Multiple Kernel Representation Learning on Networks

Abdulkadir Çelikkanat, Yanning Shen, Fragkiskos D. Malliaros

Multiple Kernel Representation Learning on Networks

Abdulkadir Çelikkanat, Yanning Shen, and Fragkiskos D. Malliaros

Abstract—Learning representations of nodes in a low dimensional space is a crucial task with numerous interesting applications in network analysis, including link prediction, node classification, and visualization. Two popular approaches for this problem are *matrix factorization* and *random walk*-based models. In this paper, we aim to bring together the best of both worlds, towards learning node representations. In particular, we propose a weighted matrix factorization model that encodes random walk-based information about nodes of the network. The benefit of this novel formulation is that it enables us to utilize kernel functions without realizing the exact proximity matrix so that it enhances the expressiveness of existing matrix decomposition methods with kernels and alleviates their computational complexities. We extend the approach with a multiple kernel learning formulation that provides the flexibility of learning the kernel as the linear combination of a dictionary of kernels in data-driven fashion. We perform an empirical evaluation on real-world networks, showing that the proposed model outperforms baseline node embedding algorithms in downstream machine learning tasks.

Index Terms—Graph representation learning, node embeddings, kernel methods, node classification, link prediction

1 Introduction

With the development in data production, storage and consumption, graph data is becoming omnipresent; data from diverse disciplines can be represented as graph structures with prominent examples including various social, information, technological, and biological networks. Developing machine learning algorithms to analyze, predict, and make sense of graph data has become a crucial task with a plethora of cross-disciplinary applications [1], [2]. The major challenge in machine learning on graph data concerns the encoding of information about the graph structural properties into the learning model. To this direction, *network representation learning* (NRL), a recent paradigm in network analysis, has received substantial attention thanks to its outstanding performance in downstream tasks, including link prediction and classification. Representation learning techniques mainly target to embed the nodes of the graph into a lower-dimensional space in such a way that desired relational properties among graph nodes are captured by the similarity of the representations in the embedding space [3], [4], [5], [6], [7], [8].

The area of NRL has been highly impacted by traditional nonlinear dimensionality reduction approaches [5], [9]. Specifically, many proposed models had initially concentrated on learning node embeddings relying on matrix factorization techniques that encode structural node similarity [5], [6], [7]. Nevertheless, most of those approaches are not very efficient for large scale networks, mainly due to the high computational and memory cost required to perform matrix factorization. Besides, most such models require the exact realization of the target matrix [1], [2].

Inspired by the field of natural language processing [10], random-walk based embedding has gained considerable attention (e.g., [3], [4], [11], [12], [13], [14]). Typically, these methods firstly produce a set of node sequences by following certain random walk strategy. Then, node representations are learned by optimizing the relative co-occurrence probabilities of nodes within the random walks. Although such random walk models follow similar approaches in modeling the relationships between nodes and optimizing the embedding vectors, their difference mainly stems from the way in which node sequences are sampled [3], [4].

On a separate topic, *kernel functions* have often been introduced along with popular learning algorithms, such as PCA [15], SVM [16], Spectral Clustering [17], and Collaborative Filtering [18], to name a few. Most of traditional learning models are insufficient to capture the underlying substructures of complex datasets, since they rely on linear techniques to model nonlinear patterns existing in data. Kernel functions [19], on the other hand, allow mapping non-linearly separable points into a (generally) higher dimensional feature space, so that the inner product in the new space can be computed without needing to compute the exact feature maps—bringing further computational benefits. Besides, to further reduce model bias, *multiple kernel learning* approaches have been proposed to learn optimal combinations of kernel functions [20]. Nevertheless, despite their wide applications in various fields [19], [21], kernel and multiple kernel methods have not been thoroughly investigated for learning node embeddings.

In this paper, we aim at combining matrix factorization and random walks in a kernelized model for learning node

- *A. Çelikkanat is with the Department of Applied Mathematics and Computer Science, Technical University of Denmark, 2800 Kgs. Lyngby, Denmark. The research has been mostly conducted while the author was with Paris-Saclay University, CentraleSupélec, Inria, Centre for Visual Computing (CVN), 91190 Gif-Sur-Yvette, France*
  E-mail: abdcelikkanat@gmail.com
- *Y. Shen is with the Department of EECS and the Center for Pervasive Communications and Computing, University of California, Irvine, CA 92697, USA*
  E-mail: yannings@uci.edu
- *F. D. Malliaros is with Paris-Saclay University, CentraleSupélec, Inria, Centre for Visual Computing (CVN), 91190 Gif-Sur-Yvette, France*
  E-mail: fragkiskos.malliaros@centralesupelec.fr

Manuscript received XXX; revised XXX.

embeddings. The potential advantage of such a modeling approach is that it allows for leveraging and combining the elegant mathematical formulation offered by matrix factorization with the expressive power of random walks to capture a notion of "stochastic" node similarity in an efficient way. More importantly, this formulation enables leveraging kernel functions in the node representation learning task. Because of the nature of matrix factorization-based models, node similarities can be viewed as inner products of vectors lying in a latent space—which allows to utilize kernels towards interpreting the embeddings in a higher dimensional feature space using non-linear maps. Besides, multiple kernels can be utilized to learn more discriminative node embeddings. Note that, although graph kernels is a well-studied topic in graph learning [22], it mainly focuses on graph classification—a task outside of the scope of this paper. To the best of our knowledge, random walk-based multiple kernel matrix factorization has not been studied before for learning node embeddings.

The main contributions of the present work can be summarized as follows:

- We propose KERNELNE (Kernel Node Embeddings), a novel approach for learning node embeddings by incorporating kernel functions with models relying on weighted matrix factorization, encoding random walk-based structural information of the graph. We further examine the performance of the model with different types of kernel functions.
- To further improve expressiveness, we introduce MKERNELNE, a multiple kernel learning formulation of the model. It extends the kernelized weighted matrix factorization framework by learning a linear combination of a predefined set of kernels.
- We demonstrate how both models (single and multiple kernel learning) leverage negative sampling to efficiently compute node embeddings.
- We extensively evaluate the proposed method's performance in node classification and link prediction. We show that the proposed models generally outperform well-known baseline methods on various real-world graph datasets. Besides, due to the efficient model optimization mechanism, the running time is comparable to the one of random walk models.

Notation. We use the notation $\mathbf{M}$ to denote a matrix, and the term $\mathbf{M}_{(v,u)}$ represents the entry at the v'th row and u'th column of the matrix $\mathbf{M}$. $\mathbf{M}_{(v,:)}$ indicates the v'th row of the matrix.

Source code. The C++ implementation of the proposed methodology can be reached at: `https://abdcelikkanat.github.io/projects/kernelNE/`.

2 RELATED WORK

Node embeddings. Traditional dimension reduction methods such as Principal Component Analysis (PCA) [23], Locally Linear Embeddings [9], and ISOMAP [24] that rely on the matrix factorization techniques, have considerably influenced early approaches on learning node embeddings. For instance, HOPE [7] learns embedding vectors by factorizing a higher-order proximity matrix using SVD, while M-NMF [25] leverages non negative matrix factorization aiming to preserve the underlying community structure of the graph. Similarly, NETMF [26] and its scalable variant NETSMF [27], learn embeddings by properly factorizing a target matrix that has been designed based on the Positive Pointwise Mutual Information (PPMI) of random walk proximities. Similarly, SDAE [28] uses the PPMI matrix with stacked denoising autoencoders to learn the representations. SDNE [29] adapts a deep neural network architecture to capture the highly non-linear network patterns. Nevertheless, as we have mentioned in Sec. 1, most of these models suffer from high time complexity, while at the same time, they assume an exact realization of the target matrix. For a detailed description of matrix factorization models for node embeddings, the reader can refer to [1], [2], [30].

To further improve the expressiveness as well as to alleviate the computational burden of matrix factorization approaches, models that rely on random walk sampling have been developed (e.g., [3], [4], [11], [12], [13], [14], [31], [32], [33]). These models aim at modeling *center-context* node relationships in random walk sequences, leveraging advances on learning word embeddings in natural language processing [10]. Due to the flexible way that random walks can be defined, various formulations have been proposed, with the two most well-known being DEEPWALK [3] and NODE2VEC [4]. Most of these models though, are based on shallow architectures which do not allow to model complex non-linear data relationships, limiting the prediction capabilities on downstream tasks (e.g., node classification) [30]. There are also recent efforts towards developing scalable models relying on fast random projections for very large networks [34], [35], [36], [37].

As will be presented shortly, in this paper we aim to bring together the best of both worlds, by proposing an efficient random walk-based matrix factorization framework that allows to learn informative embeddings.

Kernel methods. Although most algorithms in the broader field of machine learning have been developed for the linear case, real-world data often requires nonlinear models capable of unveiling the underlying complex relationships towards improving the performance of downstream tasks. To that end, kernel functions allow computing the inner product among data points in a typically high-dimensional feature space, in which linear models could still be applied, without explicitly computing the feature maps [19]. Because of the generality of the inner product, kernel methods have been widely used in a plethora of models, including Support Vector Machine [16], PCA [15], spectral clustering [17], [38], collaborative filtering [18], and non-negative matrix factorization for image processing tasks [39], to name a few.

Kernel functions have also been utilized in the field graph analysis. At the node level, diffusion kernels and their applications [40] constitute notable instances. At the graph level, *graph kernels* [41], such as the random walk and Weisfeiler-Lehman kernels [22], have mainly been utilized to measure the similarity between a pair of graphs for applications such as graph classification. Besides, they have also used for capturing the temporal changes in time-evolving networks [42]. Recent approaches have also been proposed to leverage graph kernels for node classification, but in a

supervised manner [43]. Other related applications of kernel methods on graphs include topology inference [44], [45], signal reconstruction [46], and anomaly detection [47].

The expressiveness of kernel methods can further be enhanced using multiple kernel functions in a way that the best possible combination of a set of predefined kernels can be learned [20], [48]. Besides improving prediction performance, multiple kernels have also been used to combine information from distinct heterogeneous sources (please see [20] and [49] for a detailed presentation of several multiple kernel learning algorithms and their applications). Examples include recent approaches that leverage multi-kernel strategies for graph and image datasets [50], [51].

Despite the widespread applications of kernel and multiple kernel learning methods, utilizing them for learning node embeddings via matrix factorization in an unsupervised way is a problem that has not been thoroughly investigated. Previous works that are close to our problem settings, which follow a methodologically different factorization approach (e.g., leveraging nonnegative matrix factorization [18]), targeting different application domains (e.g., collaborative filtering [52]). The multiple kernel framework for graph-based dimensionality reduction proposed by Lin et al. [53] is also close to our work. Nevertheless, their work focuses mainly on leveraging multiple kernel functions to fuse image descriptors in computer vision tasks properly.

In this paper we propose novel unsupervised models for node embeddings, that implicitly perform weighted matrix decomposition in a higher-dimensional space through kernel functions. The target pairwise node proximity matrix is properly designed to leverage information from random walk sequences, and thus, our models do not require the exact realization of this matrix. Both single and multiple kernel learning formulations are studied. Emphasis is also put on optimization aspects to ensure that node embeddings are computed efficiently.

3 Modeling and Problem Formulation

Let $G = (\mathcal{V}, \mathcal{E})$ be a graph where $\mathcal{V} = \{1, \ldots, n\}$ and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ are the vertex and edge sets, respectively. Our goal is to find node representations in a latent space, preserving properties of the network. More formally, we define the general objective function of our problem as a weighted matrix factorization [54], as follows:

$$\arg\min_{\mathbf{A},\mathbf{B}} \underbrace{\left\|\mathbf{W} \odot (\mathbf{M} - \mathbf{A}\mathbf{B}^\top)\right\|_F^2}_{\text{Error term}} + \underbrace{\frac{\lambda}{2}\left(\|\mathbf{A}\|_F^2 + \|\mathbf{B}\|_F^2\right)}_{\text{Regularization term, } \mathcal{R}(\mathbf{A},\mathbf{B})}, \quad (1)$$

where $\mathbf{M} \in \mathbb{R}^{n \times n}$ is the target matrix constructed based on the desired properties of a given network, which is used to learn node embeddings $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{n \times d}$, and $\|\cdot\|_F$ denotes the *Frobenius* norm. We will use $\mathcal{R}(\mathbf{A}, \mathbf{B})$ to denote the regularization term of Eq. (1). Each element $\mathbf{W}_{v,u}$ of the weight matrix $\mathbf{W} \in \mathbb{R}^{n \times n}$ captures the importance of the approximation error between nodes v and u, and $\odot$ indicates the *Hadamard* product. Depending on the desired graph properties that we are interested in encoding, there are many possible alternatives to choose matrix $\mathbf{M}$; such include the number of common neighbors between a pair of nodes, higher-order node proximity based on the *Adamic-Adar* or *Katz* indices [7], as well leveraging multi-hop information [6]. Here, we will design $\mathbf{M}$ as a sparse binary matrix utilizing information of random walks over the network.

As we have mentioned above, random walk-based node embedding models (e.g., [3], [4], [11], [14], [55], [56], [57]) have received great attention because of their good prediction performance and efficiency on large scale-networks. Typically, those models generate a set of node sequences by simulating random walks; node representations are then learned by optimizing a model which defines the relationships between nodes and their *contexts* within the walks. More formally, for a random walk $\mathbf{w} = (w_1, \ldots, w_L)$, the *context set* of *center* node $w_l \in \mathcal{V}$ appearing at the position l of the walk $\mathbf{w}$ is defined by $\{w_{l-\gamma}, \ldots, w_{l-1}, w_{l+1}, \ldots, w_{l+\gamma}\}$, where γ is the *window size* which is the furthest distance between the *center* and *context* nodes. The embedding vectors are then obtained by maximizing the likelihood of occurrences of nodes within the context of given center nodes. Here, we will also follow a similar random walk strategy, formulating the problem using a matrix factorization framework.

Let $\mathbf{M}_{(v,u)}$ be a binary value which equals to 1 if node u appears in the context of v in any walk. Also, let $\mathbf{F}_{(v,u)}$ be $2 \cdot \gamma \cdot \#(v)$, where $\#(v)$ indicates the total number of occurrences of node v in the generated walks. Setting each term $\mathbf{W}_{(v,u)}$ as the square root of $\mathbf{F}_{(v,u)}$, the objective function in Eq. (1) can be expressed under a random walk-based formulation as follows:

$$\begin{aligned}
&\arg\min_{\mathbf{A},\mathbf{B}} \left\|\mathbf{W} \odot \left(\mathbf{M} - \mathbf{A}\mathbf{B}^\top\right)\right\|_F^2 + \mathcal{R}(\mathbf{A},\mathbf{B}) \\
&= \arg\min_{\mathbf{A},\mathbf{B}} \left\|\sqrt{\mathbf{F}} \odot \left(\mathbf{M} - \mathbf{A}\mathbf{B}^\top\right)\right\|_F^2 + \mathcal{R}(\mathbf{A},\mathbf{B}) \\
&= \arg\min_{\mathbf{A},\mathbf{B}} \sum_{v\in\mathcal{V}}\sum_{u\in\mathcal{V}} \mathbf{F}_{(v,u)}\Big(\mathbf{M}_{(v,u)} - \Big\langle \mathbf{A}[u], \mathbf{B}[v]\Big\rangle\Big)^2 + \mathcal{R}(\mathbf{A},\mathbf{B}) \\
&= \arg\min_{\mathbf{A},\mathbf{B}} \sum_{v\in\mathcal{V}}\sum_{u\in\mathcal{V}} 2\cdot\gamma\cdot\#(v)\Big(\mathbf{M}_{(v,u)} - \Big\langle \mathbf{A}[u], \mathbf{B}[v]\Big\rangle\Big)^2 + \mathcal{R}(\mathbf{A},\mathbf{B}) \\
&= \arg\min_{\mathbf{A},\mathbf{B}} \sum_{v\in\mathcal{V}}\sum_{u\in\mathcal{V}}\sum_{s\in\mathcal{V}} \#(v,s)\Big([u=s] - \Big\langle \mathbf{A}[u], \mathbf{B}[v]\Big\rangle\Big)^2 + \mathcal{R}(\mathbf{A},\mathbf{B}) \\
&= \arg\min_{\mathbf{A},\mathbf{B}} \sum_{v\in\mathcal{V}}\sum_{s\in\mathcal{V}} \#(v,s)\sum_{u\in\mathcal{V}}\Big([u=s] - \Big\langle \mathbf{A}[u], \mathbf{B}[v]\Big\rangle\Big)^2 + \mathcal{R}(\mathbf{A},\mathbf{B}) \\
&= \arg\min_{\mathbf{A},\mathbf{B}} \sum_{(v,s)\in\mathcal{V}^2} \#(v,s)\sum_{u\in\mathcal{V}}\Big([u=s] - \Big\langle \mathbf{A}[u], \mathbf{B}[v]\Big\rangle\Big)^2 + \mathcal{R}(\mathbf{A},\mathbf{B}) \\
&= \arg\min_{\mathbf{A},\mathbf{B}} \sum_{\mathbf{w}\in\mathcal{W}}\sum_{l=1}^{L}\sum_{\substack{|j|\le\gamma \\ j\neq 0}}^{\gamma}\sum_{u\in\mathcal{V}}\Big([\{w_{l+j}=u] - \Big\langle \mathbf{A}[u], \mathbf{B}[w_l]\Big\rangle\Big)^2 + \\
&\qquad\qquad \mathcal{R}(\mathbf{A},\mathbf{B}), \qquad (2)
\end{aligned}$$

where each $\mathbf{w} \in \mathcal{V}^L$ indicates a random walk of length L in the collection $\mathcal{W}$, $\#(v,s)$ denotes the number of appearances of the center and context pairs (v,s) in the collection $\mathcal{W}$, $[\cdot]$ is the the Iverson bracket, and $\mathcal{R}(\mathbf{A},\mathbf{B})$ is the regularization term. Note that, in the equation above, the last line follows from the fact that $\#(v,s)$ is equal to

zero for the pairs which are not center-context, and the term $\sum_{\mathbf{w}\in\mathcal{W}}\sum_{l=1}^{L}\sum_{|j|\leq\gamma}^{\gamma}$ traverses all center-context pairs in the collection. Matrix $\mathbf{A}$ in Eq. (2) indicates the embedding vectors of nodes when they are considered as *centers*; those will be the embeddings that are used in the experimental evaluation. The choice of matrix $\mathbf{M}$ and the reformulation of the objective function, offers a computational advantage during the optimization step. Moreover, such formulation also allows us to further explore *kernelized* version in order to exploit possible non-linearity of the model.

4 Kernel-based Representation Learning

Most matrix factorization techniques that aim to find latent low-dimensional representations (e.g., [7], [26], [27]), adopt the *Singular Value Decomposition* (SVD) provides the best approximation of the objective function stated in Eq. (1), as long as the weight matrix is uniform [58]. Nevertheless, in our case the weight matrix is not uniform, therefore we need the exact realization of the target matrix in order to perform SVD. To overcome this limitation, we leverage kernel functions to learn node representations via matrix factorization.

Let (X, d_X) be a metric space and $\mathbb{H}$ be a Hilbert space of real-valued functions defined on X. A Hilbert space is called *reproducing kernel Hilbert space (RKHS)* if the point evaluation map over $\mathbb{H}$ is a continuous linear functional. Furthermore, a *feature map* is defined as a function $\Phi : \mathsf{X} \to \mathbb{H}$ from the input space X into *feature space* $\mathbb{H}$. Every feature map defines a *kernel* $\mathsf{K} : \mathsf{X} \times \mathsf{X} \to \mathbb{R}$ as follows:

$$\mathsf{K}(\mathbf{x}, \mathbf{y}) := \langle \Phi(\mathbf{x}), \Phi(\mathbf{y}) \rangle \qquad \forall (\mathbf{x}, \mathbf{y}) \in \mathsf{X}^2.$$

It can be seen that $\mathsf{K}(\cdot, \cdot)$ is symmetric and positive definite due to the properties of an inner product space.

A function $g : \mathsf{X} \to \mathbb{R}$ is *induced by* K, if there exists $h \in \mathbb{H}$ such that $g = \langle h, \Phi(\cdot) \rangle$ for a feature vector Φ of kernel K. Note that, this is independent of the definition of the feature map Φ and space $\mathbb{H}$ [59]. Let $\mathcal{I}_\kappa := \{g : \mathsf{X} \to \mathbb{R} \mid \exists h \in \mathbb{H} \text{ s.t. } g = \langle h, \Phi(\cdot) \rangle\}$ be the set of induced functions by kernel K. Then, a continuous kernel K on a compact metric space $(\mathsf{X}, d_\mathsf{X})$ is *universal*, if the set $\mathcal{I}_\kappa$ is dense in the space of all continuous real-valued functions $\mathcal{C}(\mathsf{X})$. In other words, for any function $f \in \mathcal{C}(\mathsf{X})$ and $\epsilon > 0$, there exists $g_h \in \mathcal{I}_\kappa$ satisfying

$$\|f - g_h\|_\infty \leq \epsilon,$$

where g_h is defined as $\langle h, \Phi(\cdot) \rangle$ for some $h \in \mathbb{H}$.

In this paper, we consider universal kernels, since we can always find $h \in \mathbb{H}$ satisfying $|\langle h, \phi(x_i) \rangle - \alpha_i| \leq \epsilon$ for given $\{x_1, \ldots, x_N\} \subset \mathsf{X}$, $\{\alpha_1, \ldots, \alpha_N\} \subset \mathbb{R}$ and $\epsilon > 0$ by Proposition 1. If we choose α_i's as the entries of a row of our target matrix $\mathbf{M}$, then the elements h and $\phi(x_i)$ indicate the corresponding row vectors of $\mathbf{A}$ and $\mathbf{B}$, respectively. Then, we can obtain a decomposition of the target matrix by repeating the process for each row. However, the element h might not always be in the range of feature map Φ; in this case, we will approximate the correct values of $\mathbf{M}$.

Proposition 1 (Universal kernels [59])**.** *Let (X, d) be a compact metric space and $\mathsf{K}(\cdot, \cdot)$ be a universal kernel on X. Then, for all compact and mutually disjoint subsets $\mathsf{S}_1, \ldots, \mathsf{S}_n \subset \mathsf{X}$, all $\alpha_1, \ldots, \alpha_n \in \mathbb{R}$, and all $\epsilon > 0$, there exists a function g induced by K with $\|g\|_\infty \leq \max_i |\alpha_i| + \epsilon$ such that*

$$\left\| g_{|\mathsf{S}} - \sum_{i=1}^{n} \alpha_i \mathbb{1}_{\mathsf{S}_i} \right\|_\infty \leq \epsilon,$$

where $\mathsf{S} := \bigcup_{i=1}^{n} \mathsf{S}_i$ and $g_{|\mathsf{S}}$ is the restriction of g to S.

Universal kernels also provide a guarantee for the injectivity of the feature maps, as shown in Lemma 2; therefore, we can always find $y \in \mathsf{X}$, such that $\Phi(y) = h$ if $h \in \Phi(\mathsf{X})$. Otherwise, we can learn an approximate pre-image solution by using a gradient descent technique [60].

Lemma 2 ([59])**.** *Every feature map of a universal kernel is injective.*

Proof. Let $\Phi(\cdot) : \mathsf{X} \to \mathbb{H}$ be a feature vector of the kernel $\mathsf{K}(\cdot, \cdot)$. Assume that Φ is not injective, so we can find a pair of distinct elements $x, y \in \mathsf{X}$ such that $\Phi(x) = \Phi(y)$ and $x \neq y$. By Proposition 1, for any given $\epsilon > 0$, there exists a function $g = \langle h, \Phi(\cdot) \rangle$ induced by K for some $h \in \mathbb{H}$, which satisfies

$$\left\| g_{|S} - (\mathbb{1}_{S_1} - \mathbb{1}_{S_2}) \right\|_\infty \leq \epsilon,$$

where $\mathsf{S} := \mathsf{S}_1 \cup \mathsf{S}_2$ for the compact sets $\mathsf{S}_1 = \{x\}$ and $\mathsf{S}_2 = \{y\}$. Then, we have that $|\langle \Phi(x), h \rangle - \langle \Phi(y), h \rangle)| \geq 2 - 2\epsilon$. In other words, we obtain $\Phi(x) \neq \Phi(y)$, which contradicts our initial assumption. □

4.1 Single Kernel Node Representation Learning

Following the kernel formulation described above, we can now perform matrix factorization in the feature space by leveraging kernel functions. In particular, we can move the inner product from the input space X to the feature space $\mathbb{H}$, by reformulating Eq. (2) as follows:

$$\begin{aligned}
&\arg\min_{\mathbf{A},\mathbf{B}} \sum_{\mathbf{w}\in\mathcal{W}} \sum_{l=1}^{L} \sum_{\substack{|j|\leq\gamma \\ j\neq 0}} \sum_{u\in\mathcal{V}} \left([w_{l+j}=u] - \left\langle \Phi(\mathbf{A}_{(u,:)}), \Phi(\mathbf{B}_{(w_l,:)}) \right\rangle \right)^2 \\
&\qquad\qquad + \mathcal{R}(\mathbf{A}, \mathbf{B}) \\
&= \arg\min_{\mathbf{A},\mathbf{B}} \sum_{\mathbf{w}\in\mathcal{W}} \sum_{l=1}^{L} \sum_{\substack{|j|\leq\gamma \\ j\neq 0}} \sum_{u\in\mathcal{V}} \left([w_{l+j}=u] - \mathsf{K}\left(\mathbf{A}_{(u,:)}, \mathbf{B}_{(w_l,:)} \right) \right)^2 \\
&\qquad\qquad + \mathcal{R}(\mathbf{A}, \mathbf{B}). \quad (3)
\end{aligned}$$

In this way, we obtain a kernelized matrix factorization model for node embeddings based on random walks. For the numerical evaluation of our method, we use the following universal kernels [59], [61]:

$$\mathsf{K}_G(\mathbf{x}, \mathbf{y}) = \exp\left(\frac{-\|\mathbf{x} - \mathbf{y}\|^2}{\sigma^2} \right) \qquad \sigma \in \mathbb{R}$$

$$\mathsf{K}_S(\mathbf{x}, \mathbf{y}) = \frac{1}{\left(1 + \|\mathbf{x} - \mathbf{y}\|^2\right)^\sigma} \qquad \sigma \in \mathbb{R}_+,$$

where K_G and K_S correspond to the *Gaussian* and *Schoenberg* kernels respectively. We will refer to the proposed kernel-based node embeddings methodology as KernelNE (the

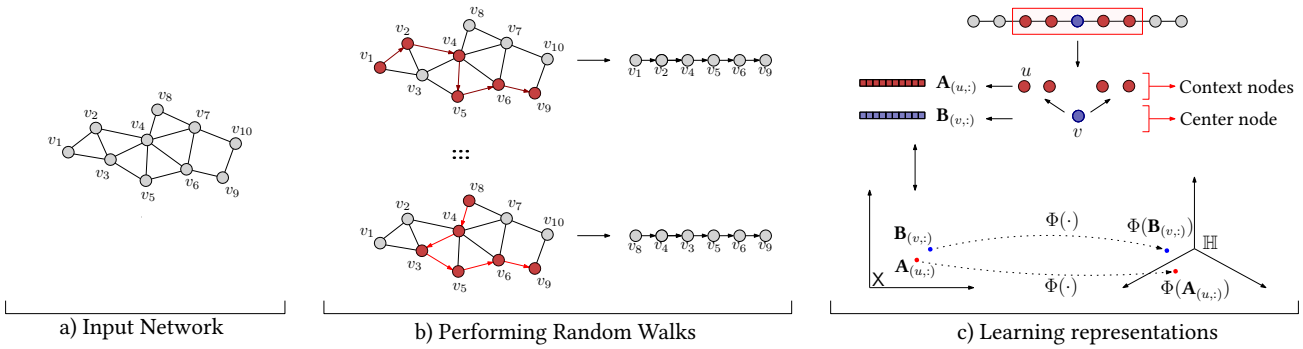

Fig. 1. Schematic representation of the KERNELNE model. Node sequences are firstly generated by following a random walk strategy. By using the co-occurrences of node pairs within a certain window size, node representations are learned by optimizing their maps in the feature space.

two different kernels will be denoted by GAUSS and SCH). A schematic representation of the basic components of the proposed model is given in Fig. 1.

4.1.1 Model Optimization

The estimation problem for both parameters $\mathbf{A}$ and $\mathbf{B}$ is, unfortunately, non-convex. Nevertheless, when we consider each parameter separately by fixing the other one, it turns into a convex problem. By taking advantage of this property, we employ *Stochastic Gradient Descent* (SGD) [62] in the optimization step of each embedding matrix. Note that, for each context node w_{l+j} in Eq. (3), we have to compute the gradient for each node $u \in \mathcal{V}$, which is computationally intractable. However, Eq. (3) can be divided into two parts with respect to the values of $[w_{l+j} = u] \in \{0, 1\}$, as follows:

$$
\begin{aligned}
&\sum_{u\in\mathcal{V}} \Big([w_{l+j}{=}u] - \mathsf{K}\Big(\mathbf{A}_{(u,:)}, \mathbf{B}_{(w_l,:)}\Big)\Big)^2 \\
&= \Big(1{-}\mathsf{K}\Big(\mathbf{A}_{(u^+,:)}, \mathbf{B}_{(w_l,:)}\Big)\Big)^2 + \sum_{u^-\in\mathcal{V}\setminus\{w_{l+j}\}} \Big(\mathsf{K}\Big(\mathbf{A}_{(u^-,:)}, \mathbf{B}_{(w_l,:)}\Big)\Big)^2 \\
&\approx \Big(1{-}\mathsf{K}\Big(\mathbf{A}_{(u^+,:)}, \mathbf{B}_{(w_l,:)}\Big)\Big)^2 + |\mathcal{V}\setminus\{w_{l+j}\}| \underset{u^-\sim}{\mathbb{E}} \Big[\mathsf{K}(\mathbf{A}_{(u^-,:)}, \mathbf{B}_{(w_l,:)})\Big]^2 \\
&= \underbrace{\Big(1 - \mathsf{K}\Big(\mathbf{A}_{(u^+,:)}, \mathbf{B}_{(w_l,:)}\Big)\Big)^2}_{\text{positive sample}} + \underbrace{k \underset{u^-\sim p^-}{\mathbb{E}} \Big[\mathsf{K}(\mathbf{A}_{(u^-,:)}, \mathbf{B}_{(w_l,:)})\Big]^2}_{\text{negative sample}},
\end{aligned}
$$

where $k := |\mathcal{V}| - 1$ and $u^+ := w_{l+j}$. To this end, we apply *negative sampling* [10] which is a variant of *noise-contrastive estimation* [63], proposed as an alternative to solve the computational problem of hierarchical softmax. For each context node $u^+ \in \mathcal{C}_{\mathbf{w}}(w_l)$, we sample k negative instances u^- from the noise distribution p^-. Then, we can rewrite the objective function Eq. (3) in the following way:

$$
\begin{aligned}
\mathcal{F}_S := \arg\min_{\mathbf{A},\mathbf{B}} \sum_{\boldsymbol{w}\in\mathcal{W}} \sum_{l=1}^{L} \sum_{\substack{|j|\le\gamma \\ j\neq 0}} \Bigg(\Big(1{-}\,\mathsf{K}(\mathbf{A}_{(w_{l+j},:)}, \mathbf{B}_{(w_l,:)})\Big)^2 \\
+ \sum_{\substack{r=1 \\ u_r^-\sim p^-}}^{k} \mathsf{K}\Big(\mathbf{A}_{(u_r^-,:)}, \mathbf{B}_{(w_l,:)}\Big)^2 \Bigg) + \mathcal{R}(\mathbf{A}, \mathbf{B}). \quad (4)
\end{aligned}
$$

Equation (4) corresponds to the objective function of the proposed KERNELNE model. In the following subsection, we will study how this model could be further extended to leverage multiple kernels.

4.2 Multiple Kernel Node Representation Learning

Selecting a proper kernel function $\mathsf{K}(\cdot,\cdot)$ and the corresponding parameters (e.g., the bandwidth of a Gaussian kernel) is a critical task during the learning phase. Nevertheless, choosing a single kernel function might impose potential bias, causing limitations on the performance of the model. Having the ability to properly utilize multiple kernels could increase the expressiveness of the model, capturing different notions of similarity among embeddings [20]. Besides, learning how to combine such kernels, might further improve the performance of the underlying model. In particular, given a set of base kernels $\{\mathsf{K}_i\}_{i=1}^K$, we aim to find an optimal way to combine them, as follows:

$$
\mathsf{K}^{\mathbf{c}}(\mathbf{x}, \mathbf{y}) = f_{\mathbf{c}}\Big(\{\mathsf{K}_i(\mathbf{x}, \mathbf{y})\}_{i=1}^K | \mathbf{c}\Big),
$$

where the combination function $f_{\mathbf{c}}$ is parameterized on $\mathbf{c} \in \mathbb{R}^K$ that indicates kernel weights. Due to the generality of the multiple kernel learning framework, $f_{\mathbf{c}}$ can be either a linear or nonlinear function.

In this paragraph, we examine how to further strengthen the proposed kernelized weighted matrix factorization, by *linearly* combining multiple kernels. Let $\mathsf{K}_1, \ldots, \mathsf{K}_K$ be a set of kernel functions satisfying the properties presented in the previous paragraph. Then, we can restate the objective function as follows:

$$
\begin{aligned}
\mathcal{F}_M := \arg\min_{\mathbf{A},\mathbf{B},\mathbf{c}} \sum_{\boldsymbol{w}\in\mathcal{W}} \sum_{l=1}^{L} \sum_{\substack{|j|\le\gamma \\ j\neq 0}} \Bigg(\Big(1{-}\sum_{i=1}^{K} c_i \mathsf{K}_i(\mathbf{A}_{(w_{l+j},:)}, \mathbf{B}_{(w_l,:)})\Big)^2 \\
+ \sum_{\substack{r=1 \\ u_r^-\sim p^-}}^{k} \Big(\sum_{i=1}^{K} c_i \mathsf{K}_i(\mathbf{A}_{(u_r^-,:)}, \mathbf{B}_{(w_l,:)})\Big)^2 \Bigg) \\
+ \frac{\lambda}{2}\Big(\|\mathbf{A}\|_F^2 + \|\mathbf{B}\|_F^2\Big) + \frac{\beta}{2}\|\mathbf{c}\|_2^2, \quad (5)
\end{aligned}
$$

where $\mathbf{c} := [c_1, \ldots, c_K]^\top \in \mathbb{R}^K$. Here, we introduce an additional parameter c_i representing the contribution of the corresponding kernel K_i. $\beta > 0$ is a trade-off parameter, and similarly, the coefficients $c_1, \ldots, c_K$ are optimized by fixing the remaining model parameters $\mathbf{A}$ and $\mathbf{B}$. Equation (5) corresponds to the objective function of the proposed

multiple kernel learning model MKERNELNE. Unlike the common usage of multiple kernels methods [20], here we do not constrain the coefficients to be non-negative. We interpret each entry of the target matrix as a linear combination of inner products of different kernels' feature maps. As discussed in the previous sections, our main intuition relies on obtaining more expressive embeddings by projecting the factorization step into a higher dimensional space.

Algorithm 1 summarizes the pseudocode of the proposed approach. For a given collection of random walks $\mathcal{W}$, we first determine the center-context node pairs (w_l, w_{l+j}) in the node sequences. Recall that, for each *center* node in a walk, its surrounding nodes within a certain distance γ, define its *context*. Furthermore, the corresponding embedding vectors $\mathbf{B}_{(w_l,:)}$ of center node w_l and $\mathbf{A}_{(w_{l+j},:)}$ of context w_{l+j} are updated by following the rules which we describe in detail below. Note that, we obtain two different representations, $\mathbf{A}_{(v,:)}$ and $\mathbf{B}_{(v,:)}$, for each node $v \in \mathcal{V}$ since the node can have either a center or context role in the walks.

The gradients in Alg. 2 are given below. For notation simplicity, we denote each $\mathsf{K}_i(\mathbf{A}_{(u,:)}, \mathbf{B}_{(v,:)})$ by $\mathsf{K}_i(u, v)$.

$$\nabla_{\mathbf{A}_{(x,:)}} \mathcal{F}_M := [x=u]\Big(-2\sum_{i=1}^{K} c_i \nabla_{\mathbf{A}_{(x,:)}} \mathsf{K}_i(x,v)\Big)\Big(1-\sum_{j=1}^{K} c_i \mathsf{K}_i(x,v)\Big) + 2\sum_{\substack{r=1 \\ u_r^- \sim p^-}}^{k} \Big[x=u_r^-\Big]\Big(\sum_{i=1}^{K} c_i \nabla_{\mathbf{A}_{(u_r^-,:)}} \mathsf{K}_i(u_r^-, v)\mathsf{K}_i(u_r^-, v)\Big) + \lambda \mathbf{A}_{(x,:)}$$

$$\nabla_{\mathbf{B}_{(v,:)}} \mathcal{F}_M := -2\Big(\sum_{i=1}^{K} c_i \nabla_{\mathbf{B}_{(v,:)}} \mathsf{K}_i(u,v)\Big)\Big(1-\sum_{i=1}^{K} c_i \mathsf{K}_i(u,v)\Big) + 2\sum_{\substack{r=1 \\ u_r^- \sim p^-}}^{k} \Big(\sum_{i=1}^{K} c_i \nabla_{\mathbf{B}_{(v,:)}} \mathsf{K}_i(u_r^-, v)\mathsf{K}_i(u_r^-, v)\Big) + \lambda \mathbf{B}_{(v,:)}$$

$$\nabla_{c_t} \mathcal{F}_M = -2\Big(1-\sum_{i=1}^{K} c_i \mathsf{K}_i(u,v)\Big)\mathsf{K}_t(u,v) + 2\sum_{\substack{r=1 \\ u_r^- \sim p^-}}^{k} \Big(\sum_{i=1}^{K} c_i \mathsf{K}_i(u_r^-, v)\Big)\mathsf{K}_t(u_r^-, v) + \beta c_t$$

4.3 Complexity Analysis

For the generation of walks, the biased random walk strategy proposed in NODE2VEC is used, which can be performed in $\mathcal{O}(|\mathcal{W}| \cdot L)$ steps [4] for the pre-computed transition probabilities, where L indicates the walk length and $\mathcal{W}$ denotes the set of walks. For the algorithm's learning procedure, we can carry out Line 5 of Algorithm 1 at most $2\gamma \cdot |\mathcal{W}| \cdot L$ times for each center-context pair, where γ represents the window size. The dominant operation in Algorithm 2 is the multiplication operation of update rules in Lines 6, 7, and 9; the running time can be bounded by $\mathcal{O}(k \cdot K \cdot d)$, where k is the number of negative samples generated per center-context pair, K is the number of kernels, and d is the representation size. To sum up, the overall running time of the proposed approach can be bounded by $\mathcal{O}(\gamma \cdot |\mathcal{W}| \cdot L \cdot K \cdot d \cdot k)$ steps.

Algorithm 1 MKERNELNE

Input: Graph $G = (\mathcal{V}, \mathcal{E})$
Representation size d
Set of walks $\mathcal{W}$
Window size γ
Kernel function K
Kernel parameter(s) σ
Output: Embedding matrix $\mathbf{A}$

1: Initialize matrices $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{n \times d}$
`/* Extract center-context node pairs */`
2: **for each** $\mathbf{w} = (w_1, \ldots, w_L) \in \mathcal{W}$ **do**
3: **for** $l \leftarrow 1$ **to** L **do**
4: **for** $j \neq 0 \leftarrow -\gamma$ **to** γ **do**
5: $\mathbf{A}, \mathbf{B}, \mathbf{c} \leftarrow$ UPDATEEMB$(\mathbf{A}, \mathbf{B}, \mathbf{c}, w_l, w_{l+j}, \mathsf{K}, \sigma)$
6: **end for**
7: **end for**
8: **end for**

Algorithm 2 UPDATEEMB

Input: Graph $G = (\mathcal{V}, \mathcal{E})$
Embedding matrices $\mathbf{A}$ and $\mathbf{B}$
Kernel coefficients $\mathbf{c} = (c_1, \ldots, c_K)$
Kernel function(s) K
Center and context nodes v and u
Learning rate η
Distribution for generating negative samples p^-
Output: Embedding matrix $\mathbf{A}$

1: node_list $\leftarrow [u]$
`/* Extract negative samples */`
2: **for** $s \leftarrow 1$ **to** k **do**
3: node_list $\leftarrow$ SAMPLENODE(p^-)
4: **end for**
`/* Update embedding vectors */`
5: **for each** x in node_list **do**
6: $\mathbf{A}_{(x,:)} \leftarrow \mathbf{A}_{(x,:)} - \eta \nabla_{\mathbf{A}_{(x,:)}} \mathcal{F}_M$
7: $\mathbf{B}_{(v,:)} \leftarrow \mathbf{B}_{(v,:)} - \eta \nabla_{\mathbf{B}_{(v,:)}} \mathcal{F}_M$
`/* Update individual kernel weights */`
8: **if** number of kernels > 1 **then**
9: $\mathbf{c} \leftarrow \mathbf{c} - \eta \nabla_c \mathcal{F}_M$
10: **end if**
11: **end for**

5 EXPERIMENTS

This section presents the experimental set-up details, the datasets, and the baseline methods used in the evaluation. The performance of the proposed single and multiple kernel models is examined for node classification and link prediction tasks on various real-world datasets. The experiments have been performed on a computer with 16Gb RAM.

5.1 Baseline Methods

We consider nine baseline models to compare the performance of our approach. (i) DEEPWALK [3] performs uniform random walks to generate the contexts of nodes; then, the SKIP-GRAM model [10] is used to learn node embeddings. (ii) NODE2VEC [4] combines SKIP-GRAM with biased random walks, using two extra parameters that control the walk in order to simulate a *BFS* or *DFS* exploration. In the experiments, we set those parameters to

TABLE 1
Statistics of networks $|\mathcal{V}|$: number of nodes, $|\mathcal{E}|$: number of edges, $|\mathcal{K}|$: number of labels and $|\mathcal{C}c|$: number of connected components.

	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{E}\rvert$	$\lvert\mathcal{K}\rvert$	$\lvert\mathcal{C}c\rvert$	**Avg. Degree**
CiteSeer	3,312	4,660	6	438	2.81
Cora	2,708	5,278	7	78	3.90
DBLP	27,199	66,832	4	2,115	4.91
PPI	3,890	38,739	50	35	19.92
AstroPh	17,903	19,7031	-	1	22.01
HepTh	8,638	24,827	-	1	5.74
Facebook	4,039	88,234	-	1	43.69
Gnutella	8,104	26,008	-	1	6.42

1.0. In our approach we sample context nodes using this biased random walk strategy. (iii) LINE [64] learns nodes embeddings relying on first- and second-order proximity information of nodes. (iv) HOPE [7] is a matrix factorization approach aiming at capturing similarity patterns based on a higher-order node similarity measure. In the experiments, we consider the *Katz* index, since it demonstrates the best performance among other proximity indices. (v) NETMF [26] targets to factorize the matrix approximated by pointwise mutual information of center and context pairs. The experiments have been conducted for large window sizes ($\gamma = 10$) due to its good performance. (vi) VERSE [33] learns the embedding vectors by optimizing similarities among nodes. As suggested by the authors, we set $\alpha = 0.85$ for the value of the hyper-parameter in the experiments. (vii) PRONE [34] learns the representations by relying on an efficient sparse matrix factorization and the extracted embeddings are improved with spectral propagation operations. (viii) GEMSEC [65] leverages the community structure of real-world graphs, learning node embeddings and the cluster assignments simultaneously. We have used the best performing number of cluster value from the set $\{5, 10, 15, 25, 50, 75, 100\}$. (ix) Lastly, M-NMF [66] extracts node embeddings under a modularity-based community detection framework based on non-negative matrix factorization. We have observed that the algorithm poses good performance by setting its parameters $\alpha = 0.1$ and $\beta = 5$. We performed parameter tuning for the number of communities using values from the following set: $\{5, 15, 20, 25, 50, 75, 100\}$.

Those baseline methods are compared against instances of KERNELNE and MKERNELNE using different kernel functions (GAUSS and SCH).

5.2 Datasets

In our experiments, we use eight networks of different types. To be consistent, we consider all network as undirected in all experiments, and the detailed statistics of the datasets are provided in Table 1. (i) *CiteSeer* [67] is a citation network obtained from the *CiteSeer* library. Each node of the graph corresponds to a paper, while the edges indicate reference relationships among papers. The node labels represent the subjects of the paper. (ii) *Cora* [68] is another citation network constructed from the publications in the machine learning area; the documents are classified into seven categories. (iii) *DBLP* [69] is a co-authorship graph, where an edge exists between nodes if two authors have co-authored at least one paper. The labels represent the research areas. (iv) *PPI* (*Homo Sapiens*) [4] is a protein-protein interaction network for Homo Sapiens, in which biological states are used as node labels. (v) *AstroPh* [70] is a collaboration network constructed from papers submitted to the *ArXiv* repository for the Astro Physics subject area, from January 1993 to April 2003. (vi) *HepTh* [70] network is constructed in a similar way from the papers submitted to *ArXiv* for the *High Energy Physics - Theory* category. (vii) *Facebook* [71] is a social network extracted from a survey conducted via a *Facebook* application. (viii) *Gnutella* [72] is the peer-to-peer file-sharing network constructed from the snapshot collected in August 2002 in which nodes and edges correspond to hosts and connections among them.

5.3 Parameter Settings

For random walk-based approaches, we set the window size (γ) to 10, the number of walks (N) to 80, and the walk length (L) to 10. We use the embedding size of $d = 128$ for each method. The instances of KERNELNE and MKERNELNE are fed with random walks generated by NODE2VEC. For the training process, we adopt the negative sampling strategy [10] as described in Subsection 4.1. The negative samples are generated proportionally to its frequency raised to the power of 0.75. For our methods and the baseline models needing to generate negative node instances, we consistently sample 5 negative nodes for a fair comparison. In our experiments, the initial learning rate of stochastic gradient descent is set to 0.025; then it is decreased linearly according to the number of processed nodes until the minimum value, 10^{-4}. For the kernel parameters, the value of σ has been chosen as 2.0 for the single kernel version of the model (KERNELNE). For MKERNELNE, we considered three kernels and their parameters are set to $1.0, 2.0, 3.0$ and $1.0, 1.5, 2.0$ for MKERNELNE-GAUSS and MKERNELNE-SCH, respectively. As an exception for the *PPI* network, we employed $0.5, 1.0, 1.5$ for MKERNELNE-GAUSS since it shows better performance due to the network's structure. The regularization parameters are always set to $\lambda = 10^{-2}$ and $\beta = 0.1$.

5.4 Node Classification

Experimental setup. In the node classification task, we have access to the labels of a certain fraction of nodes in the network (training set), and our goal is to predict the labels of the remaining nodes (test set). After learning the representation vectors for each node, we split them into varying sizes of training and test sets, ranging from 1% up to 90%. The experiments are carried out by applying an one-vs-rest logistic regression classifier with L_2 regularization [73]. We report the average performance of 50 runs for each representation learning method.

Experimental results. Tables 2 to 5 report the Micro-F_1 and Macro-F_1 scores of the classification task. With boldface and underline we indicate the best and second-best performing model, respectively. As can be observed the single and multiple kernel versions of the proposed methodology outperform the baseline models, showing different characteristics depending on the graph dataset. While the *Gaussian*

TABLE 2
Node classification task for varying training sizes on *Citeseer*. For each method, the rows show the Micro-F_1 and Macro-F_1 scores, respectively.

		1%	2%	3%	4%	5%	6%	7%	8%	9%	10%	20%	30%	40%	50%	60%	70%	80%	90%
Baselines	DEEPWALK	0.367	0.421	0.445	0.462	0.475	0.488	0.496	0.502	0.508	0.517	0.550	0.569	0.580	0.587	0.593	0.596	0.597	0.597
		0.313	0.374	0.402	0.419	0.434	0.446	0.455	0.461	0.467	0.476	0.506	0.524	0.534	0.540	0.544	0.547	0.547	0.546
	NODE2VEC	0.404	0.451	0.475	0.491	0.504	0.514	0.523	0.529	0.537	0.543	0.571	0.583	0.591	0.595	0.600	0.600	0.601	0.603
		0.342	0.397	0.424	0.443	0.456	0.466	0.476	0.483	0.489	0.497	0.525	0.535	0.542	0.546	0.549	0.549	0.549	0.550
	LINE	0.253	0.300	0.332	0.353	0.369	0.384	0.395	0.407	0.412	0.418	0.459	0.476	0.487	0.494	0.500	0.505	0.507	0.512
		0.183	0.243	0.280	0.303	0.321	0.334	0.346	0.357	0.363	0.367	0.407	0.423	0.434	0.440	0.445	0.448	0.450	0.454
	HOPE	0.198	0.197	0.201	0.204	0.205	0.207	0.211	0.208	0.212	0.216	0.235	0.253	0.265	0.276	0.288	0.299	0.304	0.316
		0.064	0.060	0.061	0.065	0.065	0.066	0.070	0.068	0.072	0.075	0.099	0.121	0.136	0.150	0.164	0.178	0.186	0.202
	VERSE	0.434	0.476	0.489	0.512	0.517	0.532	0.538	0.536	0.551	0.554	0.576	0.590	0.594	0.607	0.614	0.615	0.616	**0.620**
		0.377	0.423	0.446	0.462	0.466	0.484	0.490	0.492	0.505	0.507	0.526	0.545	0.545	0.558	0.563	0.564	0.565	**0.567**
	PRONE	0.244	0.287	0.344	0.351	0.376	0.394	0.417	0.434	0.444	0.452	0.514	0.530	0.547	0.554	0.563	0.562	0.554	0.558
		0.172	0.225	0.275	0.294	0.313	0.336	0.358	0.381	0.383	0.395	0.454	0.470	0.488	0.498	0.502	0.502	0.497	0.502
	NETMF	0.328	0.401	0.445	0.473	0.492	0.507	0.517	0.525	0.533	0.538	0.567	0.579	0.586	0.590	0.592	0.594	0.599	0.601
		0.264	0.346	0.392	0.421	0.440	0.454	0.466	0.474	0.481	0.487	0.516	0.528	0.535	0.538	0.540	0.542	0.547	0.548
	GEMSEC	0.337	0.384	0.415	0.439	0.449	0.459	0.475	0.479	0.484	0.491	0.517	0.532	0.538	0.545	0.551	0.552	0.556	0.558
		0.288	0.339	0.376	0.400	0.412	0.422	0.435	0.439	0.445	0.451	0.476	0.488	0.494	0.497	0.501	0.499	0.502	0.501
	M-NMF	0.222	0.264	0.295	0.317	0.338	0.340	0.364	0.370	0.376	0.382	0.423	0.439	0.448	0.449	0.455	0.456	0.461	0.457
		0.109	0.161	0.205	0.229	0.255	0.261	0.285	0.293	0.301	0.306	0.353	0.370	0.380	0.381	0.389	0.390	0.394	0.390
KERNELNE	GAUSS	0.422	0.465	0.490	0.504	0.518	0.528	0.535	0.543	0.551	0.555	0.588	**0.600**	0.607	0.611	**0.616**	0.616	0.619	**0.620**
		0.367	0.415	0.440	0.456	0.471	0.479	0.488	0.496	0.504	0.508	**0.540**	**0.551**	**0.558**	**0.562**	**0.564**	**0.566**	**0.567**	0.566
	SCH	0.441	0.497	0.517	0.531	0.539	0.544	0.551	0.555	0.558	0.561	0.580	0.591	0.597	0.602	0.608	0.610	0.614	0.609
		0.365	0.428	0.451	0.465	0.476	0.480	0.488	0.492	0.496	0.498	0.518	0.531	0.537	0.542	0.547	0.549	0.551	0.546
MKERNELNE	GAUSS	0.431	0.493	0.514	0.530	0.539	0.547	0.552	0.559	0.563	0.566	**0.590**	**0.600**	**0.609**	**0.613**	0.615	**0.619**	**0.620**	**0.620**
		0.362	0.434	0.455	0.473	0.482	**0.491**	**0.497**	**0.504**	**0.508**	**0.511**	0.537	0.546	0.555	0.559	0.562	**0.566**	**0.567**	0.565
	SCH	**0.443**	**0.500**	**0.525**	**0.538**	**0.547**	**0.553**	**0.558**	**0.563**	**0.567**	**0.570**	0.588	0.597	0.603	0.609	0.612	0.614	0.616	0.615
		0.368	**0.433**	**0.458**	**0.475**	**0.483**	0.490	0.495	0.500	0.505	0.508	0.527	0.537	0.544	0.550	0.553	0.557	0.557	0.556

TABLE 3
Node classification task for varying training sizes on *Cora*. For each method, the rows show the Micro-F_1 and Macro-F_1 scores, respectively.

		1%	2%	3%	4%	5%	6%	7%	8%	9%	10%	20%	30%	40%	50%	60%	70%	80%	90%
Baselines	DEEPWALK	0.522	0.617	0.660	0.688	0.703	0.715	0.724	0.732	0.742	0.747	0.782	0.799	0.808	0.815	0.821	0.825	0.827	0.832
		0.442	0.569	0.628	0.664	0.682	0.698	0.709	0.717	0.727	0.735	0.771	0.788	0.798	0.806	0.811	0.815	0.816	0.821
	NODE2VEC	0.570	0.659	0.695	0.720	0.734	0.743	0.752	0.759	0.765	0.770	0.800	0.816	0.824	0.831	0.835	0.839	0.842	0.845
		0.489	0.612	0.662	0.694	0.714	0.724	0.735	0.743	0.751	0.755	0.788	0.804	0.813	0.820	0.824	0.828	0.830	0.832
	LINE	0.351	0.416	0.463	0.498	0.521	0.546	0.566	0.581	0.598	0.609	0.673	0.701	0.719	0.728	0.735	0.741	0.743	0.747
		0.223	0.306	0.373	0.425	0.457	0.492	0.519	0.543	0.565	0.578	0.659	0.691	0.710	0.720	0.727	0.733	0.736	0.738
	HOPE	0.278	0.284	0.297	0.301	0.302	0.301	0.302	0.302	0.302	0.302	0.303	0.302	0.302	0.302	0.303	0.304	0.303	0.306
		0.070	0.067	0.067	0.066	0.067	0.067	0.066	0.066	0.066	0.066	0.067	0.067	0.067	0.067	0.067	0.068	0.070	0.074
	VERSE	0.605	0.680	0.708	0.727	0.737	0.753	0.761	0.763	0.766	0.774	0.798	0.813	0.818	0.828	0.829	0.827	0.837	0.829
		0.527	0.628	0.675	0.704	0.717	0.737	0.746	0.752	0.754	0.764	0.788	0.804	0.809	0.822	0.821	0.818	0.833	0.819
	PRONE	0.337	0.380	0.440	0.475	0.513	0.529	0.574	0.604	0.624	0.628	0.729	0.754	0.785	0.794	0.804	0.803	0.810	0.816
		0.202	0.223	0.319	0.363	0.434	0.446	0.520	0.559	0.579	0.589	0.712	0.743	0.776	0.784	0.793	0.793	0.801	0.809
	NETMF	0.534	0.636	0.693	0.716	0.735	0.748	0.757	0.767	0.770	0.773	**0.807**	**0.821**	0.828	0.834	0.839	0.841	0.839	0.844
		0.461	0.591	0.667	0.694	0.717	0.731	0.741	0.751	0.757	0.760	**0.797**	**0.811**	**0.819**	0.824	0.830	0.832	0.831	0.835
	GEMSEC	0.397	0.470	0.497	0.530	0.551	0.568	0.578	0.587	0.596	0.601	0.643	0.674	0.698	0.714	0.728	0.735	0.741	0.744
		0.317	0.406	0.439	0.477	0.504	0.527	0.535	0.546	0.556	0.562	0.611	0.646	0.672	0.689	0.704	0.713	0.719	0.722
	M-NMF	0.419	0.507	0.549	0.580	0.604	0.622	0.633	0.642	0.652	0.656	0.700	0.717	0.725	0.732	0.736	0.736	0.744	0.742
		0.354	0.459	0.507	0.550	0.575	0.598	0.609	0.622	0.632	0.638	0.687	0.706	0.716	0.722	0.728	0.728	0.735	0.734
KERNELNE	GAUSS	0.628	0.696	0.721	0.739	0.748	0.759	0.766	0.772	0.775	0.780	0.806	0.820	**0.829**	**0.837**	**0.843**	**0.846**	**0.849**	**0.851**
		0.566	**0.664**	**0.696**	0.721	0.730	0.743	0.752	0.758	0.762	0.767	0.794	0.809	0.818	**0.826**	**0.832**	**0.836**	**0.838**	**0.840**
	SCH	0.619	0.695	0.722	0.736	0.745	0.750	0.755	0.759	0.763	0.765	0.783	0.790	0.796	0.800	0.803	0.806	0.807	0.812
		0.521	0.631	0.671	0.697	0.712	0.721	0.728	0.734	0.743	0.745	0.771	0.780	0.786	0.790	0.792	0.795	0.795	0.799
MKERNELNE	GAUSS	**0.631**	**0.701**	**0.731**	**0.748**	**0.757**	**0.764**	**0.770**	**0.775**	**0.778**	**0.781**	0.801	0.812	0.819	0.823	0.827	0.828	0.833	0.833
		0.562	0.656	**0.696**	**0.723**	**0.736**	**0.746**	**0.755**	**0.761**	**0.765**	**0.769**	0.791	0.801	0.808	0.813	0.817	0.818	0.822	0.820
	SCH	0.623	0.699	0.728	0.742	0.751	0.757	0.762	0.767	0.770	0.772	0.788	0.797	0.803	0.806	0.812	0.812	0.817	0.818
		0.527	0.637	0.686	0.708	0.723	0.733	0.741	0.749	0.753	0.756	0.777	0.787	0.793	0.796	0.801	0.802	0.805	0.804

kernel comes into prominence on the *Citeseer* and *Dblp* networks, the *Schoenberg* kernel shows good performance for the *PPI* network. We further observe that leveraging multiple kernels with MKERNELNE often has superior performance compared to the single kernel, especially for smaller training ratios, which corroborates the effectiveness of the data-driven multiple kernel approach. In only a few cases, NODE2VEC shows comparable performances. The reason stems from the structure of the network and the used random walk sequences. Since the distribution of the node labels does not always follow the network structure, we observe such occasional cases due to the complexity of the datasets concerning the node classification task.

5.5 Link Prediction

For the link prediction task, we remove half of the network's edges while retaining its connectivity, and we learn node embedding on the residual network. For networks consisting of disconnected components, we consider the giant component among them as our initial graph. The removed edges constitute the positive samples for the testing set; the same number of node pairs that do not exist in the original graph is sampled at random to form the negative samples. Then, the entries of the feature vector corresponding to each node

TABLE 4
Node classification task for varying training sizes on *Dblp*. For each method, the rows show the Micro-F_1 and Macro-F_1 scores, respectively.

		1%	2%	3%	4%	5%	6%	7%	8%	9%	10%	20%	30%	40%	50%	60%	70%	80%	90%
Baselines	DeepWalk	0.518	0.550	0.573	0.588	0.597	0.603	0.607	0.611	0.614	0.616	0.627	0.630	0.632	0.633	0.634	0.635	0.635	0.636
		0.464	0.496	0.515	0.527	0.535	0.540	0.543	0.547	0.550	0.551	0.560	0.563	0.564	0.565	0.566	0.566	0.567	0.567
	node2vec	0.557	0.575	0.590	0.599	0.606	0.612	0.615	0.618	0.621	0.623	0.633	0.636	0.638	0.639	0.640	0.641	0.641	0.641
		0.497	0.517	0.531	0.541	0.546	0.551	0.554	0.557	0.559	0.561	0.568	0.571	0.573	0.574	0.574	0.575	0.574	0.574
	LINE	0.525	0.553	0.567	0.576	0.581	0.585	0.588	0.590	0.593	0.594	0.603	0.606	0.608	0.610	0.610	0.611	0.611	0.611
		0.430	0.469	0.488	0.498	0.505	0.510	0.514	0.517	0.520	0.522	0.532	0.536	0.538	0.540	0.540	0.541	0.541	0.541
	HOPE	0.377	0.379	0.379	0.379	0.379	0.379	0.379	0.379	0.379	0.379	0.379	0.380	0.381	0.383	0.385	0.387	0.388	0.391
		0.137	0.137	0.137	0.137	0.137	0.137	0.137	0.137	0.137	0.138	0.139	0.140	0.143	0.146	0.149	0.152	0.156	0.160
	VERSE	0.550	0.566	0.585	0.593	0.599	0.605	0.610	0.613	0.615	0.617	0.629	0.629	0.632	0.635	0.633	0.632	0.634	0.636
		0.492	0.508	0.528	0.534	0.539	0.545	0.552	0.551	0.554	0.555	0.565	0.566	0.568	0.571	0.567	0.566	0.570	0.571
	ProNE	0.475	0.525	0.550	0.564	0.574	0.581	0.589	0.593	0.596	0.600	0.615	0.624	0.626	0.628	0.629	0.632	0.632	0.634
		0.363	0.436	0.457	0.478	0.493	0.498	0.510	0.517	0.520	0.520	0.543	0.553	0.554	0.557	0.559	0.562	0.563	0.564
	NetMF	0.564	0.577	0.586	0.589	0.593	0.596	0.599	0.601	0.604	0.605	0.613	0.617	0.619	0.620	0.620	0.623	0.623	0.623
		0.463	0.490	0.503	0.506	0.510	0.513	0.517	0.518	0.521	0.522	0.528	0.530	0.532	0.531	0.531	0.533	0.533	0.533
	GEMSEC	0.511	0.538	0.555	0.566	0.575	0.583	0.588	0.591	0.593	0.597	0.607	0.611	0.613	0.614	0.615	0.615	0.616	0.615
		0.453	0.477	0.492	0.501	0.508	0.513	0.518	0.519	0.521	0.523	0.531	0.534	0.534	0.535	0.535	0.537	0.537	0.536
	M-NMF	0.473	0.501	0.518	0.527	0.533	0.540	0.542	0.545	0.548	0.551	0.563	0.568	0.571	0.574	0.574	0.577	0.577	0.579
		0.302	0.345	0.369	0.383	0.392	0.400	0.406	0.410	0.414	0.418	0.436	0.443	0.447	0.450	0.451	0.454	0.455	0.455
KernelNE	Gauss	0.558	0.575	0.587	0.595	0.601	0.606	0.610	0.613	0.615	0.617	0.627	0.630	0.632	0.633	0.634	0.635	0.636	0.636
		0.498	0.517	0.528	0.536	0.542	0.545	0.548	0.551	0.552	0.554	0.562	0.564	0.566	0.566	0.567	0.567	0.568	0.569
	Sch	0.599	0.609	0.613	0.617	0.620	0.621	0.623	0.624	0.625	0.627	0.632	0.635	0.636	0.637	0.638	0.638	0.638	0.640
		0.512	0.531	0.539	0.546	0.549	0.552	0.554	0.556	0.557	0.559	0.565	0.568	0.569	0.570	0.571	0.571	0.571	0.572
MKernelNE	Gauss	0.603	0.614	**0.620**	**0.624**	**0.627**	**0.630**	**0.631**	**0.633**	**0.635**	**0.636**	**0.643**	**0.646**	**0.647**	**0.648**	**0.649**	**0.650**	**0.649**	**0.650**
		0.528	**0.547**	**0.554**	**0.560**	**0.564**	**0.566**	**0.568**	**0.570**	**0.572**	**0.573**	**0.579**	**0.582**	**0.582**	**0.583**	**0.584**	**0.584**	**0.584**	**0.584**
	Sch	**0.607**	**0.615**	0.619	0.621	0.624	0.625	0.627	0.628	0.630	0.631	0.637	0.639	0.641	0.641	0.642	0.643	0.643	0.643
		0.517	0.538	0.546	0.551	0.555	0.557	0.560	0.561	0.563	0.564	0.570	0.572	0.574	0.574	0.575	0.576	0.576	0.576

TABLE 5
Node classification task for varying training sizes on *PPI*. For each method, the rows show the Micro-F_1 and Macro-F_1 scores, respectively.

		1%	2%	3%	4%	5%	6%	7%	8%	9%	10%	20%	30%	40%	50%	60%	70%	80%	90%
Baselines	DeepWalk	0.093	0.110	0.122	0.131	0.137	0.143	0.147	0.151	0.155	0.156	0.176	0.188	0.197	0.206	0.212	0.218	0.223	0.226
		0.056	0.076	0.090	0.099	0.107	0.113	0.118	0.123	0.126	0.129	0.151	0.164	0.173	0.181	0.186	0.190	0.193	0.192
	node2vec	0.097	0.115	0.128	0.136	0.143	0.148	0.154	0.157	0.161	0.164	0.185	0.196	0.205	0.211	0.216	0.221	0.224	0.226
		0.058	**0.078**	**0.092**	**0.101**	**0.108**	**0.116**	**0.122**	**0.125**	0.129	**0.132**	0.155	0.167	0.175	0.180	0.185	0.188	0.189	0.190
	LINE	0.091	0.109	0.123	0.133	0.142	0.149	0.156	0.162	0.166	0.170	0.197	0.210	0.219	0.226	0.231	0.235	0.238	0.242
		0.046	0.063	0.075	0.084	0.093	0.099	0.106	0.111	0.116	0.120	0.148	0.164	0.174	0.181	0.187	0.191	0.193	0.192
	HOPE	0.067	0.068	0.069	0.069	0.069	0.069	0.069	0.070	0.069	0.069	0.069	0.071	0.073	0.077	0.081	0.086	0.089	0.093
		0.019	0.019	0.019	0.019	0.019	0.019	0.019	0.019	0.019	0.018	0.019	0.020	0.022	0.025	0.027	0.030	0.032	0.033
	VERSE	0.089	0.110	0.126	0.132	0.139	0.150	0.153	0.156	0.162	0.158	0.185	0.195	0.212	0.216	0.224	0.228	0.234	0.246
		0.052	0.072	0.090	0.099	0.107	**0.116**	0.120	0.124	**0.131**	0.129	0.157	0.165	0.183	0.185	0.193	0.191	0.200	0.200
	ProNE	0.085	0.096	0.116	0.129	0.145	0.146	0.155	0.163	0.170	0.174	0.207	0.226	0.237	0.242	**0.248**	**0.256**	**0.258**	**0.257**
		0.039	0.052	0.065	0.079	0.091	0.096	0.102	0.111	0.117	0.123	0.155	**0.180**	**0.190**	**0.196**	**0.205**	**0.216**	**0.214**	**0.206**
	NetMF	0.073	0.084	0.091	0.097	0.104	0.112	0.116	0.120	0.124	0.129	0.155	0.170	0.180	0.187	0.192	0.195	0.198	0.199
		0.031	0.043	0.052	0.059	0.066	0.073	0.078	0.083	0.086	0.091	0.116	0.129	0.137	0.143	0.147	0.150	0.152	0.150
	GEMSEC	0.074	0.078	0.081	0.082	0.082	0.085	0.087	0.087	0.087	0.089	0.101	0.112	0.122	0.131	0.138	0.143	0.150	0.151
		0.050	0.059	0.062	0.063	0.064	0.067	0.069	0.070	0.071	0.073	0.087	0.098	0.106	0.113	0.120	0.122	0.127	0.125
	M-NMF	0.085	0.097	0.103	0.112	0.115	0.119	0.122	0.123	0.126	0.128	0.137	0.143	0.148	0.153	0.155	0.162	0.165	0.168
		0.057	0.071	0.080	0.089	0.093	0.097	0.102	0.103	0.104	0.108	0.119	0.125	0.131	0.135	0.136	0.141	0.142	0.141
KernelNE	Gauss	0.087	0.102	0.112	0.121	0.128	0.134	0.138	0.142	0.145	0.148	0.167	0.180	0.186	0.192	0.196	0.200	0.202	0.203
		0.038	0.051	0.060	0.067	0.073	0.078	0.083	0.086	0.089	0.092	0.112	0.125	0.132	0.138	0.142	0.146	0.148	0.147
	Sch	0.103	0.126	0.142	0.154	0.164	0.172	0.179	0.185	0.190	**0.195**	**0.220**	**0.232**	**0.239**	**0.244**	**0.248**	0.250	0.254	0.256
		0.050	0.069	0.081	0.091	0.100	0.107	0.114	0.119	0.124	0.128	0.156	0.171	**0.181**	0.187	0.193	0.196	0.200	0.198
MKernelNE	Gauss	0.104	**0.128**	0.144	**0.156**	0.165	**0.174**	**0.181**	**0.186**	**0.191**	**0.195**	**0.220**	0.231	0.238	0.242	0.246	0.249	0.251	0.254
		0.053	0.071	0.083	0.094	0.104	0.109	0.117	0.122	0.126	0.131	**0.158**	0.172	0.181	0.187	0.192	0.195	0.198	0.197
	Sch	**0.105**	**0.128**	**0.145**	**0.156**	**0.167**	0.173	0.180	**0.186**	**0.191**	0.194	0.219	0.230	0.236	0.241	0.245	0.247	0.249	0.251
		0.053	0.071	0.085	0.094	0.104	0.110	0.115	0.122	0.126	0.130	0.155	0.169	0.177	0.184	0.188	0.192	0.193	0.193

pair (u, v) in the test set are computed based on the element-wise operation $|\mathbf{A}_{(v,i)} - \mathbf{A}_{(u,i)}|^2$, for each coordinate axis i of the embedding vectors $\mathbf{A}_{(u,:)}$ and $\mathbf{A}_{(v,:)}$. In the experiments, we use logistic regression with L_2 regularization.

Experimental results. Table 6 shows the *Area Under Curve* (AUC) scores for the link prediction task. As we can observe, both the KernelNE and MKernelNE models perform well across different datasets compared to the other baseline models. Comparing now at the AUC score of KernelNE and MKernelNE, we observe that both models show quite similar behavior. In the case of the single kernel model KernelNE, the Schoenberg kernel (Sch) performs significantly better than the Gaussian one. On the contrary, in the MKernelNE model both kernels achieve similar performance. These results are consistent to the behavior observed in the node classification task.

5.6 Parameter Sensitivity

In this subsection, we examine how the performance of the proposed models is affected by the choice of parameters.

The effect of dimension size. The dimension size d is a critical parameter for node representation learning approaches

TABLE 6
Area Under Curve (AUC) scores for the link prediction task.

	Baselines									KERNELNE		MKERNELNE	
	DEEPWALK	NODE2VEC	LINE	HOPE	PRONE	VERSE	NETMF	GEMSEC	M-NMF	GAUSS	SCH	GAUSS	SCH
Citeseer	0.828	0.827	0.725	0.517	0.719	0.754	0.818	0.713	0.634	0.807	**0.882**	0.850	0.863
Cora	0.779	0.781	0.693	0.548	0.667	0.737	0.767	0.716	0.616	0.765	**0.818**	0.792	0.807
DBLP	0.944	0.944	0.930	0.591	0.931	0.917	0.889	0.843	0.587	0.949	0.957	**0.958**	0.957
PPI	0.860	**0.861**	0.731	0.827	0.784	0.570	0.748	0.771	0.806	0.749	0.796	0.803	0.784
AstroPh	0.961	0.961	0.961	0.703	0.965	0.930	0.825	0.697	0.676	0.915	0.970	**0.978**	0.969
HepTh	0.896	0.896	0.827	0.623	0.862	0.840	0.844	0.708	0.633	0.897	**0.915**	0.914	0.914
Facebook	0.984	0.983	0.954	0.836	0.982	0.981	0.975	0.696	0.690	0.984	0.988	**0.989**	**0.989**
Gnutella	0.679	0.694	0.623	**0.723**	0.592	0.415	0.646	0.501	0.709	0.594	0.667	0.647	0.663

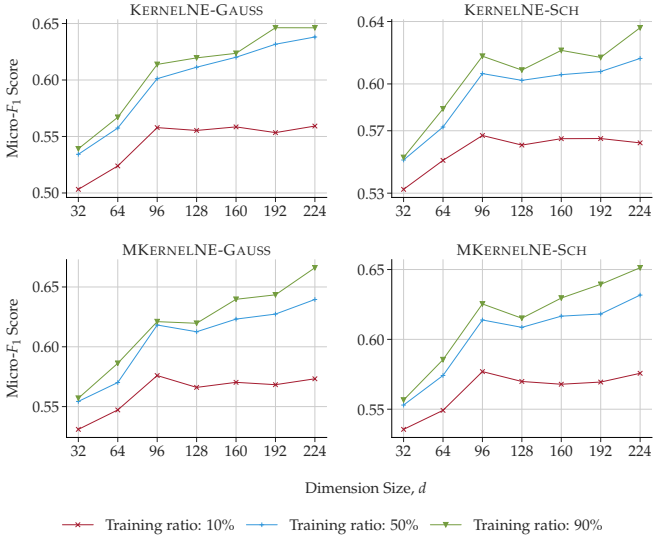

Fig. 2. Influence of the dimension size d on the *CiteSeer* network.

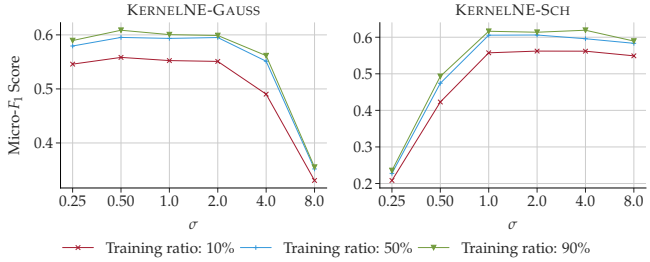

Fig. 3. Influence of kernel parameters on the *CiteSeer* network.

since the desired properties of networks are aimed to be preserved in a lower-dimensional space. Figure 2 depicts the Micro-F_1 score of the proposed models for varying embedding dimension sizes, ranging from $d = 32$ up to $d = 224$ over the *Citeseer* network. As it can be seen, all the different node embedding instances, both single and multiple kernel ones, have the same tendency with respect to d; the performance increases proportionally to the size of the embedding vectors. Focusing on 10% training ratio, we observe that the performance gain almost stabilizes for embedding sizes greater than 96.

The effect of kernel parameters. We have further studied the behavior of the kernel parameter σ of the Gaussian and Schoenberg kernel respectively (as described in Sec. 4.1). Figure 3 shows how the Micro F_1 node classification score is affected with respect to σ for various training ratios on the *Citeseer* network. For the Gaussian kernel, we observe that the performance is almost stable for σ values varying from 0.25 up to 2.0, showing a sudden decrease after 4.0. For the case of the Schoenberg kernel, we observe an almost opposite behavior. Recall that, parameter σ has different impact on these kernels (Sec. 4.1). In particular, the Micro-F_1 score increases for σ values ranging from 0.25 to 0.50, while the performance almost stabilizes in the range of 4.0 to 8.0.

5.7 Running Time Comparison

For running time comparison of the different models, we considered all the networks employed in the experiments. Additionally, we have generated an *Erdös-Renyi* $\mathcal{G}_{n,p}$ graph model [74], by setting the number of nodes $n = 10^5$ and the edge probability $p = 10^{-4}$. Table 7 reports the running time (in seconds) for both the baseline and the proposed models.

For this particular experiment, we have run all the models on a machine of 64GB memory with a single thread when it is possible. The symbol "-" signifies that either the corresponding method cannot run due to excessive memory requirements or that it requires more than one day to complete. As we can observe, both the proposed models have comparable running time—utilizing multiple kernels with MKERNELNE improves performance on downstream tasks without heavily affecting efficiency. Besides, KERNELNE runs faster than LINE in most cases, while MKERNELNE with two kernels shows also a comparable performance. It is also important to note that although matrix factorization-based models such as M-NMF are well-performing in some tasks, they are not very efficient because of the excessive memory demands (please see Sec. 2 for more details). On the contrary, our kernelized matrix factorization models are more efficient and scalable by leveraging negative sampling. The single kernel variant also runs faster than the random walk method DEEPWALK.

In addition to the running time comparison of the proposed approach against the baseline models, we further

TABLE 7
Comparison of running time (in seconds) on an Erdös-Renyi random graph and the real-world networks used in the experiments. The symbol "-" denotes that the corresponding method is unable to run due to excessive memory requirements or it takes more than one day to complete.

	DEEPWALK	NODE2VEC	LINE	HOPE	VERSE	PRONE	NETMF	GEMSEC	M-NMF	KERNELNE	MKERNELNE
Citeseer	136	42	2,020	26	294	5	34	2,012	530	93	186
Cora	139	40	2,069	27	256	5	22	2,064	392	87	170
Dblp	1,761	335	2,717	216	5,582	15	324	113,432	35,988	1,079	1,942
PPI	316	248	3,332	45	558	4	14	4,320	871	133	256
AstroPh	1,274	318	3,152	147	3,855	13	135	46,768	16,875	555	1,055
HepTh	804	128	3,866	77	1,264	5	84	15,838	3,776	346	625
Facebook	217	75	2,411	32	525	4	10	3,353	924	114	226
Gnutella	890	127	3,850	82	1,130	5	69	11,907	3,523	330	575
Erdös-Renyi	9,525	1,684	3,449	1,163	25,341	54	1,089	-	-	3,845	6,486

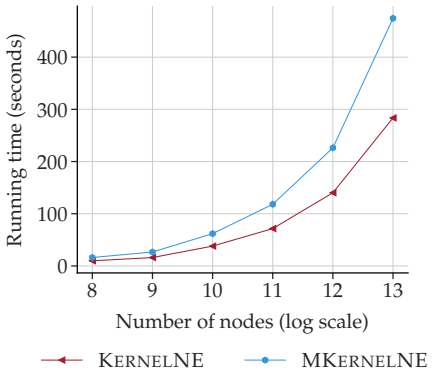

Fig. 4. Running time of KERNELNE and MKERNELNE models on Erdös-Renyi random graphs of different sizes.

examine the running time of KERNELNE and MKERNELNE (using three base kernels) over Erdös-Rényi graphs of varying sizes, ranging from 2^8 to 2^{13} nodes. In the generation of random graphs, we set the edge probabilities so that the expected node degree of graphs is 10. Figure 4 shows the running time of the proposed models. Since the Gaussian and Schoenberg kernels have similar performance, we report the running time for the Gaussian kernel only. As we observe, considering multiple kernels does not significantly affect the scalability properties of the MKERNELNE model.

5.8 Visualization and Clustering of Embedding Vectors

Modularity is a measure designed to assess the quality of the clustering structure of a graph [74]. High modularity implies good clustering structure—the network consists of substructures in which nodes densely connected with each other. Here, we perform a simple visualization and clustering experiment to examine the ability of the different node embedding models to capture the underlying community structure, preserving network's modularity in the embedding space. Note that, to keep the settings of the experiment simple, we leverage the raw embedding vectors visualizing them in the two-dimensional space (instead of performing visualization with t-SNE [75] or similar algorithms).

We perform experiments on the *Dolphins* [76] toy network, which contains 62 nodes and 159 edges. We use the LOUVAIN algorithm [77] to detect the communities in the network. Each of these detected five communities is represented with a different color in Figure 5a. We also use the proposed and the baseline models to learn node embeddings in two-dimensional space.

As we can observe in Fig. 5, different instances of MKERNELNE learn embeddings in which nodes of the different communities are better distributed in the two-dimensional space. Besides, to further support this observation, we run the k-MEANS clustering algorithm [73] on the embedding vectors, computing the corresponding *normalized mutual information* (NMI) scores [78], assuming as ground-truth communities the ones computed by the LOUVAIN algorithm in the graph space. While the NMI scores for NODE2VEC and NETMF are 0.532 and 0.572 respectively, KERNELNE-SCH achieves 0.511 while for KERNELNE-GAUSS we have 0.607. The NMI scores significantly increase for proposed multiple kernel models MKERNELNE-SCH and MKERNELNE-GAUSS, which are 0.684 and 0.740 respectively.

6 CONCLUSION

In this paper, we have studied the problem of learning node embeddings with kernel functions. We have first introduced KERNELNE, a model that aims at interpreting random-walk based node proximity under a weighted matrix factorization framework, allowing to utilize kernel functions. To further boost performance, we have introduced MKERNELNE, extending the proposed methodology to the multiple kernel learning framework. Besides, we have discussed how parameters of both models can be optimized via negative sampling in an efficient manner. Extensive experimental evaluation showed that the proposed kernelized models substantially outperform baseline NRL methods in node classification and link prediction tasks.

The proposed kernelized matrix factorization opens further research directions in network representation learning that we aim to explore in future work. To incorporate the sparsity property prominent in real-world networks, a probabilistic interpretation [79] of the proposed matrix factorization mode would be suitable. Besides, it would be interesting to examine how the proposed models could be extended in the case of dynamic networks.

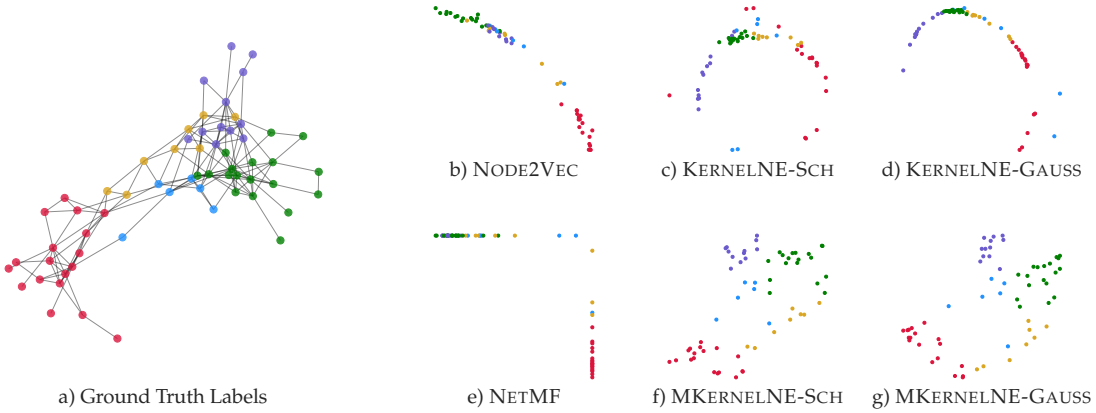

Fig. 5. The visualization of embeddings learned in 2D space for *Dolphins*. The colors indicate community labels computed by the LOUVAIN algorithm.

Acknowledgements. Supported in part by ANR (French National Research Agency) under the JCJC project GraphIA (ANR-20-CE23-0009-01).

REFERENCES

[1] W. L. Hamilton, R. Ying, and J. Leskovec, "Representation learning on graphs: Methods and applications," *IEEE Data Eng. Bull.*, vol. 40, no. 3, pp. 52–74, 2017.
[2] D. Zhang, J. Yin, X. Zhu, and C. Zhang, "Network representation learning: A survey," *IEEE Trans. on Big Data*, vol. 6, pp. 3–28, 2020.
[3] B. Perozzi, R. Al-Rfou, and S. Skiena, "DeepWalk: Online learning of social representations," in *KDD*, 2014, pp. 701–710.
[4] A. Grover and J. Leskovec, "Node2Vec: Scalable feature learning for networks," in *KDD*, 2016, pp. 855–864.
[5] M. Belkin and P. Niyogi, "Laplacian eigenmaps and spectral techniques for embedding and clustering," in *NIPS*, 2001, pp. 585–591.
[6] S. Cao, W. Lu, and Q. Xu, "GraRep: Learning graph representations with global structural information," in *CIKM*, 2015, pp. 891–900.
[7] M. Ou, P. Cui, J. Pei, Z. Zhang, and W. Zhu, "Asymmetric transitivity preserving graph embedding," in *KDD*, 2016, pp. 1105–1114.
[8] D. Berberidis, A. N. Nikolakopoulos, and G. B. Giannakis, "Adaptive diffusions for scalable learning over graphs," *IEEE Trans. on Signal Processing*, vol. 67, no. 5, pp. 1307–1321, 2019.
[9] S. T. Roweis and L. K. Saul, "Nonlinear dimensionality reduction by locally linear embedding," *Science*, vol. 290, no. 5500, pp. 2323–2326, 2000.
[10] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *NIPS*, 2013, pp. 3111–3119.
[11] D. Nguyen and F. D. Malliaros, "BiasedWalk: Biased sampling for representation learning on graphs," in *IEEE Big Data*, 2018, pp. 4045–4053.
[12] A. Çelikkanat and F. D. Malliaros, "TNE: A latent model for representation learning on networks," in *NeurIPS Relational Representation Learning Workshop*, 2018.
[13] A. Çelikkanat and F. D. Malliaros, "Exponential family graph embeddings," in *AAAI*. AAAI Press, 2020, pp. 3357–3364.
[14] S. Chanpuriya and C. Musco, "InfiniteWalk: Deep network embeddings as laplacian embeddings with a nonlinearity," in *KDD*, 2020, p. 1325–1333.
[15] B. Schölkopf, A. Smola, and K.-R. Müller, "Kernel principal component analysis," in *ICANN*, 1997, pp. 583–588.
[16] B. Scholkopf and A. J. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. Cambridge, MA, USA: MIT Press, 2001.
[17] I. S. Dhillon, Y. Guan, and B. Kulis, "Kernel k-means: Spectral clustering and normalized cuts," in *KDD*, 2004, pp. 551–556.
[18] X. Liu, C. Aggarwal, Y.-F. Li, X. Kong, X. Sun, and S. Sathe, "Kernelized matrix factorization for collaborative filtering," in *SDM*, 2016, pp. 378–386.
[19] T. Hofmann, B. Schölkopf, and A. J. Smola, "Kernel methods in machine learning," *Ann. Stat.*, vol. 36, no. 3, pp. 1171–1220, 2008.
[20] M. Gönen and E. Alpaydın, "Multiple kernel learning algorithms," *J. Mach. Learn. Res.*, vol. 12, no. null, p. 2211–2268, Jul. 2011.
[21] S. Wang, Q. Huang, S. Jiang, and Q. Tian, "S3MKL: Scalable semi-supervised multiple kernel learning for real-world image applications," *IEEE Trans. on Multimedia*, vol. 14, pp. 1259–1274, 2012.
[22] N. M. Kriege, F. D. Johansson, and C. Morris, "A survey on graph kernels," *Appl. Netw. Sci.*, vol. 5, no. 1, p. 6, 2020.
[23] J. I. T. and C. Jorge, "Principal component analysis: a review and recent developments," *Phil. Trans. R. Soc.*, 2016.
[24] J. B. Tenenbaum, V. d. Silva, and J. C. Langford, "A global geometric framework for nonlinear dimensionality reduction," *Science*, vol. 290, no. 5500, pp. 2319–2323, 2000.
[25] X. Fu, K. Huang, N. D. Sidiropoulos, and W. Ma, "Nonnegative matrix factorization for signal and data analytics: Identifiability, algorithms, and applications," *IEEE Signal Processing Magazine*, vol. 36, no. 2, pp. 59–80, 2019.
[26] J. Qiu, Y. Dong, H. Ma, J. Li, K. Wang, and J. Tang, "Network embedding as matrix factorization: Unifying DeepWalk, LINE, PTE, and Node2Vec," in *WSDM*, 2018, pp. 459–467.
[27] J. Qiu, Y. Dong, H. Ma, J. Li, C. Wang, K. Wang, and J. Tang, "NetSMF: Large-scale network embedding as sparse matrix factorization," in *WWW*, 2019, pp. 1509–1520.
[28] S. Cao, W. Lu, and Q. Xu, "Deep neural networks for learning graph representations," *AAAI*, vol. 30, no. 1, Feb. 2016.
[29] D. Wang, P. Cui, and W. Zhu, "Structural deep network embedding," in *SIGKDD*, 2016, p. 1225–1234.
[30] W. L. Hamilton, *Graph Representation Learning*. Morgan and Claypool Publishers, 2020.
[31] A. Çelikkanat and D. F. Malliaros, "Topic-aware latent models for representation learning on networks," *Pattern Recognition Letters*, vol. 144, pp. 89–96, 2021.
[32] S. Abu-El-Haija, B. Perozzi, R. Al-Rfou, and A. A. Alemi, "Watch your step: Learning node embeddings via graph attention," in *NeurIPS*, vol. 31, 2018.
[33] A. Tsitsulin, D. Mottin, P. Karras, and E. Müller, "Verse: Versatile graph embeddings from similarity measures," in *WWW*, 2018, pp. 539–548.
[34] J. Zhang, Y. Dong, Y. Wang, J. Tang, and M. Ding, "Prone: Fast and scalable network representation learning," in *IJCAI*, 2019, pp. 4278–4284.
[35] D. Yang, P. Rosso, B. Li, and P. Cudre-Mauroux, "Nodesketch: Highly-efficient graph embeddings via recursive sketching," in *KDD*, 2019, pp. 1162–1172.
[36] Z. Zhang, P. Cui, H. Li, X. Wang, and W. Zhu, "Billion-scale network embedding with iterative random projection," in *ICDM*, 2018, pp. 787–796.
[37] H. Chen, S. F. Sultan, Y. Tian, M. Chen, and S. Skiena, "Fast and accurate network embeddings via very sparse random projection," in *CIKM*, 2019, pp. 399–408.
[38] C. Alzate and J. A. K. Suykens, "Multiway spectral clustering with out-of-sample extensions through weighted kernel pca," *IEEE PAMI*, vol. 32, no. 2, pp. 335–347, 2010.

[39] Fei Zhu, P. Honeine, and M. Kallas, "Kernel nonnegative matrix factorization without the pre-image problem," in *MLSP*, 2014, pp. 1–6.

[40] R. I. Kondor and J. Lafferty, "Diffusion kernels on graphs and other discrete input spaces," in *ICML*, 2002, pp. 315–322.

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

[42] K. Melnyk, S. Klus, G. Montavon, and T. O. F. Conrad, "Graphkke: graph kernel koopman embedding for human microbiome analysis," *Applied Network Science*, vol. 5, no. 1, p. 96, 2020.

[43] Y. Tian, L. Zhao, X. Peng, and D. Metaxas, "Rethinking kernel methods for node representation learning on graphs," in *NIPS*, 2019, pp. 11 686–11 697.

[44] Y. Shen, B. Baingana, and G. B. Giannakis, "Kernel-based structural equation models for topology identification of directed networks," *IEEE Trans. on Signal Processing*, vol. 65, no. 10, pp. 2503–2516, 2017.

[45] D. Romero, V. N. Ioannidis, and G. B. Giannakis, "Kernel-based reconstruction of space-time functions on dynamic graphs," *IEEE J. Sel. Top. Signal Process.*, vol. 11, no. 6, pp. 856–869, 2017.

[46] D. Romero, M. Ma, and G. B. Giannakis, "Kernel-based reconstruction of graph signals," *IEEE Trans. on Signal Processing*, vol. 65, no. 3, pp. 764–778, 2017.

[47] G. Mateos, S. Segarra, A. G. Marques, and A. Ribeiro, "Connecting the dots: Identifying network structure via graph signal processing," *IEEE Signal Process Mag.*, vol. 36, no. 3, pp. 16–43, 2019.

[48] F. R. Bach, G. R. G. Lanckriet, and M. I. Jordan, "Multiple kernel learning, conic duality, and the smo algorithm," in *ICML*, 2004, p. 6.

[49] T. Wang, L. Zhang, and W. Hu, "Bridging deep and multiple kernel learning: A review," *Information Fusion*, vol. 67, pp. 3 – 13, 2021.

[50] A. Salim, S. Shiju, and S. Sumitra, "Design of multi-view graph embedding using multiple kernel learning," *Engineering Applications of Artificial Intelligence*, vol. 90, p. 103534, 2020.

[51] R. Wang, X.-J. Wu, and J. Kittler, "Graph embedding multi-kernel metric learning for image set classification with grassmannian manifold-valued features," *IEEE Trans. on Multimedia*, vol. 23, pp. 228–242, 2021.

[52] S. An, J. Yun, and S. Choi, "Multiple kernel nonnegative matrix factorization," in *ICASSP*, 2011, pp. 1976–1979.

[53] Y. Lin, T. Liu, and C. Fuh, "Multiple kernel learning for dimensionality reduction," *IEEE PAMI*, pp. 1147–1160, 2011.

[54] N. Srebro and T. Jaakkola, "Weighted low-rank approximations," in *ICML*, 2003, pp. 720–727.

[55] A. Epasto and B. Perozzi, "Is a single embedding enough? learning node representations that capture multiple social contexts," in *WWW*, 2019, pp. 394–404.

[56] G. H. Nguyen, J. B. Lee, R. A. Rossi, N. K. Ahmed, E. Koh, and S. Kim, "Dynamic network embeddings: From random walks to temporal random walks," in *IEEE Big Data*, 2018, pp. 1085–1092.

[57] C. Lin, P. Ishwar, and W. Ding, "Node embedding for network community discovery," in *ICASSP*, 2017, pp. 4129–4133.

[58] C. Eckart and G. Young, "The approximation of one matrix by another of lower rank," *Psychometrika*, vol. 1, pp. 211–218, 1936.

[59] I. Steinwart, "On the influence of the kernel on the consistency of support vector machines," *JMLR*, vol. 2, pp. 67–93, 2002.

[60] P. Honeine and C. Richard, "Preimage problem in kernel-based machine learning," *IEEE Signal Process Mag.*, vol. 28, pp. 77–88, 2011.

[61] C. A. Micchelli, Y. Xu, and H. Zhang, "Universal kernels," *J. Mach. Learn. Res.*, vol. 7, pp. 2651–2667, Dec. 2006.

[62] L. Bottou, "Stochastic gradient learning in neural networks," in *In Proceedings of Neuro-Nimes. EC2*, 1991.

[63] A. Mnih and K. Kavukcuoglu, "Learning word embeddings efficiently with noise-contrastive estimation," in *NIPS*, 2013, pp. 2265–2273.

[64] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "LINE: Large-scale information network embedding," in *WWW*, 2015, pp. 1067–1077.

[65] B. Rozemberczki, R. Davies, R. Sarkar, and C. Sutton, "GEMSEC: Graph embedding with self clustering," in *ASONAM*, 2019, pp. 65–72.

[66] X. Wang, P. Cui, J. Wang, J. Pei, W. Zhu, and S. Yang, "Community preserving network embedding," in *AAAI*, 2017, pp. 203–209.

[67] H. Chen, B. Perozzi, Y. Hu, and S. Skiena, "Harp: Hierarchical representation learning for networks," in *AAAI*, 2018.

[68] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Gallagher, and T. Eliassi-Rad, "Collective classification in network data," *AI magazine*, 2008.

[69] B. Perozzi, V. Kulkarni, H. Chen, and S. Skiena, "Don't walk, skip!: Online learning of multi-scale network embeddings," in *ASONAM*, 2017, pp. 258–265.

[70] J. Leskovec, J. Kleinberg, and C. Faloutsos, "Graph evolution: Densification and shrinking diameters," *ACM Trans. Knowl. Discov. Data*, vol. 1, no. 1, 2007.

[71] J. Leskovec and J. J. Mcauley, "Learning to discover social circles in ego networks," in *NIPS*, 2012, pp. 539–547.

[72] M. Ripeanu, A. Iamnitchi, and I. Foster, "Mapping the gnutella network," *IEEE Internet Computing*, vol. 6, no. 1, pp. 50–57, 2002.

[73] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," *JMLR*, vol. 12, pp. 2825–2830, 2011.

[74] M. Newman, *Networks: An Introduction*. Oxford Scholarship Online, 2010.

[75] L. van der Maaten and G. Hinton, "Visualizing data using t-sne," *JMLR*, vol. 9, no. 86, pp. 2579–2605, 2008.

[76] D. Lusseau, K. Schneider, O. J. Boisseau, P. Haase, E. Slooten, and S. M. Dawson, "The bottlenose dolphin community of doubtful sound features a large proportion of long-lasting associations," *Behavioral Ecology and Sociobiology*, vol. 54, no. 4, pp. 396–405, 2003.

[77] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *J. Stat. Mech.*, vol. 2008, no. 10, p. P10008, 2008.

[78] F. D. Malliaros and M. Vazirgiannis, "Clustering and community detection in directed networks: A survey," *Physics Reports*, 2013.

[79] A. Mnih and R. R. Salakhutdinov, "Probabilistic matrix factorization," in *NIPS*, 2008, pp. 1257–1264.

Abdulkadir Çelikkanat is currently a postdoctoral researcher at the Section for Cognitive Systems of the Technical University of Denmark. He completed his Ph.D. at the Centre for Visual Computing of CentraleSupélec, Paris-Saclay University, and he was also a member of the OPIS team at Inria Saclay. Before his Ph.D. studies, he received his Bachelor degree in Mathematics and Master's degree in Computer Engineering from Bogaziçi University. His research mainly focuses on the analysis of graph-structured data. In particular, he is interested in graph representation learning and its applications for social and biological networks.

Yanning Shen is an assistant professor at University of California, Irvine. She received her Ph.D. degree from the University of Minnesota (UMN) in 2019. She was a finalist for the Best Student Paper Award at the 2017 IEEE International Workshop on Computational Advances in Multi-Sensor Adaptive Processing, and the 2017 Asilomar Conference on Signals, Systems, and Computers. She was selected as a Rising Star in EECS by Stanford University in 2017, and received the Microsoft Academic Grant Award for AI Research in 2021. Her research interests span the areas of machine learning, network science, data science and signal processing.

Fragkiskos D. Malliaros is an assistant professor at Paris-Saclay University, CentraleSupélec and associate researcher at Inria Saclay. He also co-directs the Master Program in Data Sciences and Business Analytics (CentraleSupélec and ESSEC Business School). Right before that, he was a postdoctoral researcher at UC San Diego (2016-17) and at École Polytechnique (2015-16). He received his Ph.D. in Computer Science from École Polytechnique (2015) and his M.Sc. degree from the University of Patras, Greece (2011). He is the recipient of the 2012 Google European Doctoral Fellowship in Graph Mining, the 2015 Thesis Prize by École Polytechnique, and best paper awards at TextGraphs-NAACL 2018 and ICWSM 2020 (honorable mention). In the past, he has been the co-chair of various data science-related workshops, and has also presented twelve invited tutorials at international conferences in the area of graph mining. His research interests span the broad area of data science, with focus on graph mining, machine learning, and network analysis.