SAGE: SCALABLE ATTRIBUTED GRAPH EMBEDDINGS FOR GRAPH CLASSIFICATION

Lingfei Wu *
IBM Research AI
IBM T.J. Waston Research Center
Yorktown Heights, NY 10598, USA
wuli@us.ibm.com

Zhen Zhang * **& Arye Nehorai**
Department of Electrical and Systems Engineering
Washington University in St. Louis
St. Louis, MO 63112, USA
{zhen.zhang,nehorai}@wustl.edu

Liang Zhao
Department of Information Science and Technology
George Mason university
Fairfax, VA 22030, USA
lzhao9@gmu.edu

Fangli Xu
AI Squirrel Learning
Yixue Education Inc.
Highland Park, NJ, 08904, USA
lili@yixue.us

ABSTRACT

Graph-structured data analysis is an increasingly popular topic with applications in many fields. Despite rich existing works in Graph Kernels and Graph Neural Networks, most of these approaches only consider the connectivity structure and discrete node attributes, while ignoring the graphs with continuous node and edge attributes. In this paper, we propose a scalable algorithm for graph embeddings, which encode both the node attribute and edge attribute information. We develop a unifying embedding framework for node-attributed and edge-attributed graphs, by the virtue of the adjoint graph. Extensive graph classification experiments demonstrate the superior performance of our graph embeddings.

1 INTRODUCTION

Graph-structured data analysis has been widely used in many application domains such as computational biology, drug discovery, and social network analysis. Recent years have seen a surge of interests in graph representation learning. Graph kernels are one of the most effective methods learning graph feature map and measuring the similarity between graphs. While highly scalable graph kernels have been developed for graphs with thousands of nodes (Shervashidze et al., 2009), most of existing graph kernels are designed for unlabeled graphs or graphs with discrete node attributes (node labels) (Shervashidze et al., 2011). However, graphs appeared in real-world applications often come with continuous-valued measurements or properties attached to both nodes and edges. Therefore, a natural and open challenge is that *how to develop scalable kernels and embeddings on graphs, both nodes and edges of which have discrete and continuous attributes?*

Inspired by the recently proposed generic distance to kernel learning framework (Wu et al., 2018), we present a new approach to construct a positive-definite graph kernel via an infinite-dimensional feature map given by the energy distance between node embedding sets of graphs. Based on this kernel, we can then derive our graph embedding via a randomized kernel approximation. The linear inner products of these embeddings approximate the exact kernels. This particular randomized approximation allows us to easily obtain a scalable attributed graph embedding (SAGE) as well as to efficiently compute the graph kernels. In this paper, we present Scalable Attributed Graph Embeddings (SAGE) for graphs with any type of node attributes and edge attributes. SAGE is able to encode both the topological connectivity and attributes information of graphs. We develop an unifying embedding framework for node-attributed and edge-attributed graphs, by the virtue of the adjoint graph. The computation of this novel graph embedding scales linearly with both the number of graphs and the embedding size.

*Both authors contributed equally

Related work. A popular line of research focuses on developing graph kernels, without obtaining explicit graph representations. Most existing ones are based on the R-convolution framework (Haussler, 1999), whose key idea is decomposing a whole graph into substructures, such as subtrees (Shervashidze et al., 2011), graphlets (Shervashidze et al., 2009), and shortest paths (Borgwardt & Kriegel, 2005), and then quantitatively measuring the similarities among them. All the kernels are designed for graphs without or with only discrete attributes. Recently, the subgraph matching kernels (Kriege & Mutzel, 2012) and the graph invariant kernels (Orsini et al., 2015) were proposed for handling continuous node attributes. More recently, graph kernels that are based on either geometric node embeddings, such as (Johansson et al., 2014), (Johansson & Dubhashi, 2015), and (Nikolentzos et al., 2017), or the return probability feature embeddings (Zhang et al., 2018) were proposed. However, all the above work consider only the node attributes while ignoring the edge attributes. In our paper, we develop a framework for graph kernel design and embedding, which can be applied to graphs with both node and edge attributes.

Our contributions. We highlight our contributions as follows: (i.) We propose a unifying graph embedding framework, which, to the best of our knowledge, is the first one that considers both the node and edge attributes. (ii.) We introduce a novel strategy of converting edge-attributed graphs to node-attributed graphs, opening a door to involving edge attributes information for many other graph kernels. (iii.) We mathematically and empirically show that our proposed graph embedding SAGE exhibits linear complexity in the number of graphs and the size of embeddings. (iv.) We conduct extensive experiments on 25 graph benchmark datasets and demonstrate the superior performance of our approach in terms of both the classification accuracy and computational time compared against 13 state-of-the-art graph kernel based and graph neural network (GNNs) based baselines.

2 Graph Node Embeddings and Dissimilarity Measure

We consider the attributed graph $G = \{\mathcal{V}, \mathcal{E}, \mathcal{A}_N, \mathcal{A}_E\}$ of n nodes and m edges, where $\mathcal{V} = \{v_1, v_2, ..., v_n\}$ is the set of nodes, $\mathcal{E}$ is the set of edges, and $\mathcal{A}_N$ and $\mathcal{A}_E$ are respectively the node attributes and edge attributes set.

Graph node embeddings. We employ the return probability features (RPF) (Zhang et al., 2018) as the Euclidean space embedding of nodes, because they can well capture the topology of graphs and are invariant to graph isomorphisms. As in (Zhang et al., 2018), given each node $v_i \in G$, we represent it with a vector $\vec{\boldsymbol{p}}_i$, defined as $\vec{\boldsymbol{p}}_i = \left[\boldsymbol{P}_G^1(i,i), \boldsymbol{P}_G^2(i,i), ..., \boldsymbol{P}_G^S(i,i)\right]^T$, where $\boldsymbol{P}_G^1 = \boldsymbol{D}_G^{-1}\boldsymbol{A}_G$ is the transition probability matrix of the random walk on G, and $\boldsymbol{P}_G^s(i,i)$, $s = 1, 2, ..., S$ is the $s-$step return probability of random walks starting from v_i. Integrating RPF and the node attributes $\{\vec{\boldsymbol{a}}_i\}_{i=1}^n \subseteq \mathcal{A}_N$, we can represent the graph with the set

$$\mathcal{S}(G) = \{(\vec{\boldsymbol{p}}_i, \vec{\boldsymbol{a}}_i)\}_{i=1}^n \subseteq \mathbb{R}^S \times \mathcal{A}_N. \tag{1}$$

Graph dissimilarity measure. We use the energy distance (Székely, 2003) as the dissimilarity measurement of set representations. Let $\mathcal{S}_1 = \{x_1, x_2, ..., x_{n_1}\}$ and $\mathcal{S}_2 = \{y_1, y_2, ..., y_{n_2}\}$ be two sets in $\mathcal{X}$. Let d be a metric (here called the "ground distance") on $\mathcal{X}$. The energy distance, D_E, between $\mathcal{S}_1$ and $\mathcal{S}_2$ is defined as $D_E^2(S_1, S_2) = 2A - B - C$, where A, B, and C equal to $\frac{1}{n_1 n_2}\sum_{i=1}^{n_1}\sum_{j=1}^{n_2} d(x_i, y_j)$, $\frac{1}{n_1^2}\sum_{i=1}^{n_1}\sum_{j=1}^{n_1} d(x_i, x_j)$, and $\frac{1}{n_2^2}\sum_{i=1}^{n_2}\sum_{j=1}^{n_2} d(y_i, y_j)$, respectively.

Constructing the ground distance d. Now we construct the ground distance d such that it can well encode the difference of both the structure and node attributes information. Without loss of generality, we assume that the graphs have both discrete and continuous attributes, i.e., $\mathcal{S}(G) = \{(\vec{\boldsymbol{p}}_i, \vec{\boldsymbol{a}}_i^{(d)}, \vec{\boldsymbol{a}}_i^{(c)})\}_{i=1}^n$,where $\vec{\boldsymbol{a}}_i^{(c)}$ is the continuous attribute vector, and $\vec{\boldsymbol{a}}_i^{(d)}$ is the one-hot vector that represents the discrete attribute. We provide a novel ground distance by leveraging the power of tensors. That is, we represent each node v_i as a rank-one tensor, i.e., $x_i = \vec{\boldsymbol{p}}_i \otimes \vec{\boldsymbol{a}}_i^{(d)} \otimes \vec{\boldsymbol{a}}_i^{(c)}$. Applying the fact that $\langle \vec{\boldsymbol{u}}_1 \otimes \vec{\boldsymbol{v}}_1 \otimes \vec{\boldsymbol{w}}_1, \vec{\boldsymbol{u}}_2 \otimes \vec{\boldsymbol{v}}_2 \otimes \vec{\boldsymbol{w}}_2 \rangle_{\mathcal{T}} = \langle \vec{\boldsymbol{u}}_1, \vec{\boldsymbol{u}}_2 \rangle \cdot \langle \vec{\boldsymbol{v}}_1, \vec{\boldsymbol{v}}_2 \rangle \cdot \langle \vec{\boldsymbol{w}}_1, \vec{\boldsymbol{w}}_2 \rangle$, we can obtain d by computing the distance between two rank-one tensors x and y, i.e., $d^2(x, y) = \|x - y\|_{\mathcal{T}}^2 = \langle x, x \rangle_{\mathcal{T}} + \langle y, y \rangle_{\mathcal{T}} - 2\langle x, y \rangle_{\mathcal{T}}$.

3 SCALABLE ATTRIBUTED GRAPH EMBEDDINGS

We extend the recently proposed distance to kernel learning framework "D2KE" (Wu et al., 2018) to graphs, constructing the graph kernel and scalable graph embeddings. Let $\mathcal{G}$ be the space of all graphs. We define the function k on $\mathcal{G} \times \mathcal{G}$,

$$k(G_x, G_y) = \int_{\mathcal{G}} p(G_\omega)\phi_{G_\omega}(G_x)\phi_{G_\omega}(G_y)\mathrm{d}G_\omega, \tag{2}$$

where $\phi_{G_\omega}(G) = \exp(-\gamma D_E(G, G_\omega))$ and G_ω is a random graph, and $p(G_\omega)$ is a probability distribution over all random graphs in $\mathcal{G}$. As shown in (Wu et al., 2018), we have $k(G_x, G_y) \to \exp(-\gamma D_E(G_x, G_y))$ as $\gamma \to \infty$, which implies that k is a similarity measurement. Moreover, it can be checked that k is positive definite. Consequently, k is a graph kernel.

Node-attributed graph embeddings. The exact computation of k is intractable, since the integral over $\mathcal{G}$ has no analytic solutions. Inspired by randomized kernel approximation introduced in (Rahimi & Recht, 2008), we approximate the graph kernel k by using a finite number of random graphs. Let G_{ω_i}, $i = 1, 2, ..., R$ be random graphs sampled from a probability distribution p on $\mathcal{G}$. We consider the Euclidean space embedding, $\vec{\boldsymbol{Z}}^N(G)$, for node-attributed graph G, i.e.,

$$\vec{\boldsymbol{Z}}^N(G) = \frac{1}{\sqrt{R}}\left[\phi_{G_{\omega_1}}(G), \phi_{G_{\omega_2}}(G), ..., \phi_{G_{\omega_R}}(G)\right]^T \in \mathbb{R}^R. \tag{3}$$

Then, as $R \to \infty$, we have $\langle \vec{\boldsymbol{Z}}^N(G_x), \vec{\boldsymbol{Z}}^N(G_y)\rangle = \frac{1}{R}\sum_{i=1}^R \phi_{G_{\omega_i}}(G_x)\phi_{G_{\omega_i}}(G_y) \to k(G_x, G_y)$. We refer to the Appendix A for the strategy of sampling random graphs.

Edge-attributed graph embeddings. Now we consider the edge-attributed graphs. We employ the adjoint graph (Whitney, 1992) (also called the "line graph" or the "edge graph"), which translates the property of the original graph from edges to nodes. Consequently, we can consider the problem of embedding edge-attributed graphs as that of embedding node-attributed graphs.

Given a graph G, we can construct its adjoint graph[1] G^* such that i) the nodes in G^* represent the edges in G; ii) two nodes in G^* are adjacent if the corresponding edges in G have the same ending nodes. A toy graph is shown in Fig. 1. We can see that the edges in G are transformed to the nodes in G^*. Simultaneously, the edge attributes are converted into the node attributes. Therefore, the node embeddings of G^* are equivalent to the edge embeddings of G.

Now we can operate the return probability feature extraction on G^*. Let $\boldsymbol{q}_j$ and $\vec{\boldsymbol{b}}_j$ respectively be the edge structural representation and edge attributes of edge e_i in G. We can then characterize the graph with the set (similar with equation 1), i.e., $\mathcal{S}(G^*) = \{(\vec{\boldsymbol{q}}_j, \vec{\boldsymbol{b}}_j)\}_{j=1}^m \subseteq \mathbb{R}^S \times \mathcal{A}_E$. With $\mathcal{S}(G^*)$, we can get the edge-attributed graph representation,

$$\vec{\boldsymbol{Z}}^E(G) = \vec{\boldsymbol{Z}}^N(G^*) \in \mathbb{R}^R. \tag{4}$$

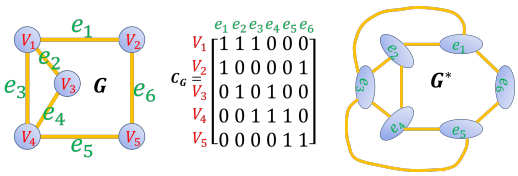

Figure 1: Left: A graph G. Middle: The corresponding incidence matrix C_G. Right: The adjoint graph G^* converted from G. For example, in G, the edges e_1, e_2, and e_3 share a common node V_1. So in G^*, the nodes e_1, e_2, and e_3 are connected.

Two types of graph embedding fusion. In some cases, the graph has both node and edge attributes. Therefore, we will have both the node-attributed embedding $\vec{\boldsymbol{Z}}^N(G)$ and the edge-attributed embedding $\vec{\boldsymbol{Z}}^E(G)$. We fuse them to obtain the final embedding vector $\vec{\boldsymbol{Z}}(G)$ of G,

$$\vec{\boldsymbol{Z}}(G) = \left[\vec{\boldsymbol{Z}}_0^T(G), \vec{\boldsymbol{Z}}_1^T(G), \vec{\boldsymbol{Z}}_2^T(G)\right]^T \in \mathbb{R}^{3R}, \tag{5}$$

where $\vec{\boldsymbol{Z}}_0(G) = \vec{\boldsymbol{Z}}^N(G)$, $\vec{\boldsymbol{Z}}_1(G) = \vec{\boldsymbol{Z}}^N(G) \circ \vec{\boldsymbol{Z}}^E(G)$, and $\vec{\boldsymbol{Z}}_2(G) = |\vec{\boldsymbol{Z}}^N(G) - \vec{\boldsymbol{Z}}^E(G)|$.

In algorithm 1, we summarize the procedure of obtaining the vector embedding of graph $G = \{\mathcal{V}, \mathcal{E}, \mathcal{A}_N, \mathcal{A}_E\}$. We refer to the Appendix C for the complexity analysis of our algorithm.

[1] We describe in detail the relationship between the original graph and the adjoint graph in Appendix B

Algorithm 1 Scalable Attributed Graph Embeddings

Input: The attributed graph sets $\{G_i\}_{i=1}^N$.
Output: The graph embeddings $\{\vec{\boldsymbol{Z}}(G_i)\}_{i=1}^N$.
1: Compute the node embeddings $\{\mathcal{S}(G_i)\}_{i=1}^N$and the edge embeddings $\{\mathcal{S}(G_i^*)\}_{i=1}^N$.
2: Sampling random graphs $\{G_{\omega_i}\}_{i=1}^R$ (See the supplementary material).
3: Compute $\vec{\boldsymbol{Z}}^N(G_i)$ (see equation 3) and $\vec{\boldsymbol{Z}}^E(G_i)$ (see equation 4), $i = 1, 2, ..., N$.
4: Compute $\vec{\boldsymbol{Z}}(G_i)$ (see equation 5), $i = 1, 2, ..., N$.

4 EXPERIMENTS

We conduct experiments to evaluate the effectiveness and efficiency of SAGE, with the goal of answering the following questions: **Q1**: How does SAGE perform and scale with respect to the size of graph embeddings? **Q2**: How does SAGE scale with respect to the number of graphs? **Q3**: On graph classification tasks, how does SAGE perform, compared with other state-of-the-art methods? **Q4**: How much improvement does SAGE achieve by considering the node and edge attributes?

Setup. We compare SAGE with 13 state-of-the-art graph kernels and graph neural networks. Detailed descriptions of these 25 datasets and 13 baselines are provided in Appendix D. We directly employ a linear SVM implemented in LIBLINEAR (Fan et al., 2008) since SAGE is a graph-level embedding. The cost parameter C is selected from $\{10^k, k \in \{-3, -2, ...2, 3\}\}$. We set the node embedding size $S = 50$ and select R in the range of $[4, \min(2^k \leq N, 2048)]$, where N the size of the graph dataset. The ranges of hyperparameters γ and $D_{\max}$ are $\{10^{-3}, 10^{-2}, 10^{-1}, 1, 10^1\}$ and [3:3:30], respectively. All parameters of the SVM and hyperparameters of our method were optimized only on the training dataset. We perform 10-fold cross-validation to evaluate the performance of SAGE using 9 folds for training and 1 for testing, and repeat the experiments ten times to report the averaged accuracies and standard deviations.

Table 1: Classification accuracy (in %) for graphs without node labels

Datasets	WL	GK	DGK	PSCN	DGCNN	RetGK	**SAGE**
COLLAB	74.8± 0.2	72.8± 0.3	73.1± 0.3	72.6± 2.2	73.7± 0.5	80.7± 0.3 (2505s)	**82.3± 0.4**(1347s)
IMDB-BINARY	70.8± 0.5	65.9± 1.0	67.0± 0.6	71.0± 2.3	70.0± 1.0	71.9± 1.0 (189.6s)	**73.8 ± 0.5** (28.6s)
IMDB-MULTI	**49.8± 0.5**	43.9± 0.4	44.6± 0.5	45.2± 2.8	47.8± 0.9	47.7± 0.3 (158.7s)	49.4± 0.7 (8.7s)
REDDIT-BINARY	68.2± 0.2	77.3± 0.2	78.0± 0.4	86.3± 1.6	–	**92.6± 0.3**(9.8h)	92.0± 0.2(3453s)
REDDIT-MULTI(5K)	51.2± 0.3	41.0± 0.2	41.3± 0.2	49.1± 0.2	–	56.1± 0.5(20.7h)	**57.2± 0.4** (2341s)
REDDIT-MULTI(12K)	32.6± 0.3	31.8± 0.1	32.2± 0.1	41.3± 0.4	–	**48.7± 0.2**(36.7h)	48.2± 0.2(3987s)

Table 2: Classification accuracy (in %) for graphs with node labels

Datasets	WL	SP	GK	DGK	PSCN	DGCNN	RetGK	**SAGE**
ENZYMES	53.4± 0.9	38.6± 1.5	–	53.4± 0.9	–	–	**60.4± 0.8** (47.8s)	56.7± 0.7 (21.7s)
PROTEINS	71.2± 0.8	73.3± 0.9	71.7± 0.6	75.7± 0.5	75.0± 2.5	75.5± 0.9	75.8± 0.6 (166.6s)	**77.2± 0.3** (4.6s)
MUTAG	84.4± 1.5	85.2± 2.3	81.6± 2.1	87.4± 2.7	89.0± 4.4	85.8± 1.7	90.3± 1.1(3.5s)	**91.1± 0.7** (0.2s)
PTC-FM	55.2± 2.3	60.5± 1.7	–	–	–	–	61.6± 1.6 (12.1s)	**64.7± 1.0** (0.3s)
PTC-FR	63.9± 1.4	61.6± 1.0	–	–	–	–	66.1± 1.6 (12.5s)	**67.7 ± 0.8** (5.7s)
PTC-MM	60.6± 1.1	62.9± 1.4	–	–	–	–	65.8± 1.1 (10.6s)	**68.2 ± 0.8** (2.5s)
PTC-MR	55.4± 1.5	57.8± 2.1	57.3± 1.1	60.1± 2.6	62.3± 5.7	58.6± 2.5	**62.5± 1.6** (12.3s)	62.1± 0.9 (2.3s)
NCI1	**85.4± 0.3**	74.8± 0.4	62.3± 0.3	80.3± 0.5	76.3± 1.7	74.4± 0.5	84.5± 0.2 (1953.4s)	81.9± 0.2 (140.4s)
DD	78.6± 0.4	>24h	–	78.5± 0.3	76.2± 2.6	79.4± 0.9	**81.6± 0.3** (1693s)	80.2± 0.5 (801s)

Table 3: Classification accuracy (in %) for graphs with both node labels and node attributes

Datasets	SP	CSM	HGK-WL	GH	GIK	P2K	RetGK	**SAGE**
ENZYMES	65.7± 1.1	69.8± 0.7	67.6± 1.0	68.8± 1.0	71.7± 0.8	69.2± 0.4	**72.2± 0.8** (73.5s)	69.5± 0.7 (15.3s)
PROTEINS	75.7± 0.4	–	76.7± 0.4	72.2± 0.3	76.8± 0.5	73.4± 0.5	**77.6± 0.5** (252.6s)	**77.6 ± 0.4** (35.6s)
PROTEINS_full	75.8± 0.5	–	–	73.1± 0.4	–	76.1 ± 0.3	75.6± 0.3 (270.6s)	**77.4 ± 0.3** (4.6s)
BZR	78.2± 1.2	79.4± 1.2	–	–	–	86.3± 1.1	86.4± 1.2 (34.5s)	**87.5± 0.7** (1.0s)
COX2	74.5± 1.3	74.4± 1.7	–	–	–	80.5± 1.8	79.5± 0.7 (47.9s)	**83.5± 0.5** (22.8s)
DHFR	77.6± 1.5	79.9± 1.1	–	–	–	79.5± 0.8	80.8± 0.4 (124.4s)	**81.6± 0.4** (41.8s)

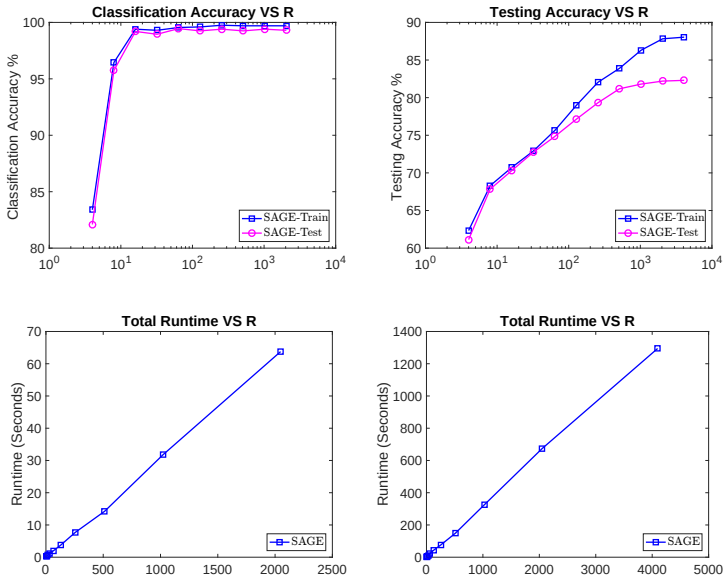

Figure 2: Accuracy and runtime of SAGE when varying R on datasets AIDS.

Experimental results of graph classification. To comprehensively answer the questions **Q3** and **Q4**, Tables 1, 2, 3, and 4 show that SAGE consistently outperforms or matches other graph kernel-based and GNN-based baselines on all benchmark datasets, highlighting the importance of developing a graph embedding method that can take full advantages of all graph information.

Impacts of R on accuracy and running time. Fig.2 provides positive answer to our motivating question **Q1**: we observe that SAGE converges quickly to its optimal performance when increasing R from a small number to relatively large number (up to 4096). Another observation is that SAGE exhibits linear scalability with respect to R, which is consistent with our computational analysis.

Impact of N on running time. As shown in Fig.3, these results yields encouraging answers to our motivating question **Q2**: SAGE shows the linear scalability in terms of the number of graphs. This is a highly desired property of our SAGE embeddings, since most graph kernels have quadratic complexity in the number of graphs, rendering them hard to scale well.

Table 4: Classification accuracy for graphs with node labels and attributes, and edge labels and attributes. RetGK only handles node labels and attributes while SAGE considers all of them.

Datasets	RetGK	**SAGE**
BZR_MD	63.4± 1.5(15.6s)	**70.7± 0.7** (6.6s)
COX2_MD	63.3± 2.2 (35.3s)	**67.0 ± 0.9** (25.9s)
DHFR_MD	66.2± 1.6 (36.9s)	**70.5 ± 0.8** (26.7s)
ER_MD	72.8± 1.4 (61.5s)	**75.7 ± 0.9** (42.4s)
Cuneiform	38.4± 1.9 (25.4s)	**45.8 ± 1.1** (13.3s)
AIDS	99.4± 0.1 (662.4s)	**99.5 ± 0.3** (2.2s)

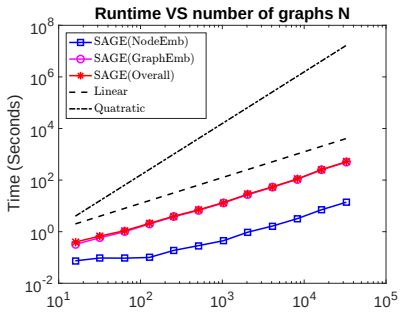

Figure 3: Runtime of SAGE when varying number of graphs N.

5 Conclusion

In this work, we developed a scalable algorithm for representing graphs, which may have node and edge attributes. To this end, we proposed a two-step embedding framework, i.e., the graph node-level embeddings and the graph-level embeddings. Leveraging the adjoint graphs, we translated the properties of graphs from edges to nodes, which provides a unifying view for embedding node-attributed and edge-attributed graphs.

REFERENCES

Karsten M Borgwardt and Hans-Peter Kriegel. Shortest-path kernels on graphs. In *ICDM*. IEEE, 2005.

Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. Liblinear: A library for large linear classification. *JMLR*, 9(Aug), 2008.

Aasa Feragen, Niklas Kasenburg, Jens Petersen, Marleen de Bruijne, and Karsten Borgwardt. Scalable kernels for graphs with continuous attributes. In *NIPS*, 2013.

David Haussler. Convolution kernels on discrete structures. Technical report, Department of Computer Science, University of California, 1999.

Catalin Ionescu, Alin Popa, and Cristian Sminchisescu. Large-scale data-dependent kernel approximation. In *AIStats*, 2017.

Fredrik Johansson, Vinay Jethava, Devdatt Dubhashi, and Chiranjib Bhattacharyya. Global graph kernels using geometric embeddings. In *ICML*, 2014.

Fredrik D Johansson and Devdatt Dubhashi. Learning with similarity functions on graphs using matchings of geometric embeddings. In *KDD*. ACM, 2015.

Kristian Kersting, Nils M. Kriege, Christopher Morris, Petra Mutzel, and Marion Neumann. Benchmark data sets for graph kernels, 2016. http://graphkernels.cs.tu-dortmund.de.

Nils Kriege and Petra Mutzel. Subgraph matching kernels for attributed graphs. *arXiv preprint arXiv:1206.6483*, 2012.

Christopher Morris, Nils M Kriege, Kristian Kersting, and Petra Mutzel. Faster kernels for graphs with continuous attributes via hashing. In *ICDM*. IEEE, 2016.

Marion Neumann, Roman Garnett, Christian Bauckhage, and Kristian Kersting. Propagation kernels: efficient graph kernels from propagated information. *Machine Learning*, 102(2), 2016.

Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. In *ICML*, 2016.

Giannis Nikolentzos, Polykarpos Meladianos, and Michalis Vazirgiannis. Matching node embeddings for graph similarity. In *AAAI*, 2017.

Francesco Orsini, Paolo Frasconi, and Luc De Raedt. Graph invariant kernels. In *IJCAI*, 2015.

Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In *NIPS*, 2008.

Nino Shervashidze, SVN Vishwanathan, Tobias Petri, Kurt Mehlhorn, and Karsten Borgwardt. Efficient graphlet kernels for large graph comparison. In *AIStats*, 2009.

Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-lehman graph kernels. *JMLR*, 12(Sep), 2011.

Gábor J Székely. E-statistics: The energy of statistical samples. *Bowling Green State University, Department of Mathematics and Statistics Technical Report*, 3(05), 2003.

Hassler Whitney. Congruent graphs and the connectivity of graphs. In *Hassler Whitney Collected Papers*. Springer, 1992.

Bo Wu, Yang Liu, Bo Lang, and Lei Huang. Dgcnn: Disordered graph convolutional neural network based on the gaussian mixture model. *arXiv preprint arXiv:1712.03563*, 2017.

Lingfei Wu, Ian En-Hsu Yen, Fangli Xu, Pradeep Ravikumar, and Michael Witbrock. D2ke: From distance to kernel and embedding. *arXiv preprint arXiv:1802.04956*, 2018.

Pinar Yanardag and SVN Vishwanathan. Deep graph kernels. In *KDD*. ACM, 2015.

Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. In *NIPS*, 2018.

Zhen Zhang, Mianzhi Wang, Yijian Xiang, Yan Huang, and Arye Nehorai. Retgk: Graph kernels based on return probabilities of random walks. In *NIPS*, 2018.

Feng Zhou and Fernando De la Torre. Factorized graph matching. In *CVPR*. IEEE, 2012.

6 APPENDIX

6.1 APPENDIX A: SAMPLING RANDOM NODE-ATTRIBUTED GRAPHS

Here we provide a simple but effective approach for sampling random attributed graphs, which is similar with the data-dependent random features sampling strategy proposed in (Ionescu et al., 2017). The "data-dependent" strategy means that we sample graphs from the training dataset $\{G_i\}_{i=1}^{N_{\text{tr}}}$. However, unlike the existing methods that select a representative set of a whole graph, we propose to sample parts of graphs as random graphs. This sampling method will generate random graphs with various topological connectivities and can help to identify hidden global structures. We also note that in order to compute $\phi_{G_\omega}(G)$, it is sufficient to directly use the node embeddings set sampled from the node embedding space. Algorithm 2 gives our random graph sampling procedure.

Algorithm 2 Random Node-Attributed Graphs Generation

Input: The node embedding sets $\{\mathcal{S}(G_i)\}_{i=1}^{N_{\text{tr}}}$ of the training graph dataset, maximal size of the random graph $D_{\max}$.
Output: The node embeddings $\mathcal{S}(G_\omega)$ of a random graph G_ω.
1: Uniformly draw a number k in $\{1, 2, ..., N_{\text{tr}}\}$.
2: Uniformly draw a number D_k in $\{1, 2, ..., D_{\max}\}$.
3: Randomly draw D_k nodes, i.e., $\{v_{i_1}, v_{i_2}, ..., v_{i_{D_k}}\}$ in the graph G_k.
4: Return node embeddings $\mathcal{S}(G_\omega) = \{(\vec{\boldsymbol{p}}_{i_m}, \vec{\boldsymbol{a}}_{i_m})\}_{m=1}^{D_k}$, where $\{(\vec{\boldsymbol{p}}_i, \vec{\boldsymbol{a}}_i)\}_{i=1}^{n_k} = \mathcal{S}(G_k)$ is the node embedding set of the graph G_k of n_k nodes.

6.2 APPENDIX B: RELATIONSHIP BETWEEN THE ORIGINAL GRAPH AND THE ADJOINT GRAPH

Given a graph $G = \{\mathcal{V}, \mathcal{E}, \mathcal{A}_N, \mathcal{A}_E\}$, we can use the binary[2] adjacency matrix $\boldsymbol{A}_G$ or the incidence matrix $\boldsymbol{C}_G$ to characterize the connectivity information of G. The adjacency matrix $\boldsymbol{A}_G$ is defined as

$$\boldsymbol{A}_G(i,j) = \begin{cases} 1, & \text{if } (v_i, v_j) \in \mathcal{E} \\ 0, & \text{otherwise} \end{cases}, \tag{6}$$

The incidence matrix $\boldsymbol{C}_G$ (see an example in Fig. 1) describes the node-edge correspondence, i.e.,

$$\boldsymbol{C}_G(i,e) = \boldsymbol{C}_G(j,e) = \begin{cases} 1, & \text{if } e = (v_i, v_j) \in \mathcal{E} \\ 0, & \text{otherwise} \end{cases}. \tag{7}$$

Let G^* be the adjoint graph of G, and let $\boldsymbol{A}_{G^*}$ be the adjacency matrix of G^*. Then,

$$\boldsymbol{C}_G \boldsymbol{C}_G^T = \boldsymbol{D}_G + \boldsymbol{A}_G, \tag{8a}$$

$$\boldsymbol{C}_G^T \boldsymbol{C}_G = \boldsymbol{I} + \boldsymbol{A}_{G^*}, \tag{8b}$$

where $\boldsymbol{D}_G$ is the degree matrix of G, and $\boldsymbol{I}$ is the identity matrix.

The following theorem demonstrate that in almost all the cases, the adjoint graph can preserve all the structure information of the original graph.

[2]Note that similarly with the setting in (Zhou & De la Torre, 2012), $\boldsymbol{A}_G$ is a binary matrix, even for weighted graphs. The weights are treated as edge attributes.

Theorem 1 *(Whitney, 1992) Two graphs are isomorphic if and only if their adjoint graphs are isomorphic, with the exception of the triangle graph K_3 and the claw $K_{1,3}$.*

Therefore, the edge embedding can still encode rich topological structure of graphs.

6.3 APPENDIX C: COMPLEXITY ANALYSIS OF SAGE

Now we analyze the total time complexity of embedding N graphs with n nodes and m edges. i) It respectively takes $O(Nn^3 + NSn^2)$ and $O(Nm^3 + NSm^2)$ to compute the $S-$dimensional RPFs of all original graphs and those of all corresponding adjoint graphs (Zhang et al., 2018). ii) It respectively takes $O(NRn^2)$ and $O(NRm^2)$ to get $\vec{\boldsymbol{Z}}^N(G)$ and $\vec{\boldsymbol{Z}}^E(G)$, since the time complexity of computing the energy distance is quadratic. iii) It takes $O(NR)$ to fuse them together. Note that the main cost comes from the node/edge embedding. Fortunately, in (Zhang et al., 2018), the authors introduced a Monte-Carlo method, which can significantly reduce the complexity.

6.4 APPENDIX D: EXPERIMENTAL SETTINGS

Datasets. To fully investigate the capability of SAGE to learn topology structure of graphs as well as different node and edge information, we evaluate SAGE on four types of graph benchmark datasets Kersting et al. (2016). i) Graphs without node labels (non-attributed): COLLAB, IMDB-BINARY, IMDB-MULTI, REDDIT-BINARY, REDDIT-MULTI(5K), and REDDIT-MULTI(12K) are derived from social networks; ii) Graphs with node labels (discrete attributes): ENZYMES, PROTEINS, and DD are derived from proteins. MUTAG, PTC-FM, PTC-FR, PTC- MM, PTC-MR, and NCI1 are derived from chemical compouds; iii) Graphs with both node labels and node attributes: PROTEINS_full are proteins and BZR, COX2, and DHFR are chemical compounds; iv) Graphs with both node and edge information (labels and/or attributes): BZR_MD, COX2_MD, DHFR_MD, ER_MD, and AIDS are derived from chemical compounds. Cuneiform is derived from the hand writing.

Setup. In our experiments, we directly employ a linear SVM implemented in LIBLINEAR Fan et al. (2008) since SAGE is a graph-level embedding. The cost parameter C is selected from $\{10^{-3}, 10^{-2}, 10^{-1}, 1, 10^1, 10^2, 10^3\}$. We set the node embedding size $S = 50$ and select R in the range of $[4, \min(2^k \leq N, 2048)]$, where N the size of the graph dataset. The ranges of hyper-parameters γ and $D_{\max}$ are $\{10^{-3}, 10^{-2}, 10^{-1}, 1, 10^1\}$ and [3:3:30], respectively. All parameters of the SVM and hyperparameters of our method were optimized only on the training dataset. For all these datasets, we perform 10-fold cross-validation to evaluate the performance of SAGE, using 9 folds for training and 1 for testing. To eliminate the random effects, we repeat the experiments ten times (thus 100 runs per dataset) and report the averaged prediction accuracies and standard deviations.

Baselines. For performance comparison on graph classification, we compare SAGE with many state-of-the-art graph kernels as well as graph neural networks.
Graph kernels. We use the following kernel baselines: i) Weisfeler-Leman Graph Kernel (WL) Shervashidze et al. (2011), ii) Shortest Path Kernel (SP) Borgwardt & Kriegel (2005), iii) Graphlet Kernel (GK) Shervashidze et al. (2009), iv) Subgraph matching kernels for attributed graphs (CSM) Kriege & Mutzel (2012), v) Hashing Weisfeiler-Lehman graph kernels for attributed graphs (HGK-WL) Morris et al. (2016), vi) GraphHoppers kernels for attributed graphs (GH) Feragen et al. (2013), vii) Graph invariant kernels for attributed graphs (GIK) Orsini et al. (2015), viii) Propagation kernels for attributed graphs (P2K) Neumann et al. (2016), ix) Graph kernels based on return probabilities of random walks (RetGK) Zhang et al. (2018).

GNN-based methods. We use the following GNN-based baselines: i) Deep Graphlet Kernel (DGK) Yanardag & Vishwanathan (2015), ii) PATCHY-SAN convolutional neural network (PSCN) Niepert et al. (2016), iii) Deep Graph Convolutional Neural Networks (DGCNN) Wu et al. (2017), iv) Hierarchical graph representation learning with differentiable pooling (DIFPOOL) Ying et al. (2018). The first model is based on Word2Vec model while the other three are built on convolutional neural networks. For all baselines (except RetGK), we directly quote the best reported values from their papers. Since RetGK, which is built on the same node embeddings, has shown to have the best performance compared to other baselines Zhang et al. (2018), the accuracies and computational time of RetGK are obtained from our own experiments on the same execution environment.

6.5 APPENDIX E: ABLATION STUDY OF SAGE

Table 5: Ablation Study of SAGE for classification accuracy (in %) on graphs with node and edge information

Datasets	SAGE-plain	SAGE-node	**SAGE-node-edge**
BZR_MD	62.0± 1.2	69.3± 0.7	**70.7± 0.7**
COX2_MD	50.3± 0.9	63.3± 1.0	**67.0 ± 0.9**
DHFR_MD	67.2± 0.6	69.2± 0.7	**70.5 ± 0.8**
ER_MD	66.6± 0.8	73.6± 0.7	**75.7 ± 0.9**
Cuneiform	7.7± 0.4	37.3± 0.9	**45.8 ± 1.1**
AIDS	99.4 ± 0.3	**99.5 ± 0.3**	**99.5 ± 0.3**

Finally, to pursue the answer for question **Q4**, we perform ablation study of SAGE on graph datasets that have both node labels (attributes) and edge labels (attributes). Table 5 shows the classification accuracies of three variants of SAGE: i) SAGE-plain without considering any attribute information, ii) SAGE-node considering only node label and/or node attributes, and iii) SAGE-node-edge considering both node and edge information. We can observe that SAGE-node-edge consistently performs better than both SAGE-node and SAGE-plain with significant improvement. It can be easily seen that the improvements that SAGE-node gains over SAGE-plain are larger than the improvements that SAGE-node-edge gains over SAGE-node. We suspect that this is because the node labels/attributes typically carry more important graph information. But anyway, the edge labels/attributes can also provide important information since the improvement of SAGE-node-edge over SAGE-node is significant.