DGSD: Distributed graph representation via graph statistical properties

Anwar Said [a], Saeed-Ul Hassan [a,*], Suppawong Tuarob [b], Raheel Nawaz [c], Mudassir Shabbir [a]

[a] *Department of Computer Science, Information Technology University, Lahore, Pakistan*
[b] *Faculty of Information and Communication Technology, Mahidol University, Thailand*
[c] *Department of Operations, Technology, Events and Hospitality Management, Manchester Metropolitan University, United Kingdom*

A R T I C L E I N F O

Article history:
Received 14 July 2020
Received in revised form 6 January 2021
Accepted 13 February 2021
Available online 20 February 2021

MSC:
00-01
99-00

Keywords:
Graph embedding
Distributed computing
Batch processing
Graph classification

A B S T R A C T

Graph encoding methods have been proven exceptionally useful in many classification tasks — from molecule toxicity prediction to social network recommendations. However, most of the existing methods are designed to work in a centralized environment that requires the whole graph to be kept in memory. Moreover, scaling them on very large networks remains a challenge. In this work, we propose a distributed and permutation invariant graph embedding method denoted as *Distributed Graph Statistical Distance* (DGSD) that extracts graph representation on independently distributed machines. DGSD finds nodes' local proximity by considering only nodes' degree, common neighbors and direct connectivity that allows it to run in the distributed environment. On the other hand, the linear space complexity of DGSD makes it suitable for processing large graphs. We show the scalability of DGSD on sufficiently large random and real-world networks and evaluate its performance on various bioinformatics and social networks with the implementation in a distributed computing environment.

1. Introduction

Encoding the graph-structured data (also known as graph embedding) is an important research topic where the primary goal is to represent a graph into a fixed-length feature-vector [1]. Graph embedding methods not only facilitate machine learning over networks but also enable us to find a similarity between them, e.g., inexact graph matching [2]. It benefits a wide range of applications, including molecule toxicity prediction, brain networks comparison, topic prediction from online social networks, link prediction, and solving graph analytic problems. For example, given a brain network constructed from neurons and their physical connections, we can perform disease classification [3]. Such problems fall under the well-established category of graph classification where a machine learning model is trained to differentiate between graphs among different classes [1].

In the era of big data, the graph-structured data is massive in size and is increasing exponentially due to a rise in the number of objects and data produced by the individual object. Current online social networks and biological networks are the prominent examples where a large number of entities involve in a single process [4]. Processing and computing representation from these large networks on a single machine with limited memory is not feasible and there is an increasing need for graph representation methods that can execute efficiently in parallel on different machines [5]. One possible solution that suffices the need is the distributed computing that can come into play to cope with gigantic graphs. Unfortunately, most of the research in graph representation has been focused on centralized algorithms that require the *whole network to be kept in memory* such as the Graph Kernels [6,7] and Graph Convolutional Networks (GCNs) [8–10]. In addition to the memory requirements, they also face many challenges. For example, the kernel methods are computationally expensive and do not provide desired results in various scenarios. A prominent example is the well-known Weisfeiler–Lehman (WL) kernel method [11] that fails on regular graphs as it begins by partitioning the vertices according to vertex valency [12]. Whereas GCNs require node attributes to learn from the network structure and have limited scalability [13]. Other recent statistical representation methods are either computationally demanding or having low discrimination power especially when the structure of the networks is significantly dense [14].

A desired property to encode large graphs is independent and parallel execution of the embedding method on different distributed machines. However, encoding nodes' global positions in

* Corresponding author.
E-mail addresses: anwar.said@itu.edu.pk (A. Said), saeed-ul-hassan@itu.edu.pk (S.-U. Hassan), suppawong.tua@mahidol.edu (S. Tuarob), r.nawaz@mmu.ac.uk (R. Nawaz), mudassir.shabbir@itu.edu.pk (M. Shabbir).

https://doi.org/10.1016/j.future.2021.02.005
0167-739X/© 2021 Elsevier B.V. All rights reserved.

the embedding space requires sufficient information of the nodes — up to several-hop neighbors, e.g., GCNs and graph kernels. It is accepted that the key aspect to extract graph representation is to consider the neighborhood of the nodes [1,15]. Neighborhood information allows us to locate nodes in the embedding space. Naturally, a node u is closed to node v than the node w, if v has direct connectivity with u as well as more common neighbors. This observation motivates us to search for a distance-measure on graphs that captures nodes' position in the embedding space. This search leads us to discover a simple but powerful statistical representation method; *Distributed Graph Statistical Distance* (DGSD) that preserves nodes' position in the embedding space. The DGSD leverages nodes' local neighborhood information using node degree, common neighbors, and direct connectivity to capture their position in the graph. On one hand, DGSD is a distributed graph encoding method that extracts feature-vectors from graphs using nodes' pair-wise distances in a distributed fashion. This is due to the fact that only the information of common neighbors, adjacency and degree are necessary for each node to find its position in the embedding space. On the other hand, its linear space complexity allows the scalability on sufficiently large graphs. Fig. 1 demonstrates the pipeline of DGSD.

The work in this study only focuses on undirected graphs but can also be extended to directed graphs. Our comprehensive empirical results on various datasets show that the DGSD representations are powerful enough to provide better accuracy results against state-of-the-art algorithms, even though DGSD uses no graph meta information and rely only on graph structural data. The main contributions of this study are as follows:

- We propose DGSD, a scalable and distributed graph representation technique to encode large and arbitrary sized graphs in a distributed environment with parallel processing.
- We show scalability and expressiveness of the proposed algorithm on the graph classification task using large scale empirical analysis on several real-world and synthetic network datasets. The results showed an improved performance on several datasets against state-of-the-art methods.
- We implemented DGSD in multiprocessing environment in Python and C programming languages and made both the implementations publicly available to foster reproducibility of the results.

The rest of the paper is organized as follows. Section 2 provides an overview of the graph representation approaches. Section 3 introduces the DGSD's based graph representations on both distributed and centralized environments. Section 4 presents the DGSD evaluation on various real-world and synthetic network datasets and Section 5 concludes the paper.

2. Related work

Enormous applications and the empirical success of graph embedding methods have attracted a lot of interest from the scientific community. Due to that, a large body of work exists. In this section, we overview state-of-the-art graph representation methods developed for the task of graph classification. In particular, we focus on the graph kernels, statistical and spectral representation approaches that use graph theoretical properties for extracting graph representations.

Generally, a kernel $k(\pi, \pi')$ is a measure of similarity between objects π and π', that satisfies two main requirements: it must be positive semi-definite and symmetric, that is $k(\pi, \pi') = k(\pi', \pi)$ [6,16–22]. The authors in [11] introduced one of the powerful graph kernel methods known as Weisfeiler–Lehman that works on vertex color refinement. Initially, it assigns colors to each node based on the vertices' degrees — vertices having the same degrees will have the same color. And then recursively refines vertices' colors concerning their neighbor colors. Despite its successful results on various benchmark datasets, this method cannot distinguish regular graphs. Neighborhood Hash Kernel (NHK) assumes labeled nodes and quantifies graphs by updating their node labels and counting the number of common labels between them. The deep graph kernel [18] compares graphs based on the number of subgraphs or motifs and employs the word embedding model. However, extracting subgraphs or motifs is a computationally expensive task that makes it impractical on significantly large networks. The shortest path kernel [16] is another well-known graph kernel that encodes and compares graphs based on the shortest paths between all pairs of vertices. Similarly, the random walk kernel [21] quantifies graphs with respect to the number of common walks among them.

Higher-order proximity is an important aspect to embed graphs into vector spaces, as it requires both local and global-level information of the graph. Therefore, most of the existing works use global-level graph-theoretic measures for graph embedding. FGSD [7] constructs graph representations as a histogram from nodes' pair-wise distances computed from the graph spectrum where the graph spectrum contains both local and global level information. Similarly, NetLSD [14] considers a heat diffusion process on a graph and constructs feature vector using Laplacian spectrum. Recently, the authors in [6] use the Wasserstein distance between the node feature distributions to distinguish graphs. The authors in [23] use graph simple statistical properties such as average degree, clustering coefficient and other nodes' centrality measures for encoding graphs. For graph comparison, Maretic et al. recently propose using the Wasserstein distance which uses the distribution of smooth graph signals, and comparing them using the Wasserstein distance [24]. A similar framework based on the optimal transport theory and discrete graph matching in a continuous domain is proposed in [25,26]. We can see that all of these measures use global-level graph measures to compute graph embeddings where the embeddings are based on nodes distances or similarities. Such recent approaches have shown promising results on the graph classification task; however, they require the entire graph to be loaded in memory. To process huge graphs, recently, the authors in [27] have proposed a streaming algorithm that approximates feature vectors by estimating counts of sub-graph. The algorithm does not require the whole graph to be kept in memory, but instead processes it in batches. More recently, NetKI [28], a nearly linear time graph descriptor has been proposed. NetKI is based on the idea of network Kirchhoff index to extract representations from graphs and scalable on sufficiently large graphs. Other popular graph comparison approaches include [29–33].

Recently, the graph neural network models have also been successfully applied on graphs [34–36]. There has been a surge in such approaches in the last few years that work both on the node classification [10] and graph classification task [37,38]. We refer the reader to [1,39] for further reading on graph neural network models.

3. Methodology

The notion of distributed computing systems is a useful and widely adopted tool for parallel processing [40]. Generally, it refers to a group of independent computers, each having its own memory and operating system, that communicate with each other over a network to solve a problem collectively. Distributed computing has several advantages over centralized computing, for example, enhanced reliability, resource sharing, and increased performance. Currently, the two main bottlenecks of centralized

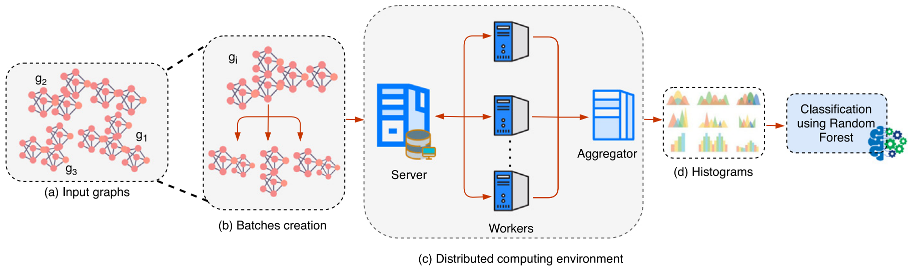

Fig. 1. Architecture diagram of DGSD. (a) a set of labeled graphs is given is an input to the DGSD. In (b), batches of approximately equal size are created from each graph and send them to the distributed computing environment where a single batch is assigned to each worker. In (c), a distributed computing environment is shown where m workers are communicating with a server and an aggregator machine. Worker machines communicate with server to process the given batch and once it completes the processing and constructs the feature vector $\mathcal{R}$ (histogram), it forwards it to the aggregator machine. The aggregator machine receives all the desired histograms from the workers, and aggregate them to create a single feature vector (for each graph). In the end, a Random Forest classifier is applied to the resultant feature matrix to perform the graph classification task.

computing are the space and computing power. The problem becomes even harder when the issue of graph representations comes into play — due to the excessive need for space and computational resources. The existing approaches require a large amount of time to generate representations while most of them are even not scalable on moderate size of networks.

In this paper, we propose to use the framework of distributed computing systems for encoding large graphs in parallel using multiple machines with limited storage. In particular, we consider the following realistic conditions while encoding graphs:

- **Independent architecture:** each worker independently processes the input vertices and communicates only with the central controller. No worker can access the data stored on other machines.
- **No knowledge of the input graph:** the workers have no prior information about the input graph.
- **Process once:** each vertex is processed only once by a single worker.
- **Deleting past information:** Past vertices' information is deleted and cannot be accessed again.
- **Limited storage:** Each worker machine has a limited memory; in particular the memory is not large enough to store the graph.

Considering the above conditions, in the following section, we define basic notations and the problem of distributed graph representation for the graph classification task.

3.1. Basic setup and notations

Let $G = (V, E)$ be an undirected graph with a set of vertices $V = \{1, 2, 3, \ldots, n\}$, and a set of edges $E \subseteq V \times V$. The adjacency matrix $\mathcal{A}$ of G is an $n \times n$ matrix where, $\mathcal{A}[i, j] = 1$ if $(i, j) \in E$, and 0 otherwise. The neighborhood, $N(i)$, of a vertex $i \in V$ represents a set of vertices that are adjacent to i, while $\deg(i)$ indicates the size of it's neighborhood.

A Degree Sum Matrix, $\mathcal{D}$, for a given graph is $n \times n$ symmetric matrix where each entry represents the sum of degrees of the corresponding pair of vertices in G, i.e., :

$$\mathcal{D}[i, j] = \begin{cases} \deg(i) + \deg(j) & \text{if } i \neq j \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

Let $N(i, j)$ denote the common neighborhood of the vertices i and j, which is the set of vertices different from i and j, that are adjacent to both the vertices, i.e., $N(i, j) = N(i) \cap N(j) \backslash \{i, j\}$. Let $\Gamma_{n \times n}$ represent the common neighborhood matrix of the graph G where each entry is defined as

$$\Gamma[i, j] = \begin{cases} |N(i, j)| & \text{if } i \neq j \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

Γ is a hollow matrix where all the diagonal elements are zero. The off-diagonal elements are assumed to be integer values ranging from 0 to $n - 2$.

3.2. DGSD based graph representation

In this section, we first describe and motivate DGSD graph embedding that uses the multiset of node pair-wise statistical distances. Then, we present two graph representation algorithms, one for the distributed computing environment and the other for the centralized environment. Lastly, we theoretically show the uniqueness, time and space complexities of DGSD algorithm in both the distributed and centralized cases.

Problem definition: In this work, we are interested in encoding large graphs in a distributed environment into a Euclidean space where the graphs cannot be processed on a single machine due to efficiency or memory limitation. For our distributed algorithm, we will assume m independent machines (workers) connected to a central controller machine, and an aggregator machine. We consider a distributed computing model where workers communicate directly to the central controller, and each of m workers has a limited storage capacity. Given a collection of graph-label pairs, $\mathcal{C} = \{(G_1, y_1), (G_2, y_2), \ldots, (G_l, y_l)\}$, the goal is to find a function $\phi : G \rightarrow \mathbb{R}^d$, that can map a given graph into a low dimensional feature vector, h_G, in a distributed environment. Here, although we consider undirected graphs, our measure can be easily extended to directed graphs as well. We also aim to preserve vertices' local and global information in the embedding space and ensure that we always have the same representations for isomorphic graphs.

Graph representation methods usually rely on either a kernel function, node pair-wise distance or end-to-end learning [7,16,36]. Among them, pair-wise distance measures have shown promising results in the last few years. These are tractable measures having theoretical guarantees and produce very accurate classification results. However, the existing methods use global scale measures such as the graph spectrum [7,14], Kirchhoff Index, Earth-mover distance, and graph compression methods [24,41] that require the whole graph to be kept in memory, and thus limit their ability to scale to large graphs.

Most of the real-world social networks follow the phenomenon of six-degree-of-separation where it is possible to connect any two pairs of vertices with a few links [42]. This is true about other networks as well, such as molecular networks and neural networks. Because of the existence of strong ties and the natural dense connectivity of these networks, their diameters are usually small, i.e. $\leq$ 6. We note that many of the state-of-the-art benchmark datasets for graph classification also follow the same phenomenon. To illustrate this, we show the count

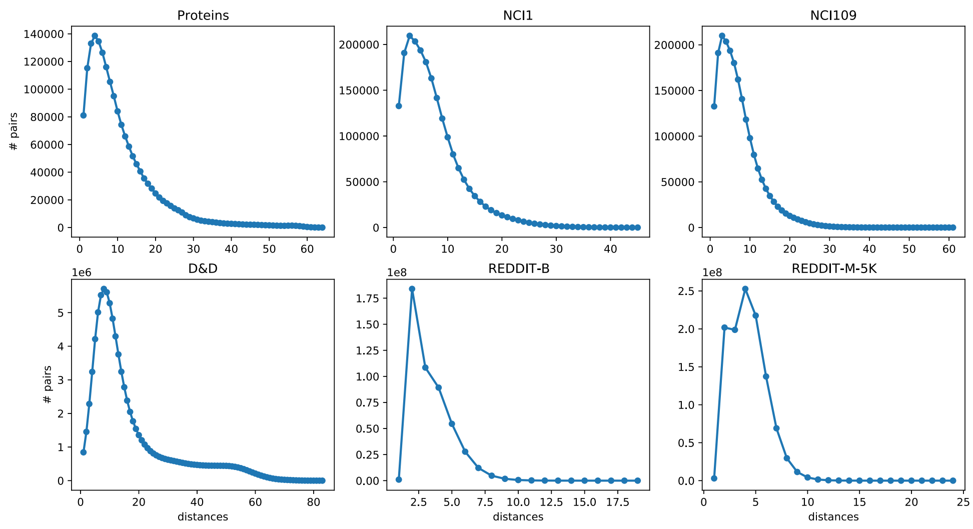

Fig. 2. Count of shortest path pair-wise distances on six graph classification benchmark datasets having number of graphs $\geq$ 1000. X-axis shows the shortest-path distances while y-axis represents the count of pairs found against each distance x.

of pair-wise shortest path distances on six well-known graph classification benchmark datasets in Fig. 2. On the y-axis, the count of vertex pairs corresponding to the shortest path distances on x-axis has been shown. We empirically found 46% pairs in bioinformatics datasets and 93% pairs in social network datasets lie in range $\leq$ 4. We conclude that for the most part, in real-world graphs, topological information is concentrated among pairs of vertices that are just a few hops away from each other. We use this observation while designing our graph embeddings. This increases the computational efficiency of the algorithm, enabling it to run in a distributed environment.

Here we present DGSD, a novel distributed graph representation method that extracts expressive representation from graphs. It encodes graphs using vertex pair-wise statistical distances. DGSD is a graph distance measure that relies on three different nodes' statistics; nodes' degrees, common neighborhood and direct connectivity. The combination of these measures has been widely adopted in many graph analytic problems such as community detection [43,44] and representation methods [45]. The main idea behind DGSD is that the vertices that share many common neighbors, $\Gamma(i, j)$, with respect to the sum of their degrees, $\deg(i) + \deg(j)$, and have direct connectivity, δ, should be closed to each other in the embedding space. The definition of DGSD is as follows.

DGSD definition: for $i, j \in V$, we define the distance between i and j on G in distributed environment as follows:

$$\mathcal{S}[i, j] = \frac{|N(i)| + |N(j)|}{|N(i)| + |N(j)| + |N(i, j)| + \delta} \tag{3}$$

Here δ is a Kronecker delta which is 1 if i and j are adjacent otherwise 0. We can easily see that the lower bound of DGSD distances is 2/3 if $i \neq j$ otherwise 1 in the upper case. The lower value of DGSD distance indicates a smaller distance (high robustness) between pairs of vertices, while 1 indicates the highest distance or dissimilarity between the pairs. To illustrate this, consider the example in Fig. 3.

We consider the green vertex as a source and show distances to a few other vertices. From green to dark blue vertex, there are two common neighbors; blue and gray and has a direct link, therefore, the distance between them is 0.7. Similarly, with the blue vertex, there is one common neighbor and a direct link,

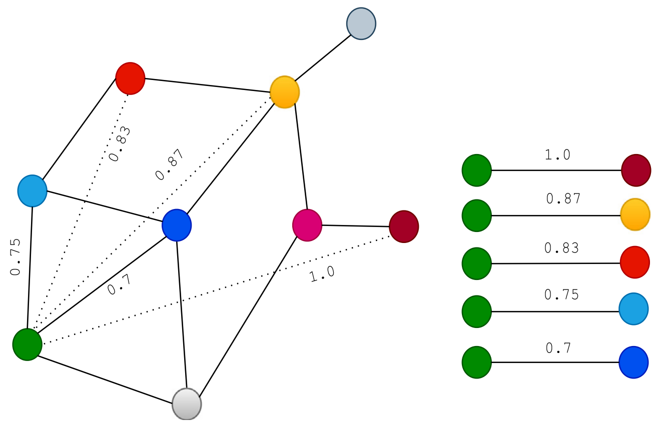

Fig. 3. DGSD based distances shown from green vertex to few other vertices in the graph. The vertices share more common neighbors and direct links have the minimum distance.

the distance is increased to 0.75. We see only a slight increase in the distance indicating the robust connection between the two vertices. We note the increase in the distance, 0.83 from 0.75, with the red vertex, because of the removal of the direct link between them. The distance is increased further to 0.87, with the yellow vertex, with the increment in the degree of the yellow vertex. With the brown vertex, there is no direct link and common neighbors, and thus distance is 1, which implies the less robust or dissimilarity between the vertices. We note that DGSD maintains vertices' positions with respect to their local neighborhood. If the vertices have no common neighbors and are not adjacent, the distance is maximum. This nature of DGSD allows us to deploy it in a distributed environment where we only need the neighbors of the corresponding two vertices to find the distance between them. We denote the multiset of distances of G by $\mathcal{S}$. DGSD extracts local and global structure information from graphs. The extraction of local proximity can be seen from considering the local neighbors' information such as common neighbors, nodes' degree and direct connectivity. This allows the method to encode the local position of the vertices. On the other hand, for vertices that are not adjacent and have no common neighbors, DGSD assigns maximum distance, i.e, 1, that helps to extract the global structure information. Motivated by the fact that DGSD encodes nodes' local and global structure information, we use $\mathcal{S}$ for the graph classification task, where it requires to perform comparison among graphs. Based on $\mathcal{S}$, we define the graph embedding as a histogram $\mathcal{R}$ of the multiset $\mathcal{S}$. Thus, the comparison of $\mathcal{R}_{G_1}$ and $\mathcal{R}_{G_2}$ implicitly evaluates the similarity between G_1 and G_2.

We describe DGSD for the distributed environment in Algorithm 1. There are three main components of the DGSD graph representation: the central controller, workers and the aggregator. Here, we first describe the central controller which takes the input graph from the source. In step 1, the central controller converts vertices' labels to integers and creates batches $\mathcal{B}$ of vertices from the input graph where the size of each batch is set to $\mathcal{B}_i = |V|/m$. The number of batches is decided according to the number of workers, m, connected with the central controller. Next, the controller broadcasts the batches to the workers — each batch to a single worker. The controller listens to workers to entertain their queries. In order to reduce the inter-process communication time, we use centralized files that keep nodes' neighborhood information.

Each worker m_i initially receives a batch $\mathcal{B}$ from the controller. Since DGSD requires only local information, it implies the necessity of only neighbor information of the corresponding vertices. Thus, neighbors for each node are required to compute nodes' pairwise distances. In step 5, the worker requests the controller to compute the total vertex count, n, of the graph and iterates on the received batch $\mathcal{B}$ in step 7. For each vertex i, the worker reads i's neighbors from the centralized files, iterates over all the vertices and computes $N(i,j)$. Step 12 uses Eq. (3) to compute the distance between i and j. Once the distance is computed, the worker deletes j's neighbors and appends the distance to the list l. At the end, the worker computes the required histogram and sends it to the aggregator. The aggregator machine receives histograms from all the workers and aggregates (sum) them to compute the final representation $\mathcal{R}$. One might observe that only adjacent vertices are considered for computing distances between vertices, which can extract only local neighborhood information. However, we note that we consider computing the distance of each vertex with every other vertex. So vertices in closed proximity will have small distances and 1 otherwise. This ensures preserving global information of vertices in the embedding space in quadratic time complexity. One can also consider multiple-hop neighbors for computing the distances, however, we can observe that it directly increase the complexity to exponential which makes the algorithm intractable for large-scale graphs even in a distributed environment.

Algorithm 1 Compute DGSD representation in distributed environment

```
Input: $G$, $bins$
    $G = (V, E)$ is undirected, unweighted graph. $bins$ indicate the length of the feature vector
Output: $\mathcal{R}$
    $\mathcal{R}$ is the desired feature vector of the graph $G$
    Central controller
 1: convert nodes' labels to integers and create $m$ batches ($\mathcal{B}$) of approximately equal size
 2: broadcast each batch $\mathcal{B}_{mi}$ to a worker $m_i$
 3: listen workers and entertain their queries
    Worker (each worker $m_i \in m$):
 4: receive $\mathcal{B}_{mi}$ from the controller
 5: $n \leftarrow$ get vertex count from the controller
 6: initialize an empty list $l$
 7: for $i \leftarrow 0$ to $|\mathcal{B}_{mi}|$ do
 8:     $N(i) \leftarrow$ get $i$'s neighbors from the controller
 9:     for $j \leftarrow 0$ to $n$ do
10:         $N(j) \leftarrow$ get $j$'s neighbors from the file
11:         $\delta \leftarrow 1$ if $j \in N(i)$ otherwise 0
12:         $\mathcal{S}[i,j] \leftarrow \frac{|N(i)|+|N(j)|}{|N(i)|+|N(j)|+|N(i,j)|+\delta}$
13:         $l \leftarrow l \cup \mathcal{S}[i,j]$
14:         delete $N(j)$
15:     end for
16: end for
17: $\mathcal{R} \leftarrow$ histogram($l$, $bins$)
18: broadcast $\mathcal{R}$ to the aggregator
    Aggregator
19: $\mathcal{R}_i \leftarrow$ receive feature-vector $\mathcal{R}_i$ from each of $m_i$ worker
20: for $i \leftarrow 1$ to $|m|$ do
21:     $\mathcal{R} \leftarrow \mathcal{R} \oplus \mathcal{R}_i$        /*element-wise addition*/
22: end for
23: return $\mathcal{R}$
```

3.3. DGSD representations on centralized machine

The interesting fact about DGSD is the easy deployment for both the centralized and distributed computing environments. Similar to the distributed environment, DGSD representations can be easily computed on a single machine using Eq. (3). Using matrix notations, DGSD can also be computed using D and Γ matrices on single machine as follows:

$$\mathcal{S}[i,j] = \frac{D[i,j]}{D[i,j] + \Gamma[i,j] + \delta} \tag{4}$$

Following Eq. (4), we can compute the distance matrix $\mathcal{S}$ for a graph as follows:

$$\mathcal{S} = \frac{D}{D + \Gamma + \mathcal{A} + \mathcal{I}} \tag{5}$$

In the denominator, the adjacency matrix $\mathcal{A}$ is used to consider direct links between pairs of nodes while identity matrix $\mathcal{I}$ is used to avoid zero values on the diagonal. The pseudocode of DGSD representation is presented in algorithm 2.

Algorithm 2 DGSD representations on centralized machine

```
Input: Graph $G = (V, E)$, adjacency matrix $\mathcal{A}$, identity matrix $\mathcal{I}$, $bins$
Output: $\mathcal{R}$
1: compute $D$ and $\Gamma$ matrices
2: $\mathcal{S} \leftarrow \frac{D}{D+\Gamma+\mathcal{A}+\mathcal{I}}$
3: set $h_g \leftarrow \{S[i,j] | \forall (i,j) \in V\}$
4: compute $\mathcal{R} \leftarrow$histogram($h_g$, $bins$)
5: return $\mathcal{R}$
```

Initially, we compute D and Γ matrices from G. In step 2, we compute the distance matrix $\mathcal{S}$ using Eq. (5) and a multiset h_G of $\mathcal{S}$ in step 4. Finally, we create the feature-vector as a histogram $\mathcal{R}$ from the multiset h_G. Note that $\mathcal{R}$ inherits all properties of h_G which is made possible by defining $\mathcal{R}$ as the histogram of h_G. Without loss of generality, we say that $\mathcal{R}_G = \mathcal{R}_{G^\pi}$ or $\mathcal{R}_G = \mathcal{R}(PAP^\intercal)$ under permutation matrix P of node labels or permutation π of vertex labels. Further, the output embedding is sparse which is a desirable property for the machine learning task.

3.4. Uniqueness, time and space complexity of DGSD representations

To analyze the expressiveness and stability of DGSD, here we show its uniqueness in terms of isomorphic graphs. Using D, Γ, $\mathcal{A}$

Table 1
Datasets characteristics. avg. deg represents average degree while d represents diameter of the graphs.

Dataset	$\lvert\mathcal{G}\rvert$	y	$avg.\lvert V\rvert$	$avg.\lvert E\rvert$	$min.\lvert V\rvert$	$max.\lvert V\rvert$	avg. deg	avg. d
Mutag	188	2	17.93	19.79	10	28	2.2	8.21
Proteins	1113	2	39.06	14.69	4	620	3.72	11.55
PTC	344	2	25.56	72.81	2	109	1.02	7.52
AIDS	2000	2	15.58	16.19	2	94	0.066	7.87
NCI1	4110	2	29.87	32.3	3	111	2.04	13.3
NCI109	4110	2	29.56	32.13	4	111	2.172	13.12
D & D	1178	2	284.3	715.65	30	5748	2.51	19.89
COLLAB	5000	3	74.49	2457.21	32	492	56.98	1.864
IMDB-B	1000	2	19.77	97.53	12	136	9.76	1.86
IMDB-M	1500	3	13.00	65.93	7	89	10.14	1.47
REDDIT-B	2000	2	429.61	497.75	6	3782	2.3	9.7
REDDIT-M-5K	4999	5	508.50	594.87	22	3648	2.338	11.96
REDDIT-M-12K	11 929	11	391.41	456.89	2	3782	2.33	10.90
Tox21	14 184	2	18.39	19.32	2	122	2.045	0.15

and $\mathcal{I}$ matrices, one can easily show that DGSD always produces same representations for isomorphic graphs. Assume graphs G_1 and G_2 are isomorphic, then their respective D matrices are equal up to permutation ($D_{G_1} = PD_{G_2}P^\intercal$), and also employ the equality of their Γ and $\mathcal{A}$ matrices up to permutation. Then we can specify $\mathcal{S}_{G_1}$ in terms of $\mathcal{S}_{G_2}$ as follows:

$$P\mathcal{S}_{G_1}P^\intercal = \frac{PD_{G_2}P^\intercal}{P(D_{G_2} + \Gamma_{G_2} + \mathcal{A} + \mathcal{I})P^\intercal} \tag{6}$$

Since $\mathcal{S}$ operates on D and Γ, we have $D_{G_1} = PD_{G_2}P^\intercal$ and $\Gamma_{G_1} = P\Gamma_{G_2}P^\intercal$. This implies $\mathcal{S}_{G_1} = P\mathcal{S}_{G_2}P^\intercal$. The same procedure holds for distributed scenario as well.

The space complexity of DGSD distributed representation on each worker machine is the cost of storing two nodes u and v's neighbors and a cost c for storing the histogram $\mathcal{R}$. Thus, the total space complexity is $O(\deg(i) + \deg(j) + c) = O(\deg(i))$. The space of l is negligible by computing $\mathcal{R}$ automatically. The total running time of DGSD is $O(|\mathcal{B}|n)$, since each worker iterates on a batch of nodes and computes common neighbors for each node. On a centralized machine, the running time complexity of DGSD is $O(n^2)$.

4. Experimental setup and results

We evaluate DGSD in terms of classification accuracy on different benchmark datasets. We also evaluate DGSD's scalability on sufficiently large random and social networks with varying numbers of machines in a distributed environment. For results comparison, we used state-of-the-art graph encoding methods including the recent spectral representation methods; NetLSD, heat $h(g)$ and wave $h(w)$ kernels [14], FGSD [7], the statistical method NetSIMILE [23] and the well-known graph kernels including Shortest Path [16], Neighborhood Hash Kernel (NHK) [31], Edge Histogram Kernel (EHK) [46], and Graphlet Sampling Kernel (GK) [47]. In GNNs, we use the well-known Graph Isomorphism Network (GIN) [37] and the latest GENeralized Graph Convolution (GENConv) [38] models for comparison. The latest spectral representation methods have shown promising results on the graph classification task with low time complexities while the rest are well-known kernel methods. The results are compared on well-known real-world bioinformatics, social networks benchmark datasets and synthetic networks. Bioinformatics datasets include MUTAG, PTC, Proteins, NCI1, NCI109, AIDS, D&D and Tox21 [48]. In social network datasets, six datasets are chosen: COLLAB, IMDB-B, IMDB-M, REDDIT-B, REDDIT-5M and REDDIT-M-12K. Necessary characteristics of these datasets are presented in Table 1.

The datasets we have chosen for the evaluation are state-of-the-art datasets available online on several platforms[1] except Tox21 dataset. Tox21 data challenge dataset [48] is a toxicity prediction challenge dataset. The challenge was initiated to produce highly reliable measurements/datasets that can be used worldwide for validating measures in applications to toxicity prediction. Tox21 challenge composed of twelve sub-challenges divided into two main panels: (1) nuclear receptor (NR) signaling pathways and (2) stress response (SR) pathways. Seven of the sub-challenges dealt with NR while five are related to NR pathways. Each challenge requires the prediction of a different type of toxicity where each sub-task consists of active and inactive pathways. We have considered the balanced and combined version of the dataset described in [49] for the evaluation.

We implemented DGSD in C programming language with MPI (Message Passing Interface)[2] and igraph.[3] libraries. MPI is a well-known library for implementing parallel and distributed programming which ensures efficient distributed mechanisms. We have also implemented DGSD in Python and released a Python package. The DGSD code and package details are made publicly available to encourage the reproducibility of the results.[4]

4.1. Numerical results on real-world datasets

The histograms are generated in the range [0,1] as the maximum possible value is 1 and the minimum is 0, however, range $[2/3, 1]$ can also be set for the histogram generation. The number of bins is chosen from the set $\{20, 100, 200, 500\}$ independently for different datasets. For graph kernel experimentation, we use Grakel[5] library with the default parameter setting. For spectral measures; NetLSD, FGSD, and NetSIMILE, we use the same parameter setting presented in the actual papers and use the same code made available by the authors. For GNNs experiments, we use Pytorch Geometric[6] where the implementations of GINs and GENConv models are available. The number of epochs was set to 350, the learning rate to 0.01, and the number of layers was set to 2. One Hot Degree nodes' feature vectors were considered throughout the experiments to provide a fair comparison against other approaches. We use early stopping criteria with 30 number of epochs and $80-20$ train–test split with 64 batch size was used. We ran all the models 5 times and reported average test accuracy for the stability of the results. For the graph classification task, we use a Random Forest classifier with 500 estimators and reported the classification accuracies accordingly. The same parameter setup is kept for all the experiments on all datasets

1 https://chrsmrrs.github.io/datasets/.

2 http://www.mpich.org/.

3 https://igraph.org/.

4 https://github.com/Anwar-Said/DGSD.

5 https://ysig.github.io/GraKeL/dev/.

6 https://pytorch-geometric.readthedocs.io/en/latest/.

Table 2
Classification accuracy comparison against well-known graph kernels and recent methods. Blue results indicate highest classification accuracy while **bold** indicate results within top 2% of the highest results. >D indicates computations exceed 24 hours. Note that the length of the embedding vectors (number of bins) were chosen from the set {20, 100, 200, 500}.

Dataset	SP	NHK	EHK	GK	NetSIMILE	FGSD	NetLSD		GINs	GENConv	DGSD
							$w(g)$	$h(g)$			
Mutag	**86.60**	85.06	85.37	77.01	83.42	**88.26**	82.40	83.31	88.42	84.74	**87.70**
PTC	59.00	**60.58**	57.54	57.56	55.80	**60.70**	57.22	53.49	62.32	60.0	**61.32**
Proteins	74.12	74.29	59.56	73.22	69.71	70.25	68.10	72.14	76.95	**76.41**	73.68
NCI1	71.65	75.52	50.04	58.12	68.87	79.75	61.94	67.25	68.30	74.60	73.48
NCI109	71.48	75.23	50.37	58.97	67.45	80.44	60.38	64.64	69.47	73.95	72.01
AIDS	**99.24**	**99.2**	**99.60**	**98.75**	**97.95**	**98.5**	93.7	**99.69**	**99.75**	99.8	99.8
D&D	**77.94**	75.81	58.65	$> D$	73.86	75.9	70.21	72.33	73.56	74.79	78.52
Tox21	**73.23**	**73.48**	68.27	**72.86**	**72.96**	**72.87**	**72.54**	**72.62**	63.47	63.20	73.72

Table 3
Graph classification accuracy on social network datasets. Results in **bold** indicate the best reported accuracy while >M indicates memory error. Note that the length of the embedding vectors (number of bins) were chosen from the set {20, 100, 200, 500}.

Dataset	NetSIMILE	FGSD	NetLSD		GINs	GENConv	DGSD
			$w(g)$	$h(g)$			
COLLAB	79.96	77.04	74.46	70.58	73.0	72.3	**79.44**
IMDB-B	74.00	73.5	71.11	71.9	79.4	75.1	75.0
IMDB-M	49.06	49.5	47.73	47.13	51.8	55.87	50.13
REDDIT-B	88.05	**88.95**	77.75	82.74	77.2	75.9	90.3
REDDIT-M-5K	**51.83**	50.2	40.34	40.44	48.72	46.83	53.33
REDDIT-M-12K	44.17	>M	27.97	29.0	>M	>M	46.55

Table 4
Classification accuracy comparison on bioinformatics datasets using custom SVM kernels: KL divergence, Wasserstein distance and the standard RBF SVM kernel.

Dataset	RBF	KL	Wasserstein
Mutag	82.79	66.52	66.52
PTC	56.08	44.19	55.81
Proteins	59.56	41.42	59.57
NCI1	62.92	38.15	50.02
NCI109	62.08	38.45	50.37
AIDS	99.25	80.0	80.0
D&D	71.47	58.66	36.07
Tox21	64.47	49.79	41.18

and algorithms. 10-fold cross-validation is used for evaluating the results. Tables 2 and 3 report the experimental results on various graph classification benchmark datasets.

We can see from the classification accuracy that DGSD produced comparative results against state-of-the-art models. On bioinformatics datasets, DGSD outperformed on AIDS, D&D, and Tox21 datasets, while the results on Mutag and PTC datasets are within 2% (absolute) from the top results. Note that on AIDS dataset, DGSD and GENConv produced the same classification accuracy. On Mutag, PTC, and Proteins datasets, GINs outperformed all other methods while FGSD produced the best results on NCI1 and NCI109 datasets. on social datasets, NetSIMILE outperformed DGSD with a slight improvement on COLLAB dataset, while DGSD performed best on three REDDIT datasets. GINs and GENConv performed best on IMDB-B and IMDB-M datasets respectively. Overall, the results demonstrate that DGSD has shown encouraging results on all the benchmark datasets.

Results on Support Vector Machines (SVM) with custom kernels: KL divergence and Wasserstein distance are well-known methods for measuring similarity between histograms. Since DGSD uses a histogram as a feature vector, we consider KL divergence and Wasserstein distance as custom kernels to evaluate their classification performance on the graph classification task. We initially deployed DGSD to generate feature vectors and then applied SVM instead of the Random Forest algorithm to perform the classification. We use the default parameters $C = 1.0$ and $\gamma =$'scale' for SVM and consider 10-fold cross-validation for the evaluation. We also considered standard RBF kernel for results comparison. The classification accuracies on the graph classification task are reported in Table 4. We can see from the table that the results of RBF kernel are quite closed to the state-of-the-art results presented in Table 2 using Random Forest algorithm. While KL and Wasserstein distance has shown lower results on almost all the datasets. This implies that these methods are not suitable to capture/learn complex boundaries resulting in poor classification accuracies.

4.2. Numerical results on synthetic networks

We also evaluate DGSD on computer-generated networks for graph classification task. We form a binary classification problem and create two classes G and G'. We use the Erdős–Rényi model to generate graphs with different probabilities for both classes. For each experiment, we generate two hundreds number of graphs in each class, where each graph consists of 100 nodes. In each iteration, we set $p = 0.5 - (i * c)$ for one class and $0.5 + (i * c)$ for the other class. The value of c is set to .005 and $i = 0, \dots, 10$. The difference in p for both the classes increases with the increase in i, as it makes one class denser and the other sparser.

In the second experiments, the first class comprises of random graphs while cliques of size k are planted in the second class graphs $G(k)$ apart from the random graphs. We set $p = 1/2$ and $k = 2t\sqrt{n}$ for $t = 0, 0.5, \dots, 4$. For each value of t, we generate N graphs of each class with $n = 100$ for both. The number of landmarks was set to 100 for both tasks.

Fig. 4 shows a comparison of DGSD against NetLSD, FGSD and NetSIMILE in terms of graph classification accuracy. On the $x - axis$ of the left figures, the difference in probabilities has been shown while the classification accuracy is shown on the $y - axis$. We can see that when $p' - p = 0.0$, the graphs of both the classes are the same and thus the classification accuracies are random; either 40 or 60% of all the methods. However, as the difference in probabilities increases among the classes, the classification accuracies of the algorithms also increase. We can observe from the results that only difference of $p' - p = 0.1$ in both the classes, all the methods have shown 100% accuracies where DGSD also competes well against state-of-the-art methods. This indicates the expressiveness of the embeddings generated by DGSD. Similarly, in the right figure, different sizes of cliques have been planted as shown in $x - axis$ to differentiate between classes. We can see that the embedding methods accurately distinguish among the classes when $t = 1.5$. In all cases, we can see that DGSD performs competitively to state-of-the-art methods.

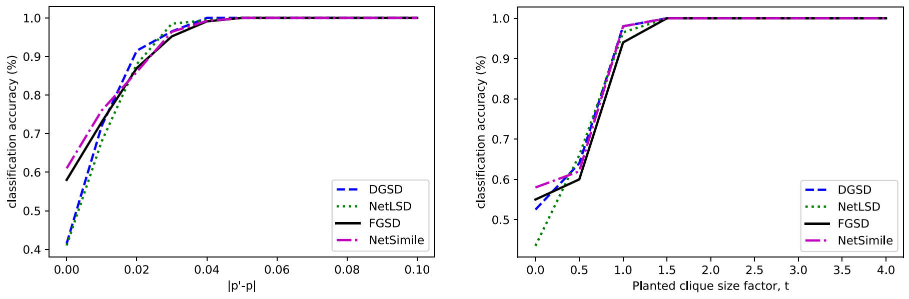

Fig. 4. 10-fold cross-validation accuracy (%) on (left) $G(n, p)$ vs. $G(n, p')$ for varying $|p - p'|$ and (right) $G(n, p)$ vs. $G(n, p, k)$ for $n = 200, p = 1/2, k = 2t\sqrt{n}$ and varying t.

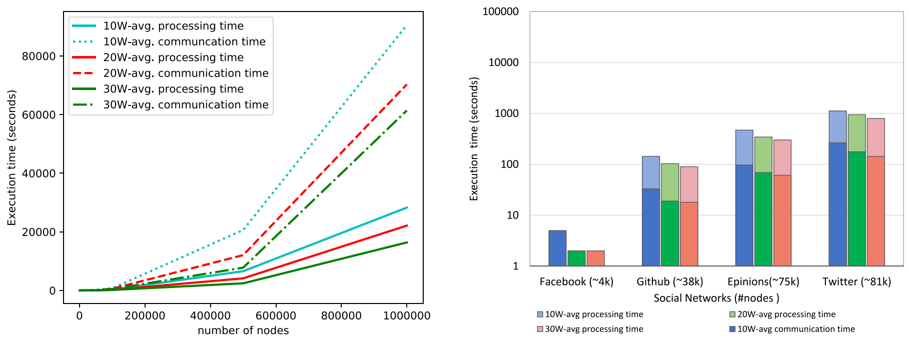

Fig. 5. Running time on large Erdős–Rényi random graphs and real world social networks. For each graph, the DGSD's average processing times and average communication times are shown separately. Running times on 10 workers (10W), 20 workers (20W) and 30 workers (30W) have been shown to highlight the importance of distributing computing on processing large graphs.

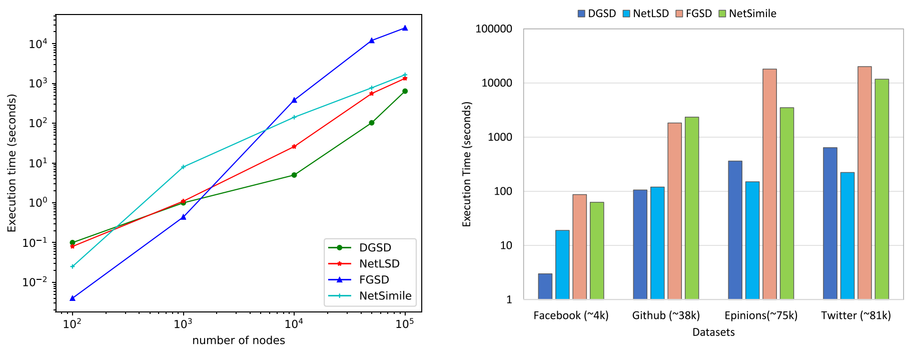

Fig. 6. Comparison of DGSD running time with state-of-the-art methods on synthetic and real-world networks. The figure on the left shows the running times comparison on synthetic networks of different sizes generated through Erdős–Rénye model, while the right figure presents a comparison on real-world social networks.

4.3. Large real world and random networks scalability analysis

To better highlight the scalability of DGSD on sufficiently large networks in a distributed environment, we perform experiments on large Erdős–Rényi random networks and real-world social networks. For the experiments, we use Intel (R) Xeon (R) 4110 CPU 2.10 GHz machine with 32 processors and 512 GB of RAM. The networks we considered for the experiments consist of 10 000, 50 000, 100 000, 500 000 and 1 million nodes generated with probabilities 0.001, 0.0001, 0.0001, 0.00001, and 0.00001 respectively. Similarly, we consider Facebook, Github, Epinions and Twitter social network datasets from SNAP repository [50]. The numbers of nodes in these datasets are 4039, 37 700, 75 879 and 81 306 respectively. We have shown the results in Fig. 5 which highlight the scalability of DGSD on sufficiently large networks up to million nodes and 5 millions edges. In the left figure, average processing and communication time of 10, 20 and 30 workers have been shown. We define communication time as the amount of time spent between the communication of a worker and server. The processing time is the time spent on all other operations on the worker machine. We can see from the results that increasing the number of workers reduces the processing as well as the communication time. We also show running time comparison of DGSD with state-of-the-art methods in Fig. 6. The running time represents the time of computing representation by each algorithm from the given graph.

5. Conclusion and future works

The task of encoding large graphs in a restricted environment where the available memory is limited is challenging. In this paper, we presented a distributed graph representation method that leverages graph statistical measures to encode graphs into embedding space. DGSD finds nodes' local proximity by considering simple graph statistical measures such as nodes' degree, common neighbors and direct connectivity. This nature of DGSD allows it to run on independently distributed machines where only the first-hop neighbors are necessary to be kept in memory for computing the distances of the corresponding nodes. Additionally, the linear time space and computational complexities of DGSD allow it to run on huge graphs. Through extensive experiments on benchmark datasets for the graph classification task, DGSD outperforms state-of-the-art methods on several benchmark datasets. In particular, the proposed method was found effective with dense social network datasets like IMDB and REDDIT. This demonstrates the benefits of very simple graph statistical measures such as degree, direct connectivity and common neighbors in the design of graph descriptors.

DGSD is the initial effort to tackle the problem of graph embeddings in parallel in a distributed environment using simple graph statistical measures. Several aspects of DGSD can be explored in the future to infer improved embeddings. Among which, neighbor selection strategy with a conditional approach may be useful to improve the performance of the algorithm. On the other hand, it can also be explored in the context of dynamic networks for several other general-purpose data mining tasks. Since DGSD involves extensive communication among client and server machines, it is also worth studying to explore this aspect and come up with a solution to reduce the communication overhead.

CRediT authorship contribution statement

Anwar Said: Conceptualization, Methodology, Investigation, Writing - original draft, Editing. **Saeed-Ul Hassan:** Supervision, Writing - original draft, Editing. **Suppawong Tuarob:** Data curation, Investigation, Writing - review & editing. **Raheel Nawaz:** Validation, Software, Revision. **Mudassir Shabbir:** Supervision, Methodology, Investigation, Validation.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] H. Cai, V.W. Zheng, K.C.-C. Chang, A comprehensive survey of graph embedding: Problems, techniques, and applications, IEEE Trans. Knowl. Data Eng. 30 (9) (2018) 1616–1637.
[2] F. Emmert-Streib, M. Dehmer, Y. Shi, Fifty years of graph matching, network alignment and network comparison, Inform. Sci. 346 (2016) 180–197.
[3] W.L. Hamilton, Z. Ying, J. Leskovec, Representation learning on graphs: Methods and applications, IEEE Data Eng. Bull. 40 (2017) 52–74.
[4] P. Langfelder, S. Horvath, Wgcna: an r package for weighted correlation network analysis, BMC Bioinform. 9 (1) (2008) 559.
[5] A. Narayanan, M. Chandramohan, R. Venkatesan, L. Chen, Y. Liu, S. Jaiswal, graph2vec: Learning distributed representations of graphs, arXiv preprint arXiv:1707.05005.
[6] M. Togninalli, E. Ghisu, F. Llinares-López, B. Rieck, K. Borgwardt, Wasserstein weisfeiler-lehman graph kernels, in: Advances in Neural Information Processing Systems 32, Curran Associates, Inc., 2019, pp. 6436–6446.
[7] S. Verma, Z.-L. Zhang, Hunt for the unique, stable, sparse and fast feature learning on graphs, in: Advances in Neural Information Processing Systems, 2017, pp. 88–98.
[8] Z. Ying, D. Bourgeois, J. You, M. Zitnik, J. Leskovec, Gnnexplainer: Generating explanations for graph neural networks, in: Advances in Neural Information Processing Systems, 2019, pp. 9240–9251.
[9] Q. Liu, M. Nickel, D. Kiela, Hyperbolic graph neural networks, in: Advances in Neural Information Processing Systems, 2019, pp. 8228–8239.
[10] T.N. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks, in: International Conference on Learning Representations (ICLR), 2017.
[11] N. Shervashidze, P. Schweitzer, E.J.v. Leeuwen, K. Mehlhorn, K.M. Borgwardt, Weisfeiler-lehman graph kernels, J. Mach. Learn. Res. 12 (Sep) (2011) 2539–2561.
[12] S. Verma, Z.-L. Zhang, Deep universal graph embedding neural network, arXiv preprint arXiv:1909.10086.
[13] C. Morris, M. Ritzert, M. Fey, W.L. Hamilton, J.E. Lenssen, G. Rattan, M. Grohe, Weisfeiler and leman go neural: Higher-order graph neural networks, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 33, 2019, pp. 4602–4609.
[14] A. Tsitsulin, D. Mottin, P. Karras, A. Bronstein, E. Müller, Netlsd: hearing the shape of a graph, in: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, ACM, 2018, pp. 2347–2356.
[15] B. Nie, S. Sun, Knowledge graph embedding via reasoning over entities, relations, and text, Future Gener. Comput. Syst. 91 (2019) 426–433.
[16] K.M. Borgwardt, H.-P. Kriegel, Shortest-path kernels on graphs, in: Fifth IEEE International Conference on Data Mining (ICDM'05), IEEE, 2005, pp. 8–pp.
[17] Z. Lei, Y. Sun, Y. Nanehkaran, S. Yang, M.S. Islam, H. Lei, D. Zhang, A novel data-driven robust framework based on machine learning and knowledge graph for disease classification, Future Gener. Comput. Syst. 102 (2020) 534–548.
[18] T.-S. Kuo, K.-S. Tseng, J.-W. Yan, Y.-C. Liu, Y.-C. Frank Wang, Deep aggregation net for land cover classification, in: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops, 2018.
[19] P. Yanardag, S. Vishwanathan, Deep graph kernels, in: Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2015, pp. 1365–1374.
[20] R. Kondor, H. Pan, The multiscale laplacian graph kernel, in: Advances in Neural Information Processing Systems, 2016, pp. 2990–2998.
[21] T. Gärtner, P. Flach, S. Wrobel, On graph kernels: Hardness results and efficient alternatives, in: Learning Theory and Kernel Machines, Springer, 2003, pp. 129–143.
[22] J. Tsurumi, T. Haga, Y. Ujiie, T. Sasaki, H. Matsushima, Payload-based statistical intrusion detection for in-vehicle networks, in: Trends and Applications in Knowledge Discovery and Data Mining: PAKDD 2018 Workshops, Springer, 2018.
[23] M. Berlingerio, D. Koutra, T. Eliassi-Rad, C. Faloutsos, Network similarity via multiple social theories, in: Proceedings of the 2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining, ACM, 2013, pp. 1439–1440.
[24] H.P. Maretic, M. El Gheche, G. Chierchia, P. Frossard, Got: An optimal transport framework for graph comparison, in: Advances in Neural Information Processing Systems, 2019, pp. 13876–13887.
[25] R. Flamary, N. Courty, A. Rakotomamonjy, D. Tuia, Optimal transport with laplacian regularization, 2014.
[26] T. Yu, J. Yan, Y. Wang, W. Liu, et al., Generalizing graph matching beyond quadratic assignment model, in: Advances in Neural Information Processing Systems, 2018, pp. 853–863.
[27] Z.R. Hassan, M. Shabbir, I. Khan, W. Abbas, Estimating descriptors for large graphs, arXiv preprint arXiv:2001.10301.
[28] Anwar Said, Saeed-Ul Hassan, Waseem Abbas, Mudassir Shabbir, NetKI: A kirchhoff index based statistical graph embedding in nearly linear time, Neurocomputing 433 (2021) 108–118.
[29] T. Kajdanowicz, P. Kazienko, W. Indyk, Parallel processing of large graphs, Future Gener. Comput. Syst. 32 (2014) 324–337.
[30] C. Morris, N.M. Kriege, K. Kersting, P. Mutzel, Faster kernels for graphs with continuous attributes via hashing, in: 2016 IEEE 16th International Conference on Data Mining (ICDM), IEEE, 2016, pp. 1095–1100.
[31] S. Hido, H. Kashima, A linear-time graph kernel, in: 2009 Ninth IEEE International Conference on Data Mining, IEEE, 2009, pp. 179–188.
[32] G. Nikolentzos, P. Meladianos, S. Limnios, M. Vazirgiannis, A degeneracy framework for graph similarity, in: IJCAI, 2018, pp. 2595–2601.
[33] S.S. Du, K. Hou, R.R. Salakhutdinov, B. Poczos, R. Wang, K. Xu, Graph neural tangent kernel: Fusing graph neural networks with graph kernels, in: Advances in Neural Information Processing Systems, 2019, pp. 5724–5734.
[34] Y. Li, D. Tarlow, M. Brockschmidt, R. Zemel, Gated graph sequence neural networks, arXiv preprint arXiv:1511.05493.
[35] D.K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, R.P. Adams, Convolutional networks on graphs for learning molecular fingerprints, in: Advances in Neural Information Processing Systems, 2015, pp. 2224–2232.
[36] W. Hamilton, Z. Ying, J. Leskovec, Inductive representation learning on large graphs, in: I. Guyon, U.V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, R. Garnett (Eds.), Advances in Neural Information Processing Systems 30, Curran Associates, Inc., 2017, pp. 1024–1034.
[37] K. Xu, W. Hu, J. Leskovec, S. Jegelka, How powerful are graph neural networks? in: International Conference on Learning Representations (ICLR), 2019.
[38] G. Li, C. Xiong, A. Thabet, B. Ghanem, Deepergcn: All you need to train deeper gcns, arXiv preprint arXiv:2006.07739.
[39] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, S.Y. Philip, A comprehensive survey on graph neural networks, IEEE Trans. Neural Netw. Learn. Syst..

[40] A.D. Kshemkalyani, M. Singhal, Distributed Computing: Principles, Algorithms, and Systems, Cambridge University Press, 2011.
[41] A. Ahmed, Z.R. Hassan, M. Shabbir, Interpretable multi-scale graph descriptors via structural compression, Inform. Sci..
[42] A.-L. Barabási, et al., Network Science, Cambridge university press, 2016.
[43] V.D. Blondel, J.-L. Guillaume, R. Lambiotte, E. Lefebvre, Fast unfolding of communities in large networks, J. Stat. Mech. Theory Exp. 2008 (10) (2008) P10008.
[44] A. Said, R.A. Abbasi, O. Maqbool, A. Daud, N.R. Aljohani, Cc-ga: A clustering coefficient based genetic algorithm for detecting communities in social networks, Appl. Soft Comput. 63 (2018) 59–70.
[45] A. Grover, J. Leskovec, node2vec: Scalable feature learning for networks, in: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016, pp. 855–864.
[46] M. Sugiyama, K. Borgwardt, Halting in random walk kernels, in: Advances in Neural Information Processing Systems, 2015, pp. 1639–1647.
[47] N. Pržulj, Biological network comparison using graphlet degree distribution, Bioinformatics 23 (2) (2007) e177–e183.
[48] T.D. Challenge, Tox21 data challenge 2014, 2014.
[49] F. Stefaniak, Prediction of compounds activity in nuclear receptor signaling and stress pathway assays using machine learning algorithms and low-dimensional molecular descriptors, Front. Environ. Sci. 3 (2015) 77.
[50] J. Leskovec, A. Krevl, SNAP Datasets: Stanford large network dataset collection, 2014, http://snap.stanford.edu/data.

Anwar Said is a Ph.D. research scholar in Scientometrics Lab, Department of Computer Science at Information Technology University, Lahore, Pakistan. He received M.Phil. (2016) degree in Computer Science from Quaid-i-Azam University, Islamabad, Pakistan. His research interests are in the area of graph representation, social network analysis, and data science.

Saeed-Ul Hassan is the Director of Artificial Intelligence Lab and a faculty member at Information Technology University (ITU) in Pakistan, a former Post-Doctorate Fellow at the United Nations University — with more than 15 years of hands-on experience of advanced statistical techniques, artificial intelligence, and software development client work. He earned his Ph.D. in the field Information Management from Asian Institute of Technology. He has also served as a Research Fellow at National Institute of Informatics in Japan. Dr. Saeed's research interests lie within the areas of Data Science, Artificial Intelligence, Scientometrics, Information Retrieval and Text Mining. Dr. Hassan is also the recipient of James A. Linen III Memorial Award in recognition of his outstanding academic performance. More recently, he has been awarded Eugene Garfield Honorable Mention Award for Innovation in Citation Analysis by Clarivate Analytics, Thomson Reuters.

Suppawong Tuarob received his Ph.D. in Computer Science and Engineering and MS in Industrial Engineering both from the Pennsylvania State University, and his BSE and MSE both in Computer Science and Engineering from the University of Michigan-Ann Arbor. Currently, he is an Assistant Professor of Computer Science at Mahidol University, Thailand. His research involves data mining in large-scale scholarly, social-media and healthcare domains by applying multiple cutting-edge techniques, such as machine learning, topic modeling, and sentiment analysis.

Raheel Nawaz has served in various senior leadership positions in the private Higher and Further Education sector; and was an Army Officer before that. He is currently the Director of the Digital Technology Solutions and a Reader in Analytics and Digital Education with Manchester Metropolitan University (MMU). He has founded and/or headed several research units specializing in artificial intelligence, digital transformations, data science, digital education, and apprenticeships in higher education. He has led on numerous funded research projects in the U.K., EU, South Asia, and Middle East. He holds adjunct or honorary positions with several research, higher education, and policy organizations, both in the U.K. and overseas.

Mudassir Shabbir is an Assistant Professor in the Department of Computer Science at the Information Technology University, Lahore, Pakistan. He received his Ph.D. from Division of Computer Science, Rutgers University, NJ USA in 2014. Previously, Mudassir has worked at Lahore University of Management Sciences, Pakistan, Los Alamos National Labs, NM, Bloomberg L.P. New York, NY, and at Rutgers University. He was Rutgers Honors Fellow for 2011–12. His main area of research is Algorithmic and Discrete Geometry and has developed new methods for the characterization and computation of succinct representations of large data sets with applications in nonparametric statistical analysis. He also works in Combinatorics and Graph Theory.