Learning Deep Graph Representations via Convolutional Neural Networks

Wei Ye, Omid Askarisichani, Alex Jones, and Ambuj Singh

Abstract—Graph-structured data arise in many scenarios. A fundamental problem is to quantify the similarities of graphs for tasks such as classification. R-convolution graph kernels are positive-semidefinite functions that decompose graphs into substructures and compare them. One problem in the effective implementation of this idea is that the substructures are not independent, which leads to high-dimensional feature space. In addition, graph kernels cannot capture the high-order complex interactions between vertices. To mitigate these two problems, we propose a framework called DeepMap to learn deep representations for graph feature maps. The learned deep representation for a graph is a dense and low-dimensional vector that captures complex high-order interactions in a vertex neighborhood. DeepMap extends Convolutional Neural Networks (CNNs) to arbitrary graphs by generating aligned vertex sequences and building the receptive field for each vertex. We empirically validate DeepMap on various graph classification benchmarks and demonstrate that it achieves state-of-the-art performance.

Index Terms—Deep learning, representation learning, convolutional neural networks, feature maps, graph kernels, graphlet, shortest path, Weisfeiler-Lehman

1 Introduction

Irregular data arise in many scenarios, such as 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 such data. One fundamental problem with graph-structured data is computing their similarities, needed for downstream tasks such as classification. Graph kernels have been developed and widely used to measure the similarities between graph-structured data. This paper deals with graph kernels that are instances of the family of R-convolution kernels [1]. The key idea is to recursively decompose graphs into their substructures such as graphlets [2], subtrees [3], [4], walks [5], [6], paths [7], [8], and then compare these substructures from two graphs. A typical definition for graph kernels is $\mathcal{K}(\mathcal{G}_1, \mathcal{G}_2) = \langle \phi(\mathcal{G}_1), \phi(\mathcal{G}_2) \rangle$, where $\langle \cdot, \cdot \rangle$ denotes the dot product between two vectors, $\phi(\mathcal{G}_i) = [\psi(\mathcal{G}_i, A_1), \psi(\mathcal{G}_i, A_2), \ldots, \psi(\mathcal{G}_i, A_m)]$ is a vector that contains the number of occurrences of substructure A_j $(1 \leq j \leq m)$ (denoted as $\psi(\mathcal{G}_i, A_j)$) in graph $\mathcal{G}_i$ $(i = 1, 2)$. We call $\phi(\mathcal{G})$ the feature map[1] (please see Definition 2) of graph $\mathcal{G}$.

Although graph kernels are efficient methods to compute graph similarities, they still have the following two main issues: First, the substructures extracted from graphs are not independent. For instance, by adding/deleting vertices or edges, one graphlet can be derived from another graphlet. Fig. 1 shows that graphlet $G_3^{(3)}$ can be derived from graphlet $G_2^{(3)}$ by adding an edge. This dependency (redundancy) remains in graph feature maps. Because of this dependency between substructures, the dimension of the graph feature map often grows exponentially and thus it leads to low effectiveness. Second, graph kernels use the hand-crafted features without considering the complex interactions between vertices. Thus, high-order information in the neighborhood of a vertex is not integrated into graph feature maps.

To solve the first main issue, Deep Graph Kernels (DGK) [9] leverages techniques from natural language processing to learn latent representations for substructures. Then the similarity matrix between substructures is computed and integrated into the computation of the graph kernel matrix. If the number of substructures is high, it will cost a lot of time and memory to compute the similarity matrix. In addition, DGK uses natural language processing models to learn latent representations for substructures without proving that the frequency of substructures extracted from graphs follows a power-law distribution, which is observed in natural language. For example, the Weisfeiler-Lehman subtree kernel (WL) [3], [4] decomposes graphs into subtree patterns and then counts the number of common subtree patterns across graphs. If the subtree is of depth zero (i.e., only one root vertex), we can represent it using the vertex degree. However, the vertex degree distribution of a graph does not always follow a power-law distribution. Thus, the learned representations for substructures are not accurate. To solve the second main issue, people develop graph neural networks (GNNs) [10], [11], [12], [13] to extract the complex high-order interactions in a vertex neighborhood.

1. In this work, feature map and representation are used in an exchangable manner.

• *The authors are with the Department of Computer Science, University of California, Santa Barbara, CA 93106 USA. E-mail: {weiye, omid55, alexjones, ambuj}@cs.ucsb.edu.*

Manuscript received 2 Dec. 2019; revised 14 July 2020; accepted 31 July 2020. Date of publication 4 Aug. 2020; date of current version 1 Apr. 2022.
(Corresponding author: Wei Ye.)
Recommended for acceptance by J. Tang.
Digital Object Identifier no. 10.1109/TKDE.2020.3014089

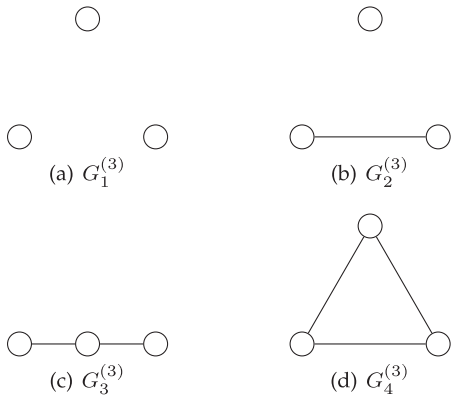


Fig. 1. Non-isomorphic subgraphs (graphlets) of size $k = 3$.

To mitigate these two main issues, we develop a CNN architecture on the vertex feature map (please see Definition 3) extracted from each vertex in a graph. The method is called DEEPMAP since it learns deep representations for graph feature maps. The learned deep representation of a graph is a dense and low-dimensional vector that captures complex high-order interactions in a vertex neighborhood. Typically, a CNN contains several convolutional and dense layers. CNNs exploit spatial locality of an input and thus the learned "filters" produce the strongest response to the spatially local input pattern. Stacking many such layers leads to non-linear filters that can capture appropriate patterns. The extension of CNNs from images to graphs of arbitrary size and shape faces one main challenge: as opposed to images whose pixels are spatially ordered, vertices in graphs do not have spatial or temporal order. Vertices across different graphs are hard to align, and thus the receptive fields of CNNs cannot be directly applied on vertices in graphs. To develop a CNN applicable to arbitrary graphs, we propose to solve two main problems: (1) Generate a vertex sequence for each graph such that these sequences are aligned. (2) Determine the receptive field for each vertex in each vertex sequence.

Our contributions are summarized as follows:

- We analyze the graph feature maps of three popular graph kernels and then propose the definition of vertex feature maps.
- We develop a new CNN model DEEPMAP on the vertex feature maps to mitigate the two main issues. The extension of CNN from images to graphs of arbitrary size and shape is achieved by two steps: (1) We use eigenvector centrality [14] as a measure to generate aligned vertex sequences. (2) We use a breadth-first search (BFS) method for constructing the receptive field for each vertex in each vertex sequence.
- We empirically validate DEEPMAP on a number of graph classification benchmarks and demonstrate that it achieves state-of-the-art performance.

The rest of the paper is organized as follows: We describe related work in Section 2. Section 3 covers the ideas of graph feature maps of three popular graph kernels. Section 4 introduces the core ideas behind our approach DEEPMAP, including the definition of vertex feature maps and the extension of CNN to arbitrary graphs. Using the benchmark graph datasets, Section 5 compares DEEPMAP with related techniques. Section 6 makes some discussions. And Section 7 concludes the paper.

2 Related Work

2.1 Graph Kernels

R-convolution graph kernels can be based on walks [5], [6], paths [7], [8], graphlets [15], and subtree patterns [3], [4], [16], [17], etc. RetGK [6] 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. The shortest-path graph kernel (SP) [7] counts the number of pairs of shortest paths that have the same source and sink labels and the same length in two graphs. The Tree++ [8] graph kernel is proposed for the problem of comparing graphs at multiple levels of granularities. It first uses a path-pattern graph kernel to build 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. To capture graph similarity at multiple levels of granularities, Tree++ incorporates a new concept called super path into the path-pattern graph kernel. The super path contains truncated BFS trees rooted at the vertices in a path. The graphlet kernel (GK) [15] proposes to use the method of random sampling to extract graphlets from graphs. The idea of random sampling is motivated by the observation 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 Weisfeiler-Lehman subtree kernel (WL) [3], [4] is based on the Weisfeiler-Lehman test of graph isomorphism [18] for graphs. In each iteration, the Weisfeiler-Lehman test of graph isomorphism augments vertex labels by concatenating their neighbors' labels and then compressing the augmented labels into new labels. The compressed labels correspond to the subtree patterns. WL counts common original and compressed labels in two graphs.

There are also some graph kernels [19], [20] focusing on computing global similarities between graphs. The paper [19] computes the Jensen-Shannon divergence between probability distributions over graphs, without the need of decomposing the graph into substructures. The paper [20] designs two novel graph kernels to capture global properties of unlabeled graphs. The kernels are based on the Lovász number and are called the Lovász ϑ kernel and the SVM-ϑ kernel. Both of these two kernels still need to enumerate all subsets of nodes from two graphs and compute the Lovász number for each subset. The number of possible subsets of nodes still exponentially increases with the increasing size of the graph.

Recently, some research works such as [9], [21] focus on augmenting the existing graph kernels or fusing GNNs with graph kernels [22]. DGK [9] 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 [21] 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. Graph Neural Tangent Kernel (GNTK) [22] is inspired by the connections between over-parameterized neural networks and kernel methods [23], [24]. It is a model that inherits both advantages from GNNs and graph kernels. It can extract powerful features from graphs as GNNs and is easy to train and analyze as graph kernels. It is equivalent to infinitely wide GNNs trained by gradient descent.

2.2 Graph Neural Networks

In addition to the above-described literature, there are also some literature from the field of graph neural networks (GNNs) [10], [11], [12], [13], [25], [26], [27], [28], [29] related to our work. SpectralNet [26] develops an extension of spectral networks [30] for deep learning on graphs. A spectral network generalizes a convolutional network through the Graph Fourier Transform. Graph-CNN [28] proposes a strictly localized spectral filters that uses Chebyshev polynomials for approximately learning K-order spectral graph convolutions [31]. Both SpectralNet and Graph-CNN first construct similarity graphs from a dataset and then classify data points into different classes. They are not applicable to graphs of arbitrary size and shape. GCN [27] introduces a simple and well-behaved layer-wise propagation rule for graph convolutional networks. The propagation rule is derived from the first-order approximation of spectral graph convolutions. GAT [29] computes the latent representations for each vertex in a graph, by attending over its neighbors, following a self-attention strategy. It specifies different weights to different vertices in a neighborhood. GraphSAGE [32] is developed for the inductive representation learning on graphs. It learns a function to generate embeddings for each node, by sampling and aggregating features from a node's local neighborhood. GCN, GAT and GraphSAGE are designed for the classification of vertices in a graph.

Neural Graph Fingerprints (NGF) [13] introduces a convolutional neural network on graphs for learning differentiable molecular fingerprints, by replacing each discrete operation in circular fingerprints with a differentiable analog. NGF develops a local message-passing architecture that propagates information to a depth of R neighborhood. DCNN [25] extends convolutional neural networks to graphs by introducing a diffusion-convolution operation, based on which diffusion-based representations can be learned from graphs and used as an effective basis for vertex classification and graph classification. DGCNN [12] first designs a novel special graph convolution layer to extract multi-scale vertex features. Then, in order to sequentially read graphs of differing vertex orders, DGCNN designs a novel SortPooling layer that sorts graph vertices in a consistent order so that traditional neural networks can be trained on graphs. GIN [10] is proposed to analyze the expressive power of GNNs to capture different graph structures. Both DGCNN and GIN are inspired by the close connection between GNNs and the Weisfeiler-Lehman test of graph isomorphism. The inputs to DGCNN and GIN are the one-hot encodings of vertex labels. PATCHYSAN [11] generalizes CNNs from images to arbitrary graphs. It first orders vertices by the graph canonicalization tool NAUTY [33], and then performs three operations: (1) vertex sequence selection, (2) neighborhood assembly, and (3) graph normalization. There is another work that also uses CNNs on graphs. DeepTrend 2.0 [34] proposes a CNN-based model on a sensor network for traffic flow prediction. It converts the sensor network into an image, in which neighboring pixels represent sensors that have a strong correlation. In this way, the local similarity of the image is fullfilled. But neighboring sensors in a sensor network may not be mapped to the neighboring pixels in the image.

3 GRAPH FEATURE MAPS

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 $\mathbf{x}(i)$. We use $\mathbf{x} = [x_1, \ldots, x_n]$ to denote creating a vector by stacking scalar x_i along the columns. We consider an undirected labeled graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, l)$, 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 [35], 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 of all distinct edges. A path is a trail that consists of all distinct vertices and edges. The depth of a subtree is the maximum length of paths between the root and any other vertex in the subtree.

Definition 1 (Graph Isomorphism). *Two undirected labeled graphs $\mathcal{G}_1 = (\mathcal{V}_1, \mathcal{E}_1, l_1)$ and $\mathcal{G}_2 = (\mathcal{V}_2, \mathcal{E}_2, l_2)$ are isomorphic (denoted by $\mathcal{G}_1 \simeq \mathcal{G}_2$) 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 $\mathcal{G}_2$; (2) and such that $l_1(v) = l_2(\varphi(v))$.*

Let $\mathcal{X}$ be a non-empty set and let $\mathcal{K} : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ be a function on $\mathcal{X}$. 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 $\phi(\mathcal{G})$ denote a mapping from a graph to a vector which contains the number of occurrences of the atomic substructures in graph $\mathcal{G}$. Then, the kernel on two graphs $\mathcal{G}_1$ and $\mathcal{G}_2$ is defined as $\mathcal{K}(\mathcal{G}_1, \mathcal{G}_2) = \langle \phi(\mathcal{G}_1), \phi(\mathcal{G}_2) \rangle$.

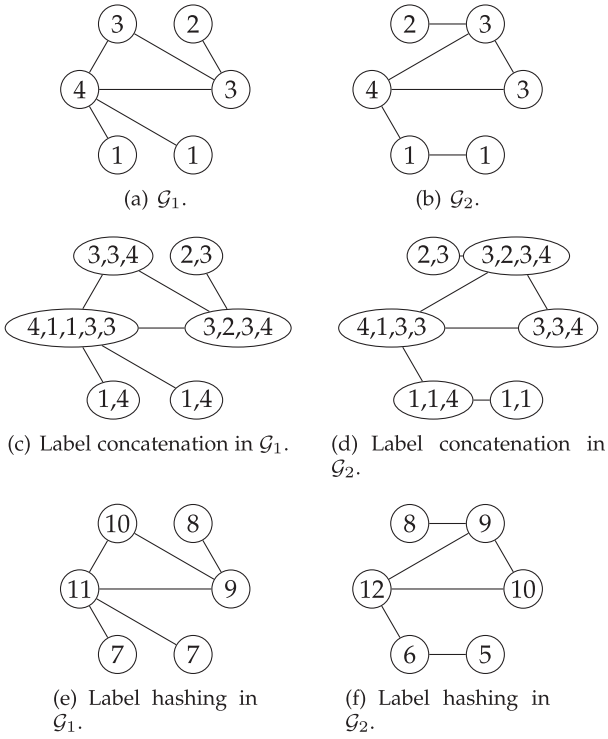
We define graph feature maps as follows:

Definition 2 (Graph Feature Maps). *Define a map $\psi : \{\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_n\} \times \Sigma \rightarrow \mathbb{N}$ such that $\psi(\mathcal{G}, A)$ is the number of occurrences of the atomic substructure A in graph $\mathcal{G}$. Then the feature map of graph $\mathcal{G}$ is defined as follows:*

$$\phi(\mathcal{G}) = [\psi(\mathcal{G}, A_1), \psi(\mathcal{G}, A_2), \ldots, \psi(\mathcal{G}, A_m)], \tag{1}$$

where m is the number of unique atomic substructures and depends on graphs.

In the following, let us elaborate the mechanisms of three popular graph kernels, i.e., the graphlet kernel (GK) [15],



(a) $\mathcal{G}_1$. (b) $\mathcal{G}_2$. (c) Label concatenation in $\mathcal{G}_1$. (d) Label concatenation in $\mathcal{G}_2$. (e) Label hashing in $\mathcal{G}_1$. (f) Label hashing in $\mathcal{G}_2$.

Fig. 2. Illustration of one iteration of the WL test of graph isomorphism algorithm for graphs. For subfigures (a) and (b), $\Sigma = \{1, 2, 3, 4\}$. For subfigures (e) and (f), $\Sigma = \{5, 6, 7, 8, 9, 10, 11, 12\}$.

the shortest-path kernel (SP) [7] and the Weisfeiler-Lehman subtree kernel (WL) [4], and relate them to our definitions.

A graphlet G (as shown in Fig. 1) is a non-isomorphic subgraph of size k induced from graph $\mathcal{G}$. Let $\mathsf{G}^{(k)}$ be the multiset[2] of size-k graphlets. Then, for graph $\mathcal{G}$, its feature map is defined as follows:

$$\phi(\mathcal{G}) = \Big[\psi(\mathcal{G}, G_1^{(k)}), \psi(\mathcal{G}, G_2^{(k)}), \ldots, \psi(\mathcal{G}, G_m^{(k)})\Big], \tag{2}$$

where m stands for the number of unique graphlets of size k in $\mathsf{G}^{(k)}$, $\psi(\mathcal{G}, G_i^{(k)})(1 \leq i \leq m)$ denotes the frequency of the unique graphlet $G_i^{(k)}$ occuring in graph $\mathcal{G}$. Exhaustive enumation of all graphlets of size k is prohibitively expensive, especially for large graphs. Usually, we use some sampling techniques such as the random sampling scheme proposed in [15] to sample a number of q graphlets of size k from graph $\mathcal{G}$, and then count the frequency of each unique graphlet occurring in these q samples.

Let P denote the multiset of all shortest-paths in graph $\mathcal{G}$. For each shortest-path $P = (s, v_1, v_2, \ldots, t) \in \mathsf{P}$ where s denotes the source vertex and t denotes the sink vertex, we use a triplet $(l(s), l(t), len(P))$ to denote it, where $len(P)$ is the length of the shortest-path P. For example, in Fig. 2b, the triplet for the shortest-path between the two vertices with labels 2 and 4 respectively is (2, 4, 2). Then, for graph $\mathcal{G}$, its feature map is defined as follows:

$$\phi(\mathcal{G}) = [\psi(\mathcal{G}, S_1), \psi(\mathcal{G}, S_2), \ldots, \psi(\mathcal{G}, S_m)], \tag{3}$$

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

where m denotes the number of unique triplets in P, and $\psi(\mathcal{G}, S_i)(1 \leq i \leq m)$ denotes the number of a unique triplet S_i occurring in graph $\mathcal{G}$.

The Weisfeiler-Lehman test of graph isomorphism [18] 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 graph isomorphism algorithm augments vertex labels by first concatenating their neighbors' labels and then hashing the augmented labels into new labels. The hashed labels correspond to subtree patterns.

For example, in Fig. 2b, a subtree pattern of height one rooted at the vertex with label 4 can be denoted as a string of concatenated labels of vertices "4, 1, 3, 3" which is augmented as "12' by the Weisfeiler-Lehman test of graph isomorphism. Let $\mathsf{T}^{(h)}$ denote the multiset of all subtree patterns of height h in graph $\mathcal{G}$, then the feature map of $\mathcal{G}$ is defined as follows:

$$\phi(\mathcal{G}^{(h)}) = \Big[\psi(\mathcal{G}^{(h)}, T_1^{(h)}), \psi(\mathcal{G}^{(h)}, T_2^{(h)}), \ldots, \psi(\mathcal{G}^{(h)}, T_m^{(h)})\Big], \tag{4}$$

where $\mathcal{G}^{(0)}$ is the original graph $\mathcal{G}$ and $\mathcal{G}^{(h)}$ is the augmented graph at the hth iteration of the Weisfeiler-Lehman test of graph isomorphism. We call graphs $\mathcal{G}^{(0)}, \mathcal{G}^{(1)}, \ldots, \mathcal{G}^{(h)}$ a sequence of Weisfeiler-Lehman graphs. m denotes the number of unique subtree patterns in $\mathsf{T}^{(h)}$, and $\psi(\mathcal{G}^{(h)}, T_i^{(h)})(1 \leq i \leq m)$ denotes the number of a unique subtree pattern $T_i^{(h)}$ occurring in graph $\mathcal{G}^{(h)}$.

The feature map of WL is the concatenation of the feature maps at all the iterations

$$\phi(\mathcal{G}) = \Big[\phi(\mathcal{G}^{(0)}), \phi(\mathcal{G}^{(1)}), \ldots, \phi(\mathcal{G}^{(h)})\Big]. \tag{5}$$

4 Deep Graph Feature Maps

In this section, we develop a new convolutional neural network (CNN) model for learning deep graph feature maps, which is called DeepMap. The extension of CNNs from images whose pixels are spatially ordered to graphs of arbitrary size and shape is challenging. We first align vertices across graphs. Then, we build the receptive field for each vertex.

4.1 CNNs on Graphs

We define the vertex feature maps as follows:

Definition 3 (Vertex Feature Maps). *Define a map* $\psi : \{v_1, v_2, \ldots, v_{|\mathcal{V}|}\} \times \Sigma \rightarrow \mathbb{N}$ *where* $v_i \in \mathcal{V}(1 \leq i \leq |\mathcal{V}|)$ *such that* $\psi(v, A)$ *is the number of occurrences of the atomic substructure* A *that contains* v *in graph* $\mathcal{G}$*. Then the feature map of vertex* v *is defined as follows:*

$$\phi(v) = [\psi(v, A_1), \psi(v, A_2), \ldots, \psi(v, A_m)], \tag{6}$$

where m *is the number of unique atomic substructures and depends on graphs.*

From Definitions 2 and 3, we can observe that the feature map of a graph equals to the sum of the feature maps of all the vertices in that graph. Note that this pooling-like feature

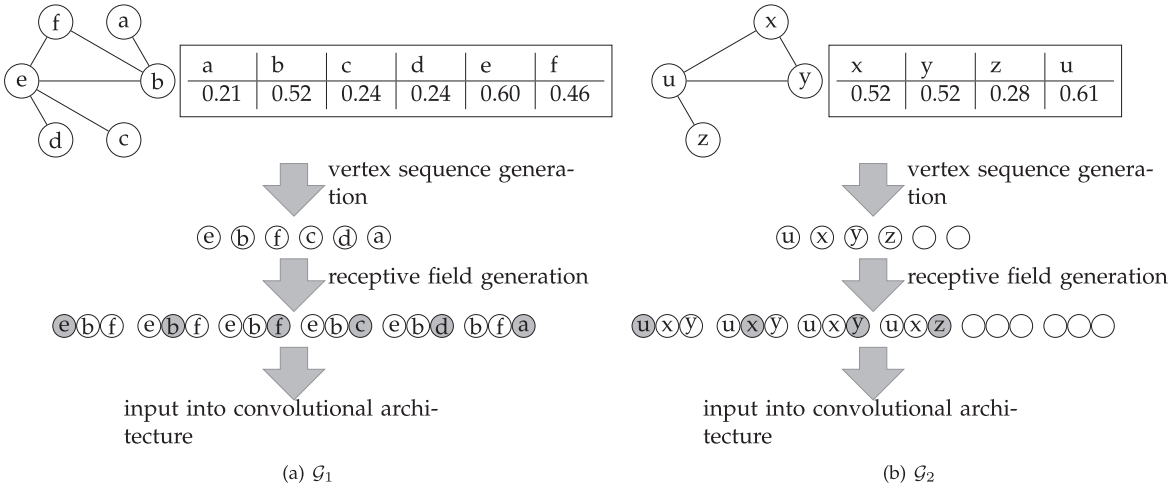


Fig. 3. Illustration of the generation of vertex sequences and the corresponding receptive fields for undirected graphs $\mathcal{G}_1$ and $\mathcal{G}_2$. The two tables beside graphs $\mathcal{G}_1$ and $\mathcal{G}_2$ contain the eigenvector centrality value of each vertex.

map is permutation-invariant and size-invariant. In other words, the feature map is invariant to the ordering of vertices and the sizes of graphs

$$\phi(\mathcal{G}) = \sum_{i=1}^{|\mathcal{V}|} \phi(v_i). \tag{7}$$

From the definitions of vertex feature maps (Equation (6)) and graph feature maps (Equation (1)), we can see that several important issues are not taken into account to compute graph feature maps. As described in Section 1, they are: (1) Substructures are not independent and thus it leads to high-dimensional feature space. (2) The complex high-order interactions in the neighborhood of a vertex are not considered. To mitigate these two issues, in this work, we develop a CNN architecture on the vertex feature maps. The learned deep graph representation is of low-dimension. Furthermore, convolution operation in CNN can capture the complex high-order interactions in the neighborhood of a vertex. One main challenge in developing CNNs for graphs of arbitrary size and shape is that unlike images whose pixels are spatially ordered, vertices in graphs do not have a spatial or temporal order. Vertices across different graphs are difficult to align, and thus the receptive fields of CNNs cannot be directly applied on vertices in graphs.

An image can be considered as a rectangle grid graph whose vertices represent pixels. A CNN of a stride length one on an image can be considered as traversing a sequence of pixels (vertices), from left to right and top to bottom. As indicated above, pixels are spatially ordered and they are aligned across images. Thus, the order of pixels in the sequence that is traversed by a CNN is unique. To make CNNs applicable to graphs, we first need to generate a vertex sequence for each graph such that the sequences are aligned across graphs. In this work, we use eigenvector centrality [14] 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. We generate a vertex sequence in each graph by sorting their eigenvector centrality values from high to low. Since graphs are of arbitrary size, we use the size w of the graph that has the largest number of vertices as the length of the sequence. In this case, for sequences whose lengths are less than w, we concatenate them with dummy vertices to make their lengths equal to w. The dummy vertices' feature maps are set to zero vectors so that they do not contribute to the convolution.

After generating a vertex sequence for each graph, we need to determine the receptive field for each vertex in the sequence. Assume that the size of the receptive field is r. We perform a breadth-first search (BFS) on the original graph for constructing the receptive field. If the number of the one-hop neighbors of a vertex is greater than or equal to $r-1$, we select the top $r-1$ largest one-hop neighbors with respect to their eigenvector centrality values. If the number of the one-hop neighbors of a vertex is less than $r-1$, we first select all the one-hop neighbors and then select vertices from the two-hop neighbors, the three-hop neighbors, and so on, until the receptive field has exact r vertices. If the size of a graph is less than r, we use dummy vertices for padding purposes. Note that the vertices in the receptive field are also sorted in descending order according to their eigenvector centrality values.

We use Fig. 3 to demonstrate the generation procedure for vertex sequences and their corresponding receptive fields. In the first row of Fig. 3, the tables demonstrate the eigenvector centrality value of each vertex. In the second row of Fig. 3, the vertex sequence is generated by sorting the eigenvector centrality values of vertices in descending order. Since the size of graph $\mathcal{G}_2$ is less than that of graph $\mathcal{G}_1$, we concatenate two dummy vertices (indicated by two blank vertices) in the generated sequence. In the third row of Fig. 3, for each vertex (indicated in gray) in the vertex sequence, we use BFS to generate its receptive field. Here, the size of the receptive field is three. Finally, we use a convolutional architecture to learn deep graph feature maps from the feature maps of the vertices.

Fig. 4 demonstrates our convolutional architecture. The architecture has three one-dimensional convolution layers which have rectified linear units (ReLU). Three one-

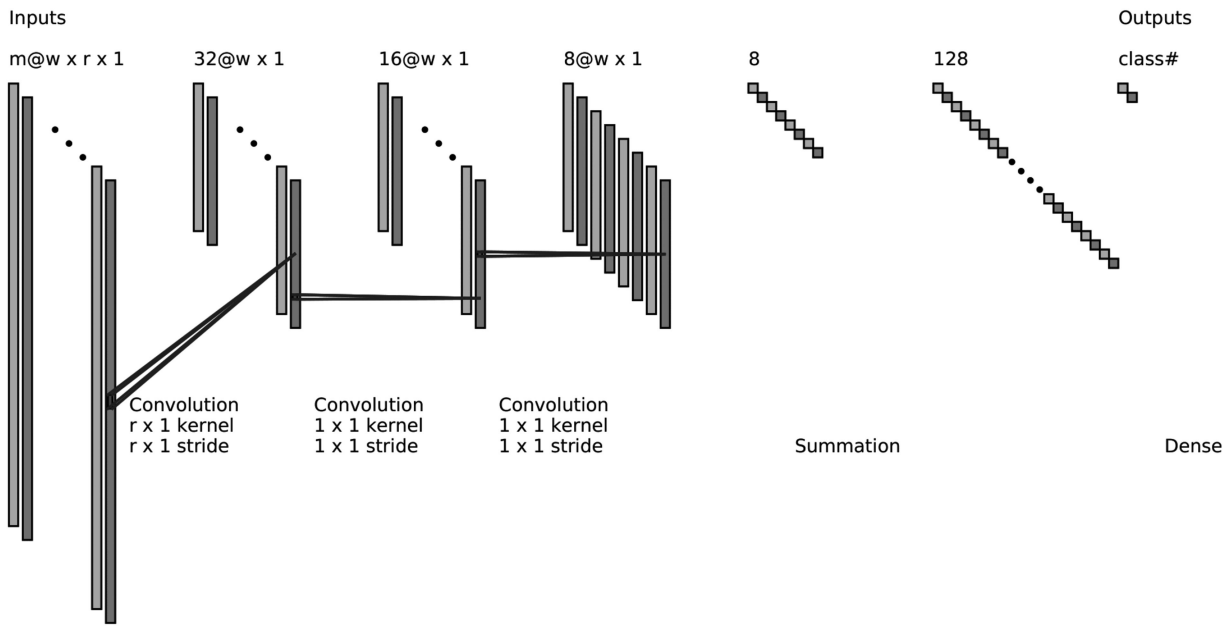


Fig. 4. Convolutional architecture. In the first layer, m is the dimension of the vertex feature map, w is the maximum number of vertices in a set of graphs, and r is the size of the receptive field.

dimensional convolution layers are used to aggregate the feature maps of each vertex with those of its neighbors. After the last convolution layer, we use a summation layer to add the feature map of every vertex in a graph together and the resulting feature map is the deep graph feature map. This summation layer just functions Equation (7). After the summation layer, we use a dense (fully-connected) layer with rectified linear units (ReLU), followed by a dropout layer and a softmax layer, for graph classification.

Theorem 1. *If two graphs $\mathcal{G}_1$ and $\mathcal{G}_2$ are isomorphic, their deep graph feature maps after the summation layer are the same.*

Proof. If $\mathcal{G}_1 \simeq \mathcal{G}_2$, we have the following:

- $|\mathcal{V}_1| = |\mathcal{V}_2|$ and $|\mathcal{E}_1| = |\mathcal{E}_2|$.
- $\mathcal{G}_1$ and $\mathcal{G}_2$ have the same degree sequence.
- Vertex v_1 from $\mathcal{G}_1$ and vertex $\varphi(v_1)$ from $\mathcal{G}_2$ have the same feature map $\boldsymbol{\phi}(v_1) = \boldsymbol{\phi}(\varphi(v_1))$.
- Vertex sequence $S_1 = \big(v_1, v_2, \ldots, v_{|\mathcal{V}_1|}\big)$ generated from $\mathcal{G}_1$ is identical to vertex sequence $S_2 = \big(\varphi(v_1), \varphi(v_2), \ldots, \varphi(v_{|\mathcal{V}_2|})\big)$ generated from $\mathcal{G}_2$.
- The receptive field for each vertex in sequence S_1 is the same as that of the corresponding vertex in sequence S_2.

Thus, $\mathcal{G}_1$ and $\mathcal{G}_2$ have the same deep graph feature maps after the summation layer. □

Note that if we use the sampling technique to sample graphlets around two corresponding vertices from two isomorphic graphs, the vertex feature maps may not be the same. Thus, the deep graph feature maps may not be the same.

4.2 Algorithm

The pseudo-code for DEEPMAP is given in Algorithm 1. Lines 1–7 compute the feature map for each vertex in each graph. The feature map could be graphlet feature map, shortest-path feature map, or subtree feature map. The user can choose one kind of them. If using random sampling for graphlets, the time complexity to compute feature maps for all vertices in n graphs is $\mathcal{O}(n \cdot w \cdot d^3)$ [15], where w is the largest number of vertices in a set of graphs, and d is the maximum degree number. If using the Floyd–Warshall algorithm to find all pairs of shortest paths, the time complexity to compute feature maps for all vertices in n graphs is $\mathcal{O}(n \cdot w^3)$ [7]. If using subtrees, the time complexity to compute feature maps for all vertices in n graphs is $\mathcal{O}(n \cdot h \cdot e)$ [4], where e is the largest number of edges and we assume $e > w$, and h is the iteration of the Weisfeiler-Lehman test of graph isomorphism.

For each graph, line 11 generates the vertex sequence by sorting vertices in descending order with respect to their eigenvector centrality values. We use power iteration to compute eigenvector centrality for each graph. The time complexity for line 11 is bounded by $\mathcal{O}(e + w \cdot \log w)$, where $\mathcal{O}(e)$ is the time complexity of power iteration, and $\mathcal{O}(w \cdot \log w)$ is the time complexity for the fast sort. If the size of a graph is less than the designated length w of the vertex sequence, line 13 appends its vertex sequence with dummy vertices. For each vertex v in the sequence, lines 15–19 construct its receptive field and append the corresponding vertex feature maps to $\boldsymbol{\Phi}'$. Line 17 uses the breadth-first search (BFS) starting from v on the original graph to find the top $r-1$ largest neighbors w. r.t. their eigenvector centrality values, and sort them in descending order. We assume the number of edges is greater than the number of vertices. Thus, the time complexity for BFS is bounded by $\mathcal{O}(e)$. Thus, the time complexity for lines 10–20 is $\mathcal{O}(n \cdot (e + w \cdot \log w + w \cdot e))$. Finally, we input $\boldsymbol{\Phi}'$ and the graph labels $\mathcal{Y}$ into CNNs for graph classification.

The time complexity of two-dimensional CNNs is $\mathcal{O}\big(\sum_{l=1}^{c} n_{ic} \cdot s_l^2 \cdot n_f \cdot m_{oc}^2\big)$ [36], where l is the index of a convolutional layer, c is the number of convolutional layers, n_{ic} is the number of input channels, s_l is the length of the receptive field, n_f is the number of filters in the lth layer, m_{oc} is

the size of output channels. In our one-dimensional CNNs, c is set to three. In the first layer, n_{ic} is the length m of the vertex feature maps extracted by counting the substructures around the vertices, s_l is r, n_f is w, m_{oc} is set to 32. In the second layer, n_{ic} is 32, s_l is set to one, n_f is w, m_{oc} is set to 16. In the third layer, n_{ic} is 16, s_l is also set to one, n_f is w, m_{oc} is set to 8. Thus, the time complexity of our one-dimensional CNNs is bounded by $\mathcal{O}(m \cdot r \cdot w)$. The dense layer has 128 units. The dropout layer has a dropout rate of 0.5. The worst-case (when using the shortest-path feature map) time complexity of DeepMap is $\mathcal{O}(n \cdot w^3 + n \cdot w \cdot e + m \cdot w \cdot r)$.

Algorithm 1. DeepMap

Input: A set of graphs $\{\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_n\}$ and their corresponding labels $\mathcal{Y} = \{y_1, y_2, \ldots, y_n\}$, the size r of the receptive field
Output: Classification accuracy acc

$\mathbf{\Phi} \leftarrow []$ /* feature maps initialization. */
foreach graph $\mathcal{G}_i (1 \le i \le n)$ **do**
  $\mathbf{X} \leftarrow []$;
  foreach vertex $v \in \mathcal{G}_i$ **do**
    $\mathbf{x} \leftarrow \phi(v)$;
    $\mathbf{X}$.append($\mathbf{x}$);
  $\mathbf{\Phi}$.append($\mathbf{X}$);
$w \leftarrow \max_{\mathcal{G}_i}$ length($\mathcal{G}_i$) $(1 \le i \le n)$;
$\mathbf{\Phi}' \leftarrow []$;
foreach graph $\mathcal{G}_i (1 \le i \le n)$ **do**
  generate vertex sequence $S_i = \left(v_{\sigma_1}, v_{\sigma_2}, \ldots, v_{\sigma_{|\mathcal{V}_i|}}\right)$ by sorting vertices according to their eigenvector centrality values /* $\sigma_1, \sigma_2, \ldots, \sigma_{|\mathcal{V}_i|}$ is a permutation of $1, 2, \ldots, |\mathcal{V}_i|$. */
  if length($\mathcal{G}_i$) $< w$ **then**
    append $w-$length($\mathcal{G}_i$) dummy vertices to S_i;
  $\mathbf{X}' \leftarrow []$;
  foreach vertex v in sequence S_i **do**
    if v is not a dummy vertex **then**
      append $\mathbf{X}(v_{\sigma_1}), \mathbf{X}(v_{\sigma_2}), \ldots, \mathbf{X}(v), \ldots, \mathbf{X}(v_{\sigma_{r-1}})$ to $\mathbf{X}'$ /* use the breadth-first search (BFS) starting from v on the original graph to find the top $r-1$ largest neighbors w.r.t. their eigenvector centrality values, and sort them in descending order. */
    else
      append number r of zero vectors $\mathbf{0}$ (for dummy vertices) to $\mathbf{X}'$;
  $\mathbf{\Phi}'$.append($\mathbf{X}'$);
$acc \leftarrow$ CNNs $\mathbf{\Phi}', \mathcal{Y}$ /* 10-fold cross-validation. */
return acc;

5 Experimental Evaluation

In this section, we conduct experiments on the benchmark graph datasets to compare DeepMap with state-of-the-art graph kernels and GNNs. DeepMap is built on three kinds of vertex feature maps: graphlet (GK [15]), shortest-path (SP [7]), and subtree patterns (WL [4]). The corresponding three versions of DeepMap are denoted as DeepMap-GK, DeepMap-SP, and DeepMap-WL, respectively.

5.1 Experimental Setup

We run all the experiments on a server with a 32-core Intel (R) Xeon(R) Silver 4110 CPU@2.10 GHz, 128 GB memory, a quad-core GeForce RTX 2080 GPU, and Ubuntu 18.04.1 LTS operating system, Python version 3.6. DeepMap is implemented with the Tensorflow wrapper Keras. We make our code publicly available at Github.[3]

We compare DeepMap with six state-of-the-art graph kernels, i.e., GNTK [22], DGK [9], RetGK [6], GK [15], SP [7], and WL [4]. We also compare DeepMap with four state-of-the-art GNNs, i.e., GIN [10], PatchySan [11], DCNN [25], and DGCNN [12]. We perform 10-fold cross-validation and report the average classification accuracies and standard deviations.

For the comparison methods, we set their parameters according to their original papers. The graphlet size of GK is selected from $\{3, 4, 5\}$. The depth of the subtree used in WL is selected from $\{0, 1, 2, 3, 4, 5\}$. For DeepMap, we use a single network architecture for all the experiments. We use the RMSPROP optimizer with initial learning rate 0.01 and decay the learning rate by 0.5 if the number of epochs with no improvement in the loss reaches five. We select the number of batch size from $\{32, 256\}$. Following GIN [10], for DeepMap and other GNNs, the number of epochs is set as the one that has the best cross-validation accuracy averaged over the ten folds. For graph kernels, we use a binary C-SVM [37] as the classifier. The parameter C for each fold is independently tuned from $\{1, 10, 10^2, 10^3\}$ using the training data from that fold.

TABLE 1
Statistics of the Benchmark Datasets Used in the Experiments

Dataset	Size	Class #	Avg. Node#	Avg. Edge#	Label #
SYNTHIE	400	4	95.00	172.93	N / A
KKI	83	2	26.96	48.42	190
BZR_MD	306	2	21.30	225.06	8
COX2_MD	303	2	26.28	335.12	7
DHFR	467	2	42.43	44.54	9
NCI1	4110	2	17.93	19.79	37
PTC_MM	336	2	13.97	14.32	20
PTC_MR	344	2	14.29	14.69	18
PTC_FM	349	2	14.11	14.48	18
PTC_FR	351	2	14.56	15.00	19
ENZYMES	600	6	32.63	62.14	3
PROTEINS	1113	2	39.06	72.82	3
IMDB-BINARY	1000	2	19.77	96.53	N / A
IMDB-MULTI	1500	3	13.00	65.94	N / A
COLLAB	5000	3	74.49	2457.78	N / A

N / A means the dataset has no vertex labels.

5.2 Datasets

In order to test the effectiveness of DeepMap, we use benchmark datasets whose statistics are given in Table 1. For datasets without vertex labels, we use vertex degrees as their vertex labels.

Synthetic Dataset. SYNTHIE [38] contains 400 graphs and can be divided into four classes. They are generated from two Erdős-Rényi graphs with edge probability 0.2.

Brain Network Dataset. KKI [39] is a brain network constructed from the whole brain functional resonance image (fMRI) atlas. Each vertex corresponds to a region of interest

3. https://github.com/yeweiysh/DeepMap

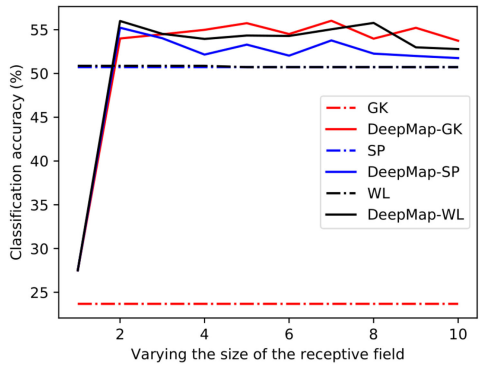


Fig. 5. Parameter sensitivity studies for the deep map models and their corresponding graph kernels on the benchmark dataset SYNTHIE.

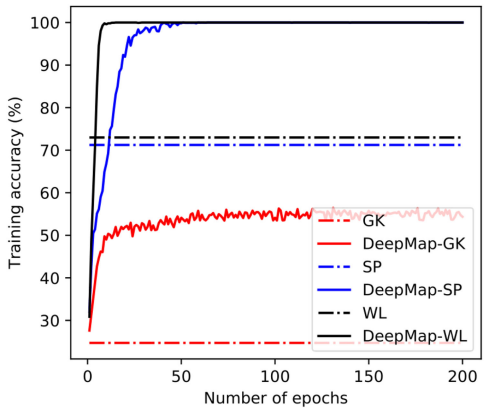


Fig. 6. Representational power studies for the deep map models and their corresponding graph kernels on the benchmark dataset SYNTHIE.

(ROI), and each edge indicates correlations between two ROIs. KKI is constructed for the task of Attention Deficit Hyperactivity Disorder (ADHD) classification.

Chemical Compound Datasets. The chemical compound datasets BZR_MD, COX2_MD, and DHFR are from [40]. 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 ligands for the benzodiazepine receptor. COX2 is a dataset of cyclooxygenase-2 inhibitors. DHFR is a dataset of 756 inhibitors of dihydrofolate reductase. BZR_MD and COX2_MD are derived from BZR and COX2, respectively, by removing explicit hydrogen atoms. The chemical compounds in the datasets BZR_MD and COX2_MD are represented as complete graphs. NCI1 [41] 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 PTC [42] 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). ENZYMES is a dataset of protein tertiary structures from [43], consisting of 600 enzymes from six Enzyme Commission top-level enzyme classes. The dataset PROTEINS is from [43]. 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.

Movie Collaboration Dataset. IMDB-BINARY and IMDB-MULTI datasets are from [9]. IMDB-BINARY contains movies of different actor/actress and genre information. For each collaboration graph, vertices represent actors/actresses. Edges denote that two actors/actresses appear in the same movie. The collaboration graphs are generated on Action and Romance genres. And for each actor/actress, a corresponding ego-network is derived and labeled with its genre. IMDB-MULTI is a multi-class version of IMDB-BINARY and includes a balanced set of ego-networks derived from Comedy, Romance, and Sci-Fi genres.

Scientific Collaboration Dataset. COLLAB [44] is derived from three public collaboration datasets, i.e., High Energy Physics, Condensed Matter Physics, and Astro Physics. Each graph represents an ego-network of a researcher from a research field. The label represents the research field (High Energy Physics, Condensed Matter Physics, and Astro Physics) of a researcher.

5.3 Results

In this section, we first evaluate DeepMap with varying the size r of the receptive field, then compare DeepMap with baselines on representational power and classification accuracy.

5.3.1 Parameter Sensitivity

We test the parameter sensitivity of the deep map models and their corresponding graph kernels on the synthetic dataset SYNTHIE. The results are shown in Fig. 5. Because graph kernels do not have the parameter of the size r of the receptive field, their classification accuracies do not change. When the size of the receptive field equals one, i.e., no neighborhood information is used in the deep map models, we can see that the deep map models perform poorly (classification accuracy is around 27 percent). When the size of the receptive field exceeds two, all the three deep map models are superior to their corresponding graph kernels. The performance of DeepMap-SP decreases with the increasing size of the receptive field, which can be explained by the small world experiments [45]. The experiments are often associated with the phrase "six degrees of separation", which means every two vertices in a graph can be connected by a shortest-path of length at most six. When the size of the receptive field of a vertex exceed seven (including the vertex itself), extra neighbors deteriorate the discrimination power. We can also observe a similar tendency of DeepMap-WL because it is built on subtree patterns which are also constrained by the "six degrees of separation". For DeepMap-GK, its performance increases with the increasing size of the receptive field. DeepMap-GK is built on graphlets. For each vertex, we randomly sample 20 graphlets of size five. More neighbors provide more distinct information and thus improve the discrimination power.

5.3.2 Representational Power

We test the representational power of the deep map models and their corresponding graph kernels on the benchmark dataset SYNTHIE in Fig. 6. We use the average training accuracy over the ten folds to evaluate the representational power. We can see that the deep map models dramatically improve the representational power of their corresponding graph kernels. DeepMap-WL and DeepMap-SP converge

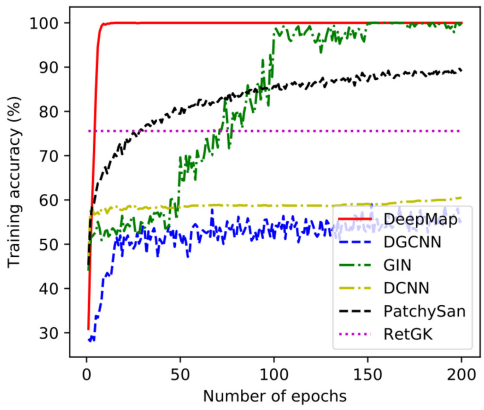


Fig. 7. Representational power studies for DeepMap and other baselines on the benchmark dataset SYNTHIE. To reduce clutter, for graph kernels, we only show the result of the graph kernel that has the highest representational power.

faster than DeepMap-GK. We use the best results of the deep map models in Fig. 6 as the final results for DeepMap. We compare the representational power of DeepMap with other baselines in Fig. 7. We can observe that DeepMap has better representational power and converges faster than other GNNs. DeepMap is superior to all the other baselines with a large margin.

5.3.3 Classification Accuracy

Table 2 shows the classification accuracies of the deep map models and their corresponding graph kernels on the benchmark datasets. We can see from the table that the deep map models outperform their corresponding graph kernels in most cases. On the dataset IMDB-MULTI, SP is better than DeepMap-SP, with a gain of 5.3 percent. On the dataset NCI1 and COLLAB, WL outperforms DeepMap-WL. However, on the other datasets, e.g., BZR_MD, DeepMap-GK has a gain of 28.1 percent over GK, DeepMap-SP has a gain of 7.2 percent over SP, and DeepMap-WL has a gain of 19.9 percent over WL, respectively.

Table 3 shows the classification accuracies of DeepMap and other graph kernels and GNNs on the benchmark datasets. DeepMap outperforms all the GNNs on most of the datasets. DGK is also based on the graph feature maps. We can see that our model DeepMap is superior to DGK with a large margin on all the datasets. On the dataset ENZYMES

TABLE 2
Comparison of Classification Accuracy (± Standard Deviation) of the Deep Map Models to Their Corresponding Graph Kernels on the Benchmark Datasets

Dataset	GK	DeepMap-GK	SP	DeepMap-SP	WL	DeepMap-WL
SYNTHIE	23.68±2.11	**54.48±4.34**	50.73±1.74	**54.03±2.38**	50.88±1.04	**54.53±6.16**
KKI	51.88±3.19	**56.77±9.69**	50.13±3.46	**62.92±7.94**	50.38±2.77	**61.65±15.0**
BZR_MD	49.27±2.15	**63.11±10.0**	68.60±1.94	**73.55±5.76**	59.67±1.47	**71.56±6.66**
COX2_MD	48.17±1.88	**52.44±7.36**	65.70±1.66	**72.28±9.37**	56.30±1.55	**69.66±7.32**
DHFR	61.01±0.23	**61.64±2.07**	77.80±0.98	**81.35±4.08**	82.39±0.90	**85.17±2.19**
NCI1	62.11±0.19	**63.26±2.04**	73.12±0.29	**79.90±1.78**	**84.79±0.22**	83.07±1.07
PTC_MM	50.82±6.20	**66.68±5.71**	62.18±2.22	**66.30±4.87**	67.18±1.62	**69.59±7.39**
PTC_MR	49.68±2.03	**63.38±6.04**	59.88±2.02	**67.73±6.61**	61.32±0.89	**63.59±5.31**
PTC_FM	51.94±4.05	**62.83±6.23**	61.38±1.66	**64.45±5.04**	64.44±2.09	**65.16±5.62**
PTC_FR	49.54±6.00	**65.82±1.07**	66.91±1.46	**68.39±3.57**	66.17±1.02	**67.82±5.03**
ENZYMES	23.88±1.78	**30.50±3.88**	41.07±0.77	**50.33±4.70**	51.98±1.24	**54.33±6.11**
PROTEINS	71.44±0.25	**73.77±2.33**	75.77±0.58	**76.19±2.91**	75.45±0.20	**75.47±3.26**
IMDB-BINARY	67.03±0.79	**69.60±4.80**	72.20±0.78	**74.60±4.74**	72.26±0.78	**78.10±5.26**
IMDB-MULTI	40.83±0.57	**42.80±2.84**	**50.89±0.90**	48.33±2.70	50.39±0.49	**53.33±3.89**
COLLAB	72.84±0.28	**73.92±2.03**	N / A	N / A	**78.90±1.90**	75.54±2.78

TABLE 3
Comparison of Classification Accuracy (± Standard Deviation) of DeepMap to Other Competitors on the Benchmark Datasets

Dataset	DeepMap	DGCNN	GIN	DCNN	PatchySan	DGK	RetGK	GNTK
SYNTHIE	**54.53±6.16**	47.50±7.99	53.48±3.64	54.18±4.49	44.25±14.36	52.43±1.02	49.95±1.96	53.98±0.87
KKI	**62.92±7.94**	56.25±18.8	60.34±12.5	48.93±7.50	43.75±13.98	51.25±4.17	48.50±2.99	46.75±5.75
BZR_MD	**73.55±5.76**	64.67±9.32	70.53±8.00	59.61±11.2	67.00±9.48	58.50±1.52	62.77±1.69	66.47±1.20
COX2_MD	**72.28±9.37**	64.00±8.86	65.97±5.70	51.29±5.31	65.33±7.78	51.57±1.71	59.47±1.66	64.27±1.55
DHFR	**85.17±2.19**	70.67±4.95	82.15±4.02	59.80±2.45	77.00±3.59	64.13±0.89	82.33±0.66	73.48±0.65
NCI1	83.07±1.07	71.73±2.14	82.70±1.70	57.10±0.69	78.60±1.90	80.31±0.46	**84.50±0.20**	84.20±1.50
PTC_MM	**69.59±7.39**	62.12±14.1	67.19±7.41	63.04±2.71	56.58±9.01	67.09±0.49	67.90±1.40	65.94±1.21
PTC_MR	**67.73±6.61**	55.29±9.38	62.57±5.18	55.65±4.92	55.25±7.98	62.03±1.68	62.50±1.60	58.32±1.00
PTC_FM	**65.16±5.62**	60.29±6.69	64.22±2.36	63.50±3.78	58.38±9.27	64.47±0.76	63.90±1.30	63.85±1.20
PTC_FR	**68.39±3.57**	65.43±11.3	66.97±6.17	66.24±3.83	61.00±5.61	67.66±0.32	67.80±1.10	66.97±0.56
ENZYMES	54.33±6.11	43.83±6.85	50.50±6.01	17.50±2.67	22.50±7.08	53.43±0.91	**60.40±0.80**	32.35±1.17
PROTEINS	**76.19±2.91**	73.06±4.81	**76.20±2.80**	66.47±1.10	75.90±2.80	75.68±0.54	75.80±0.60	75.60±4.20
IMDB-BINARY	**78.10±5.26**	70.03±0.86	75.10±5.10	71.38±2.08	71.00±2.29	66.96±0.56	72.30±0.60	76.90±3.60
IMDB-MULTI	**53.33±3.89**	47.83±0.85	52.30±2.80	45.02±1.73	45.23±2.84	44.55±0.52	48.70±0.60	52.80±4.60
COLLAB	75.54±2.78	73.76±2.52	80.20±1.90	76.24±0.60	72.60±2.20	73.09±0.25	81.00±0.30	**83.60±1.00**

TABLE 4
Comparison of Classification Accuracy (± Standard Deviation) of DeepMap to Other GNNs With the Same Input of Vertex Feature Maps

Dataset	DeepMap	DGCNN	GIN	DCNN	PatchySan
SYNTHIE	**54.53±6.16**	47.25±7.86	53.68±8.25	50.67±4.41	42.00±10.36
KKI	62.92±7.94	56.25±18.87	**64.93±17.15**	53.93±7.22	48.75±15.26
BZR_MD	**73.55±5.76**	64.33±8.90	73.00±10.70	68.73±3.46	67.33±8.41
COX2_MD	**72.28±9.37**	59.00±9.30	65.76±7.65	61.98±4.99	62.00±10.13
DHFR	**85.17±2.19**	79.33±5.56	80.16±5.27	76.51±6.47	71.00±16.76
NCI1	**83.07±1.07**	71.05±2.03	75.38±2.03	77.34±0.98	80.14±1.58
PTC_MM	**69.59±7.39**	61.21±12.27	68.40±7.78	64.64±2.74	62.00±7.69
PTC_MR	**67.73±6.61**	54.12±7.74	64.87±8.41	57.57±4.26	58.88±8.19
PTC_FM	**65.16±5.62**	58.53±6.86	61.89±8.54	57.78±4.07	58.38±5.09
PTC_FR	**68.39±3.57**	65.43±11.38	66.08±5.99	62.99±4.17	58.25±8.81
ENZYMES	**54.33±6.11**	35.33±5.02	37.50±3.59	42.75±1.81	25.17±5.19
PROTEINS	76.19±2.91	**76.58±4.37**	75.10±5.04	65.55±3.36	65.50±6.80
IMDB-BINARY	**78.10±5.26**	69.20±5.73	74.10±3.18	74.55±2.50	68.70±5.27
IMDB-MULTI	**53.33±3.89**	47.67±4.41	49.87±3.14	48.32±3.40	43.33±7.25
COLLAB	75.54±2.78	73.5±2.1	71.68±2.10	**76.50±1.26**	72.38±2.18

and NCI1, RetGK outperforms DeepMap. On the dataset COLLAB, GNTK is better than DeepMap. However, DeepMap dramatically outperforms the worst method DCNN with a gain of 210.5 percent. On the dataset COX2_MD, DeepMap has a gain of 9.6 percent over the second-best method GIN and has a gain of 40.9 percent over the worst method DCNN.

In the next experiment, we input the vertex feature maps to other GNNs. Table 4 shows the classification accuracies. We want to investigate if DeepMap has a better architecture for vertex feature maps. Even with the same inputs as DeepMap, all the other GNNs cannot defeat DeepMap in most cases. On the dataset KKI, GIN achieves the best classification result, with a gain of 3.2 percent over DeepMap. On the dataset PROTEINS, DGCNN is slightly better than DeepMap. On the dataset COLLAB, DCNN achieves the best result.

5.4 Runtime

Table 5 shows the average runtime of each epoch of DeepMap and other GNNs on the real-world datasets. DeepMap is competitive to other GNNs. PTC_MM, PTC_MR, PTC_FM and PTC_FR are four similar datasets. The runtime of DeepMap on these four datasets are differing because DeepMap uses different kinds of vertex feature maps and their dimensions are different. For datasets NCI1, ENZYMES, IMDB-BINARY and IMDB-MULTI, DeepMap performs the worst because the vertex feature maps built on the shortest-path or subtree patterns are of high dimension. Each epoch of GIN costs over 1s because GIN uses five layers of MLPs (multilayer perceptrons) that are hard to train.

TABLE 5
Runtime of Each Epoch of DeepMap and Other GNNs

Dataset	DeepMap	DGCNN	GIN	DCNN	PatchySan
SYNTHIE	166.7ms	313.5ms	1.4s	338.5ms	566.0ms
KKI	428.8ms	61.5ms	1.1s	63.1ms	343.9ms
BZR_MD	99.2ms	224.0ms	1.1s	93.3ms	366.0ms
COX2_MD	106.9ms	200.5ms	1.2s	95.0ms	367.8ms
DHFR	564.2ms	442.5ms	1.2s	375.8ms	654.1ms
NCI1	7.3s	3.0s	1.6s	3.4s	2.5s
PTC_MM	104.3ms	212.5ms	1.1s	138.3ms	381.2ms
PTC_MR	213.0ms	212.5ms	1.1s	148.1ms	390.5ms
PTC_FM	430.3ms	217.5ms	1.1s	147.2ms	382.9ms
PTC_FR	121.1ms	219.5ms	1.1s	143.8ms	385.0ms
ENZYMES	9.9s	359.5ms	1.2s	279.1ms	530.6ms
PROTEINS	334.1ms	727.5ms	1.2s	1.2s	887.2ms
IMDB-BINARY	2.9s	638.0ms	1.2s	514.0ms	932.8ms
IMDB-MULTI	2.6s	882.0ms	1.3s	665.7ms	1.1s
COLLAB	8.4s	6.3s	3.8s	10.4s	4.1s

6 Discussion

Similar to PatchySan, our method DeepMap also imposes an order for graph vertices to make alignments across graphs. However, DeepMap is different from PatchySan in three aspects: (1) DeepMap adopts eigenvector centrality to impose an order for graph vertices, which is more efficient than Nauty used in PatchySan. (2) PatchySan samples a number (equals to the average degree) of vertices from graphs to construct a vertex sequence. DeepMap uses all the vertices in a graph to generate a vertex sequence. Compared with PatchySan, DeepMap makes full use of all the vertex information in a graph. (3) The input to PatchySan is the one-hot encoding of each vertex label, while the input to DeepMap is the vertex feature map built on the graphlet, shortest-path, or subtree patterns. Compared with the one-hot encoding of each vertex label, vertex feature maps include richer information. The disadvantage of using the vertex feature map is that the dimension may be very high and it leads to low efficiency for CNNs.

As discussed in Section 2, DeepTrend 2.0 maps a sensor network to an image. Neighboring sensors may not be mapped to neighboring pixels. Differing from DeepTrend 2.0, DeepMap maps a graph into a vertex sequence. Then, for each vertex in the vertex sequence, DeepMap decides its receptive field using BFS on the original graph. All the vertices in a receptive field are neighboring vertices in the original graph. One problem with this formalization is that the size of the input vertex sequence into CNNs is r times that of the original graph. This may also cause the low-efficiency problem. DeepMap uses a summation layer as a readout function for the whole graph. The sum function loses the

local distribution of each deep vertex feature map. A possible alternative is to use a concatenation layer that concatenates all the deep vertex feature maps into a vector. DeepMap is built on the hand-crafted vertex feature maps used in graph kernels. It is not an end-to-end framework. Recently, researchers have been focusing on deriving neural architectures from graph kernels [46]. It is very interesting to research on this direction, designing an end-to-end neural learning architecture that is inspired from the mechanisms of graph kernels. Another interesting direction is to design a new graph neural network that can realize different levels of embeddings, including node level, edge level, group level and graph level.

As discussed before, R-convolution graph kernels just decompose graphs into substructures and compare these substructures. Thus, they cannot capture the high-order complex interactions between vertices. For example, the random walk graph kernel [5], [47], [48] conduct random walk on each vertex in two graphs. Each random walk is denoted as a string of node labels and edge labels. Then, the random walk graph kernels just count the number of common random walks (the same strings of node labels and edge labels) in these two graphs. Because random walk is conducted on the first-order transition matrix of the graph structure, the random walk graph kernel cannot capture the high-order complex interactions between vertices. To this end, one possible extension is to conduct random walk on the high-order transition matrix of the graph structure. We leave this for a future work.

7 Conclusion

In this paper, we have proposed the deep map models to learn deep representations for graphs. DeepMap extends CNNs from images to graphs of arbitrary shape and size, by solving the problems of vertex alignment across graphs and vertex receptive field generation. DeepMap can be built on the vertex feature maps of any substructures. By resolving the two main problems that derived from R-convolutional graph kernels, DeepMap dramatically improves the performances of R-convolutional graph kernels and also outperforms several state-of-the-art graph neural networks. The learned deep feature map of each vertex can also be considered as vertex embedding and used for vertex classification. In the future, we would like to develop new architectures that integrate the mechanisms of graph kernels for graph neural networks.

Acknowledgments

The authors would like to thank anonymous reviewers for their constructive and helpful comments. This work was supported partially by the US 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] D. Haussler, "Convolution kernels on discrete structures," Dept. Comput. Sci., Univ. California, Santa Cruz, CA, USA, *Tech. Rep. UCSC-CRL-99–10*, 1999.

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

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

[4] N. Shervashidze, P. Schweitzer, E. J. V. Leeuwen, K. Mehlhorn, and K. M. Borgwardt, "Weisfeiler-Lehman graph kernels," *J. Mach. Learn. Res.*, vol. 12, no. Sep., pp. 2539–2561, 2011.

[5] S. V. N. Vishwanathan, N. N. Schraudolph, R. Kondor, and K. M. Borgwardt, "Graph kernels," *J. Mach. Learn. Res.*, vol. 11, no. Apr., pp. 1201–1242, 2010.

[6] 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.

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

[8] W. Ye, Z. Wang, R. Redberg, and A. Singh, "Tree++: Truncated tree based graph kernels," *IEEE Trans. Knowl. Data Eng.*, to be published, doi: 10.1109/TKDE.2019.2946149.

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

[10] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" in *Proc. Int. Conf. Learn. Represent.*, 2018.

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

[12] M. Zhang, Z. Cui, M. Neumann, and Y. Chen, "An end-to-end deep learning architecture for graph classification," in *Proc. AAAI Conf. Artif. Intell.*, 2018, pp. 4438–4445.

[13] D. K. Duvenaud *et al.*, "Convolutional networks on graphs for learning molecular fingerprints," in *Proc. 28th Int. Conf. Neural Inf. Process. Syst.*, 2015, pp. 2224–2232.

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

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

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

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

[18] 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.

[19] L. Bai, L. Rossi, A. Torsello, and E. R. Hancock, "A quantum Jensen–Shannon graph kernel for unattributed graphs," *Pattern Recognit.*, vol. 48, no. 2, pp. 344–355, 2015.

[20] F. Johansson, V. Jethava, D. Dubhashi, and C. Bhattacharyya, "Global graph kernels using geometric embeddings," in *Proc. 31st Int. Conf. Mach. Learn.*, 2014, pp. II-694–II-702.

[21] 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.

[22] S. S. Du, K. Hou, B. Póczos, R. Salakhutdinov, R. Wang, and K. Xu, "Graph neural tangent kernel: Fusing graph neural networks with graph kernels," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2019, pp. 5724–5734.

[23] A. Jacot, F. Gabriel, and C. Hongler, "Neural tangent kernel: Convergence and generalization in neural networks," in *Proc. 32nd Int. Conf. Neural Inf. Process. Syst.*, 2018, pp. 8580–8589.

[24] S. Arora, S. S. Du, W. Hu, Z. Li, R. R. Salakhutdinov, and R. Wang, "On exact computation with an infinitely wide neural net," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2019, pp. 8141–8150.

[25] J. Atwood and D. Towsley, "Diffusion-convolutional neural networks," in *Proc. 30th Int. Conf. Neural Inf. Process. Syst.*, 2016, pp. 2001–2009.

[26] M. Henaff, J. Bruna, and Y. LeCun , "Deep convolutional networks on graph-structured data," 2015, *arXiv:1506.05163*.

[27] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proc. Int. Conf. Learn. Representations*, 2016.

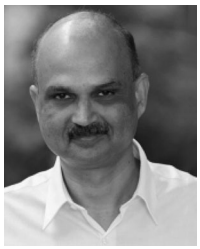
[28] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Proc. 30th Int. Conf. Neural Inf. Process. Syst.*, 2016, pp. 3844–3852.

[29] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," in *Proc. Int. Conf. Learn. Representations*, 2018.
[30] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," in *Proc. Int. Conf. Learn. Representations*, 2014.
[31] D. K. Hammond, P. Vandergheynst, and R. Gribonval, "Wavelets on graphs via spectral graph theory," *Appl. Comput. Harmon. Anal.*, vol. 30, no. 2, pp. 129–150, 2011.
[32] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 1024–1034.
[33] B. D. McKay and A. Piperno, "Practical graph isomorphism, II," *J. Symbolic Comput.*, vol. 60, pp. 94–112, 2014.
[34] X. Dai *et al.*, "DeepTrend 2.0: A light-weighted multi-scale traffic prediction model using detrending," *Transp. Res. Part C: Emerg. Technol.*, vol. 103, pp. 142–157, 2019.
[35] F. Harary, *Graph Theory*. Reading, MA, USA: Addison-Wesley, 1969.
[36] K. He and J. Sun, "Convolutional neural networks at constrained time cost," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2015, pp. 5353–5360.
[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] 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.
[39] S. Pan, J. Wu, X. Zhu, G. Long, and C. Zhang, "Task sensitive feature exploration and learning for multitask graph classification," *IEEE Trans. Cybern.*, vol. 47, no. 3, pp. 744–758, Mar. 2017.
[40] 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. Chem. Inf. Comput. Sci.*, vol. 43, no. 6, pp. 1906–1915, 2003.
[41] 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.
[42] N. Kriege and P. Mutzel, "Subgraph matching kernels for attributed graphs," in *Proc. 29th Int. Conf. Mach. Learn.*, 2012, pp. 291–298.
[43] K. M. Borgwardt, C. S. Ong, S. Schönauer, S. Vishwanathan, A. J. Smola, and H.-P. Kriegel, "Protein function prediction via graph kernels," *Bioinformatics*, vol. 21, no. suppl_1, pp. i47–i56, 2005.
[44] J. Leskovec, J. Kleinberg, and C. Faloutsos, "Graphs over time: Densification laws, shrinking diameters and possible explanations," in *Proc. 11th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2005, pp. 177–187.
[45] S. Milgram, "The small world problem," *Psychol. Today*, vol. 2, no. 1, pp. 60–67, 1967.
[46] T. Lei, W. Jin, R. Barzilay, and T. Jaakkola, "Deriving neural architectures from sequence and graph kernels," in *Proc. 34th Int. Conf. Mach. Learn.*, 2017, pp. 2024–2033.
[47] 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.
[48] H. Kashima, K. Tsuda, and A. Inokuchi, "Marginalized kernels between labeled graphs," in *Proc. 20th Int. Conf. Mach. Learn.*, 2003, pp. 321–328.

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

Omid Askarisichani received the BSc degree in computer engineering, in 2011, and the MSc degree in artificial intelligence from the Sharif University of Technology, Tehran, Iran, in 2014. He is currently working toward the PhD degree in the Dynamo Lab, University of California, Santa Barbara, California. Prior to joining Dynamo Lab in 2015, he spent few years as a software engineer in industry. His research interests include complex networks, analysis of financial data, and applied machine learning.

Alex Jones received the BSc degree in computer science and engineering with a minor in mathematics from the University of Southern California, Los Angeles, California, in 2015. He is currently working toward the PhD degree in the Dynamo Lab, University of California, Santa Barbara, California. He is interested in dynamic network representations of social systems and how algorithms and optimization theory can be used to better understand, drive, and measure these systems.



Ambuj Singh received the PhD degree from the University of Texas at Austin, Austin, Texas, in 1989. He is a professor of computer science with the University of California, Santa Barbara. He joined UCSB's Computer Science Department after his PhD. 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.