from a set of positive integers \( \Sigma \) to nodes. Without loss of generality, \( |\Sigma| \leq |\mathcal{V}| \). An edge \( e \) is denoted by two nodes \( u, v \) that are connected to it. The depth of a BFS tree is the maximal length of the shortest paths between the root and any other node in the BFS tree.

Let \( \mathcal{X} \) be a nonempty set and let \( \kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R} \) be a function on \( \mathcal{X} \). If there is a real Hilbert space \( \mathcal{H} \) and a mapping \( \phi : \mathcal{X} \to \mathcal{H} \) satisfying that \( \kappa(x, y) = \langle \phi(x), \phi(y) \rangle \) for all \( x, y \) in \( \mathcal{X} \), where \( \langle \cdot, \cdot \rangle \) denotes the inner product of \( \mathcal{H} \), then \( \kappa \) is a kernel on \( \mathcal{X} \). \( \phi \) is called a feature map and \( \mathcal{H} \) is called a feature space. \( \kappa \) is a symmetric and positive-semidefinite kernel, which can be inputted into support vector machines (SVM) for classification. Similarly for graphs, let \( \phi(G) \) be a mapping from a graph to a vector, each element of which denotes the frequency of the substructures (such as shortest path) in graph \( G \). Then, the kernel on two graphs \( G_1 \) and \( G_2 \) is defined as \( \kappa(G_1, G_2) = \langle \phi(G_1), \phi(G_2) \rangle \), which denotes the sum of substructure similarities. R-convolution graph kernels use such kind of aggregation of substructure similarities, ignoring the distribution of substructures. In this article, we adopt the Wasserstein distance to better compute the graph similarity, capturing the distribution of the substructures.

B. Wasserstein Distance

The Wasserstein distance is a distance function defined between probability distributions on a given metric space \( M \). Let \( \mu \) and \( \nu \) be two probability distributions on \( M \) and \( g \) be a ground distance such as the Euclidean distance. The \( p \)th \((p \geq 1)\) Wasserstein distance between \( \mu \) and \( \nu \) is defined as follows:

\[
W_p(\mu, \nu) := \left( \inf_{\gamma \in \Pi(\mu, \nu)} \int_{M \times M} g(x, y)^p d\gamma(x, y) \right)^{1/p}
\]

(1)

where \( \Gamma(\mu, \nu) \) denotes the collection of all probability distributions on \( M \times M \) with marginal distributions \( \mu \) and \( \nu \), respectively.

The one-dimensional \((p = 1)\) Wasserstein distance is equivalent to the Earth Mover’s Distance [35]. Intuitively, if \( \mu \) and \( \nu \) are viewed as a unit amount of earth piled on \( M \), \( W_1(\mu, \nu) \) is the minimum cost of turning one pile into the other, which is assumed to be the amount of earth moved times the distance it has to be moved. In this article, each graph is represented as a finite set of node feature maps. Thus, we use the one-dimensional Wasserstein distance because it supports the distance computation between two sets of vectors. We use sum instead of integral in the computation of \( W_1(\mu, \nu) \), which can be formulated and solved as an optimal transport problem [43]

\[
\min \sum_{i=1}^{l} \sum_{j=1}^{2} f_{i,j} g(x_i, y_j)
\]

subject to

\[
\begin{align*}
& \sum_{i=1}^{l} \sum_{j=1}^{2} f_{i,j} = \frac{1}{|\mathcal{V}_1|} \forall j \in \{1, \ldots, |\mathcal{V}_2|\} \\
& \sum_{j=1}^{2} f_{i,j} = \frac{1}{|\mathcal{V}_2|} \forall i \in \{1, \ldots, |\mathcal{V}_1|\} \\
& f_{i,j} \geq 0 \forall i \in \{1, \ldots, |\mathcal{V}_1|\}, \forall j \in \{1, \ldots, |\mathcal{V}_2|\}
\end{align*}
\]

(2)

where \( |\mathcal{V}_1| \) and \( |\mathcal{V}_2| \) stand for the number of nodes in graphs \( G_1 \) and \( G_2 \), respectively. \( x_j \) is the feature map of node \( v_j \) in graph \( G_1 \) and \( y_j \) is the feature map of node \( u_j \) in graph \( G_2 \). \( f_{i,j} \) is the \((i, j)\)th element of the flow matrix \( F \in \mathbb{R}^{ |\mathcal{V}_1| \times |\mathcal{V}_2| } \), \( g(x_i, y_j) = ||x_i - y_j||_2 \) denotes the Euclidean distance between two node feature maps.

After finding the optimal flow matrix \( F \) that minimizes the above objective function, \( W_1(X, Y) \) between the node feature maps of graphs \( G_1 \) and \( G_2 \) can be computed as follows:

\[
W_1(X, Y) = \frac{\sum_{i=1}^{l=1} \sum_{j=1}^{2} f_{i,j} g(x_i, y_j)}{\sum_{i=1}^{l=1} \sum_{j=1}^{2} f_{i,j}}.
\]

(3)

IV. MULTISCALE WASSERSTEIN SHORTEST-PATH GRAPH KERNELS

A. Shortest-Path Graph Feature Map

The SP [5] uses a triplet that contains the labels of the source and sink nodes and the length of the shortest path as its representation. This coarse representation loses the information of the intermediate nodes. To alleviate this situation, we consider all the information of nodes in the shortest path. We first describe the representation of the shortest path used in our graph kernel as follows.

Definition 1: (Shortest-path representation). Given an undirected and labeled graph \( G = (\mathcal{V}, \mathcal{E}, \ell) \), rooted at each node \( v \in \mathcal{V} \), we construct a truncated BFS tree \( T_v = (\mathcal{V}', \mathcal{E}', \ell) \) \((\mathcal{V}' \subseteq \mathcal{V} \) and \( \mathcal{E}' \subseteq \mathcal{E} \) of depth \( d \). For every node \( v' \) in this BFS tree, the shortest path is constructed by concatenating all the distinct nodes and edges from the root \( v \) to \( v' \), i.e., \( P = \langle v, v_1, v_2, \ldots, v' \rangle \). The shortest-path representation is the concatenated labels of all the nodes in this shortest path, i.e., \( l(P) = l(v), l(v_1), l(v_2), \ldots, l(v') \).

Guided by the small-world property, we construct a truncated BFS tree of depth \( d \) rooted at each node to restrict the maximal length of the shortest path in a graph. Fig. 2(a) and 2(b) shows two undirected labeled graphs \( G_1 \) and \( G_2 \), respectively. Fig. 2(c)
and 2(d) shows two truncated BFS trees of depth \( d = 1 \) rooted at the nodes with label 4 in \( G_1 \) and \( G_2 \), respectively. All the shortest paths starting from the root of the BFS tree in Fig. 2(c) can be represented by “4,” “4,1,” “4,3,” and “4,3” and those of the root of the BFS tree in Fig. 2(d) can be represented by “4,” “4,1,” “4,2,” “4,3,” and “4,3.” Note that we also consider the shortest path of length zero, which only contains one node that plays the roles of both source and sink node. We only consider the shortest paths in BFS trees instead of finding all the shortest paths in a graph as used in SP, considering the small-world property. For a set of graphs, we construct a truncated BFS tree of depth \( d \) rooted at each node and generate all its shortest paths, whose representations are kept in a multiset\(^2\) \( M \).

We define the shortest-path graph feature map as follows.

**Definition 2**: (Shortest-path graph feature map). Let \( \mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_n \) be the multisets of shortest-path representations extracted from graphs \( G_1, G_2, \ldots, G_n \), respectively. Let the union of all the multisets be a set \( \mathcal{U} = \mathcal{M}_1 \cup \mathcal{M}_2 \cup \cdots \cup \mathcal{M}_n = \{(l(P_1), l(P_2), \ldots, l(P_{|T|})\} \). Define a map \( \psi : \{(G_1, G_2, \ldots, G_n) \times \Sigma \rightarrow \mathbb{N} \) such that \( \psi(G_i, l(P_j)) \) (1 \( \leq i \leq n, 1 \leq j \leq |T|) \) is the frequency of the shortest path representation \( l(P_j) \) in graph \( G_i \). Then, the shortest-path graph feature map of \( G_i \) is defined as follows:

\[
\phi(G_i) = [\psi(G_i, l(P_1)), \psi(G_i, l(P_2)), \ldots, \psi(G_i, l(P_{|T|})].
\]

**B. Multiscale Shortest-Path Graph Feature Map**

In the last section, we decompose a graph into its substructures, i.e., shortest paths. However, shortest paths cannot reveal the structural information of nodes. Thus, the shortest-path graph feature map can only capture graph similarity at fine scales. To capture graph similarity at multiple different scales, we focus on the different scales of the graph structure around a node.

To incorporate the multiple different scales of the graph structure into the shortest-path graph feature map, we first use a node. The difficulty is how to label each truncated BFS tree in the multiple different scales of the graph structure around a node to augment this node and then generate all the shortest paths starting from this node. Specifically, for each node in the shortest path \( P = \langle v, v_1, v_2, \ldots, v' \rangle \), we construct a truncated BFS tree of depth \( k \) rooted at it, which captures the graph structure at the \( k \)th scale around this node. And we have the shortest path \( P \) that goes through a sequence of BFS trees \( P = \langle T_0, T_{v_1}, T_{v_2}, \ldots, T_{v'} \rangle \). Note that compared with the normal shortest path whose components are nodes, the shortest path in this work can contain BFS trees as its components.

Our next step is to represent such kinds of shortest paths. The difficulty is how to label each truncated BFS tree in the shortest path. Thus, we need to redefine the definition of the label function \( l \) described in Section III-A as follows: \( l : T \rightarrow \Sigma \) is a function that assigns labels from a set of positive integers \( \Sigma \) to BFS trees. Thus, we can just concatenate all the labels of BFS trees in the shortest path as its representation, i.e., \( l(P) = \langle l(T_0), l(T_{v_1}), l(T_{v_2}), \ldots, l(T_{v'}) \rangle \).

For each truncated BFS tree in the shortest path, we need to hash it to an integer label. The hash method should be injective, i.e., hashing the same two BFS trees to the same integer label and different BFS trees to different integer labels. In this article, we propose a method for this purpose. We use Fig. 3(f) as an example, which shows a truncated BFS tree of depth two rooted at the node with label 4 in graph \( G_1 \). We recursively consider each subtree of depth one in this truncated BFS tree. First, we consider the subtree of depth one rooted at the node with label 4 (also shown in Fig. 2c). We sort the child nodes in ascending order with regard to their label values and use the concatenation of all the edges in the subtree as its representation. Thus, this subtree can be represented by a string “4,1, 4,3, 4,3.” Likewise, the middle subtree of depth one rooted at the node with label 3 in Fig. 3(f) can be represented by a string “3,1” and the right

---

\(^2\)A set that can contain the same element multiple times.
subtree of depth one rooted at the node with label 3 in Fig. 3(f) can be represented by a string “3,2.” Note that the roots of the latter two subtrees of depth one have the same label, which leads to arbitrariness in node order. To eliminate arbitrariness in node order, we sort the subtrees of depth one rooted at the nodes with the same label lexicographically according to their string representations. Finally, the subtree $T_6^{(1)}$ as shown in Fig. 3(f) can be represented by a string “4,1, 4,3, 4, 3-3, 1-,” which is the concatenation of the representations of all the subtrees of depth one in it. We use the dash symbol “-” to differentiate between different subtrees. This dash symbol is necessary to differentiate $T_6^{(1)}$ between the case in which the node with label 2 is attached to the root of the middle subtree of depth one rooted at the node with label 3. Our method is injective and makes the representation of each truncated BFS tree unique. Now, the label function $l: T \to \Sigma$ can assign the same positive integer label to the same trees, which have the same representation.

All unique BFS trees of depth $k$ rooted at each node in a dataset of graphs are kept in a set. For example, for BFS trees shown in Figs. 3 and 4, the set will contain $T_1^{(2)}, T_2^{(2)}, T_3^{(2)}, T_4^{(1)}, T_5^{(1)}, T_6^{(1)}, T_7^{(1)}, T_8^{(1)}, T_9^{(1)}, T_{10}^{(1)}, T_{11}^{(1)}, T_{12}^{(1)}, T_{13}^{(1)}, T_{14}^{(1)}, T_{15}^{(1)}, T_{16}^{(1)}, T_{17}^{(1)}, T_{18}^{(1)}, T_{19}^{(1)}, T_{20}^{(1)}, T_{21}^{(1)}, T_{22}^{(1)}, T_{23}^{(1)}, T_{24}^{(1)}, T_{25}^{(1)}, T_{26}^{(1)}, T_{27}^{(1)}, T_{28}^{(1)}$, and $T_6^{(1)}$. All unique BFS trees in the set are sorted lexicographically. The index of each truncated BFS tree in the set can be used as its label. For instance, $l: T_1^{(2)} \to 1, l: T_2^{(2)} \to 2, l: T_3^{(1)} \to 3, l: T_4^{(1)} \to 4, l: T_5^{(3)} \to 5, l: T_6^{(3)} \to 6, l: T_7^{(1)} \to 7, l: T_8^{(1)} \to 8, l: T_9^{(2)} \to 9, l: T_6^{(2)} \to 10, l: T_6^{(1)} \to 11$. After considering the graph structure at the second scale around node 4 in $G_2$, the shortest paths starting from node 4 are “$T_6^{(1)}$, “$T_6^{(1)}$, “$T_1^{(1)}$, “$T_6^{(1)}$, “$T_4^{(1)}”, and “$T_6^{(1)}$, “$T_6^{(1)}”. Their corresponding representations are “11,” “11,4,” “11,8,” and “11,7,” respectively.

Note that if the truncated BFS trees of the same depth rooted at two nodes are identical, the two nodes have the same scale of the graph structure around them. For example, the two nodes with label 3 on the upper side of the node with label 4 in Fig. 2(a) and on the left side of the node with label 4 in Fig. 2(b) have the same BFS trees of depth two as shown in Figs. 3(d) and 4(e), respectively. Thus, they have the same second scale of the graph structure. Fig. 3(a) and 3(b) shows another two truncated BFS trees rooted at the two nodes with the same label 1 in Fig. 2(a). We can see that they have a different second scale of the graph structure. Thus, by integrating the multiple different scales of the graph structure into the shortest paths, we can distinguish graphs at multiple different scales. If we build truncated BFS trees of depth zero rooted at each node for augmenting nodes, the two shortest paths that have the same representation “1, 4, 3” in Fig. 2(a) (The starting node is the node with label 1 on the bottom side of the node with label 4 and the ending node is the node with label 3 on the upper side of the node with label 4.) cannot be distinguished. However, if we augment nodes using truncated BFS trees of depth two (as shown in Fig. 3(a), 3(f), and 3(d) and Fig. 4(b), 4(f), and 4(e), respectively), the aforementioned two shortest paths have different representations “4, 11, 8” and “2, 10, 8,” which can be distinguished easily.

In this article, we concatenate all the shortest-path graph feature maps at multiple different scales as follows:

$$\phi(G) = \phi(G^{(0)}) \| \phi(G^{(1)}) \| \cdots \| \phi(G^{(k)})$$

where $\phi(G^{(k)})$ corresponds to the shortest-path graph feature map built on the shortest paths with truncated BFS trees of depth $k$ rooted at each node as its components.

### C. Computing Graph Kernels

We define the shortest-path node feature map as follows.

**Definition 3:** (Shortest-path node feature map). For nodes in graph $G$, define a map $\psi: \{v_1, v_2, \ldots, v_{|V|}\} \times \Sigma \to N$ where $v_i \in \forall (1 \leq i \leq |V|)$ such that $\psi(v_i, l(P_j))$ (1 $\leq i \leq |V|, 1 \leq j \leq \ell(l)$ is the frequency of the shortest path $l(P_j)$ that starts from $v_i$ in graph $G$. Then, the shortest-path node feature map of $v_i$ is defined as follows:

$$\phi(v_i) = \left[\psi(v_i, l(P_1)), \psi(v_i, l(P_2)), \ldots, \psi(v_i, l(P_{|P|}))\right].$$

Similar to (5), we generate the multiscale shortest-path node feature map as follows:

$$\phi(v_i) = \phi(v_i^{(0)}) \| \phi(v_i^{(1)}) \| \cdots \| \phi(v_i^{(k)})$$

where $\phi(v_i^{(k)})$ is the shortest-path node feature map built on $G^{(k)}$.

We can see that the multiscale shortest-path graph feature map equals the sum of all the multiscale shortest-path node feature maps in that graph, i.e.,

$$\phi(G) = \sum_{i=1}^{N} \phi(v_i).$$

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
Algorithm 1: MWSP

**Input:** A set of graphs \( \mathcal{G} = \{G_1, G_2, \ldots, G_n\} \) and the initial label function \( l : V \rightarrow \Sigma, d, k, \lambda \)

**Output:** The computed kernel matrix \( K \in \mathbb{R}^{n \times n} \)

1. \( K \leftarrow \text{zeros}(n, n) \);
2. all_graph_paths \( \leftarrow \text{Shortest-path_Generation} \) \((G, d)\);
3. all_labels \( \leftarrow \text{Graph_Augmentation} \) \((G, l, k)\);
4. for \( i \leftarrow 0 \) to \( k \) do
5. \( U^{(i)} \leftarrow \text{set}() \), \( l^{(i)} \leftarrow \text{all_labels}[i] \);
6. for \( j \leftarrow 1 \) to \( n \) do
7. graph_paths \( \leftarrow \text{all_graph_paths}[j] \);
8. for each node_path \( P \in \text{graph_path} \) do
9. \( U^{(i)} \text{.add}(l^{(i)}(P)) \);
10. \( U^{(i)} \leftarrow \text{sort}(U^{(i)}) \); /* sort lexicographically */
11. for \( j \leftarrow 1 \) to \( n \) do
12. for each node \( v \in \text{graph} \ G_j \) do
13. \( \Phi^{(i)} \leftarrow [\phi(v_1), \phi(v_2), \ldots \phi(v_{|V|})] \); /* row-wise concatenation */
14. \( \Phi_j \leftarrow [\Phi_0^{(i)} \| \Phi_1^{(i)} \| \cdots \| \Phi_n^{(i)}] \); /* column-wise concatenation */
15. for each pair of graphs \( G_i \) and \( G_j \) do
16. \( K(i, j) \leftarrow \exp(-\lambda W_1(\Phi_i, \Phi_j)) \)
17. return \( K \);

In this article, we use the multiscale shortest-path node feature maps rather than their sum aggregation, i.e., the multiscale shortest-path graph feature map, because the sum aggregation loses the distributions of shortest paths around each node. We first use (3) to compute the distance between the multiscale shortest-path node feature maps of two graphs \( G_1 \) and \( G_2 \) and then use the Laplacian radial basis function (RBF) kernel to construct our graph kernel MWSP

\[
\kappa(G_1, G_2) = \exp(-\lambda W_1(X, Y))
\]  

where \( \lambda \) is a hyperparameter and \( X \) and \( Y \) denote the matrix of the multiscale shortest-path node feature maps of graphs \( G_1 \) and \( G_2 \), respectively.

Note that the Wasserstein distance, despite being a metric, in its general form is not isometric (i.e., no metric-preserving mapping to the \( l_2 \) norm) as argued in [16], owing to that the metric space induced by it strongly depends on the chosen ground distance [44]. Thus, we cannot derive a positive semidefinite kernel from the Wasserstein distance in its general form, because the conventional methods [45] cannot be applied to it. Thus, MWSP is not a positive semidefinite kernel. However, we can still employ Krein SVM (K SVM) [46] for learning with indefinite kernels. Generally speaking, kernels that are not positive semidefinite induce reproducing kernel Krein spaces (RKKS), which is a generalization of reproducing kernel Hilbert spaces (RKHS). KSVM can solve learning problems in RKKS.

Algorithm 2: Shortest-path_Generation

**Input:** A set of graphs \( \{G_1, G_2, \ldots, G_n\} \) and the depth \( d \) of the truncated BFS tree rooted at each node

**Output:** All the shortest paths in these graphs

1. all_graph_paths \( \leftarrow [] \);
2. for \( i \leftarrow 1 \) to \( n \) do
3. graph_paths \( \leftarrow [] \);
4. for each node \( v \in G_i \) do
5. node_paths \( \leftarrow [] \);
6. Build a truncated BFS tree \( T_v \) of depth \( d \) rooted at node \( v \);
7. for each node \( v' \in T_v \) do
8. node_paths.append(P);
9. /* \( P = \{v, v_1, v_2, \ldots, v_m\} \) */
10. graph_paths.append(node_paths);
11. all_graph_paths.append(graph_paths);
12. return all_graph_paths ;

D. Algorithm

In this section, we provide the pseudocode of our MWSP graph kernel in Algorithm 1. Line 2 in Algorithm 1 uses Algorithm 2 to generate all the shortest paths in a set of graphs \( \mathcal{G} \). For each node \( v \) in graph \( G_i \) (lines 4–9 in Algorithm 2), we build a truncated BFS tree of depth \( d \) rooted at it. The time complexity of the BFS traversal of a graph is \( \mathcal{O}(|V| + |E|) \), where \( |V| \) is the number of nodes and \( |E| \) is the number of edges in a graph. For convenience, we assume that \( |E| > |V| \) and all the \( n \) graphs have the same number of nodes and edges. Thus, the time complexity of Algorithm 2 for all the \( n \) graphs is \( \mathcal{O}(n(|V| + |E|)) \). Line 3 in Algorithm 1 uses Algorithm 2 to augment/relabel each node (i.e., hashing each truncated BFS tree of depth \( k \) to an integer label, capturing the multiple different scales of the graph structure). Like Algorithm 2, for each node in each graph, Algorithm 3 also constructs a BFS tree of depth \( k \), whose time complexity is \( \mathcal{O}(kn(|V| + |E|)) \). For a total of \( kn|V| \) truncated BFS trees, we use our hash method to project them to unique integer labels, whose time complexity is bounded by \( \mathcal{O}(n|E| \log(|E|)) \). Thus, the time complexity of Algorithm 3 is \( \mathcal{O}(kn|V|(|E| + |V|) + n|E| \log(|E|)) \).

Lines 6–10 in Algorithm 1 generate all the shortest paths corresponding to the same value of \( k \) (i.e., the same scale of the graph structure). The number of shortest paths generated by Algorithm 2 is at most \( n|V|^2 \). The time complexity of lines 6–10 in Algorithm 1 is \( \mathcal{O}(n|V|^2) \) since the depth of the truncated BFS tree or the maximal length of all the shortest paths is \( d \). Line 11 in Algorithm 1 sorts the elements in \( U^{(i)} \) lexicographically. Since \( U^{(i)} \) at most contains \(|V|^2 \) shortest paths in terms of their representations. The time complexity is bounded by


Algorithm 3: Graph Augmentation

Input: A set of graphs \( \mathcal{G} = \{G_1, G_2, \ldots, G_n\} \) and the initial label function \( l : \mathcal{T} \rightarrow \Sigma \), the depth \( k \) of the truncated BFS tree rooted at each node for capturing the multiple different scales of the graph structure.

Output: The new label of each node

\[
\begin{aligned}
\text{for} & \ i \leftarrow 0 \ \text{to} \ k \ \text{do} \\
& \ l(i) \leftarrow []; \\
& \ \text{for} \ j \leftarrow 1 \ \text{to} \ n \ \text{do} \\
& \ \ \ \text{label} \leftarrow []; \\
& \ \ \ \text{foreach} \ \text{node} \ \nu \in G_i \ \text{do} \\
& \ \ \ \ \ \ \ \text{Build a truncated BFS tree} \ T_v \ \text{of depth} \ i \ \text{rooted at node} \ \nu; \\
& \ \ \ \ \ \ \ \text{label}.append(l(T_v)); \\
& \ \ \ \ l(i).append(label) \\
& \ \text{all_labels}.append(l(i)); \\
\end{aligned}
\]

\( O(d|V|^2) \) via radix sort. Lines 12–15 in Algorithm 1 count the number of occurrences of each unique shortest path in \( U^{(i)} \) around each node. For each node, the counting time complexity is bounded by \( O(q d|V|) \), where \( q \) is the maximal length of all \( U^{(i)} \) \((0 \leq i \leq k) \). Thus, the time complexity of lines 12–15 in Algorithm 1 is \( O(n q d|V|^2) \). The time complexity of lines 4–15 in Algorithm 1 is \( O(n|V|^3) \).

Lines 18–19 in Algorithm 1 first compute the Wasserstein distance between the multiscale shortest-path node feature maps of graphs \( G_i \) and \( G_j \) and then compute the Laplacian RBF kernel. The time complexity of computing the Wasserstein distance is \( O(|V|^3 \text{log}(|V|)) \) [47]. For \( n \) graphs, the time complexity to compute the kernel matrix is \( O((n^2)|V|^3 \log(|V|)) \). The total time complexity of our MWSP graph kernel is bounded by \( O(k n|V| (|E| + q d|V|)) + n|E| \text{log}(|E|) + n^2|V|^3 \text{log}(|V|)) \).

V. EXPERIMENTAL EVALUATION

A. Experimental Setup

We run all the experiments on a server with a dual-core Intel(R) Xeon(R) Gold 6226R CPU @ 2.90GHz, 256 GB memory, and Ubuntu 18.04.6 LTS operating system. MWSP is written in Python. We make our code publicly available at Github. We compare MWSP with nine state-of-the-art graph kernels, i.e., MLG [10], RetGK [4], SP [5], WL [14], FWL [32], WWL [16], GWL [25], GAWL [26], and GraphQNTK [42].

We set the parameters for our MWSP graph kernel as follows: The depth \( d \) of the truncated BFS tree rooted at each node for constructing and limiting the maximal length of the shortest paths is chosen from 0, 1, 2, ..., 6 and the depth \( k \) of the truncated BFS tree for capturing the graph structure at multiple different scales around each node is chosen from 0, 1, 2, ..., 6 as well. Both \( d \) and \( k \) are selected by cross-validation on the training data.

MWSP achieves the best classification accuracy on 12 datasets. On the ENZYMES dataset, MWSP has a gain of 10.4% over the best comparison method RetGK and a gain of 86.3% over a specific value of \( k \) or \( d \), if MWSP runs out of memory on some dataset, we will not try larger values and terminate the search. The parameters of the comparison methods are set according to their original articles. We use the implementations of MLG, SP, and WL from the GraKeL [48] library. The implementations of other methods are obtained from their official websites.

We use tenfold cross-validation with a binary C-SVM [49] (or a K SVM [46] for MWSP, FWL, and WWL) to test the classification accuracy of each method. The parameter \( C \) for each fold is independently tuned from \( \{10^{-3}, 10^{-2}, \ldots, 10^3\} \) using the training data from that fold. We also tune the parameter \( \lambda \) from \( \{10^{-4}, 10^{-3}, \ldots, 10^1\} \) for MWSP, FWL, and WWL. We repeat the experiments ten times and report the average classification accuracy and standard deviation.

In order to test the performance of our graph kernel MWSP, we adopt 14 benchmark datasets whose statistics are given in Table I. All the datasets are downloaded from Kersting et al. [50]. Chemical compound datasets include MUTAG [51], DHFR_MD [52], NCI1 [53], and NCI109 [53]. These chemical compounds are represented by graphs, of which edges represent the chemical bond types (single, double, triple, or aromatic), nodes represent atoms, and node labels indicate atom types. Molecular compound datasets contain PTC [male mice (MM), male rats (MR), female mice (FM), and female rats (FR)] [54], ENZYMES [55], PROTEINS [55], and DD [56]. IMDB-B [57] is a movie collaboration dataset, which contains movies of different actors/actresses and genre information. Graphs represent the collaborations between different actors/actresses. For each actor/actress, a corresponding collaboration graph (ego network) is derived and labeled with its genre. Nodes represent actors/actresses and edges denote that two actors/actresses appear in the same movie. The collaboration graphs are generated on the Action and Romance genres. REDDIT-B [57] contains graphs representing online discussion threads. Nodes represent users and edges represent the responses between users. This dataset has two classes, i.e., question/answer-based community and discussion-based community. Cuneiform dataset [58] contains graphs representing 30 different Hittite cuneiform signs, each of which consists of tetrahedron-shaped markings called wedges. The visual appearance of wedges can be classified according to the stylus orientation into vertical, horizontal, and diagonal. Each type of wedge is represented by a graph consisting of four nodes, i.e., right node, left node, tail node, and depth node.

B. Results

1) Classification Accuracy: Table II demonstrates the comparison of the classification accuracy of MWSP to other graph kernels on the benchmark datasets. Our method MWSP needs node labels. If datasets (e.g., social networks) do not have node labels, we follow the method described in [59], i.e., assigning node degrees as node labels for the IMDB-B dataset and setting uniform node labels for the REDDIT-B dataset. We can see that MWSP achieves the best classification accuracy on 12 datasets. On the ENZYMES dataset, MWSP has a gain of 10.4% over the best comparison method RetGK and a gain of 86.3% over...
the worst comparison method GraphQNTK. MWSP is outperformed by RetGK on two datasets, i.e., MUTAG and REDDIT-B. But RetGK is not an R-convolution graph kernel. It uses the return probability of random walk starting from and ending in the same node with multiple hops as node features and adopts the MMD [22] to compute graph similarity. Compared with all the R-convolution graph kernels, MWSP performs the best on all the datasets. GraphQNTK is not an R-convolution graph kernel either. It introduces some advanced techniques such as quantum computing and multihead quantum attention mechanism to graph neural tangent kernels. It is also outperformed by MWSP on all the datasets.

Compared with the R-convolution graph kernels MLG, SP, WL, GWL, and GAWL, which do not solve both of the two challenges, MWSP achieves the best results on all the datasets. Like our method MWSP, both the R-convolution graph kernels WWL and FWL embed each graph into the vector space and consider each graph as a set of node embeddings. The similarity between two graphs is derived by computing the similarity between their node embeddings using the Wasserstein distance. MWSP outperforms both of them on all the datasets, which proves that the multiscale shortest-path node feature map used in MWSP captures more information than the subtree pattern used in WWL and FWL.

2) Ablation Study: In this section, we compare MWSP to its two variants, i.e., WSP and MWSP-GFM. WSP is a kind of shortest-path graph kernel that is derived by not using the multiple different scales of the graph structure in MWSP. MWSP-GFM uses the multiscale shortest-path graph feature maps rather than the multiscale shortest-path node feature maps as used in MWSP. The classification accuracy of each method is given in Table III. Compared with WSP, MWSP achieves the best results on seven out of 14 datasets. On the remaining seven datasets, MWSP and WSP have the same performance. Social networks usually have multiple different scales of the graph structure. MWSP is better than WSP on the IMDB-B dataset. For the REDDIT-B dataset, because MWSP runs out of memory under the parameter setting $k = 1$, we do not conduct experiments with $k \geq 1$. Thus, MWSP and WSP have the same classification accuracy on the REDDIT-B dataset. This also happens on the DD dataset. For the other five datasets, because they do not have obvious multiple different scales of the graph structure, MWSP and WSP perform equally well. This proves that using the multiple different scales of the graph structure

| Dataset       | Size | Class # | Average Node # | Average Edge # | Node Label # | Mean SP Length | Max SP Length |
|---------------|------|---------|----------------|----------------|--------------|----------------|---------------|
| MUTAG         | 188  | 2       | 17.95          | 19.79          | 7            | 3.87           | 15            |
| DHFR_MD       | 393  | 2       | 23.87          | 283.01         | 7            | 1.00           | 1             |
| NCI           | 4110 | 2       | 29.87          | 32.30          | 37           | 7.27           | 45            |
| NCI109        | 4127 | 2       | 29.68          | 32.13          | 38           | 7.19           | 61            |
| PTC_MM        | 336  | 2       | 13.97          | 14.32          | 20           | 5.34           | 30            |
| PTC_MR        | 344  | 2       | 14.29          | 14.69          | 18           | 5.70           | 30            |
| PTC_FM        | 349  | 2       | 14.11          | 14.48          | 18           | 5.23           | 30            |
| PTC_FR        | 351  | 2       | 14.56          | 15.00          | 19           | 5.54           | 30            |
| ENZYMES       | 600  | 6       | 32.63          | 62.14          | 3            | 5.72           | 37            |
| PROTEINS      | 1113 | 2       | 39.06          | 72.82          | 3            | 10.55          | 64            |
| DD            | 1178 | 2       | 284.32         | 715.66         | 82           | 16.84          | 83            |
| IMDB-B        | 1000 | 2       | 19.77          | 96.53          | N/A          | 1.59           | 2             |
| REDDIT-B      | 2080 | 2       | 429.63         | 497.75         | N/A          | 3.38           | 19            |
| Cuneiform     | 267  | 30      | 21.27          | 44.80          | 3            | 2.30           | 3             |

| Dataset       | MWSP | MLG  | RetGK | SP     | WL  | FWL | WWL | GWL | GAWL | GraphQNTK |
|---------------|------|------|-------|-------|-----|-----|-----|-----|-----|-----------|
| MUTAG         | 89.9 | 5.5 | 84.2 | 2.6  | 84  | 1.7 | 82  | 1.0 | 85  | 1.8       |
| DHFR_MD       | 73.5 | 5.2 | 67.1 | 0.6  | 64  | 1.0 | 68  | 0.6 | 60  | 0.5       |
| NCI           | 86.3 | 1.3 | 80.8 | 1.3  | 84  | 2.0 | 85  | 2.1 | 85  | 1.6       |
| NCI109        | 86.4 | 1.1 | 81.3 | 0.8  | 80  | 1.0 | 72  | 0.3 | 82  | 0.2       |
| PTC_MM        | 71.1 | 6.5 | 62.1 | 1.1  | 67  | 1.6 | 62  | 2.2 | 67  | 1.6       |
| PTC_MR        | 67.1 | 4.0 | 62.1 | 1.2  | 65  | 1.6 | 61  | 1.9 | 63  | 1.3       |
| PTC_FM        | 67.2 | 4.1 | 61.9 | 1.8  | 63  | 1.3 | 61  | 0.9 | 60  | 1.9       |
| PTC_FR        | 70.1 | 5.3 | 63.7 | 1.9  | 67  | 1.1 | 66  | 1.5 | 66  | 1.0       |
| ENZYMES       | 66.7 | 4.7 | 57.9 | 5.4  | 60  | 0.1 | 41  | 0.8 | 52  | 1.3       |
| PROTEINS      | 76.6 | 3.4 | 76.1 | 0.8  | 75  | 0.6 | 75  | 5.0 | 74  | 3.8       |
| DD            | 80.3 | 3.3 | 74.6 | 0.7  | 79  | 0.7 | 79  | 0.3 | 79  | 0.4       |
| IMDB-B        | 75.2 | 2.7 | 68.9 | 1.1  | 73  | 1.0 | 72  | 2.0 | 78  | 2.4       |
| REDDIT-B      | 90.8 | 1.2 | N/A  | 91.5 | 0.2 | 84  | 6.0 | 80  | 3.1 | 79  | 0.2       |
| Cuneiform     | 79.8 | 0.7 | 73.1 | 1.1  | 73  | 3.1 | 74  | 0.9 | 73  | 1.7       |
can improve performance and adaptiveness. Compared with MWSP-GFM, MWSP achieves the best results on 13 out of 14 datasets, which proves that considering the distributions of the shortest paths around each node is better. On the MUTAG dataset, MWSP-GFM has the same average classification accuracy as MWSP but a higher standard deviation than MWSP. Using multiscale shortest-path node feature maps can lead to smoother classification accuracy across the ten folds than using multiscale shortest-path graph feature maps. We can see that MWSP is better than MWSP-GFM on seven datasets in terms of the standard deviation of the classification accuracy over the ten folds. MWSP and MWSP-GFM have the same standard deviation on two datasets NCI109 and PROTEINS. Compared with SP, it is evident that all of MWSP, WSP, and MWSP-GFM perform better, which proves that our strategies to improve the performance of SP are effective.

3) Parameter Sensitivity: To analyze parameter sensitivity with regard to the two parameters $d$ and $k$ of MWSP, we compute average classification accuracy over the ten folds for all the combinations of $d$ and $k$, where $d, k \in \{0, 1, 2, \ldots, 6\}$, on datasets DHFR_MD, PTC_FM, IMDB-B, and Cuneiform. DHFR_MD is a chemical compound dataset. PTC_FM is a molecular compound dataset. IMDB-B is a social network. Cuneiform is a Hittite cuneiform signs dataset. We select these four datasets to represent their categories. Fig. 5 shows the heatmaps of the results. On datasets DHFR_MD and IMDB-B, the classification accuracy remains stable when the values of $k$ and $d$ exceed a threshold. To explain this phenomenon, we check the multiscale shortest-path feature map of each node in datasets DHFR_MD and IMDB-B and find that its dimension does not change when the values of $k$ and $d$ exceed a threshold. We can see from Table I that both the mean shortest-path length and the maximal shortest-path length of DHFR_MD equal one. A BFS tree of depth one rooted at each node can capture all the shortest-path patterns in DHFR_MD. As for the IMDB-B dataset, because the maximal shortest-path length is two, a BFS tree of depth two rooted at each node can capture all the shortest-path patterns in IMDB-B. The best parameters for these four datasets are different owing to their different statistics as shown in Table I. To sum up, the best value of $k$ or $d$ is around the mean shortest-path length of these four datasets.

4) Running Time: In this section, we report the running time needed to compute the kernel matrices of MWSP and other graph kernels on the benchmark datasets in Table IV. Since the time complexity of MWSP is linear with respect to the parameters $k$ and $d$ (please see Section IV-D), the running time of MWSP will increase linearly with regard to $k$ or $d$. For convenience, the values of $k$ and $d$ are set to zero and one, respectively. Note that GWL is written in Java and runs the fastest. All the other methods are written in Python. Since the four R-convolution graph kernels SP, WL, GWL, and GAWL do not consider the distributions of substructures in the computation of graph kernels, they also run very fast on most datasets. RetGK uses a random Fourier feature map [60] to approximate the kernel matrix computation with promising running time. MLG develops a randomized projection method, which is similar to the Nyström method, to approximate the kernel matrix computation. However, MLG runs slower than WWL. Both MWSP and WWL use the Wasserstein distance to compute the graph similarity and they have a similar running time. FWL runs faster on large datasets compared with MWSP and WWL. Although GraphQNTK adopts quantum parallelism to accelerate the kernel matrix computation, it has a worse running time compared with WWL and MWSP, especially on large datasets. Our method MWSP has a similar time complexity to WWL. One possible solution to reduce the running time of MWSP is to use the Sinkhorn method [61], [62].

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
VI. CONCLUSION

In this article, we have proposed a novel multiscale graph kernel called MWSP to deal with the two challenges of the R-convolution graph kernels, i.e., they cannot compare graphs at multiple different scales and do not consider the distributions of substructures. To mitigate the first challenge, we propose the shortest-path graph/node feature map, whose element represents the frequency of each shortest path in a graph or around a node. Since the shortest-path graph/node feature map cannot compare graphs at multiple different scales, we incorporate into it the multiple different scales of the graph structure, which are captured by the truncated BFS trees of different depths rooted at each node in a graph. To mitigate the second challenge, we adopt the Wasserstein distance to compute the similarity between the multiscale shortest-path node feature maps of two graphs. Experimental results show that MWSP outperforms the state-of-the-art graph kernels on most datasets. Since the Wasserstein distance has a high time complexity, we would like to develop an efficient distance measure to compute the similarity between the node feature maps of two graphs in the future.

REFERENCES

[1] D. Haussler, “Convolution kernels on discrete structures,” Dept. Comput. Sci., Univ. California Santa Cruz, CA, USA, Tech. Rep., 1999, [Online]. Available: http://www.cse.ucsc.edu/~haussler

[2] T. Gärtner, P. Flach, and S. Wrobel, “On graph kernels: Hardness results and efficient alternatives,” in Proc. Learn. Theory Kernel Mach., New York, NY, USA: Springer-Verlag, 2003, pp. 129–143.

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

[4] Z. Zhang, M. Wang, Y. Xiang, Y. Huang, and A. Nehorai, “ReTK: Graph kernels based on return probabilities of random walks,” in Proc. Adv. Neural Inf. Process. Syst., vol. 31, 2018, pp. 3964–3974.

[5] K. M. Borgwardt and H.-P. Kriegel, “Shortest-path kernels on graphs,” in Proc. 5th IEEE Int. Conf. Data Mining (ICDM), Piscataway, NJ, USA: IEEE Press, 2005, p. 8.

[6] W. Ye, Z. Wang, R. Redberg, and A. Singh, “Tree++: Truncated tree based graph kernels,” IEEE Trans. Knowl. Data Eng., vol. 33, no. 4, pp. 1778–1789, Apr. 2021.

[7] N. Shervashidze, S. Vishwanathan, T. Petri, K. Mehlhorn, and K. Borgwardt, “Efficient graphlet kernels for large graph comparison,” in Proc. Artif. Intell. Statist., 2009, pp. 488–495.

[8] F. Costa and K. D. Grave, “Fast neighborhood subgraph pairwise distance kernel,” in Proc. Int. Conf. Mach. Learn., Haifa, Israel: Omnipress, 2010, pp. 255–262.

[9] T. Horváth, T. Gärtner, and S. Wrobel, “Cyclic pattern kernels for predictive graph mining,” in Proc. SIGKDD Int. Conf. Knowl. Discovery Data Mining, New York, NY, USA: ACM, 2004, pp. 158–167.

[10] R. Kondor and H. Pan, “The multiscale Laplacian graph kernel,” in Proc. Adv. Neural Inf. Process. Syst., 2016, pp. 2990–2998.

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

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

[13] N. Shervashidze and K. M. Borgwardt, “Fast subtree kernels on graphs,” in Proc. Adv. Neural Inf. Process. Syst., 2009, pp. 1660–1668.

[14] N. Shervashidze, P. Schweitzer, E. J. Van Leeuwen, K. Mehlhorn, and K. M. Borgwardt, “Weisfeiler–Lehman graph kernels,” J. Mach. Learn. Res., vol. 12, pp. 2553–2561, Sep. 2011.

[15] G. D. San Martino, N. Navarin, and A. Sperti, “A tree-based kernel for graphs,” Proc. SIAM Int. Conf. Data Mining, Philadelphia, PA, USA: SIAM, 2012, pp. 975–986.

[16] M. Togninalli, E. Ghisu, F. Llinares-López, B. A. Rieck, and K. M. Borgwardt, “Weisfeiler–Lehman graph kernels,” in Proc. Adv. Neural Inf. Process. Syst., 2016, pp. 1639–1647.

[17] A. Leman and B. Weisfeiler, “A reduction of a graph to a canonical form and an algebra arising during this reduction,” Nauchno-Technicheskaya Informatiya, vol. 2, no. 9, pp. 12–16, 1968.

[18] R. Kondor and T. Jебara, “A kernel between sets of vectors,” in Proc. 20th Int. Conf. Mach. Learn., 2003, pp. 361–368.

[19] D. J. Watts and S. H. Strogatz, “Collective dynamics of ‘small-world’ networks,” Nature, vol. 393, no. 6684, pp. 440–442, 1998.

[20] R. Andersen, F. Chung, and K. Lang, “Local graph partitioning using PageRank vectors,” in Proc. 47th Ann. IEEE Symp. Found. Comput. Sci., Piscataway, NJ, USA: IEEE Press, 2006, pp. 475–486.

[21] F. Chung, “The heat kernel as the pagerank of a graph,” in Proc. Nat. Acad. Sci., vol. 104, no. 50, pp. 19735–19740, 2007.

[22] 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. 1, pp. 723–773, 2012.

[23] M. Sugiyama and K. Borgwardt, “H Starting in random walk kernels,” in Proc. Adv. Neural Inf. Process. Syst., vol. 28, 2015, pp. 1639–1647.

[24] N. Prill, D. G. Corneil, and D. Jurisica, “Modeling interactome: Scale-free or geometric?” Bioinformatics, vol. 20, no. 18, pp. 3508–3515, 2004.

[25] F. Bause and N. M. Kriege, “Gradual Weisfeiler–Lehman: Slow and steady wins the race,” in Proc. Learn. Theory Kernel Mach., 2004, pp. 20–21.

[26] G. Nikolentzos and M. Vazirgiannis, “Graph alignment kernels using PageRank vectors,” in Proc. Adv. Neural Inf. Process. Syst., 2009, pp. 1660–1668.

[27] T. N. Kipf and M. Welling, “Semi-supervised classification with graph convolutional networks,” in Proc. Int. Conf. Mach. Learn. Represent., 2016, pp. 1–14.
