Neurocomputing 433 (2021) 108–118

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

NetKI: A kirchhoff index based statistical graph embedding in nearly linear time

Anwar Said [a], Saeed-Ul Hassan [a,*], Waseem Abbas [b], Mudassir Shabbir [a]

[a] *Department of Computer Sciences, Information Technology University, Lahore, Pakistan*
[b] *Electrical Engineering and Computer Science, Vanderbilt University, Nashville, TN 37212, United States*

ARTICLE INFO

Article history:
Received 25 October 2019
Revised 26 August 2020
Accepted 12 December 2020
Available online 29 December 2020
Communicated by Steven Hoi

Keywords:
Graph classification
Graph embedding
Graph mining
Kirchhoff index
Linear time

ABSTRACT

Recent advancements in learning from graph-structured data have shown promising results on the graph classification task. However, due to their high time complexities, making them scalable on large graphs, with millions of nodes and edges, remains a challenge. In this paper, we propose NetKI, an algorithm to extract sparse representation from a given graph with n nodes and m edges in $O(m\epsilon^{-2}\log^4 n)$ time. Our approach follows the notion of Kirchhoff index that encodes the structure of the graph by estimating effective resistance - relying on this approach yields nearly linear time graph representation method that allows scalability on sufficiently large graphs. Through extensive experiments, we show that NetKI provides improved results in terms of running time on large networks and the classification accuracy is within range 2% from the state-of-the-art results.

1. Introduction

Many real-world systems involve interactions between pairs of entities. Examples of such systems include social networks, financial systems, molecular graph structures, resistor networks, recommender systems, and protein–protein interaction networks. All of these systems and many more can be represented as graphs, that encode real-world objects (*i.e.*, nodes) and the pair-wise interactions (*i.e.*, edges) between them. More recently, graphs have also been used to model grid-structured data, such as images, to make the computations faster [1]. Encoding graphs into low-dimensional embeddings enables one to employ the traditional machine learning and data mining algorithms on graph-structured data. This approach facilitates researchers to solve hard problems on graphs such as graph classification. The graph classification problem refers to the understanding of complex graph structures among different classes, and it has a variety of real-world applications ranging from text classification to molecules toxicity prediction and classifying community structures in a social network [2]. However, graph classification poses several challenges such as permutation invariance, scalability, and, the runtime efficiency of the encoding. Generally, the same graph can be represented by many adjacency matrices due to a lack order on nodes of a graph, therefore, the ideal encoding procedure for graph classification should be invariant under permutations of the nodes. Moreover, it should preserve the atomic structure of the graph based on the local and global positions of pairs of nodes. Unfortunately, existing graph classification approaches often require a pairwise comparison among the graphs or are based solely on a statistical and spectral representations, which is hard to compute [3–6]. Therefore, appropriate representation methods are required to encode the atomic structure of the graph succinctly that are efficient. To address the aforementioned challenges, in this paper, we propose using a well-known measure called the Kirchhoff index for extracting graph representations. The Kirchhoff index encapsulates the atomic structure of the graph because it incorporates the information regarding the number of paths and quality of paths. It features in many other contexts, such as Markov chains (the average commute time of a Markov chain on the graph), experiment design, and Euclidean distance embeddings [7,8]. Recently, it has also been employed to quantify the resilience of networks based on noisy data in distributed networked control systems [9]. In fact, it has more equivalent descriptions including the Laplacian spectrum and the effective resistance which makes it suitable for extracting graph representations. Typically, Kirchhoff index is defined in terms of effective resistance using the analogy of graphs in an electrical circuit. A graph can be transformed into an electrical circuit by replacing each edge with a unit resistance.

* Corresponding author.
E-mail addresses: anwar.said@itu.edu.pk (A. Said), saeed-ul-hassan@itu.edu.pk (S.-U. Hassan), waseem.ee@gmail.com (W. Abbas), mudassir.shabbir@itu.edu.pk (M. Shabbir).

https://doi.org/10.1016/j.neucom.2020.12.075
0925-2312/© 2020 Elsevier B.V. All rights reserved.

The effective resistance between any two nodes in such a network refers to the equivalent electrical resistance between those two nodes. The sum of the effective resistance between all pair of nodes is referred to as the Kirchhoff index of a graph. We note that a higher value of effective resistance between nodes indicates an inferior or less robust overall connection between the nodes (as measured by the number and quality of paths between them) [8,10,11,7,12]. Since there is always a path between two nodes in a connected network (that is, any two nodes are always connected, maybe via some other nodes), the effective resistance between any two nodes is well defined. Consequently, the effective resistance of a graph, which is the sum of all pair-wise effective resistances between nodes, is also well defined. To illustrate this, consider the example in Fig. 1.

In (b), there are four paths between nodes 1 and 2 with lengths $1, 3, 4$, and 5, and the effective resistance between the nodes is 0.72. In (c), there are three paths between nodes 1 and 2 with lengths $1, 3$, and 5, and the effective resistance between is 0.73. We see only a slight increase in effective resistance (indicating a slightly less robust connection) as the path of length four does not exist anymore. The effect is minimal because the paths of smaller lengths, which are more important, are still there. In (d), there are only two paths between nodes 1 and 2 with lengths 3, and 5. The edge between nodes 1 and 2, hence the path of length 1, is removed. The effective resistance between the nodes increases significantly as the path with the shortest length having the maximum effect is deleted. In (e), there is only one path of length 3 between nodes 1 and 2, which increase the effective resistance from 2.75 to 3.0. Thus, we observe that effective resistance between two nodes (adjacent or non-adjacent) measure robustness in terms of the number and quality (length) of paths between nodes. Since Kirchhoff index measures the number as well as the quality of paths, we argue in this paper that this is a suitable measure to generate meaningful and expressive graph representations.

Tapping the recent advancements in solving linear systems in graph Laplacian and for measuring edge centrality in graphs, we propose an efficient method to compute the Kirchhoff index for graph classification task [13,14]. The major contributions of this study are as follows.

- We propose an efficient Kirchhoff index based graph classification method that encapsulates graph structure in terms of the number and quality of paths between all pairs of nodes.
- We empirically show that proposed method provides improved results in terms of running time on large networks while the classification accuracy is close to the state-of-the-art methods that are, otherwise, impractical for large graphs.
- We are making our implementation publicly available for other researchers to use and improve upon.

2. Related work

Our related work section is divided into four categories: direct methods, graph kernel methods, statistical and spectral representations, and graph-theoretic approaches. The methods in the first three categories deal with graph representations, while the last category surveys the relevant graph-theoretic approaches used or referred in this work. Table 1 summarizes the related work's necessary properties and time complexity.

2.1. Direct methods

Direct methods seek to extract features from graphs directly [3,4,15,16,5,17]. The *Graph Edit Distance* (GED) [3] approach is built upon the minimal number of modification needed to transform one graph into another. However, the problem is NP-hard even when the goal is to approximate the distance. A group-theoretic approach, the graphlet spectrum [4] characterizes graphs in terms of the position or frequency of common graphlets or motifs. Some other well-known approaches include propagation models [18], family of tractable distances [19] and heuristics-based approaches [20,21]. However, all of these methods require extensive computational power to run on any decent size graph dataset.

2.2. Kernel methods

Graph kernel methods are one of the most popular categories of graph classification. In general, these methods define a distance between pairs of elementary substructures of two graphs, and some function of the resulting matrix is the representation of distance between the two given graphs. [15,22–25,3,19,26,18,20]. Deep graph kernels [15] compute substructures or motifs of a graph, and use a word-embedding model to extract the underlying features. In order to extract both local and global-scale graph features, Kondor et al., [16] introduced the Multiscale Laplacian Graph Kernel (MLG) which captures nodes' topological information

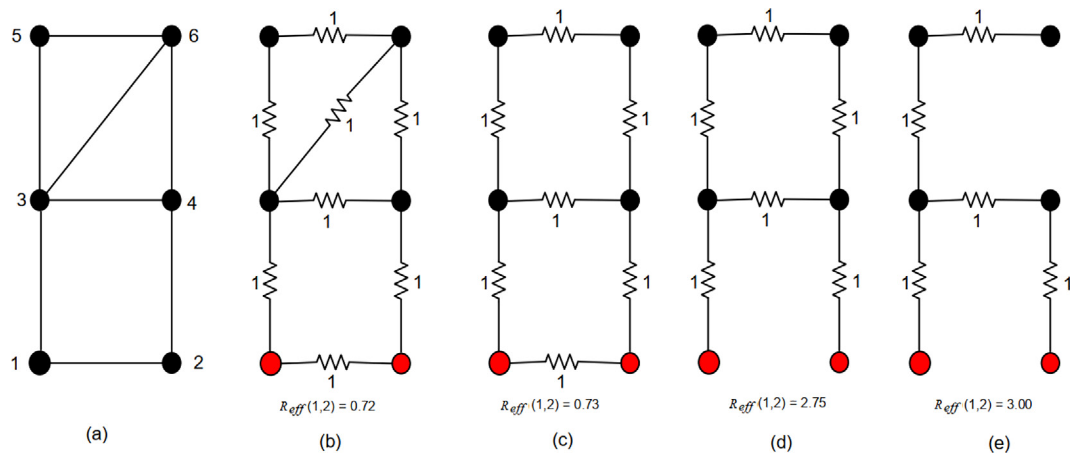

Fig. 1. (a) A graph with six nodes. (b) Each edge is replaced by a unit resistor and the effective resistance (R_{eff}) between nodes 1 and 2 is computed. The subsequent removal of edges as shown in (c), (d), and (e) results in the loss of path between nodes 1 and 2, which causes an increase in the effective resistance. The corresponding increase in the effective resistance is also shown with the figures. A loss of a good quality path results in large increase in the corresponding effective resistance.

Table 1
Summary of related work: the column “labels” indicates whether the method support (requires) labels. The column “attributes” refers to the features or other general attributes associated with the nodes. The variables represent: (n nodes, m edges, d constant, D average degree, h iterations, k eigenvalues, ϵ relative error, GCNs (Graph Convolutional Networks), Attr – Attributes, Perm – permutation, Invar – Invariance, stat – statistical, spec – spectral, rep – representation).

Method	Type	Labels	Attr.	Perm. Invar.	Complexity
GED [3]	direct	+	–	+	NP-complete
FTD [19]	direct	+	–	–	$O(n^2)$
RW[52]	kernel	+	+	+	$O(n^6)$
SP [22]	kernel	+	+	+	$O(n^3)$
NHK [28]	kernel	+	–	+	$O(dhnD)$
WL [24]	kernel	+	–	+	$O(hm)$
MLG [16]	kernel	+	–	+	$O(k^3)$
FGSD [5]	stat. rep.	–	–	+	$O(n^2)$
NetKI	stat. rep.	–	–	+	$O(m\epsilon^{-2}\log^4 n)$
NetLSD [6]	spec. rep.	–	–	+	$O(km + k^2 n)$
GCN [47]	GCNs	–	+	–	
GraphSAGE [53]	GCNs	+	+	+	
GIN [49]	GCNs	+	+	+	
DGI [50]	GCNs	+	+	+	

through graph Laplacian. Other popular graph kernels include the Weisfeiler-Lehman kernel (WL) [24], shortest path kernel (SP) [22], edge histogram kernel (EHK) [27], neighbourhood hash kernel (NHK) [28], graphlet sampling kernel (GK) [29] and graph invariant kernel [25]. However, these methods suffer from higher time complexity. In order to capture the distribution of an individual component of the graph, recently, [30] proposed a distance measure for graphs: the graph Wasserstein distance and Weisfeiler-Lehman-inspired embedding scheme derived from optimal transport theory. The proposed measure uses node feature representations and weighted edges to enhance the quality of the graph embeddings. Similarly, [31] introduced indefinite graph kernel for time series analysis by using the power of optimal transport theory. The computational overhead of these methods makes them impractical for large-scale graph classification.

2.3. Statistical and spectral representations

Statistical and spectral representation methods use a graph’s statistical properties and graph spectrum to generate graph feature vectors [32,33,5,6]. Initial works in this area [32,33] use summary-graph-statistics and handcrafted features such as nodes’ degrees to extract local graph properties. Recently, Verma and Zhang [5] introduced a histogram-based representation method, known as the Family of Graph Spectral Distances (FGSD). This approach based on computing multisets of spectral distances between all pairs of nodes using eigendecomposition. The authors in [6] introduced a spectral representation method, based on wave- and heat-trace signatures at various times, and consider the associated heat diffusion process on the graph rather than use the Laplacian spectrum to obtain graph representations. Recent approaches [5,6] have shown promising results on graph comparison. However, their time complexities make them impractical for reasonably large graphs.

Under the assumption of the same number of nodes in the graphs, [34] recently propose a graph comparison framework that uses Wasserstein distance is mean to compare graphs. The measure uses smooth graph signal distributions associated with graphs and compares them using the Wasserstein distance. Similarly, [35] uses optimal transport theory and introduces a graph comparison framework based on empirical distribution with a graph-based regularization. The main idea of the work was to compute an optimal transportation plan by limiting the displacement of a pair of nodes. Graph-based regularization [36] approach finds the similarity between adjacent nodes by their position or displacements of the transported sample. Instead of matching graph feature vectors, [37] propose to use a Wasserstein distance to compare between them. [38] uses separable functions to approximate discrete graph matching in continuous domain. Likewise, [39,40] use Gumbel-sinkhorn network to extract permutations from data. Closer to our work, the authors in [41] propose a graph streaming algorithm that doesn’t require the whole graph in memory and approximate feature vector by estimating counts of sub-graph. Some other distance measures on graphs include [42–44,40]. Many of these approaches have shown promising results on the graph classification task, however, due to their computational complexities, scaling them on sufficiently large graphs is still a challenging task.

With the advancement in deep learning approaches especially the convolutional neural networks and recurrent neural networks, recently there has been a surge in neural models for learning graph representations [45–48,1]. Graph neural models are end-to-end learning approaches based on neighborhood aggregation and recurrent neural networks [49–51].

2.4. Graph-theoretic methods

Li and Zhang [14] introduced nearly-linear time algorithms for finding nodes’ and edges’ importance in a given network. To find edge importance, the authors deployed the Sherman-Morrison formula and used two graph-theoretic approaches to attain the required goal. These methods will be discussed in details in forthcoming section, as our method is based on this idea. Spielman and Srivastava constructed spectral graph sparsifier in $O(m)$ and proved the existence of sparse graph with $O(n\log n/\epsilon^2)$ edges [54]. To construct the sparsifier, this method uses the notion of graph effective resistance approximated by solving graph Laplacian. The breakthrough results by Spielman and Teng show that it can be solved in $O(m\log^c n\log(\frac{1}{\epsilon}))$ time [55]. Kyng and Sachdeva [13] introduced a nearly-linear time algorithm that approximates cholesky factorization for Laplacian matrices [56]. This algorithm is based purely on random sampling and Gaussian elimination to solve linear systems in graph Laplacians. Recently, Cohen et al.,[57] introduced a sub-sampling iterative method (currently the best-known Laplacian solver) based on recursive preconditioning, and this aims to reduce graphs to trees. Many other follow-up works [58–61] have extended the field in the past decade, and, currently the best known algorithm has $O(m\log^{\frac{1}{2}} n\log\log^c n\log(\frac{1}{\epsilon}))$ [57] time complexity. Although [57] is the best-known Laplacian solver, we employed Kyng and Sachdeva’s [13] method in our experimental setup, since it is simple and requires no graph theoretic constructions. This enabled us to solve a much broader family of linear systems. Additionally, the code of the solver is freely available[1], and we used it in our experimental setup.

3. Preliminaries and basic notations

3.1. Graph Basics

Let $G = (V, E)$ denote an undirected connected graph where $V = \{v_1, v_2, \ldots, v_n\}$ is the set of n nodes, and $E \subseteq V \times V$ is the set of m edges. Throughout this study, we consider G as a resistor network where nodes represent junctions while edges represent resistors between the junctions. Let r_e denote a unit resistance over the edge e. We write $G'(e)$ to represent the graph obtained from G by deactivating edge e by decreasing r_e to θr_e for some small $0 < \theta < 1/2$. We denote this process by *θ-deletion*. Let L denote the $n \times n$ Laplacian matrix for G where,

$$L_{(u,v)} = \begin{cases} k_u & \text{if } u == v \\ -1 & \text{if } u \sim \text{v} \\ 0 & \text{otherwise.} \end{cases} \quad (1)$$

Where k_u is the degree of node u, and $(u \sim v)$ indicate that node u and v are neighbors. Let $L^{\theta e}$ denote the Laplacian of the graph after the deactivation of edge e. Let W be an $m \times m$ diagonal matrix with $W[x, x] = w(e_x)$ (e_x indicates an edge from the edge set $\{e_1, e_2, \ldots, e_m\} \in E$). Similarly, let $B_{m\times n}$ denotes a *signed edge-vertex incident matrix* where each entry is

$$B_{e_x,v} = \begin{cases} 1 & \text{if } v \text{ is } e_x\text{'s head,} \\ -1 & \text{if } v \text{ is } e_x\text{'s tail,} \\ 0 & \text{otherwise.} \end{cases} \quad (2)$$

Then the Laplacian matrix $L = B^\top WB$ [62]. Let L^+ represents the pseudo-inverse of the Laplacian and e_i denote the i^{th} standard basis vector or one-hot encoding of a node v_i where only the corresponding i^{th} index is 1, and 0 otherwise, and $b_{u,v} = e_u - e_v$. For each edge $e \in E$, we define $b_e = b_{u,v}$ where u and v are the end points of the edge e respectively. This implies that b_e contains only two nonzero entries: 1 (on the i^{th} index) and -1 (on j^{th} index). Using the standard basis vector b_e, it can be shown that $L = \sum_{e\in E} b_e b_e^\top$ for G. Similarly, for $G'(e)$, we have $L^{\theta e} = L - (1-\theta) b_e b_e^\top$ [14].

3.2. Network Kirchhoff Index

Extracting features from graphs is hard and involves many challenges such as the permutation invariance, scalability, and efficiency of the algorithm. To tackle these challenges, here we leverage a resistor network metaphor. The electrical flow on the resistor network can be described is as follow.

Let $\mathbf{I}_{inj}(u)$ represent currents injected at the graph node u, and $\mathbf{I}(e)$ indicate the currents induced in the edge e. Let $\phi(u)$ be the potential induced at the node u. Then the Kirchhoff current law states that the sum of total current injected to the node must be equal to the sum of current flowing out of it:

$$B^\top \boldsymbol{I} = \boldsymbol{I}_{inj} \quad (3)$$

By Ohm's law, the flow of current in an edge follows

$$\boldsymbol{I} = WB\phi \quad (4)$$

Combining (3) and (4), we get

$$\boldsymbol{I}_{inj} = B^\top (WB\phi) = L\phi \quad (5)$$

If the total amount of current passed to a node equals the total current flowing out of it, then we can write it as follows.

$$\phi = L^+ \boldsymbol{I}_{inj} \quad (6)$$

Now using L^+, the effective resistance and Kirchhoff index can be defined as below.

Effective Resistance: The effective resistance ($\mathscr{R}_{eff}$) of an edge e can be defined as the potential difference between endpoints of the edge when a unit current is passed at the head of the edge and extracted at the tail. In terms of Laplacian matrix, it can be defined as

$$R_{eff}(e) = b_e^\top L^+ b_e \quad (7)$$

Generally, the effective resistance between a pair of nodes can be calculated through well-known manipulations: series, parallel. Edges $(e_1, .., e_k)$ correspond to resistors with resistance $(r_1, .., r_k)$ Ohm in series can be replaced by a single edge with resistance $\sum_i^k r_i$. In case of parallel connectivity, edges can be replaced by one edge with resistance $\left(\sum_i^k r_i^{-1}\right)^{-1}$. The definitions of these series and parallel manipulations on resistor networks follow that the effective resistance captures both the number of paths, quality of paths and their length between pair of nodes which intuitively quantifies edges based on their electrical importance.

Rayleigh's Monotonicity Law: *"If the resistance of a circuit is increased, the effective resistance R_{eff} between any two points can only increase. If they are decreased, it can only decrease*".

By law of Rayleigh's Monotonicity, we define edges' centrality scores as follow [14].

Definition 3.1. The centrality score $C_\theta(e)$ of an edge $e \in E$ using the Kirchhoff index is defined as the increase of the Kirchhoff index of the graph obtained from G by $\theta -$ *deleting* e.

Definition 3.2. Kirchhoff Index: The Kirchhoff index of a graph G can be defined as the sum of the effective resistance over all pairs of nodes.

$$\text{Kirchhoff index(G)} = \sum_{v_i, v_j \in V,\ i<j} \mathscr{R}_{eff}(v_i, v_j) \quad (8)$$

The Kirchhoff index of a given graph quantifies graph connectivity. The less connected a graph is, the larger the Kirchhoff index will be. Similarly, when the graph is densely connected, the corresponding Kirchhoff index is smaller in general. Based on the definition of Kirchhoff index and Rayleigh's Monotonicity Law, we have [14]:

Fact 1.1: *The Kirchhoff index of a graph does not decrease by $\theta -$ deleting e.*

Fact 1.2: *"Let $\lambda_1, \lambda_2, \ldots, \lambda_{n-1}$ be the nonzero eigenvalues of L. The Kirchhoff index of graph G satisfies Kirchhoff index(G) $= n\sum_{i=1}^{n-1} \frac{1}{\lambda_i} = nTr(L^+)$*".

Based on Fact 1.1, edges scores $C_\theta(e)$ using Kirchhoff index can be computed via $\theta -$ *deletion* as Kirchhoff index encapsulates information regarding the number of paths and quality of paths. By Definition 3.1, the difference of the Kirchhoff index between G and $G'(e)$ reflects $C_\theta(e)$ and it can be computed for all the edges to encode the structure of the graph. In the forthcoming sections, we present a graph embedding algorithm using the computation of exact Kirchhoff index of the graph. And then we show linear time approximation of Kirchhoff index for encoding sufficiently large graphs.

[1] http://danspielman.github.io/Laplacians.jl/latest/usingSolvers/

4. Methodology

4.1. Graph embedding algorithm using exact Kirchhoff index

Here we present Kirchhoff index based graph representation algorithm which encodes graph structures using network Kirchhoff index. The algorithm describes the procedure for extracting graph representation by computing exact network Kirchhoff index.

Algorithm 1: Compute $\mathscr{H}$ using exact Kirchhoff index

```
Input: Graph $G = (V, E, w)$, Laplacian $L$, bin width $h$, number of bins $b$
Output: $\mathscr{H}$
1: Let $L^+$ indicate pseudo-inverse of the Laplacian matrix
2: $K \leftarrow nTr(L^+)$
3: initialize sparse matrix $\mathscr{F}_{n\times n}$ with zeros
4: for $e(u, v) \in E$ do
5:     $G'(e) \leftarrow \theta - deleting$ e
6:     $L^{\theta e} \leftarrow$Laplacian of $G'(e)$
7:     $L'(e) \leftarrow$ pseudo-inverse of $L^{\theta e}$
8:     $K^{\theta e} \leftarrow nTr(L'(e))$
9:     $C_\theta(e) \leftarrow K^{\theta e} - K$
10:    $\mathscr{F}_{uv} \leftarrow C_\theta(e)$
11: end for
12: Set $Flt \leftarrow \{\mathscr{F}_{uv} | \forall (u, v) \in |V|\}$
13: $\mathscr{H} \leftarrow$histogram$(Flt, b, h)$
14: return $\mathscr{H}$
```

The core idea of Algorithm 1 is to perform $\theta - deletion$ and compute the Kirchhoff index of the graph. Initially, the algorithm computes pseudo-inverse in step-1 and by using Fact 1.2, it computes Kirchhoff index (K) (step-2) of the G. In step 5, it performs $\theta - deleting$ e and computes the Laplacian $L^{\theta e}$, pseudo-inverse $L'(e)$ and $K^{\theta e}$(step 6–8) of the new graph $G'(e)$. Using Definition 3.1, edge e centrality score is computed in step 9. This procedure is repeated for all edges and the corresponding scores $C_\theta(e)$ are stored in matrix $\mathscr{F}$. Finally, we flatten $\mathscr{F}$ and compute the histogram $\mathscr{H}$(step 12–13).

From step 4–11, the algorithm computes Laplacian pseudo-inverse $L^{\theta e}$ and L'(e) for each edge (m times) which has $O(n^2)$ running time complexity each. Thus, the total cost of this algorithm is $O(mn^2)$.

4.2. NetKI: Algorithm for graph representation using the approximate Kirchhoff index

Here, we show an approximation algorithm that extracts feature vector from graphs in nearly linear time. Our proposed method is based on the approximation procedure proposed in [14]. Li and Zhang [14] introduced two alternate $\epsilon - approximation$ algorithms to estimate the relative importance of graph edges and an algorithm for computing nodes' importance in a given graph. To compute edge relative importance, the authors used the notion of Kirchhoff index and approximate it in two different ways. We follow the *EDGECENTCOMP2*(Theorem 1.2) approximation algorithm for estimating the edge scores which is derived through Sherman-Morrison formula [63]. For an edge $e \in E$ and a scalar $0 < \theta < 1$, we have

$$(L^{\theta e})^+ = (L - (1-\theta)b_e b_e^\top)^+$$
$$\text{By Sherman-Morrison formula, we have} \quad (9)$$
$$= L^+ + (1-\theta)\frac{L^+ b_e b_e^\top L^+}{1-(1-\theta)b_e^\top L^+ b_e}$$

Thus by the definition of the edge score, we have

$$\mathscr{C}_{\theta e} = n(1-\theta)\frac{Tr(L^+ b_e b_e^\top L^+)}{1-(1-\theta)b_e^\top L^+ b_e} \quad (10)$$

Where $(L^{\theta e})$ is the Laplacian of the graph $G'(e)$ which is obtained by $\theta - deletion$ in the original graph G. $\mathscr{C}_{\theta e}$ is the score we aim to compute as discussed in Algorithm 1. The numerator of Eq. (10) is the trace of the implicit matrix $Tr(L^+ b_e b_e^\top L^+))$ which is approximated through Hutchinson's Monte-Carlo method [64,65]. Further to apply L^+, the author used nearly-linear time Laplacian solver [57]. The denominator of the Eq. (10) is just $1-(1-\theta)R_{eff}$ derived in Eq. (7), and the authors used the random projection in [54] to approximate the effective resistance.

EDGECENTCOMP2 algorithm is based on two main procedures: solving linear systems in graph Laplacian and approximating the effective resistance. Due to the various real-world applications of the graph Laplacian including the graph partitioning, resistor networks and the theory of random walks, there has been a surge in approaches to solve it efficiently. In particular, the breakthrough results by Spielman and Teng [55] showed that linear systems in graph Laplacian could be solved in $O(m\log^c n \log(\frac{1}{\epsilon}))$ time. Various follow up works have improved the bound in the last decade. One notable Laplacian solver was proposed by Kyng and Sachdeva [13] which is purely based on random sampling and runs in expected time $O(m\log^3 n)$. The algorithm does not use any graph theoretic constructions and performs Gaussian elimination based on Cholesky factorization to solve $x = L^+ b$ in graph Laplacian. This allows us to recursively solve a much broader family of linear systems. In EDGECENTCOMP2 algorithm, the Laplacian solver proposed in [57] was used which runs in $O(m\log^{\frac{1}{2}} n \log\log^c n \log(\frac{1}{\epsilon}))$ expected time.

To approximate the effective resistance, EDGECENTCOMP2 uses the algorithm proposed in [54]. The authors in [54] used vector random projection in a subspace $\{QW^{\frac{1}{2}}BL^+X_v\}$ where Q is a random Bernoulli matrix and X_v indicates standard basis vector of node v. Adapted from [54], we show the procedure for approximating the effective resistance in Algorithm 2.

Algorithm 2: Compute approximation matrix Z

```
Input: Graph $G = (V, E)$ and $\epsilon > 0$
Output: $R_{eff}(u, v)$
1: Let $Q_{k\times n}$ be a random Bernoulli matrix, where
   $k = 24\log n/\epsilon^2$
2: compute $\mathscr{Y} = QW^{1/2}B$
3: Let $Z_{k\times n}$ be an approximate resistance matrix
4: Let $y_i$ denote the rows of $\mathscr{Y}$
5: for $i = 1$ to $k$ do
6:     $U^\top DU \leftarrow SPARSECHOLESKY(L, \epsilon)$
7:     $Z_i \leftarrow (U^+)^\top (D^+)(U^+)y_i$
8: end for
9: return $Z$
```

The Algorithm 2 uses Johnson-Lindenstrauss Lemma and Kyng and Sachdeva's Laplacian solver to estimate the effective resistance. Using the Johnson-Lindenstrauss lemma, the algorithm randomly generates $Q_{k\times n}$ Bernoulli matrix and considers it as a k-dimensional projection in $\mathbb{R}^d$ [66,13]. Then the algorithm computes $\mathscr{Y}$ matrix which takes $2m \times 24\log n/\epsilon^2 + m = O(m/\epsilon^2)$ time, since W is diagonal and B has $2m$ entries. Finally, the algorithm solves the Laplacian L for each row of $\mathscr{Y}$ matrix to form the resistance matrix Z which has $O(m\log^3 n)$ time complexity. After computing the resistance matrix, the effective resistance of each edge (u, v) can be computed as follows [54].

$$R_{eff}(u, v) \sim ||Z(\mathscr{X}_u - \mathscr{X}_v)||^2 \tag{11}$$

Where $\mathscr{X}_u$ and $\mathscr{X}_v$ represent the corresponding columns of nodes u and v in the resistance matrix Z.

Algorithm 3: Compute $\mathscr{H}$ using NetKI

```
Input: Graph $G = (V, E, w)$, Laplacian $L$, bin width $h$, number of
   bins $b, \theta$ and $\epsilon : [0, 1/2]$.
Output: $\mathscr{H}$
1: Let $z_1, \ldots, z_M$ be independent random $\pm 1$ vectors, where
   $M = \lceil \epsilon^{-2} \log(n) \rceil$
2: Let $A_{M\times|E|}$ be a solver matrix
3: Let $F_{n\times n}$ be a score matrix
4: for $i = 1$ to $M$ do
5:    $U^\top DU \leftarrow SPARSECHOLESKY(L, \epsilon)$
6:    $x_i \leftarrow (U^+)^\top (D^+)(U^+)z_i$
7:    foreach $e \in E$ do
8:       $A_{i,e} \leftarrow x_i^\top b_e b_e^\top x_i$
9:    end for
10: end for
11: $Z \leftarrow$ Compute approximation matrix$(G, \theta\epsilon/9)$
12:    for each $e(u, v) \in |E|$ do
13: $F_{u,v} \leftarrow (1-\theta)\frac{n}{M}\sum_i^M A_{i,e}/(1-(1-\theta)Z||\mathscr{X}_u - \mathscr{X}_v||^2$
14: end for
15: Set $Flt \leftarrow \{\mathscr{F}_{uv} | \forall (u, v) \in |V|\}$
16: compute $\mathscr{H} \leftarrow$histogram$(Flt, b, h)$
17: return $\mathscr{H}$
```

Using EDGECENTCOMP2, we present the approximation algorithm for generating graph representations in Algorithm 3. The core idea of the algorithm is to approximate Kirchhoff index based on Eq. (10). To apply Laplacian pseudo-inverse, we use nearly-linear time solver [13] to generate sparse Cholesky factorization (step 5). In step 6, the algorithm computes the solution vector and calculates Kirchhoff index score for each edge using Eq. (10). This process is repeated M times which compute M scores for each edge. Further, the algorithm computes approximate matrix Z using Algorithm 2 to estimate the effective resistance (step-11). Following Monte-Carlo method which approximates trace of a matrix L by $\frac{1}{M}\sum_i^M z_i^\top L z_i$ where $\mathbb{E}[z_i^\top L z_i] = Tr(L)$, the algorithm computes score matrix F. Note that z_i represents random ± 1 vectors consisting of independent Bernoulli entries. Finally, we flatten the score matrix and generate histogram $\mathscr{H}$ to perform graph classification task using a standard classification algorithm.

4.3. Modification to the EDGECENTCOMP2 algorithm

EDGECENTCOMP2 algorithm uses Cohen et al., [57] Laplacian solver, which is currently the best-known method for solving linear systems in graph Laplacian with running time $O(m\log^{1/2} n \log\log^c n \log(1/\epsilon))$. However, it requires many graph-theoretic constructions such as the recursive preconditioning, leverage score, ultra-sparsifiers, and low-stretch spanning trees, and thus, is difficult to implement. Instead, we use Kyng and Sachdeva (step 4 in Algorithm 2) - a popular nearly linear time Laplacian solver which runs in $O(m\log^3 n)$ in our experimental setup. This method is quite simple and based purely on random sampling. We refer the reader to [13] for more details. Further, we extend the algorithm to extract graph representation 2 using the approximated Kirchhoff index based edge scores.

4.4. Analysis of the algorithms

The NetKI method is based on two main procedures: SPARSECHOLESKY and the "Compute approximation matrix". The running time of SPARSECHOLESKY method is $O(m\log^3 n)$. The "Compute approximation matrix" makes a total of $O(\log n)$ calls to SPARSECHOLESKY each of which takes $O(m\log^3 n)$ time. Therefore the total running time is $O(m\log^4 n)$.

The NetKI algorithm makes $O(\epsilon^{-2}\log n)$ calls to SPARSECHOLESKY each of which takes $O(m\log^3 n)$ time, makes a call to "Compute approximation matrix" which runs in $O(m\log^4 n)$ time, and computes the histogram which runs in $O(n+b)$, where b is the number of bins. Therefore, the running time of NetKI algorithm is bounded by, $O(m\epsilon^{-2}\log^4 n) + O(m\log^4 n) + O(n+b) = O(m\epsilon^{-2}\log^4 n)$.

5. Evaluation

We performed a number of different experiments to evaluate NetKI in terms of scalability, feature sparsity, running time, classification accuracy and features comparison. The forthcoming sections describe the details of all the experimentation we performed for the evaluation of NetKI.

5.1. Scaling to large graphs

Scalability on large graphs is an essential requirement for classification of graph-structured data using machine learning. Although, recent approaches have shown promising results on various benchmark datasets, making them scalable on large graphs remains a challenge. To corroborate the scalability power of NetKI, here we present it's performance analysis in terms of time on large random graphs with millions of nodes and edges.

We generate five large random graphs of size $\{10^2, 10^3, 10^4, 10^5, 10^6\}$ using Erdős-Rényi model with probabilities $\{0.1, 0.01, 0.001, 0.0001, 0.00003\}$ respectively. The combination of these parameters generates large densely connected graphs up to 15 millions of edges. The time complexity comparison against other state-of-the-art methods including NetLSD [6] (which is currently the best known spectral representation method in terms of complexity) is shown in Fig. 2. We can see that NetKI took only 9 min on a graph with 10^5 nodes and took 30 min on a graph with 10^6 nodes. On the other hand, NetLSD (h_g) executed in 22 min on graph of size 10^5 nodes and took 3 h on 10^6 nodes. Although, FGSD performs well on small graphs, however, its performance worsens sharply as the size of the graphs increase. On the other hand, NetKI performs quite well on large graphs and outperforms all other approaches. These results illustrate that NetKI is a suitable graph embedding and classification method for large-scale graphs.

5.2. NetKI sparse representations

Representation sparsity is desirable for the task of machine learning and can be defined as the number of zero entries present in the feature vector. Sparse representations have received considerable attention in the last few years, and have found applications in various fields such as signal processing, image processing, and deep learning [67]. The main aspect in which sparse representations are superior is the adaptation on varying levels of informa-

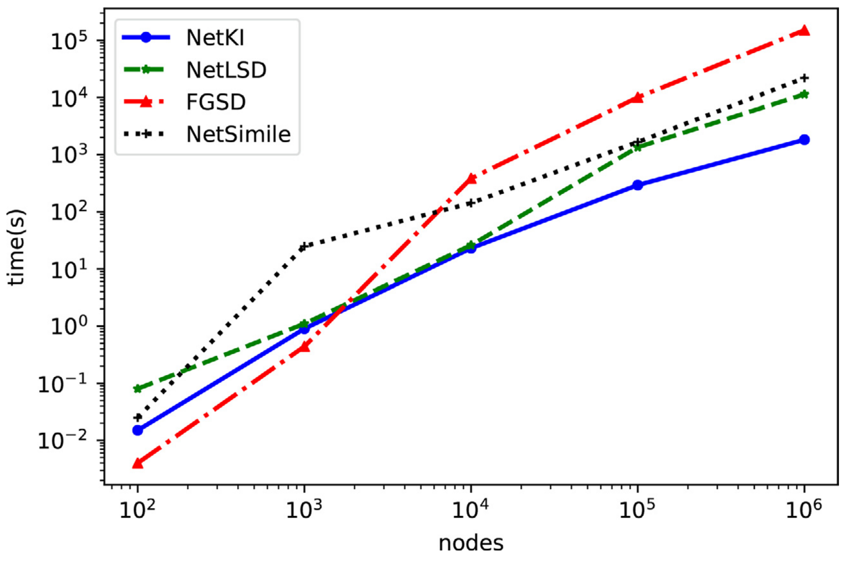

Fig. 2. Performance comparison in terms of time with state-of-the-art approaches on large random graphs. NetKI efficiently performs on large-scale graphs.

tion and its closeness to the linearity, as dense representations are difficult to learn. On the other hand, it provides an intermediate form between one-hot representation and pure distributed representation. Thus, the more sparse representation, the easier it is to generalize and learn [67,5].

We empirically show NetKI's representation sparsity and also provide comparison with FGSD in Fig. 3 (left). The NetKI shown minimum 80% representation sparsity on D&D dataset and maximum 98% on PTC dataset. To better motivate the sparse representations for the classification task, we show the classification accuracy of NetKI and FGSD on sparse and dense Erdős-Rényi random graphs in Fig. 3 (right). The classification accuracy of both the algorithms is observed better on the representations of sparse graphs as compared to the dense representations.

To create random datasets for the experiment shown in Fig. 3 (right), we form binary classification problem and generate Erdős-Rényi random graphs with different densities for each class. We consider two cases: sparse and dense graphs with 10 datasets for each case and generate 100 graphs for each class where each graph consists of 100 nodes. For sparse graphs, in each iteration i, we set $p = 0.3 + (i * c)$ for positive class and $p = 0.3 - (i * c)$ for negative class. For dense datasets, we set $p = 0.7 + (i * c)$ for one class and $p = 0.7 - (i * c)$ for the other class. The value of c is set to 0.002. We observe that with gradual increase in value of i, the difference between the graph densities of two classes widens as it makes one class denser and the other sparser and we see an improvement in the performance of the classifier.

5.3. Numerical results

We compare NetKI for classification accuracy and running time against the state-of-the-art methods, namely FGSD [5], NetLSD's wave $w(g)$ and heat $h(g)$ kernels [6], NetSimile [32], Random Walk kernel (RW) [52], Shortest Path kernel [22] Edge Histogram kernel (EHK), [27], Neighborhood Hash kernel (NHK) [28] and Graphlet sampling kernel (GK) [29]. FGSD, NetLSD and NetSimile are recent statistical and spectral representation methods that show promising results while the rest are well-known graph kernel methods.

Datasets: We used five bioinformatics [68] and six social networks benchmark datasets to evaluate NetKI and other methods on. The bioinformatics datasets include MUTAG, PTC, PROTEINS, NCI1 and D&D. The social networks include COLLAB, IMDB-B, IMDB-M, REDDIT-B, REDDIT-5 K, and REDDIT-L. We report their characteristics in Table 2.

Experimental Setup: All experiments were performed on Intel-Core i7-6800 K CPU@3.40 GHz machine with 128 GB RAM. We implemented NetKI in Python 3.7 with Networkx[2] library to interact with graphs and Laplacians.jl[3] library to solve linear systems in graph Laplacian. The NetKI implementation has been made publicly available [4]. In NetKI setting, we set the $\epsilon$and θ to 0.5 throughout all the experiments. In the preprocessing step, we apply five-number summary and replace scores above the third quartile $(Q3) + 2.5 * IQR$ with the mean value by considering the multisets of the whole dataset. In case of disconnected graph, we considered largely connected component of the graph. For computing the histogram, we set its range from (0-maxval) where maxval indicates the maximum score of the edge seen in the entire collection of graphs in the given dataset. After experimentation, the number of bins is chosen from the set $\{20, 50, 200, 500, 1000, 2000\}$ independently for different datasets. We attested this setup to be a good choice in terms of classification accuracy tradeoff. For experimenting with graph kernel methods, we used Grakel[5] library and used the default parameters setting. Moreover, we use the default parameters setting for NetSimile and NetLSD presented in their papers. NetLSD provides two functions: heat h_g and wave w_g, therefore we report results on both the functions. In FGSD setting, we performed experiments using all the provided bin width: $\{0.001, 0.0001, 0.00001\}$ as reported in the actual paper and show the best results achieved among them.

For graph classification, we employed Random Forest (RF) Classifier with 500 estimators. To make a fair comparison among all the methods, the same classifier with the same parameter setting and 10-fold cross-validation is employed in the experimental setup. The results comparison on bioinformatics datasets with graph kernels, NetSimile, FGSD and, NetLSD is reported in Table 3 while Table 4 reports results comparison on social network datasets.

On bioinformatics datasets (Table 3), we found that NetKI outperforms all other methods on PTC dataset while the classification accuracy on MUTAG, PROTEINS and D&D is within range 2.0 from the best accuracy. On NCI1 dataset, FGSD performed well and out-

[2] https://networkx.github.io/documentation/stable/
[3] http://danspielman.github.io/Laplacians.jl/latest/
[4] https://github.com/Anwar-Said/NetKI
[5] https://ysig.github.io/GraKeL/dev/

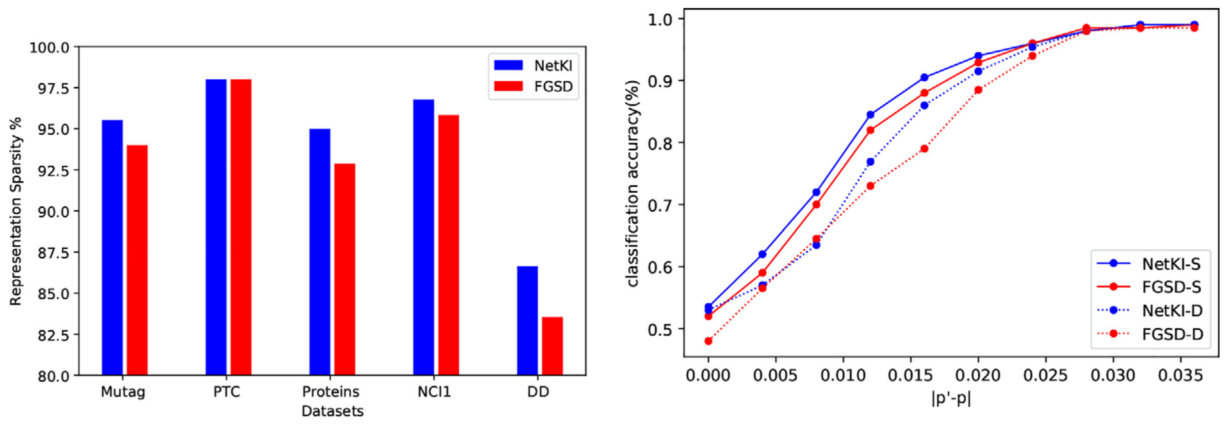

Fig. 3. (left) NetKI representations sparsity comparison with FGSD on bioinformatics datasets. (right) NetKI and FGSD performance comparison in terms of classification accuracy on sparse and dense graph datasets generated using Erdős-Rényi model with 30% and 70% densities. Note that "−S" in legends indicates performance on sparse networks while "−D" represents performance on dense network datasets.

Table 2
Datasets characteristics.

Dataset	$\|\mathcal{G}\|$	Classes	$avg.\|V\|$	$avg.\|E\|$	$min.\|V\|$	$max.\|V\|$
MUTAG	188	2	17.93	19.79	10	28
PROTEINS	1113	2	39.06	14.69	4	620
PTC	344	2	25.56	72.81	2	109
NCI1	4110	2	29.87	32.3	3	111
D & D	1178	2	284.3	715.65	30	5748
COLLAB	5000	3	74.49	2457.21	32	492
IMDB-B	1000	2	19.77	97.53	12	136
IMDB-M	1500	3	13.00	65.93	7	89
REDDIT-B	2000	2	429.61	497.75	6	3782
REDDIT-5 K	4999	5	508.50	594.87	22	3648
REDDIT-L	291	2	97491	1090822.82	4445	1632141

Table 3
classification accuracy comparison of NetKI against well-known graph kernels and recent methods is presented. **Bold** results indicate accuracy within range 2.0 from the best results of all the methods, and the blue color indicates best results. The expression '$> D$' indicates computations exceed 24 h while std indicates the standard deviation of the performance of the algorithms. Running time in seconds of each algorithm is shown in brackets with the accuracy where the minimum running time is highlighted bold.

Dataset	RW	SP	NHK	EHK	GK	NetSimile	FGSD	NetLSD [6]				NetKI	
	[52]	[22]	[28]	[27]	[29]	[32]	[5]	$w(g)$	std	$h(g)$	std	acc/time	std
MUTAG	**87.00** (119s)	**86.60** (7.45s)	85.06 (10.37s)	85.37 (3.3s)	77.01 (42.3s)	83.42 (0.14s)	**88.26** **(0.10s)**	82.40 (0.6s)	0.017	83.31 (0.55s)	0.011	**86.3** (0.50s)	0.011
PTC	51.97 (520s)	**59.00** (13.04s)	**60.58** (25.6s)	57.54 (7.17s)	57.56 (51.76s)	55.80 (1.70s)	**60.70** **(0.26s)**	57.22 (1.1s)	0.018	53.49 (0.9s)	0.005	**60.8** (0.9s)	0.019
PROTEINS	**73.84** (3580s)	**74.12**(308s)	**74.29** (387s)	59.56 (10.04s)	**73.22** (3287s)	69.71 (10.11s)	70.25 **(1.68s)**	68.10 (6.1s)	0.006	72.14(5.8s)	0.010	**73.32** (5.2s)	0.009
NCI1	>D	71.65 (821s)	75.52 (1.12hr)	50.04 (450s)	58.12 (1647s)	68.87 (29.14s)	**79.75** **(3.61s)**	61.94 (13.3s)	0.003	67.25 (13.1s)	0.002	63.52 (12.1s)	0.008
D&D	>D	**77.94** (4.5hr)	75.81 (7.4hr)	58.65 (120.7s)	>D	73.86 (59.24s)	75.9 (172.87s)	70.21 (25.5s)	0.011	72.33 (24.5s)	0.010	**76.19** **(23.5s)**	0.008

Table 4
Classification accuracy and running times on social network datasets. **'OMR'** indicates out of memory error.

Dataset	NetSimile	FGSD	NetLSD [6]				NetKI	
	[32]	[5]	$w(g)$	std	$h(g)$	std	acc/time	std
COLLAB	**79.96** (1.58hr)	77.04 **(12s)**	74.46(130.6s)	0.003	70.58(132.1s)	0.029	70.3 (120.5s)	0.004
IMDB-B	**74.00** (6.4s)	**73.5(0.99s)**	71.11(3.5s)	0.010	71.9(3.4s)	0.003	70.5 (3.3s)	0.010
IMDB-M	**49.06** (7.90s)	**49.5** **(1.25s)**	**47.73**(3.8s)	0.004	47.13(4.5s)	0.004	**48.2** (3.5s)	0.012
REDDIT-B	**88.05** (303s)	**88.95** (760s)	77.75(91.1s)	0.006	82.74(97.5s)	0.003	**87.5** **(56.3s)**	0.040
REDDIT-M-5K	**51.83** (464s)	**50.2** (1172s)	40.34 (291.2s)	0.090	40.44(307.3s)	0.005	**52.0** **(55.5s)**	0.008
REDDIT-L	OMR	>D	84.82 (12.5hr)	0.005	84.83 (12.5hr)	0.006	**88.07** **(5.5hr)**	0.006

performed all other methods. On social network datasets (Table 4), NetSimile performed quite well and outperformed all other methods on COLLAB and IMDB-B datasets. FGSD outperformed on IMDB-M and REDIIT-B while NetKI has shown highest accuracy on REDDIT-M-5 K and REDDIT-L social networks. The classification accuracy of NetKI on IMDB-M and REDDIT-B is within range 2.0 from the best result. We also report the running times of the algorithms (excluding any data loading or classification time for all algorithms) in brackets in Tables 3 and 4. We can see from the results that on small datasets such as MUTAG, and PTC, etc., the performance of FGSD is quite well as compared to the other algorithms, although it has quadratic time complexity. The main reason of this is that the FGSD implementation simply involves matrix multiplication which can be done quite fast when the size of the

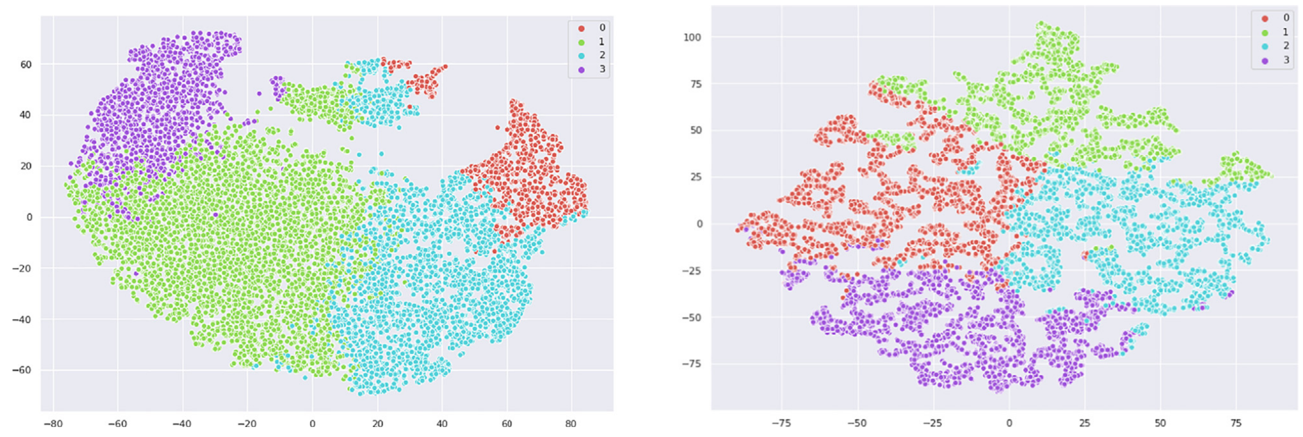

Fig. 4. NetKI (left) and NetLSD (right) clustering visualization of the feature vectors of all small graphs up to size 8 using tSNE.

matrices is small. However, as the size of the matrices increase such as in D&D and REDDIT datasets, the performance of FGSD worsens sharply. In contrast, NetKI performs quite well in terms of running time on large datasets as well as compared to other approaches. However, due to the approximation factor involved and the nature of the datasets, the performance of NetKI in terms of classification accuracy is also worse on NCI1 and COLLAB datasets. Recall that the main objective of NetKI was the scalability on sufficiently large datasets. We show in Tables 3 and 4 that the performance of NetKI is within range 2.0 from the best accuracy on most of the datasets. On the other hand, we can see from the tables and also from Fig. 2 that the running times of NetKI reduces sharply when the size of the datasets increases. To evaluate NetKI in terms of time and accuracy on very large real networks, we also reported classification accuracy and running time on REDDIT-L dataset, which is a very large social network dataset having 97491 average nodes and more than 1 million edges on average (see Table 2). We can see that NetKI outperformed NetLSD and shown quality results with 3.24% improvements in terms of accuracy and 127% improvements in terms of running time. Overall, we can see that NetKI has shown improved results in terms of running time and the classification accuracy is close to the state-of-the-art approaches that are, otherwise, impractical for large datasets. these results demonstrate that NetKI is a suitable graph embedding and classification method for large networks. NetSimile and FGSD could not able to scale on such large graphs therefore, we do not report on them.

5.4. Features comparison to NetLSD

Apart from the methodology perspective, NetKI and NetLSD are directly linked with the graph spectrum and can be observed closely related to each other. Thus, to provide more crisper analysis among them, here we present feature comparison to analyze the information extracted by both the algorithms from the same dataset. We hypothesized that high overlap in clustering on the embeddings generated through both the algorithms will reflect similar feature extraction. To test the hypothesis, we generate feature vectors from all small non-isomorphic graphs up to size 8. The total of them are 12111 number of graphs. For NetKI, we set the length of feature vector to 50 and use the same parameter setup shown above for other experiments. For NetLSD, we use default parameter setting and generate feature vectors from the same dataset. We perform K-means clustering with $k = 4$ on the resultant feature vectors of both the algorithms and find overlap between the clusters. For finding the overlap, we compute the intersection between NetKI each cluster with every cluster of NetLSD. The highest overlap was found 56% between NetKI cluster 1 and NetLSD cluster 0. The overlap between all other cluster was found $< 50\%$. We have also shown tSNE visualization of the resultant clustering of both the algorithm in Fig. 4. We can see that the distributions of both the clustering are different, while the ratio of the overlap is also small. Thus, we conclude that NetKI and NetLSD capture different information from the graph.

5.5. Discussion

Currently, large graphs with millions of nodes and edges have become very common. Examples include online social networks, collaboration networks, and biomedical networks. Scalability on such large networks especially for the encoding methods is one of the key requirements to apply state-of-the-art machine learning methods [1]. There has been tremendous growth in graph representation methods, however, very few efforts have been devoted to enhance the performance of these methods and make them scalable on sufficiently large networks. The extensive experiments on well-known state-of-the-art methods on several networks shown in Section 5.3 reveal that the existing methods are still incapable of dealing with sufficiently large networks. And thus, there is a necessity of the encoding methods that are scalable on large networks.

On large real-world datasets such as REDDIT, D&D, and other large random networks, we show that NetKI runs very fast as compared to the existing approaches while attaining quality results in terms of classification accuracy. This demonstrates the effectiveness of NetKI on different kinds of networks, in particular, when the networks are sufficiently large and well-connected. In addition to the application of such efficient approaches in various domains, recently, the graph encoding methods have also been applied with GCNs collectively to improve the performance of the algorithms [69]. Such approaches help GCNs to generalize on multiple domains data. Due to the efficiency and quality results of NetKI, we believe that it is a suitable method that can be combined with end-to-end deep learning models to design efficient and universal graph embedding models.

6. Conclusion

We introduced NetKI, a novel method for graphs classification. NetKI is purely based on graph statistical representation that can be computed efficiently in nearly linear time. We proposed using the network Kirchhoff index as a function of graph representation $f(g_h)$ and show its approximation in nearly-linear time. NetKI does not require any graph summary statistics or node attributes and relies only on graph structure. We performed extensive experiments on various real-world benchmark datasets and synthetic networks with millions of nodes and edges, and show that NetKI processes huge graphs and produce sparse representations which lead to attain good quality results.

CRediT authorship contribution statement

Anwar Said: Data curation, Methodology, Investigation, Writing - original draft. **Saeed-Ul Hassan:** Supervision, Conceptualization, Methodology, Writing - original draft. **Waseem Abbas:** Writing - original draft, Methodology, Investigation. **Mudassir Shabbir:** Supervision, Investigation, Writing - original draft.

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.

Acknowledgments

The authors (Saeed-Ul Hassan & Mudassir Shabbir) were funded by the CIPL National Center in Big Data and Cloud Computing (NCBC) grant, received from the Planning Commission of Pakistan, through Higher Education Commission (HEC) of Pakistan.

References

[1] W.L. Hamilton, Z. Ying, J. Leskovec, Representation learning on graphs: methods and applications, IEEE Data Eng. Bull. 40 (2017) 52–74.
[2] P. Goyal, E. Ferrara, Graph embedding techniques, applications, and performance: a survey, Knowl.-Based Syst. 151 (2018) 78–94.
[3] A. Sanfeliu, K.-S. Fu, A distance measure between attributed relational graphs for pattern recognition, IEEE Trans. Syst. Man Cybern. (3) (1983) 353–362.
[4] R. Kondor, N. Shervashidze, K.M. Borgwardt, The graphlet spectrum, in: Proceedings of the 26th Annual International Conference on Machine Learning ACM, 2009, pp. 529–536.
[5] S. Verma, Z.-L. Zhang, Hunt for the unique, stable, sparse and fast feature learning on graphs, Adv. Neural Inform. Process. Syst. (2017) 88–98.
[6] 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.
[7] A. Ghosh, S. Boyd, A. Saberi, Minimizing effective resistance of a graph, SIAM Rev. 50 (1) (2008) 37–66.
[8] W. Abbas, M. Egerstedt, Robust graph topologies for networked systems, IFAC Proceedings Volumes 45 (26) (2012) 85–90.
[9] W. Abbas, M. Shabbir, A.Y. Yazicioglu, A. Akber, On the trade-off between controllability and robustness in networks of diffusively coupled agents, arXiv preprint arXiv:1903.05524..
[10] W. Ellens, R.E. Kooij, Graph measures and network robustness, arXiv preprint arXiv:1311.5064..
[11] W. Ellens, F. Spieksma, P. Van Mieghem, A. Jamakovic, R. Kooij, Effective graph resistance, Linear Algebra Appl. 435 (10) (2011) 2491–2506.
[12] X. Wang, E. Pournaras, R.E. Kooij, P. Van Mieghem, Improving robustness of complex networks via the effective graph resistance, Eur. Phys. J. B 87 (9) (2014) 221.
[13] R. Kyng, S. Sachdeva, Approximate gaussian elimination for laplacians-fast, sparse, and simple, IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS), IEEE 2016 (2016) 573–582.
[14] H. Li, Z. Zhang, Kirchhoff index as a measure of edge centrality in weighted networks: Nearly linear time algorithms, in: Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, Society for Industrial and Applied Mathematics, 2018, pp. 2377–2396..
[15] 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.
[16] R. Kondor, H. Pan, The multiscale laplacian graph kernel, Adv. Neural Inform. Process. Syst. (2016) 2990–2998.
[17] J. Pang, Y. Gu, J. Xu, G. Yu, Semi-supervised multi-graph classification using optimal feature selection and extreme learning machine, Neurocomputing 277 (2018) 89–100.
[18] D. Koutra, J.T. Vogelstein, C. Faloutsos, Deltacon: a principled massive-graph similarity function, in, in: Proceedings of the 2013 SIAM International Conference on Data Mining SIAM, 2013, pp. 162–170.
[19] J. Bento, S. Ioannidis, A family of tractable graph distances, in: Proceedings of the 2018 SIAM International Conference on Data Mining, 2018, pp. 333–341.
[20] A. Fischer, C.Y. Suen, V. Frinken, K. Riesen, H. Bunke, Approximation of graph edit distance based on hausdorff matching, Pattern Recogn. 48 (2) (2015) 331–343.
[21] T. Ma, W. Shao, Y. Hao, J. Cao, Graph classification based on graph set reconstruction and graph kernel feature reduction, Neurocomputing 296 (2018) 33–45.
[22] 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..
[23] F. Li, Z. Zhu, X. Zhang, J. Cheng, Y. Zhao, Diffusion induced graph representation learning, Neurocomputing..
[24] 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.
[25] F. Orsini, P. Frasconi, L. De Raedt, Graph invariant kernels, in: Twenty-Fourth International Joint Conference on Artificial Intelligence, 2015..
[26] Z. Wang, Y. Zhao, G. Wang, Y. Li, X. Wang, On extending extreme learning machine to non-redundant synergy pattern based graph classification, Neurocomputing 149 (2015) 330–339.
[27] M. Sugiyama, K. Borgwardt, Halting in random walk kernels, in: Advances in neural information processing systems, 2015, pp. 1639–1647..
[28] S. Hido, H. Kashima, A linear-time graph kernel, in: 2009 Ninth IEEE International Conference on Data Mining, IEEE, 2009, pp. 179–188.
[29] N. Pržulj, Biological network comparison using graphlet degree distribution, Bioinformatics 23 (2) (2007) e177–e183.
[30] M. Togninalli, E. Ghisu, F. Llinares-López, B. Rieck, K. Borgwardt, Wasserstein weisfeiler-lehman graph kernels, Adv. Neural Inform. Process Syst. (2019) 6436–6446.
[31] C. Bock, M. Togninalli, E. Ghisu, T. Gumbsch, B. Rieck, K. Borgwardt, A wasserstein subsequence kernel for time series, in: 19th IEEE International Conference on Data Mining (ICDM 2019), 2019.
[32] M. Berlingerio, D. Koutra, T. Eliassi-Rad, C. Faloutsos, Network similarity via multiple social theories, in, in: Proceedings of the 2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining, ACM, 2013, pp. 1439–1440.
[33] A.M. Bronstein, M.M. Bronstein, L.J. Guibas, M. Ovsjanikov, Shape google: Geometric words and expressions for invariant shape retrieval, ACM Transactions on Graphics (TOG) 30 (1) (2011) 1..
[34] H.P. Maretic, M. El Gheche, G. Chierchia, P. Frossard, Got: An optimal transport framework for graph comparison, Adv. Neural Inform. Process. Syst. (2019) 13876–13887.
[35] R. Flamary, N. Courty, A. Rakotomamonjy, D. Tuia, Optimal transport with laplacian regularization, 2014..
[36] S. Ferradans, N. Papadakis, G. Peyré, J.-F. Aujol, Regularized discrete optimal transport, SIAM J. Imaging Sci. 7 (3) (2014) 1853–1882.
[37] G. Nikolentzos, P. Meladianos, M. Vazirgiannis, Matching node embeddings for graph similarity, in: Thirty-First AAAI Conference on Artificial Intelligence, 2017..
[38] 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..
[39] G. Mena, D. Belanger, S. Linderman, J. Snoek, Learning latent permutations with gumbel-sinkhorn networks, arXiv preprint arXiv:1802.08665..
[40] P. Emami, S. Ranka, Learning permutations with sinkhorn policy gradient, arXiv preprint arXiv:1805.07010..
[41] Z.R. Hassan, M. Shabbir, I. Khan, W. Abbas, Estimating descriptors for large graphs, arXiv preprint arXiv:2001.10301..
[42] I. Jovanović, Z. Stanić, Spectral distances of graphs, Linear Algebra Its Appl. 436 (5) (2012) 1425–1435.
[43] T. Vayer, L. Chapel, R. Flamary, R. Tavenard, N. Courty, Optimal transport for structured data with application on graphs, arXiv preprint arXiv:1805.09114..
[44] G. Peyré, M. Cuturi, J. Solomon, Gromov-wasserstein averaging of kernel and distance matrices, in: International Conference on Machine Learning, 2016, pp. 2664–2672..
[45] F. Scarselli, M. Gori, A.C. Tsoi, M. Hagenbuchner, G. Monfardini, The graph neural network model, IEEE Trans. Neural Networks 20 (1) (2009) 61–80.
[46] 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..
[47] T.N. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks, International Conference on Learning Representations (ICLR) (2017).
[48] S. Fu, W. Liu, Y. Zhou, L. Nie, Hplapgcn: hypergraph p-laplacian graph convolutional networks, Neurocomputing 362 (2019) 166–174.
[49] K. Xu, W. Hu, J. Leskovec, S. Jegelka, How powerful are graph neural networks?, International Conference on Learning Representations (ICLR) (2019), URL: https://openreviewnet/forum?id=ryGs6iA5Km.
[50] P. Veličković, W. Fedus, W.L. Hamilton, P. Liò, Y. Bengio, R.D. Hjelm, Deep graph infomax, International Conference on Learning Representations (ICLR) (2019), URL: https://openreview.net/forum?id=rklz9iAcKQ.
[51] 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: Association for the Advancement of Artificial Intelligence (AAAI), 2019..

[52] 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.
[53] 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.
[54] D.A. Spielman, N. Srivastava, Graph sparsification by effective resistances, SIAM J. Comput. 40 (6) (2011) 1913–1926.
[55] D.A. Spielman, S.-H. Teng, Nearly linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems, SIAM J. Matrix Anal. Appl. 35 (3) (2014) 835–885.
[56] D. Durfee, R. Kyng, J. Peebles, A.B. Rao, S. Sachdeva, Sampling random spanning trees faster than matrix multiplication, in, in: Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, ACM, 2017, pp. 730–742.
[57] M.B. Cohen, R. Kyng, G.L. Miller, J.W. Pachocki, R. Peng, A.B. Rao, S.C. Xu, Solving sdd linear systems in nearly m log 1/2 n time, in: Proceedings of the forty-sixth annual ACM symposium on Theory of computing, ACM, 2014, pp. 343–352..
[58] D.A. Spielman, S.-H. Teng, Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems, in: Proceedings of the STOC, vol. 4, 2004..
[59] I. Koutis, G.L. Miller, R. Peng, Approaching optimality for solving sdd linear systems, SIAM J. Comput. 43 (1) (2014) 337–354.
[60] R. Peng, D.A. Spielman, An efficient parallel solver for sdd linear systems, in: Proceedings of the forty-sixth annual ACM symposium on Theory of computing, ACM, 2014, pp. 333–342..
[61] R. Kyng, Y.T. Lee, R. Peng, S. Sachdeva, D.A. Spielman, Sparsified cholesky and multigrid solvers for connection laplacians, in: Proceedings of the forty-eighth annual ACM symposium on Theory of Computing, ACM, 2016, pp. 842–850..
[62] C. Godsil, G.F. Royle, Algebraic graph theory, vol. 207, Springer Science & Business Media, 2013..
[63] J. Sherman, W.J. Morrison, Adjustment of an inverse matrix corresponding to a change in one element of a given matrix, Ann. Math. Stat. 21 (1) (1950) 124–127.
[64] M.F. Hutchinson, A stochastic estimator of the trace of the influence matrix for laplacian smoothing splines, Commun. Stat.-Simul. Comput. 19 (2) (1990) 433–450.
[65] H. Avron, S. Toledo, Randomized algorithms for estimating the trace of an implicit symmetric positive semi-definite matrix, J .ACM (JACM) 58(2) (2011) 8..
[66] W.B. Johnson, J. Lindenstrauss, Extensions of lipschitz mappings into a hilbert space, Contemporary Math. 26 (189–206) (1984) 1.
[67] K. Huang, S. Aviyente, Sparse representation for signal classification, in: Advances in neural information processing systems, 2007, pp. 609–616..
[68] K. Kersting, N.M. Kriege, C. Morris, P. Mutzel, M. Neumann, Benchmark data sets for graph kernels, 2016.http://graphkernels.cs.tu-dortmund.de..
[69] S. Verma, Z.-L. Zhang, Deep universal graph embedding neural network, arXiv preprint arXiv:1909.10086..

Anwar Said: Mr. Said is a PhD research scholar in AI AI Lab, Department of Computer Science at Information Technology University, Lahore, Pakistan. He received MPhil (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: Dr. Hassan is the Director of AI AI Lab and the Chairperson in the De of Computer Science 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. Hassan'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.

Waseem Abbas: Dr. Abbas is a Research Assistant Professor in the Electrical Engineering and Computer Science Department at the Vanderbilt University, Nashville, TN, USA. Previously, he was an Assistant Professor at the Information Technology University Lahore in Pakistan, and a postdoctoral research scholar at the Vanderbilt University between 2014 and 2017. He received Ph.D. (2013) and M.Sc. (2010) degrees, both in Electrical and Computer Engineering, from Georgia Institute of Technology, Atlanta, GA, and was a Fulbright scholar from 2009 till 2013. His research interests are in the areas of resilience and security of network control systems, cyber-physical systems, and graph-theoretic methods in complex networks.

Mudassir Shabbir: Dr. 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.