Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI-18)

Efficient Attributed Network Embedding via Recursive Randomized Hashing

Wei Wu[1], **Bin Li**[2], **Ling Chen**[1], **Chengqi Zhang**[1]
[1] Centre for Artificial Intelligence, University of Technology Sydney, Australia
[2] School of Computer Science, Fudan University, China
william.third.wu@gmail.com, libin@fudan.edu.cn, {ling.chen, chengqi.zhang}@uts.edu.au

Abstract

Attributed network embedding aims to learn a low-dimensional representation for each node of a network, considering both attributes and structure information of the node. However, the learning based methods usually involve substantial cost in time, which makes them impractical without the help of a powerful workhorse. In this paper, we propose a simple yet effective algorithm, named NetHash, to solve this problem only with moderate computing capacity. NetHash employs the randomized hashing technique to encode shallow trees, each of which is rooted at a node of the network. The main idea is to efficiently encode both attributes and structure information of each node by recursively sketching the corresponding rooted tree from bottom (i.e., the predefined highest-order neighboring nodes) to top (i.e., the root node), and particularly, to preserve as much information closer to the root node as possible. Our extensive experimental results show that the proposed algorithm, which does not need learning, runs significantly faster than the state-of-the-art learning-based network embedding methods while achieving competitive or even better performance in accuracy.

1 Introduction

The surge of real-world network data, e.g., citation networks and social networks, has fostered network mining research. An important research that underpins many high-level applications is to represent the network by embedding each node in the network into a low-dimensional space. Based on the simplified network representation (i.e, network embedding), one can conduct node classification [Bhagat *et al.*, 2011] and link prediction [Liben-Nowell and Kleinberg, 2007], etc. In some real-world scenarios, high efficiency is strongly required with precisions guaranteed. For example, given an evolving social network, if we aim to retrieve the most similar users w.r.t. a queried user (e.g., for user search or friend recommendation), network representation should be performed very quickly to return precise results. Therefore, it is necessary to develop efficient and effective network embedding algorithms.

Recently, various network embedding algorithms have been proposed, including *plain network embedding* methods with only structure information preserved [Perozzi *et al.*, 2014; Grover and Leskovec, 2016], and *attributed network embedding* approaches with both structure and content information captured [Huang *et al.*, 2017a; 2017b]. However, plain network embedding methods may not satisfy accuracy because content information is simply ignored, while attributed network embedding methods usually involve inefficient learning procedures due to massive matrix operations.

To preserve content and structure information and dramatically reduce the computational cost in situations without powerful workhorses (e.g, query in mobile devices), one possible solution is to employ randomized hashing techniques [Broder *et al.*, 1998; Gionis *et al.*, 1999; Chi *et al.*, 2014; Wu *et al.*, 2016; 2017; Yang *et al.*, 2017], which can efficiently sketch high-dimensional data and map them into a fixed number of dimensions as an estimator. Some hashing algorithms have been used to characterize substructures in the graph, e.g., [Gibson *et al.*, 2005] discovers large dense subgraphs. By contrast, in [Li *et al.*, 2012], the graph is represented as a feature vector, where each dimension comprises a set of substructures, for classification by employing the hashing techniques to approximately count substructures.

Although there have been some research using randomized hashing techniques to approximately count substructures in a graph as features, to our knowledge there is no reported work using randomized hashing for graph embedding (node representation). In this paper, we represent each node of a graph as a shallow rooted tree through expanding its neighboring nodes, and then adopt the Locality-Sensitive Hashing (LSH) algorithm to sketch level-wise content of the rooted tree from bottom (the predefined highest-order neighboring nodes) to top (i.e., the root node). In particular, to summarize multiple attributes at each level of the rooted tree, MinHash, which is a well-known randomized LSH scheme in the bag-of-words model, is recursively employed. In the recursive operation of LSH along the rooted tree, we take into account the law of network information diffusion, i.e., exponential decay from the information source to the distance. This implies that the proposed recursive randomized hashing algorithm should be able to discount the content information level by level in the course of propagation. We name the resulting algorithm for attributed network embedding NetHash, which can encode

the attributes and structure information of the attributed network for each node using a low-dimensional discrete vector.

We provide theoretical analysis of the estimator of the similarity between two rooted trees, and also empirically evaluate effectiveness and efficiency of NetHash and the state-of-the-art methods on a number of real-world citation network and social network data sets for the tasks of multi-class node classification, link prediction and large-scale node query. In summary, our contributions are three-fold:

1. This work is the first endeavor to introduce randomized hashing techniques into attributed network embedding.
2. We propose a novel non-learning attributed network embedding algorithm named NetHash, which can efficiently preserve content and structure information for each node by expanding the node and its neighboring nodes into a rooted tree and recursively sketching the rooted tree from bottom to top.
3. The experimental results show that NetHash can achieve competitive or better accuracy with significantly reduced computational cost, compared to the state-of-the-art learning-based network embedding methods.

The remainder of the paper is organized as follows: Section 2 reviews related work of network embedding and graph hashing. Section 3 introduces the attributed network embedding problem. We describe NetHash in Section 4. The experimental results are presented in Section 5. Finally, we conclude this paper in Section 6.

2 Related Work

2.1 Network Embedding

Current network embedding approaches consist of plain network embedding and attributed network embedding [Zhang *et al.*, 2017]. The former describes each node by considering only structure information in the network, while the latter preserves both content and structure information.

Inspired by Skip-Gram [Mikolov *et al.*, 2013], DeepWalk [Perozzi *et al.*, 2014] and node2vec [Grover and Leskovec, 2016] perform random walks in the network so that the nodes in the random walks can capture contextual structure information. The former uniformly samples nodes from neighbors in the random walk, while the latter non-uniformly does. On the other hand, The first-order proximity composed of edges between nodes preserves local structure information, and the second-order proximity shows that nodes with shared neighbors are similar and captures global structure information. LINE [Tang *et al.*, 2015] explicitly defines an objective function to capture the two categories of structure information.

To incorporate content information, researchers have successively proposed attributed network embedding algorithms. TADW [Yang *et al.*, 2015], as the first approach to attributed network embedding, injects texts into matrix factorization by proving that DeepWalk is equivalent to matrix factorization. Based on TADW, HSCA [Zhang *et al.*, 2016] adds information of the first-order proximity to improve network representation. In addition to content and structure information, one node may have different contexts when interacting with different neighboring nodes. To this end, CANE [Tu *et al.*, 2017] adopts deep learning to capture such contexts.

2.2 Graph Hashing

Although most randomized hashing algorithms are designed for vectors or sets, some works aim to capture structure information by characterizing substructures in the graph. In [Gibson *et al.*, 2005], large dense subgraphs in massive graphs are identified by a variant of MinHash, where each hash function is generalized to return multiple elements to capture more neighboring information. In [Becchetti *et al.*, 2010], a slight modification of MinHash, which accelerates the standard method, is used to estimate the local number of triangles in large graphs. Some works adopt two random hashing schemes: In [Chi *et al.*, 2013], the first hashing scheme compresses a graph into a small graph, and the second one maps unlimitedly emerging cliques into a fixed-size clique set.

[Li *et al.*, 2012; Wu *et al.*, 2018] hash tree structures in the graph and approximately count substructures. Consequently, the graph is represented as a feature vector for classification, where each dimension comprises a set of substructures and the value of each dimension is related to the estimated number of the substructures. These algorithms cannot embed network nodes into a low-dimensional space, thus are not designed for node representation in our problem setting.

3 Problem Definition

Given an attributed network $G = (\mathcal{V}, \mathcal{E}, f)$, where $\mathcal{V}$ denotes nodes of the network, $\mathcal{E}$ denotes undirected edges of the network, and $f : \mathcal{V} \mapsto \mathcal{A}$ is a function that assigns attributes from an attribute set (or a global vocabulary) $\mathcal{A}$ to the nodes. Naturally, a node $v \in \mathcal{V}$, which owns attributes $f(v)$, can be represented as a 0/1 feature vector $\mathbf{v}_v$ where each dimension represents an attribute in $\mathcal{A}$. Figure 1(a) shows a toy example where an attributed network is composed of 6 nodes, 8 edges and 10 attributes. Each node is represented as a 10-dimensional 0/1 feature vector (bottom right). Given a network G, we aim to embed each node $\mathbf{v}_v \in \{0,1\}^{|\mathcal{A}|}$ into a low-dimensional vector $\mathbf{x}_v \in \mathbb{R}^K$, where $K \ll |\mathcal{A}|$[1].

4 NetHash

In this section, we first introduce MinHash, and then propose the NetHash algorithm for attributed network embedding.

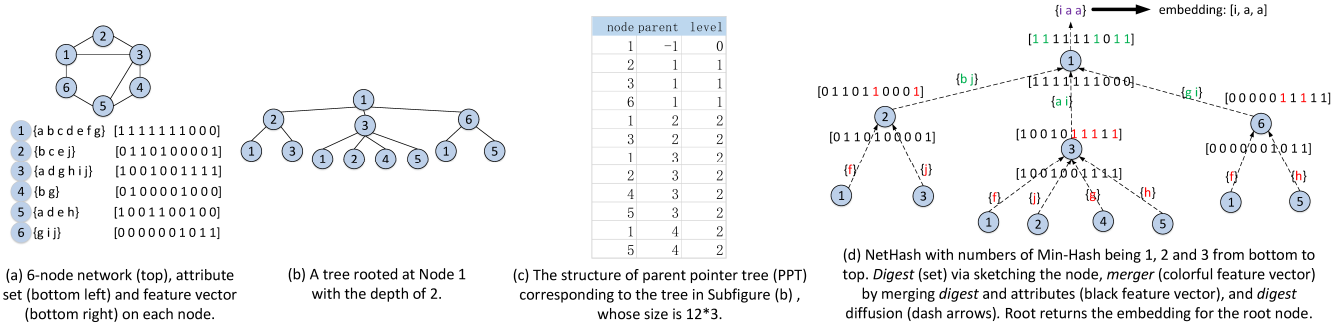
4.1 The MinHash Scheme

Given a universal set $\mathcal{U}$ and a subset $\mathcal{S} \subseteq \mathcal{U}$, MinHash is generated: Assuming a set of K random permutations, $\{\pi_k\}_{k=1}^K$, where $\pi_k : \mathcal{U} \mapsto \mathcal{U}$, are applied to $\mathcal{U}$, elements in $\mathcal{S}$ which lie in the first position of each permutation, $\{\min\big(\pi_k(\mathcal{S})\big)\}_{k=1}^K$, will be the MinHashes of $\mathcal{S}$ [Broder *et al.*, 1998].

MinHash is an approximate algorithm to compute the Jaccard similarity of two sets. It is proved that the probability of two sets, $\mathcal{S}_1$ and $\mathcal{S}_2$, to generate the same MinHash value is exactly equal to the Jaccard similarity of the two sets:

$$\Pr\Big(\min\big(\pi_k(\mathcal{S}_1)\big) = \min\big(\pi_k(\mathcal{S}_2)\big)\Big) = J(\mathcal{S}_1, \mathcal{S}_2) = \frac{|\mathcal{S}_1 \cap \mathcal{S}_2|}{|\mathcal{S}_1 \cup \mathcal{S}_2|}.$$

[1]Most algorithms embed nodes into l_2 space, while we embed nodes into l_1 space, as in [Gionis *et al.*, 1999].



(a) 6-node network (top), attribute set (bottom left) and feature vector (bottom right) on each node.

(b) A tree rooted at Node 1 with the depth of 2.

(c) The structure of parent pointer tree (PPT) corresponding to the tree in Subfigure (b) , whose size is 12*3.

(d) NetHash with numbers of Min-Hash being 1, 2 and 3 from bottom to top. *Digest* (set) via sketching the node, *merger* (colorful feature vector) by merging *digest* and attributes (black feature vector), and *digest* diffusion (dash arrows). Root returns the embedding for the root node.

Figure 1: An illustration of embedding a node (recursively sketching a rooted tree) on a network.

We can conduct multiple independent random permutations for unbiasedly estimating the similarity:

$$\hat{J}(\mathcal{S}_1, \mathcal{S}_2) = \frac{\sum_{k=1}^{K} \mathbf{1}\Big(\min\big(\pi_k(\mathcal{S}_1)\big) = \min\big(\pi_k(\mathcal{S}_2)\big)\Big)}{K}.$$

However, the explicit random permutations are expensive. In practice, a hash function as follows can be used to generate the permutated index $\pi_k(i) = \texttt{mod}\big((a_k i + b_k), c_k\big)$, where i is the index of an element from the universal set $\mathcal{U}$, $0 < a_k, b_k < c_k$ are two random integers and c_k is a big prime number such that $c_k \geq |\mathcal{U}|$ [Broder *et al.*, 1998].

4.2 The NetHash Algorithm

To simultaneously capture content and structure information in the network, we expand each node along with its neighboring nodes into a rooted tree. By recursively sketching the rooted tree from bottom to top, NetHash diffuses content information level by level. To this end, we employ the structure of *parent pointer tree* (PPT) to store the tree such that each node of the tree is visited only once. Figure 1(b) shows a tree rooted at Node 1 with depth 2. The 1st level has Nodes 2, 3 and 6, and the 2nd level has Nodes $\{1, 3\}$, $\{1, 2, 4, 5\}$, and $\{1, 5\}$. Figure 1(c) shows the corresponding PPT, which is a 12×3 array: The first column shows node identifiers on the network; the second column points to the parent node in the array, e.g., "-1" means that root Node 1 has no parent node and "1" in the 2nd row (i.e., $[2, 1, 1]$) shows that the parent node of Node 2 is in the 1st row (i.e., Node 1); the third column is the level where the node is located.

We outline NetHash in Algorithm 1. NetHash processes nodes one by one (Line 1). We first build the PPT to store the rooted tree whose root node is currently being processed (Line 2). Also, we introduce an auxiliary queue Q to store the currently visited node in the rooted tree and the corresponding diffused content information (Line 3). Subsequently, NetHash traverses the PPT from end to head (Lines 4-12). The recursive sketching process consists of two steps. Sketching a node produces a set of MinHashes called *"digest", which represents the content information diffused from the node to its parent node*, e.g., $\{b, j\}$ produced by sketching Node 2 in Figure 1(d). Actually, the step summarizes content information of each node. In the while loop, an *internal node* `merges` its own attributes with all digests, which are generated by sketching its all child nodes, to produce the

Algorithm 1 The NetHash Algorithm

Input: $G = (\mathcal{V}, \mathcal{E}, f)$; number of embedding dimensions K; entropy of degrees of network S; depth of tree D [1]; hash functions at the l-th level $\{\pi_k^{(l)}\}_{l=0,k=1}^{D,k_l}$

Output: G's embedding $\mathbf{h}$

1: **for** $r = 1, \ldots, |\mathcal{V}|$ **do**
2: Build a parent pointer tree $\mathbf{T}$ for node r;
3: Initialize an empty auxiliary queue Q;
4: **for** $v \in \mathbf{T} \uparrow$ **do**
5: $l \leftarrow$ level of v in $\mathbf{T}$;
6: $merger \leftarrow f(v)$; // initial merger from attributes on v
7: **while** Q is not empty and v is the parent node of $Q[0]$ in $\mathbf{T}$ **do**
8: $merger \leftarrow \texttt{merge}(Q.\texttt{pop()}.digest, merger)$;
9: **end while**
10: $digest \leftarrow \texttt{MinHash}(merger, \{\pi_k^{(l)}\}_{k=1}^{k_l})$;
11: $Q.\ \texttt{push}(\{digest, v\})$;
12: **end for**
13: $\mathbf{h}(r) \leftarrow Q.\texttt{pop()}.digest$;
14: **end for**

"merger", whose entries of 1 correspond to attributes or digests, e.g., $[0\,1\,1\,0\,1\,1\,0\,0\,0\,1]$ on Node 2 in Figure 1(d). Consequently, the *internal node* carries both its own and diffused content information. Essentially, this step preserves structure information by diffusing and merging because the structure (i.e., edges) is reflected by digests. Note that no `merging` operation occurs on *leaf nodes*. Finally, the root node returns its representation (Line 13).

4.3 Exponentially Decayed Diffusion

We build a framework of capturing content and structure information by recursively sketching the rooted tree. Intuitively, the lower-order neighboring nodes contribute more to representation of the root node than the higher-order ones do, which implies that NetHash should discount content information level by level in the course of propagation. To this end, we combine the recursive operation of the MinHash algorithm with the law of network information diffusion, i.e., exponential decay from the information source to the distance. Natually, the number of MinHashes, k_l, which is the amount of digest produced from a node at the l-th level, satisfies:

$$k_l = \max\{1, \texttt{int}\big(K \exp(-l\lambda)\big)\}, \tag{1}$$

[1] Note that $D = 0$ indicates that the tree has only one root node.

where K is the number of embedding dimensions, i.e., the number of MinHashes at the root node (its level is 0 and $k_0 = K$), λ is the exponential decay rate, and $\texttt{int}$ is a rounding function. If $k_l < 1$, one attribute of digest will be diffused.

Content information is diffused along edges, so information diffusion is closely related to node degrees. If treating node degrees in a network as a distribution, we can define the entropy of node degrees, S, from the perspective of information theory: $S = -\sum_{\nu \in \texttt{unique}(\{\nu_v\}_{v\in\mathcal{V}})} \Pr(\nu) \log \Pr(\nu)$, where ν_v is degree of node v and $\texttt{unique}(\{\nu_v\}_{v\in\mathcal{V}})$ is the set of unique degrees observed in the network. A large entropy suggests that the network tends to have evenly distributed degrees, which implies that content information diffused in the network is unstable since the amount of diffused content (in our case the "digest") varies widely from node to node. For example: Given a network containing 100 nodes, node degrees vary from 1 to 5, and the number of the nodes corresponding to each node degree, if evenly distributed, is 20. In this case, the amount of digest is different from node to node. The total amount of digest received by each node remarkably changes due to different degrees. If node degrees are not that evenly distributed and are more concentrated, the total amount of digest will tend to be consistent. To address the issue, such a network ought to diffuse little digest by accelerating the decay of diffusion. To this end, we use entropy as the decay rate and Eq. (1) is rewritten as

$$k_l = \max\{1, \texttt{int}\big(K \exp(-lS)\big)\}. \tag{2}$$

4.4 Theoretical Analysis

Similarity: Given two rooted trees, $\mathcal{A}^{(D)}$ and $\mathcal{B}^{(D)}$, where D is the depth of the trees, their similarity is:

$$J(\mathcal{A}^{(D)}, \mathcal{B}^{(D)}) = \mathbb{E}_{\pi^{(0)},\cdots,\pi^{(D)}}\Big[\mathbf{1}\big(\rho(\mathcal{A}) = \rho(\mathcal{B})\big)\Big],$$

$$\rho(\mathcal{X}) = \pi^{(0)}\Big(f(\mathcal{X}_*^{(0)}) \cup \pi^{(1)}\big(\cdots f(\mathcal{X}_*^{(D-1)}) \cup \pi^{(D)}(f(\mathcal{X}_*^{(D)}))\big)\Big),$$

where $\mathcal{X} \in \{\mathcal{A}, \mathcal{B}\}$, $f(\mathcal{X}_*^{(d)})$ and $\pi^{(d)}(f(\mathcal{X}_*^{(d)}))$ are attributes of all nodes at the d-th level and the digest sets produced from the corresponding nodes by MinHash $\pi^{(d)}$ at the d-th level, respectively, and the number of $\pi^{(0)}$ is K.

Bounds: We give bounds between the real similarity $J(\mathcal{A}^{(D)}, \mathcal{B}^{(D)})$ and the estimated similarity $\hat{J}(\mathcal{A}^{(D)}, \mathcal{B}^{(D)})$.

Theorem 1 *Given $K > 2\delta^{-2}s^{-1}\log\epsilon^{-1}$, $0 < \delta < 1$ and $\epsilon > 0$, for two rooted trees, $\mathcal{A}^{(D)}$ and $\mathcal{B}^{(D)}$, we have bounds:*

1. *If $J(\mathcal{A}^{(D)}, \mathcal{B}^{(D)}) \geq s$, then $\hat{J}(\mathcal{A}^{(D)}, \mathcal{B}^{(D)}) \geq (1-\delta)s$ with a probability at least $1-\epsilon$.*
2. *If $J(\mathcal{A}^{(D)}, \mathcal{B}^{(D)}) \leq s$, then $\hat{J}(\mathcal{A}^{(D)}, \mathcal{B}^{(D)}) \leq (1+\delta)s$ with a probability at least $1-\epsilon$.*

Proof 1 *Let $X_{\pi^{(0)},\cdots,\pi^{(D)}}$ be a random variable. $X_{\pi^{(0)},\cdots,\pi^{(D)}} = 1$ if $\rho(\mathcal{A}) = \rho(\mathcal{B})$, and 0 otherwise. Let $X = \sum_{k=1}^{K} X^{(k)}_{\pi^{(0)},\cdots,\pi^{(D)}}$. $\mathbb{E}_{\pi^{(0)},\cdots,\pi^{(D)}}[X^{(k)}_{\pi^{(0)},\cdots,\pi^{(D)}}] = J(\mathcal{A}^{(D)}, \mathcal{B}^{(D)}) \geq s$; $\mathbb{E}[X] \geq Ks$ as only K MinHash functions sketch the root node whatever D is. Based on the Chernoff bound [Chernoff, 1952], we have*

$$\Pr\big(X < (1-\delta)Ks\big) \leq \Pr\big(X < (1-\delta)\mathbb{E}(X)\big) \leq \mathrm{e}^{-\frac{\delta^2\mathbb{E}(X)}{2}} \leq \mathrm{e}^{-\frac{\delta^2 Ks}{2}} \tag{3}$$

Substituting Eq. (3) with $\hat{J}(\mathcal{A}^{(D)}, \mathcal{B}^{(D)}) = \frac{X}{K}$ gives

$$\Pr\big(\hat{J}(\mathcal{A}^{(D)}, \mathcal{B}^{(D)}) < (1-\delta)s\big) \leq \mathrm{e}^{-\frac{\delta^2 Ks}{2}} < \epsilon \tag{4}$$

By rearranging Eq. (4), we prove the first proposition for $K > 2\delta^{-2}s^{-1}\log\epsilon^{-1}$. Similarly, the second one is proved.

Complexity: The rooted tree can be averagely considered as a $\overline{\nu}$-ary tree, where $\overline{\nu}$ and $|\overline{\mathcal{A}}|$ denote the mean value of degrees of the network and the mean number of attributes of each node, respectively. Since NetHash traverses each node only once, the theoretical computational complexity is $O(|\mathcal{V}|\big(\sum_{d=0}^{D-1}(|\overline{\mathcal{A}}| + k_{d+1}\overline{\nu})k_d\overline{\nu}^d + |\overline{\mathcal{A}}|k_D\overline{\nu}^D\big))$. Considering the simplified version of Eq. (2), i.e., $k_d = K\mathrm{e}^{-dS}$, we have $O(|\mathcal{V}|\overline{\nu}^D K\mathrm{e}^{-DS}(\max\{|\overline{\mathcal{A}}|, K\mathrm{e}^{-(D-1)S}\}))$.

TADW and HSCA first spend $O(|\mathcal{V}|^2)$ in probability that each node randomly walks to any one in fixed steps; then in each iteration of the learning process, TADW costs at least $O\big(|\mathcal{V}|K^2 + |\mathcal{V}||\mathcal{A}|K\big)$, and HSCA does at least $O\big(|\mathcal{V}|K^2 + |\mathcal{V}||\mathcal{A}|K + |\mathcal{V}||\mathcal{A}|^2\big)$. In practice, we normally consider trees with $D \in \{1, 2, 3\}$ as the distant nodes can hardly diffuse effective attributes to the root node. Besides, in most networks, $\overline{\nu}, \overline{\nu}^D \ll |\mathcal{V}|$ due to sparsity, $|\overline{\mathcal{A}}|, K \ll |\mathcal{A}|$. Therefore, the time complexity of NetHash is practically efficient, especially in large-scale networks since it is linear with respect to $|\mathcal{V}|$.

Data Set	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{E}\rvert$	$\overline{\nu}$	$\lvert\mathcal{A}\rvert$	$\lvert\overline{\mathcal{A}}\rvert$	S	$\lvert\mathcal{L}\rvert$
Cora	2,708	5,429	3.8	1,433	18.17	2.11	7
Wikipedia	2,405	17,981	9.08	4,973	673.31	3.06	19
Flickr	7,575	239,738	63.29	12,047	24.13	4.81	9
BlogCatalog	5,196	171,743	66.11	8,189	71.10	4.89	6
ACM	1,108,140	6,121,261	11.05	586,190	59.09	3.26	-

$|\mathcal{V}|$: number of nodes; $|\mathcal{E}|$: number of edges; $\overline{\nu}$: average node degrees; $|\mathcal{A}|$: size of the attribute set; $|\overline{\mathcal{A}}|$: average number of node attributes; S: entropy of node degrees; $|\mathcal{L}|$: number of node labels.

Table 1: Summary of the five network data sets.

5 Experimental Results

In this section, we evaluate the performance of NetHash via node classification, link prediction and node retrieval.

The state-of-the-art methods: (1) DeepWalk [Perozzi *et al.*, 2014]: It captures contextual structure information based on random walks; (2) node2vec [Grover and Leskovec, 2016]: It performs a biased DeepWalk to explore diverse neighbors; (3) TADW [Yang *et al.*, 2015]: It learns node representations by combining attributes with structure in matrix factorization; (4) HSCA [Zhang *et al.*, 2016]: It adds the first-order proximity to TADW; (5) CANE [Tu *et al.*, 2017]: Besides attribute and structure information, it captures different contexts which a node expresses for different neighbors.

Data sets: (1) Cora [Yang *et al.*, 2015]: A citation network of machine learning papers. (2) Wikipedia [Yang *et al.*, 2015]: A citation network of articles in Wikipedia. (3) Flickr [Li *et al.*, 2015]: The network consists of users as nodes, following relationship as edges and interest tags of users as attributes. (4) BlogCatalog [Li *et al.*, 2015]: The

Data	Algorithms	Micro-F1(%)					Macro-F1(%)					Runtime(s)
		Training Ratio					Training Ratio					
		50%	60%	70%	80%	90%	50%	60%	70%	80%	90%	
Cora	DeepWalk	78.83	79.68	80.29	80.78	81.20	77.63	78.48	79.08	79.55	79.83	436.39
	node2vec	81.22	81.98	82.68	83.09	83.52	80.14	80.91	81.61	81.99	82.28	24.01
	TADW	85.74	86.17	86.46	86.71	86.78	84.35	84.71	85.23	85.25	85.26	66.54
	HSCA	85.70	85.79	85.97	86.25	86.38	84.15	84.25	84.41	84.62	84.82	72.17
	CANE	**85.78**	86.13	86.70	**87.27**	**87.32**	84.41	84.73	85.26	85.57	85.86	5622.72
	NetHash	85.65	**86.22**	**86.74**	87.13	87.31	**84.80**	**85.42**	**85.93**	**86.34**	**86.43**	**0.4**
Wikipedia	DeepWalk	63.71	64.63	65.33	65.83	66.55	50.63	51.60	52.28	52.47	52.11	355.56
	node2vec	63.70	64.61	65.31	65.91	66.36	50.78	51.53	52.22	52.49	52.04	24.81
	TADW	**75.58**	**76.06**	76.59	76.96	77.20	**59.39**	**60.23**	**61.10**	**61.57**	60.90	90.97
	HSCA	72.35	72.61	72.71	72.82	73.05	56.57	56.93	57.19	57.01	56.40	103.96
	CANE	72.62	73.27	73.91	74.36	74.71	57.30	58.23	58.97	59.02	58.28	10878.74
	NetHash	74.12	75.49	**76.64**	**77.51**	**78.12**	55.10	57.45	59.39	61.00	**61.28**	**2.00**

Table 2: Node classification results on citation networks.

Data	Algorithms	Training Ratio									
		50%		60%		70%		80%		90%	
		AUC	Runtime(s)	AUC	Runtime(s)	AUC	Runtime(s)	AUC	Runtime(s)	AUC	Runtime(s)
Flickr	DeepWalk	31.72	1144.68	32.08	1150.99	32.83	1156.30	33.07	1156.31	34.33	1151.60
	node2vec	17.98	289.60	16.62	354.92	16.70	423.26	17.52	479.61	18.95	528.56
	TADW	64.40	365.34	65.85	392.96	66.50	417.07	67.05	443.16	67.79	477.08
	HSCA	55.35	387.11	54.54	412.93	54.03	456.80	54.03	498.20	53.25	497.93
	NetHash	**85.07**	**0.60**	**85.19**	**0.70**	**85.54**	**0.80**	**85.79**	**1.10**	**85.44**	**1.20**
BlogCatalog	DeepWalk	**77.13**	822.70	**77.89**	819.44	**78.56**	824.86	**78.76**	831.62	**79.36**	845.66
	node2vec	56.22	92.51	61.29	109.65	66.65	125.15	70.72	141.07	74.47	159.97
	TADW	68.52	213.70	69.51	219.51	70.50	226.92	71.28	218.44	71.94	221.91
	HSCA	50.23	209.94	50.25	221.68	50.30	267.34	49.85	277.38	50.35	302.77
	NetHash	69.07	**0.60**	69.83	**0.80**	72.16	**0.90**	72.51	**1.00**	74.26	**1.10**

*Note that in the experiment, each algorithm is given a cutoff time of 100,000 seconds, and CANE is forced to stop within the cutoff time.

Table 3: Link prediction results on social networks.

network consists of bloggers as nodes, following relationship as edges and keywords in blog as attributes. (5) ACM [Tang *et al.*, 2008]: The original data contains 2,381,688 ACM papers and 10,476,564 citation relationship. After cleaning up papers without abstracts or citations, we build a citation network with papers as nodes, citation as edges and abstract as attributes. The data sets are summarized in Table 1.

Experimental Settings: For all methods, we set the embedding dimension $K = 200$, as in TADW and CANE. All compared algorithms are implemented by the authors and their parameters are set to default values. For two nodes v_1 and v_2, NetHash generates the representations of length K, $\mathbf{x}_{v_1}$ and $\mathbf{x}_{v_2}$, respectively. The Jaccard similarity between $\mathbf{x}_{v_1}$ and $\mathbf{x}_{v_2}$ is $\mathbf{Sim}_{\mathbf{x}_{v_1},\mathbf{x}_{v_2}} = \frac{\sum_{k=1}^{K} \mathbf{1}(x_{v_1,k}=x_{v_2,k})}{K}$. The runtime of NetHash consists of generating hash functions, constructing and sketching the rooted trees. All experiments are conducted on a node of Linux Cluster with 8×3.4 GHz Intel Xeon CPU (64 bit) and 32GB RAM.

5.1 Node Classification on Citation Networks

We report classification performance on Cora and Wikipedia. In the task, we adopt LIBSVM [Chang and Lin, 2011] for NetHash which uses the Jaccard similarity matrix as the precomputed kernel, and LIBLINEAR [Fan *et al.*, 2008] for the compared algorithms[2], as used in their papers. We vary the training ratio (i.e., percentage of nodes as the training set) in $\{50\%, 60\%, 70\%, 80\%, 90\%\}$, for each ratio of which we repeat the experiment 10 times and average the results.

Table 2 reports the experimental results in accuracy (Micro-F1 and Macro-F1) and runtime. NetHash defeats all plain network embedding algorithms, and achieves the competitive and even better accuracy than all attributed network embedding ones. In runtime, NetHash presents the strong advantage: it performs more efficiently than all compared methods. Generally, NetHash outperforms TADW and HSCA by one to two orders of magnitudes, and CANE by four orders.

5.2 Link Prediction on Social Networks

We conduct link prediction task on Flickr and BlogCatalog. NetHash uses the Jaccard similarity in l_1 space while the compared methods does the Euclidean distance in l_2 space (the same in Section 5.3). A training network is obtained by randomly preserving a training ratio of edges, while the removed edges act as the unobserved (test) links. Based on the training network, we compute similarity between each pair of nodes. Then, we randomly pick an unobserved link and a nonexistent one to compare their similarities. AUC values represent the probability that a randomly selected unobserved link is more similar than a randomly selected nonexistent one by repeating the picking operation 10,000 times. We compute AUC values 10 times and average the results.

Table 3 shows the experimental results of AUC and runtime. NetHash significantly outperforms the compared methods in AUC and runtime on Flickr, and it defeats all attributed network embedding algorithms on BlogCatalog. NetHash exhibits superiority in runtime – it performs much more effi-

[2]We test the compared methods on LIBSVM. DeepWalk, node2vec and CANE achieve similar accuracy while TADW and HSCA perform worse, so we just report results on LIBLINEAR.

Query: Genetic Algorithms in Search, Optimization and Machine Learning	**Runtime(s)**
DeepWalk	19,087
1. Dynamic Identification of Inelastic Shear Frames by Using Prandtl-Ishlinskii Model	
2. Application of Nontraditional Optimization Techniques for Airfoil Shape Optimization	
3. Genetic Algorithms and Neural Networks in Optimal Location of Piezoelectric ...	
4. Trajectory Controller Network and Its Design Automation Through Evolutionary ...	
5. Flow Restrictor Design for Extrusion Slit Dies for a Range of Materials: Simulation ...	
NetHash	**723**
1. Neural Networks: A Comprehensive Foundation	
2. Machine Learning	
3. A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II	
4. Introduction to Algorithms	
5. An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods	

*Note that in the experiment, each algorithm is given a cutoff time of 24 hours. TADW, HSCA and CANE are forced to stop because of overtime, while node2vec fails due to out of memory.

Table 4: Top-5 Query results on a large-scale citation network.

ciently than the compared methods by orders of magnitudes.

5.3 Case Study: Node Retrieval on a Large-scale Citation Network

In this case study, given 5 query papers on ACM that have top 5 highest citation numbers, we retrieve 5 most similar papers for each query based on their embeddings. Due to similar results and space constraint, we only report the result of the paper with the highest citation number (i.e., 6,294).

Table 4 presents the query result. All results retrieved by NetHash are closely related to algorithms and machine learning and particularly, it accurately returns one paper regarding the genetic algorithm (i.e, the third result). Moreover, from the semantic perspective, we find that the results, except the third one, and the query all focus on topics of AI and algorithms, while the papers retrieved by DeepWalk are more related to specific applications, which deviate from the query. In terms of runtime, NetHash is still very efficient when running on such a large-scale network data.

5.4 Discussion on the Results

In the above three tasks, NetHash generally achieves competitive and even better accuracy, because it preserves both content and structure information while DeepWalk and node2vec only capture structure infomration. Exceptionally, DeepWalk performs best on BlogCatalog, largely because it captures sufficient structure information by long random walks in the complex network (e.g., large average node degree and entropy). In terms of attributed network embedding algorithms, NetHash and TADW essentially preserve attributes and structure information, so they keep the same level in accuracy on Cora, Wikipedia and BlogCatalog; the reason why NetHash defeats TADW on Flickr is largely because the recursive operation of MinHash can more effectively capture structure information. Although HSCA and CANE additionally capture first-order proximity and contexts, respectively, NetHash still outperforms them on Wikipedia, Flickr and BlogCatalog because the additional information negatively impacts representation and in turn deteriorates the accuracy.

In terms of runtime, NetHash performs much more efficiently than DeepWalk and node2vec, because NetHash expands nodes into the trees with the depth being around 2 while DeepWalk and node2vec perform long random walks. We know from Section 4.4 that empirically, NetHash has lower time complexity than TADW and HSCA. CANE experimentally executes a substantial number of matrix operations in deep learning, which is costly in runtime and demands powerful workhorses. In addition, NetHash, as a randomized algorithm, avoids the iteration process in the learning-based algorithms, which is also an important factor of high efficiency.

		Micro-F1(%)					**Runtime(s)**
		Training Ratio					
		50%	60%	70%	80%	90%	
Depth (D)	1	85.41	85.99	86.55	86.98	87.28	0.1
	2	**85.65**	**86.22**	**86.74**	**87.13**	**87.31**	**0.4**
	3	85.12	85.93	86.55	86.75	87.00	0.8
Decay rate (λ)	0.5	85.11	85.99	86.55	86.98	87.28	6
	2.11	**85.65**	**86.22**	**86.74**	**87.13**	**87.31**	**0.4**
	5	85.12	85.93	86.55	86.75	87.00	0.2

Table 5: Parameter sensitivity of depth and decay rate on Cora.

5.5 Parameter Sensitivity

NetHash has two exclusive parameters, tree depth D and decay rate λ. Due to space constraint, we report only the node classification results on Cora.

Table 5 (top) shows accuracy of NetHash w.r.t. tree depth D. NetHash achieves the best accuracy on Cora when $D = 2$. The reason why the accuracy on Cora improves first when D varies from 1 to 2 is largely because more useful attributes are captured from higher-order neighboring nodes. However, when D further increases to 3, more noise will dominate in the diffusion process to deteriorate the performance, which generates the unsatisfying representation. Thus, tuning depth D is a tradeoff between noise and useful attributes, and the optimal value of D depends on the particular data. The runtime grows with D increasing. We set $D = 1$ for Wikipedia, Flickr and BlogCatalog, and $D = 2$ for Cora and ACM.

Table 5 (bottom) shows accuracy of NetHash w.r.t. exponential decay λ. The accuracy first increases and then declines. The best accuracy is achieved when λ is set to the entropy of node degrees S. A smaller λ implies more diffused information and thus, more noise; while a larger λ implies less diffused information, which is insufficient. The runtime decreases remarkably with λ increasing. Hence, we adopt the entropy of node degrees S as the decay rate.

6 Conclusion

In this paper, we propose an efficient attributed network embedding algorithm dubbed NetHash, which employs the randomized hashing technique to recursively sketch the shallow trees, each of which is rooted at a node of the network, from bottom to top, and preserves as much information closer to the root node as possible by simulating the exponential decay in network information diffusion.

We conduct extensive empirical tests of NetHash and the state-of-the-art methods on five network data sets. The experimental results show that NetHash not only achieves competitive or better performance but also performs much faster than the compared methods by orders of magnitude. In large-scale network analysis, NetHash can perform very well with a limited budget of computational capacity, which makes it more practical in the era of big data.

Acknowledgments

This work is partially supported by ARC Discovery Grant DP 180100966. Bin Li is supported by the Fudan University Startup Research Grant (SXH2301005) and Shanghai Municipal Science & Technology Commission (16JC1420401).

References

[Becchetti *et al.*, 2010] Luca Becchetti, Paolo Boldi, Carlos Castillo, and Aristides Gionis. Efficient Algorithms for Large-scale Local Triangle Counting. *ACM Transactions on Knowledge Discovery from Data*, 4(3):13, 2010.

[Bhagat *et al.*, 2011] Smriti Bhagat, Graham Cormode, and S Muthukrishnan. Node Classification in Social Networks. In *Social Network Data Analytics*, pages 115–148. 2011.

[Broder *et al.*, 1998] Andrei Z Broder, Moses Charikar, Alan M Frieze, and Michael Mitzenmacher. Min-wise Independent Permutations. In *STOC*, pages 327–336, 1998.

[Chang and Lin, 2011] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A Library for Support Vector Machines. *ACM Transactions on Intelligent Systems and Technology*, 2(3):1–27, 2011.

[Chernoff, 1952] Herman Chernoff. A Measure of Asymptotic Efficiency for Tests of a Hypothesis based on the Sum of Observations. *The Annals of Mathematical Statistics*, pages 493–507, 1952.

[Chi *et al.*, 2013] Lianhua Chi, Bin Li, and Xingquan Zhu. Fast Graph Stream Classification Using Discriminative Clique Hashing. In *PAKDD*, pages 225–236. Springer, 2013.

[Chi *et al.*, 2014] Lianhua Chi, Bin Li, and Xingquan Zhu. Context-preserving Hashing for Fast Text Classification. In *SDM*, pages 100–108, 2014.

[Fan *et al.*, 2008] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. LIBLINEAR: A Library for Large Linear Classification. *Journal of Machine Learning Research*, 9(Aug):1871–1874, 2008.

[Gibson *et al.*, 2005] David Gibson, Ravi Kumar, and Andrew Tomkins. Discovering Large Dense Subgraphs in Massive Graphs. In *VLDB*, pages 721–732, 2005.

[Gionis *et al.*, 1999] Aristides Gionis, Piotr Indyk, Rajeev Motwani, et al. Similarity Search in High Dimensions via Hashing. In *VLDB*, volume 99, pages 518–529, 1999.

[Grover and Leskovec, 2016] Aditya Grover and Jure Leskovec. node2vec: Scalable Feature Learning for Networks. In *KDD*, pages 855–864, 2016.

[Huang *et al.*, 2017a] Xiao Huang, Jundong Li, and Xia Hu. Accelerated Attributed Network Embedding. In *SDM*, pages 633–641, 2017.

[Huang *et al.*, 2017b] Xiao Huang, Jundong Li, and Xia Hu. Label Informed Attributed Network Embedding. In *WSDM*, pages 731–739, 2017.

[Li *et al.*, 2012] Bin Li, Xingquan Zhu, Lianhua Chi, and Chengqi Zhang. Nested Subtree Hash Kernels for Large-scale Graph Classification over Streams. In *ICDM*, pages 399–408, 2012.

[Li *et al.*, 2015] Jundong Li, Xia Hu, Jiliang Tang, and Huan Liu. Unsupervised Streaming Feature Selection in Social Media. In *CIKM*, pages 1041–1050, 2015.

[Liben-Nowell and Kleinberg, 2007] David Liben-Nowell and Jon Kleinberg. The Link-prediction Problem for Social Networks. *Journal of the Association for Information Science and Technology*, 58(7):1019–1031, 2007.

[Mikolov *et al.*, 2013] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed Representations of Words and Phrases and Their Compositionality. In *NIPS*, pages 3111–3119, 2013.

[Perozzi *et al.*, 2014] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. DeepWalk: Online Learning of Social Representations. In *KDD*, pages 701–710, 2014.

[Tang *et al.*, 2008] Jie Tang, Jing Zhang, Limin Yao, Juanzi Li, Li Zhang, and Zhong Su. ArnetMiner: Extraction and Mining of Academic Social Networks. In *KDD*, pages 990–998, 2008.

[Tang *et al.*, 2015] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. LINE: Large-scale Information Network Embedding. In *WWW*, pages 1067–1077, 2015.

[Tu *et al.*, 2017] Cunchao Tu, Han Liu, Zhiyuan Liu, and Maosong Sun. Cane: Context-Aware Network Embedding for Relation Modeling. In *ACL*, volume 1, pages 1722–1731, 2017.

[Wu *et al.*, 2016] Wei Wu, Bin Li, Ling Chen, and Chengqi Zhang. Canonical Consistent Weighted Sampling for Real-Value Weighted Min-Hash. In *ICDM*, pages 1287–1292, 2016.

[Wu *et al.*, 2017] Wei Wu, Bin Li, Ling Chen, and Chengqi Zhang. Consistent Weighted Sampling Made More Practical. In *WWW*, pages 1035–1043, 2017.

[Wu *et al.*, 2018] Wei Wu, Bin Li, Ling Chen, Xingquan Zhu, and Chengqi Zhang. K-Ary Tree Hashing for Fast Graph Classification. *IEEE Transactions on Knowledge and Data Engineering*, 30(5):936–949, 2018.

[Yang *et al.*, 2015] Cheng Yang, Zhiyuan Liu, Deli Zhao, Maosong Sun, and Edward Y Chang. Network Representation Learning with Rich Text Information. In *IJCAI*, pages 2111–2117, 2015.

[Yang *et al.*, 2017] Dingqi Yang, Bin Li, Laura Rettig, and Philippe Cudré-Mauroux. HistoSketch: Fast Similarity-Preserving Sketching of Streaming Histograms with Concept Drift. In *ICDM*, 2017.

[Zhang *et al.*, 2016] Daokun Zhang, Jie Yin, Xingquan Zhu, and Chengqi Zhang. Homophily, Structure, and Content Augmented Network Representation Learning. In *ICDM*, pages 609–618, 2016.

[Zhang *et al.*, 2017] Daokun Zhang, Jie Yin, Xingquan Zhu, and Chengqi Zhang. Network Representation Learning: A Survey. *arXiv preprint arXiv:1801.05852*, 2017.