NodeSketch: Highly-Efficient Graph Embeddings via Recursive Sketching

Dingqi Yang[1], Paolo Rosso[1], Bin Li[2] and Philippe Cudre-Mauroux[1]
[1]University of Fribourg, Switzerland, {firstname.lastname}@unifr.ch; [2]Fudan University, China, libin@fudan.edu.cn

ABSTRACT

Embeddings have become a key paradigm to learn graph representations and facilitate downstream graph analysis tasks. Existing graph embedding techniques either sample a large number of node pairs from a graph to learn node embeddings via stochastic optimization, or factorize a high-order proximity/adjacency matrix of the graph via expensive matrix factorization. However, these techniques usually require significant computational resources for the learning process, which hinders their applications on large-scale graphs. Moreover, the cosine similarity preserved by these techniques shows suboptimal efficiency in downstream graph analysis tasks, compared to Hamming similarity, for example. To address these issues, we propose NodeSketch, a highly-efficient graph embedding technique preserving high-order node proximity via recursive sketching. Specifically, built on top of an efficient data-independent hashing/sketching technique, NodeSketch generates node embeddings in Hamming space. For an input graph, it starts by sketching the self-loop-augmented adjacency matrix of the graph to output low-order node embeddings, and then recursively generates k-order node embeddings based on the self-loop-augmented adjacency matrix and (k-1)-order node embeddings. Our extensive evaluation compares NodeSketch against a sizable collection of state-of-the-art techniques using five real-world graphs on two graph analysis tasks. The results show that NodeSketch achieves state-of-the-art performance compared to these techniques, while showing significant speedup of 9x-372x in the embedding learning process and 1.19x-1.68x speedup when performing downstream graph analysis tasks.

CCS CONCEPTS

• **Information systems** → *Data mining*; *Web applications*;

KEYWORDS

Graph embedding, Recursive sketching, Data independent hashing

ACM Reference Format:
Dingqi Yang, Paolo Rosso, Bin Li and Philippe Cudre-Mauroux. 2019. NodeSketch: Highly-Efficient Graph Embeddings via Recursive Sketching. In *The 25th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '19), August 4–8, 2019, Anchorage, AK, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3292500.3330951

KDD '19, August 4–8, 2019, Anchorage, AK, USA

ACM ISBN 978-1-4503-6201-6/19/08....$15.00.
https://doi.org/10.1145/3292500.3330951

1 INTRODUCTION

With the increasing prominence of graph (network) data in real-world scenarios such as online social networks, biological networks and communication networks, graph embeddings (a.k.a. network embeddings) have become a key paradigm to learn node representations from a graph [3]. Specifically, these techniques represent each node in a graph as a feature vector, while still preserving key structural properties of the graph (mostly topological proximity of the nodes). Based on such node embeddings, downstream graph analysis tasks, such as node classification and link prediction, can be easily performed.

However, existing graph embedding techniques usually face computational challenges in the embedding learning process, and become inefficient when tackling large graphs. In the current literature, existing techniques can be roughly classified into two categories, i.e., graph-sampling based techniques and factorization-based techniques. First, graph-sampling based techniques such as DeepWalk [23], Node2Vec [9], LINE [26], SDNE [30] or VERSE [29] sample node pairs (directly or using random walks) from an input graph and design specific models to learn node embeddings from those samples via stochastic optimization. To ensure the quality of the learnt node embeddings, these methods usually sample a large number of node pairs, and thus require significant computational resources (CPU time in particular). Second, factorization-based techniques such as GraRep [4], HOPE [22] and NetMF [24] learn node embeddings directly from the high-order proximity/adjacency matrix of an input graph using matrix factorization. These methods usually require significant computational resources also (both CPU time and RAM) due to expensive matrix factorization operations. These computational challenges indeed hinder the application of many of those techniques on large-scale graphs (see Section 4.2 for more details).

Moreover, the learnt node embeddings from the above techniques also face computational challenges when being used in downstream tasks, in particular for tasks heavily involving similarity computation between node embeddings. A typical example is link prediction, which tries to predict potential links between pairs of nodes in a graph. For a given graph, it requires to compute the similarity between all pairs of disconnected nodes in the embedding vector space, and then to rank the node pairs according to their similarity. In the current literature, most of the existing graph embedding techniques measure node proximity using cosine distance (or dot product after normalization). However, cosine distance is known to be slow compared to other techniques. A recent work [18], for instance, applied learning-to-hash techniques [31] to graph embedding problems to learn node embeddings in Hamming space, and showed that using Hamming distance is able to speedup the KNN search (which heavily involves similarity computations) by

4x-10x compared to cosine distance. Despite this clear computational advantage of Hamming distance in downstream tasks, the graph embedding technique proposed by [18] still undergoes expensive matrix factorization operations, resulting in an inefficient embedding learning process (see Section 4.4 for more details).

Against this background and to address the above computational challenges, we explore data-independent hashing (i.e., sketching[1]) techniques [6] to efficiently and effectively solve the graph embedding problem. Specifically, sketching techniques use randomized hashing functions to create compact and fixed-size sketches for the original high-dimensional data for fast similarity approximation in Hamming space. Different from learning-to-hash techniques [31], which are data-dependent and learn dataset-specific hash functions, data-independent sketching techniques use randomized hashing functions without involving a learning process on a dataset, which is often much more efficient. However, it is not straightforward to directly apply existing sketching techniques such as minhash [2] or consistent weighted sampling [20], to the graph embedding problem due to the requirement of considering high-order proximity in graph embedding problems. More precisely, it has been widely shown that considering high-order proximity is of particular importance in learning high-quality node embeddings from a graph [4, 18, 22, 24]. While directly applying sketching techniques to the adjacency matrix of a graph captures low-order node proximity only, a straightforward extension to capture high-order proximity is to sketch the high-order adjacency/proximity matrix of the graph, measured by Katz index [22] or a high-order transition matrix (used for random walks in a graph) [4] for example. However, involving such high-order adjacency matrices dramatically increases the complexity of the process, since 1) the computation of high-order transition matrices usually involves expensive matrix multiplication/inversion operations, and 2) storing them creates significant overhead in RAM as these high-order adjacency matrices are often much denser compared to the original adjacency matrix of the graph. For example, on our Blog dataset (see Section 4.1.1), we observe that the density of its original adjacency matrix is only 0.63%, while the density of its 2nd- and 3rd-order adjacency matrices quickly raises up to 61.66% and 99.48%, respectively.

In this paper, to efficiently exploit sketching techniques for graph embeddings, we propose NodeSketch, a highly-efficient graph embedding technique preserving high-order node proximity via recursive sketching. Specifically, NodeSketch is designed on top of consistent weighted sampling [10, 13, 16, 20, 36], which is a popular data-independent sketching technique for sketching nonnegative real-valued, high-dimensional data. Our recursive skeching process works in the following way. For each node in a given graph, our technique starts by sketching its Self-Loop-Augmented (SLA) adjacency vector [1] to create its low-order (i.e., 1st- and 2nd-order proximity-preserving) embedding vector. Afterwards, to output the k-order embedding of the node, our technique sketches an approximate k-order SLA adjacency vector, which is generated by merging the node's SLA adjacency vector with the (k-1)-order embedding vectors of all the node's direct neighbors in a weighted manner. Such a recursive sketching process actually captures up-to-k-order node proximity (with an exponential decay weight when increasing k) in the resulting node embeddings. Our design ensures that NodeSketch is highly-efficient, as it involves fast vector sketching and merging operations only, without computing and storing any high-order adjacency matrix. We conduct a thorough empirical evaluation using five real-world graphs on two graph analysis tasks, including node classification and link prediction tasks. We compare NodeSketch against a sizable collection of state-of-the-art techniques from three categories, i.e., classical graph embedding techniques, learning-to-hash techniques, and sketching techniques. The results show that our NodeSketch achieves state-of-the-art performance with a remarkable speedup in the embedding learning process. In summary, our contributions are three-fold:

- We investigate data-independent sketching techniques to solve graph embedding problems, aiming to overcome the computational challenges in both the node embedding learning process and the application of node embeddings to downstream graph analysis tasks;
- We propose NodeSketch, a highly-efficient graph embedding techniques preserving high-order node proximity via recursive sketching. NodeSketch not only generates high-quality and efficient-to-use node embeddings in Hamming space by considering high-order node proximity, but is also highly-efficient in the embedding learning process due to our recursive sketching process;
- Our extensive evaluation shows that NodeSketch significantly outperforms learning-to-hash and other sketching techniques, and achieves state-of-the-art performance compared to classical graph embedding techniques. More importantly, NodeSketch is highly-efficient in the embedding learning process, showing 12x-372x speedup over classical graph embedding baselines, 9x-163x speedup over learning-to-hash baselines, and a 10x speedup over other sketching baselines. The resulting node embeddings preserving Hamming distance also lead to an improved efficiency in downstream graph analysis tasks, showing 1.19x-1.68x speedup over cosine distance.

2 RELATED WORK

2.1 Graph Embeddings

In the current literature, most of the existing graph embedding techniques focus on projecting nodes onto real-valued vector spaces, where the similarity between nodes is measured using cosine similarity (dot-product of normalized embedding vectors). These techniques can be classified into two categories. First, graph-sampling based techniques design specific models to learn node embeddings from pairs of nodes sampled from an input graph. DeepWalk [23] and Node2vec [9], JUST [12] and LBSN2Vec [37] firstly sample pairs of nodes from an input graph using random walks, and then feed them into a SkipGram-alike model [21] to output node embeddings. LINE [26] directly samples node pairs from an input graph considering specifically the 1st- and 2nd-order node proximity. VERSE [29] was recently proposed as a generalized graph-sampling based embedding learning framework that preserves a pre-selected node similarity measure. SDNE [30] and DVNE [39] use deep neural networks to learn node embeddings by sampling nodes from an input graph. As the embedding learning process of the techniques from this category usually relies on Stochastic Gradient Descent (SGD),

[1]We use the term "sketching" in this paper to exclusively refer to data-independent hashing, in order to avoid any potential confusion with learning-to-hash.

a large number of node pairs are often sampled and learnt to ensure the quality of the learnt node embeddings (convergence of the SGD algorithms), which requires significant computational resources (CPU time in particular). Second, factorization-based techniques apply matrix factorization on a high-order proximity/adjacency matrix of an input graph to output node embeddings. GraRep [4] preserves the top k-order node proximity by factorizing the top k-order transition matrices; HOPE [22] investigates several different node proximity measures and uses a generalized SVD to factorize the corresponding high-order node proximity matrices; NetMF [24] was recently proposed as a generalized matrix factorization framework unifying DeepWalk, Node2vec and LINE. Due to expensive matrix factorization operations, the techniques from this category suffer from computational challenges (RAM bottleneck in particular). In this paper, we explore sketching techniques to efficiently generate node embeddings from a graph and overcome these computational challenges.

In addition, it has been shown that cosine similarity (used by above techniques) is not efficient enough in downstream graph analysis tasks, in particular those involving intensive similarity computation, such as link prediction. To overcome this issue, INH-MF [18] was recently proposed to generate node embeddings in Hamming space using learning-to-hash techniques [31], resulting in a significant speedup in the downstream KNN search task compared to cosine similarity. However, similar to factorization-based graph embedding techniques, INH-MF still has severe computational issues in its embedding learning process, which utilizes expensive matrix factorization operations. To overcome this limitation, data-independent hashing (sketching) has been recently explored in solving *attributed* graph embedding problems. NetHash [33] has been proposed for *attributed* graph embedding, where each node in a graph is assumed to have a set of attributes describing the properties of the node. It adopts locality sensitive hashing [7] to sketch node *attributes* in a recursive manner. However, as node attributes are not always available in real-world graphs, this paper focuses on using sketching techniques to solve general graph embeddings problems without assuming the availability of node attributes, which therefore differs from the problem setting of NetHash.

2.2 Similarity-Preserving Hashing

Similarity-preserving hashing [6, 31] has been extensively studied to efficiently approximate the similarity of high dimensional data, such as documents or images [7]. Its key idea is to create compact sketches (in Hamming space) of the original high dimensional data while still preserving their similarities. According to the hashing process, the existing techniques can be classified into two categories: data-dependent hashing and data-independent hashing/sketching [6] (also called by [31] as learning-to-hash and locality sensitive hashing, respectively). First, data-dependent hashing (learning-to-hash) techniques, such as spectral hashing [32], iterative quantization [8], discrete graph hashing [19], supervised discrete hashing [25] and scalable graph hashing [14], learn dataset-specific hashing functions to closely fit the underlying data distribution in the feature space. Second, data-independent sketching techniques, such as minhash [2] and consistent weighted sampling [20], use randomized hashing functions without involving any learning process from a dataset, which is usually more efficient. In this paper, we exploit data-independent sketching techniques to build a highly-efficient solution for graph embedding problems, while considering high-order node proximity. To this end, we resort to a recursive sketching scheme. Recursive sketching has been explored mainly for data with complex structures in order to capture the internal structural information of each data instance, such as textual structures of a document [5], or subtrees in a graph [15]. These approaches create a sketch vector for each complex data instance (e.g., a graph), in order to fast approximate the similarity between two data instances (e.g., two graphs). In contrast, our objective is to create a sketch for each node in a graph, while preserving their high-order node proximity, which is different from [5, 15].

3 NODESKETCH

In this section, we first briefly introduce consistent weighted sampling techniques, and then present our proposed technique NodeSketch, followed by a theoretical analysis.

3.1 Preliminary:Consistent Weighted Sampling

Consistent weight sampling techniques were originally proposed to approximate min-max similarity for high-dimensional data [10, 13, 16, 20, 34–36]. Formally, given two nonnegative data vectors V^a and V^b of size D, their min-max similarity is defined as follows:

$$Sim_{MM}(V^a, V^b) = \frac{\sum_{i=1}^{D} \min(V_i^a, V_i^b)}{\sum_{i=1}^{D} \max(V_i^a, V_i^b)} \quad (1)$$

It is also called weighted Jaccard similarity [16], as it can be simplified to Jaccard similarity under the condition $V^a, V^b \in \{0,1\}^D$. When applying the sum-to-one normalization $\sum_{i=1}^{D} V_i^a = \sum_{i=1}^{D} V_i^b = 1$, Eq. 1 becomes the normalized min-max similarity, denoted by Sim_{NMM}. It has been shown in [16] that (normalized) min-max kernel is an effective similarity measure for nonnegative data; it can achieve state-of-the-art performance compared to other kernels such as linear kernel and intersection kernel on different classification tasks over a sizable collection of public datasets.

The key idea of consistent weight sampling techniques is to *generate data samples such that the probability of drawing identical samples for a pair of vectors is equal to their min-max similarity. A set of such samples is then regarded as the sketch of the input vector.* The first consistent weighted sampling method [20] was designed to handle integer vectors. Specifically, given a data vector $V \in \mathbb{N}^D$, it first uses a random hash function h_j to generate independent and uniform distributed random hash values $h_j(i, f)$ for each (i, f), where $i \in \{1, 2, ..., D\}$ and $f \in \{1, 2, ..., V_i\}$, and then returns $(i_j^*, f_j^*) = \text{argmin}_{i \in \{1,2,...,D\}, f \in \{1,2,...,V_i\}} h_j(i, f)$ as one sample (i.e., one sketch element S_j). The random hash function h_j depends only on (i, f), and maps (i, f) uniquely to $h_j(i, f)$. By applying L ($L \ll D$) independent random hash functions ($j = 1, 2, ..., L$), we generate sketch S (of size L) from V. Subsequently, the collision probability between two sketch elements (i_j^{a*}, f_j^{a*}) and (i_j^{b*}, f_j^{b*}), which are generated from V^a and V^b, respectively, is proven to be exactly the min-max similarity of the two vectors [10, 20]:

$$Pr[(i_j^{a*}, f_j^{a*}) = (i_j^{b*}, f_j^{b*})] = Sim_{MM}(V^a, V^b) \quad (2)$$

Therefore, the min-max similarity between V^a and V^b of large size D can be efficiently approximated by the Hamming similarity between the sketches S^a and S^b of compact size L.

To improve the efficiency of the above method and extend it to nonnegative real vectors ($V \in \mathbb{R}^D_{\geq 0}$), Ioffe [13] later proposed to directly generate one hash value for each i (with its corresponding $f \in \mathbb{N}$, $f \leq V_i$) by taking V_i as the input of the random hash value generation process, rather than generating V_i different random hash values. In such a case, V_i can also be any nonnegative real number. Based on this method, Li [16] further proposed 0-bit consistent sampling to simplify the sketch by only keeping i^*_j rather than (i^*_j, f^*_j), and empirically proved $Pr[i^{a*}_j = i^{b*}_j] \approx Pr[(i^{a*}_j, f^{a*}_j) = (i^{b*}_j, f^{b*}_j)]$. Recently, Yang et al. [36] further improved the efficiency of 0-bit consistent sampling using a much more efficient hash value generation process, where the resulting sketches have been proven to be equivalent to those generated by 0-bit consistent sampling. A succinct description of the method proposed in [36] is as follows. To generate one sketch element S_j (sample i^*_j), the method uses a random hash function h_j with an input i (seed for a random number generator) to generate a random hash value $h_j(i) \sim Uniform(0, 1)$, and then returns the sketch element as:

$$S_j = \underset{i \in \{1,2,\ldots,D\}}{\operatorname{argmin}} \frac{-\log h_j(i)}{V_i} \tag{3}$$

With a sketch length of L, the resulting sketches actually preserves normalized min-max similarity [35]:

$$Pr[S^a_j = S^b_j] = Sim_{NMM}(V^a, V^b),\ j = 1, 2, \ldots, L. \tag{4}$$

Please refer to [13, 16, 36] for more details. In this paper, we take advantage of the high efficiency of the above consistent weighted sampling technique to design NodeSketch, a highly-efficient graph embedding technique via recursive sketching.

3.2 Node Emebeddings via Recursive Sketching

Built on top of the above consistent weighted sampling technique, our proposed NodeSketch first generates low-order (1st- and 2nd-order) node embeddings from the Self-Loop-Augmented (SLA) adjacency matrix of an input graph [1], and then generate k-order node embeddings[2] based on this SLA adjacency matrix and the (k-1)-order node embeddings in a recursive manner.

3.2.1 Low-Order Node Embeddings. The adjacency matrix $\mathbf{A}$ of a graph encodes the 1st-order node proximity of the graph. It is often used by classical graph embedding techniques, such as GraRep [4] and LINE [26], to learn 1st-order node embeddings. However, directly sketching an adjacency vector V (one row of the adjacency matrix $\mathbf{A}$) actually overlooks the 1st-order node proximity and only preserves 2nd-order node proximity. To explain this, we investigate the min-max similarity between the nodes' adjacency vectors (preserved by the sketches). As the adjacency vector of a node contains only its direct neighbors, the min-max similarity between two nodes indeed characterizes the similarity between their sets of neighbors only. Figure 1 shows an toy example in its top part.

[2]Note that the k-order embeddings here actually refers to up-to-k-order embeddings in this section; we keep using k-order embeddings for the sake of clarity.

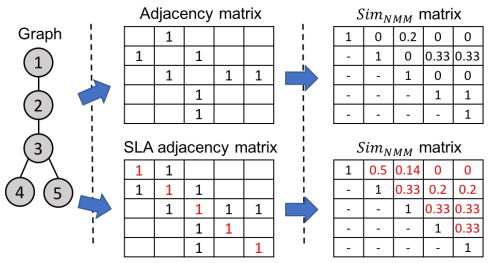


Figure 1: A toy example illustrating the adjacency and SLA adjacency matrices of a graph and their corresponding normalized min-max similarity matrices Sim_{NMM} (computed using Eq. 1 between each pair of normalized adjacency vectors). In the top part of the □gure, we see that Sim_{NMM} between the original adjacency vectors ignores the 1st-order node proximity, but preserves only the 2nd-order proximity. More precisely, we have $Sim_{NMM}(node1, node2) = 0$ as $nodes1$ and $nodes2$ do not share any common neighbor (even though they are directly connected), while $Sim_{NMM}(node1, node3) = 0.2$ as $nodes1$ and $nodes3$ have a common neighbor $node2$ (but they are not directly connected). In contrast, as shown in the bottom part of the □gure, Sim_{NMM} between the SLA adjacency vectors preserves both 1st- and 2nd-order node proximity. After adding an identity matrix (changes highlighted in red), we have $Sim_{NMM}(node1, node2) = 0.5$ and $Sim_{NMM}(node1, node3) = 0.14$, which implies that $node1$ is now "closer" (in terms of Sim_{NMM}) to $node2$ than to $node3$.

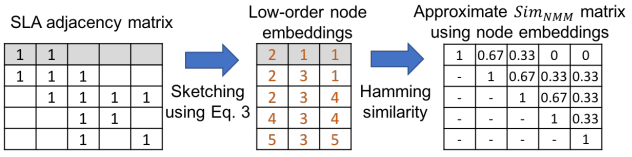


Figure 2: Low-order node embeddings by sketching the SLA adjacency vector of each node, where the sketch length (embeddings size) is set to 3. We highlight the SLA adjacency vector and the corresponding embeddings of $node1$ as an example. Based on the node embeddings, the Sim_{NMM} matrix can be e□ciently approximated by computing the hamming similarity between node embeddings.

To address this issue, we resort to the *Self-Loop-Augmented (SLA) adjacency matrix* of a graph [1]. Specifically, it is obtained by adding an identity matrix to the original adjacency matrix of the graph:

$$\widetilde{\mathbf{A}} = \mathbf{I} + \mathbf{A} \tag{5}$$

Subsequently, the min-max similarity between the resulted SLA adjacency vectors $\widetilde{V}$ (row vectors of $\widetilde{\mathbf{A}}$) is able to preserve both 1st- and 2nd-order node proximity. More precisely, when two nodes are directly connected, their SLA adjacency vectors have two more common entries than the original adjacency vectors, and thus further captures 1st-order node proximity beyond the 2nd-order proximity captured by the original adjacency vectors. Figure 1 shows the SLA adjacency matrix of the previous toy example in its bottom part.

In summary, we sketch the SLA adjacency vector of each node using Eq. 3 to generate its low-order (1st- and 2nd-order proximity preserving) sketch/embedding[3]. Figure 2 shows the low-order node embeddings generated for the graph in Figure 1.

3.2.2 High-Order Node Embeddings. NodeSketch learns high-order node embeddings in a recursive manner. Specifically, to output the k-order embedding of a node, it sketches an approximate k-order SLA adjacency vector of the node, which is generated by merging the node's SLA adjacency vector with the (k-1)-order embeddings of all the neighbors of the node in a weighted manner.

One key property of the consistent weighted sampling technique (in Eq. 3) is *the uniformity of the generated samples, which states that the probability of selecting i is proportional to V_i, i.e.,* $Pr(S_j = i) = \frac{V_i}{\sum_i V_i}$. As its proof is omitted in the original paper [36], we provide a brief proof in the supplemental materials. This uniformity property serves as the foundation of our recursive sketching process. It implies that the proportion of element i in the resulting sketch S is an unbiased estimator of V_i, where we applied sum-to-one normalization $\sum_i V_i = 1$, and thus the empirical distribution of sketch elements is an unbiased approximation of input vector V.

Based on this uniformity property, our recursive sketching process works in the following way. First, for each node r, we compute *an approximate k-order SLA adjacency vector* $\widetilde{V}^r(k)$ by merging the node's SLA adjacency vector $\widetilde{V}^r$ with the distribution of the sketch elements in the (k-1)-order embeddings of all the neighbors of the node in a weighted manner:

$$\widetilde{V}_i^r(k) = \widetilde{V}_i^r + \sum_{n \in \Gamma(r)} \frac{\alpha}{L} \sum_{j=1}^{L} \mathbb{1}_{[S_j^n(k-1)=i]} \tag{6}$$

where $\Gamma(r)$ is the set of neighbors of node r, $S^n(k-1)$ is the (k-1)-order sketch vector of node n, and $\mathbb{1}_{[cond]}$ is an indicator function which is equal to 1 when *cond* is true and 0 otherwise. More precisely, the sketch element distribution for one neighbor n, (i.e., $\frac{1}{L}\sum_{j=1}^{L} \mathbb{1}_{[S_j^n(k-1)=i]}$ where $i = 1, ..., D$) actually approximates the (k-1)-order SLA adjacency vector of the neighbor, which preserves the (k-1)-order node proximity. Subsequently, by merging the sketch element distribution for all the node's neighbors with the node's SLA adjacency vector, we indeed expand the order of proximity by one, and therefore obtain an approximate k-order SLA adjacency vector of the node. Moreover, during the summing process, we assign an (exponential decay) weight α to the sketch element distribution, in order to give less importance to higher-order node proximity. Such a weighting scheme in the recursive sketching process actually implements exponential decay weighting when considering high-order proximity, where the weights for the kth-order proximity decays exponentially with k; it is a widely used weighting scheme in measuring high-order node proximity [22, 33]. Subsequently, we generate the k-order node embeddings $S(k)$ by sketching the approximate k-order SLA adjacency vector $\widetilde{V}^r(k)$ using Eq. 3. Figure 3 shows the high-order node embeddings generated via recursive sketching for the graph of Figure 1.

In summary, Algorithm 1 shows the overall process of generating k-order node embeddings from three inputs: the SLA adjacency matrix $\widetilde{\mathbf{A}}$, the order k and decay weight α. When $k > 2$, we first generate (k-1)-order node embeddings $S(k-1)$ using Algorithm 1 again (Line 2), and then generate k-order node embeddings by sketching each node's approximate k-order SLA adjacency vector $\widetilde{V}^r(k)$ which is obtained from $\widetilde{V}^r$ and $S(k-1)$ using Eq. 6 (Line 3-6). When $k = 2$, we simply generate low-order node embeddings by directly sketching each SLA adjacency vector $\widetilde{V}$ (Line 8-10). The implementation of NodeSketch is available here[4].

Algorithm 1 NodeSketch ($\widetilde{\mathbf{A}}$, k, α)

```
1:  if k > 2 then
2:      Get (k-1)-order sketch: S(k − 1) = NodeSketch (Ã, k − 1, α)
3:      for each row (node) r in Ã do
4:          Get k-order SLA adjacency vector Ṽ^r(k) using Eq. 6
5:          Generate sketch S^r(k) from Ṽ^r(k) using Eq. 3
6:      end for
7:  else if k = 2 then
8:      for each row (node) r in Ã do
9:          Generate low-order sketch S^r(2) from Ṽ^r using Eq. 3
10:     end for
11: end if
12: return k-order sketch S(k)
```

3.3 Theoretical Analysis

3.3.1 Similarity and error bound. According to Eq. 4, the hamming similarity $H(\cdot,\cdot)$ between the k-order embeddings of two nodes a and b actually approximates the normalized min-max similarity between the k-order SLA adjacency vectors of the two nodes:

$$\mathbb{E}(H(S^a(k), S^b(k))) = Pr[S_j^a(k) = S_j^b(k)] = Sim_{NMM}(\widetilde{V}^a(k), \widetilde{V}^b(k))$$

The corresponding approximation error bound is:

$$Pr[|H - Sim_{NMM}| \geq \epsilon] \leq 2\exp(-2L\epsilon^2) \tag{7}$$

The error is bigger than ϵ with probability at most $2\exp(-2L\epsilon^2)$. Please refer to the supplemental materials for the proof.

3.3.2 Complexity. For time complexity, we separately discuss the cases of low- and high-order node embeddings. First, for low-order node embeddings where we directly apply Eq. 3 on the SLA adjacency vector of each node, the time complexity is $O(D \cdot L \cdot \overline{d})$, where D and L are the number of nodes and embedding size (sketch length), respectively, and $\overline{d}$ is the average node degree in the SLA adjacency matrix. Second, for high-order node embeddings ($k > 2$) where the recursive sketching process is involved, the time complexity is $O(D \cdot L \cdot (\overline{d} + (k-2) \cdot \min\{\overline{d} \cdot L, \overline{d}^2\}))$. In practice, we often have $\overline{d} \ll D$ due to the sparsity of real-world graphs, and also $L, k \ll D$. Therefore, the time complexity is linear w.r.t. the number of nodes D. Moreover, only involving fast hashing and merging operations makes NodeSketch highly-efficient as we show below.

For space complexity, NodeSketch is memory-efficient as it only stores the SLA adjacency matrix and the node embeddings, resulting in a space complexity of $O(D \cdot (\overline{d} + L))$. Compared to the case of storing a high-order proximity matrix (such as GraRep [4] and NetMF [24]) where the space complexity is $O(D^2)$, NodeSketch is much more memory-efficient as $\overline{d}, L \ll D$.

[3] As the sketch vector of a node is regarded as its embedding vector, we do not distinguish these two terms in this paper.

[4] https://github.com/eXascaleInfolab/NodeSketch

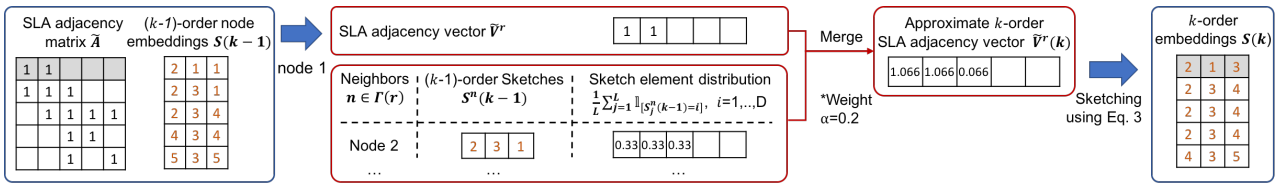


Figure 3: High-order node embeddings via recursive sketching. Here we highlight the detailed recursive sketching process for $node1$ based on the SLA adjacency matrix and (k-1)-order node embeddings (where k=3 in this example). First, we compute the approximate k-order SLA adjacency vector $\widetilde{V}^r(k)$ by summing the SLA adjacency vector of $node1$ and the sketch element distribution in the (k-1)-order embeddings of all $node1$'s neighbors ($node1$ has only one neighbor $node2$ in the graph) in a weighted manner. The exponential decay weight is set to $\alpha = 0.2$ here. Then, we generate the k-order node embeddings by sketching the approximate k-order SLA adjacency vector $\widetilde{V}^r(k)$ using Eq. 3.

Table 1: Characteristics of the experimental graphs

Dataset	Blog	PPI	Wiki	DBLP	YouTube
#Nodes	10,312	3,890	4,777	13,326	1,138,499
#Edges	333,983	76,584	184,812	34,281	2,990,443
#Labels	39	50	40	2	47

4 EXPERIMENTS

4.1 Experimental Setting

4.1.1 Dataset. We conduct experiments on the following five real-world graphs which are commonly used by existing works on graph embeddings. *BlogCatalog (**Blog**)* [27] is a social network of bloggers. The labels of a node represent the topic categories that the corresponding user is interested in. *Protein-Protein Interactions (**PPI**)* [9] is a graph of the PPI network for Homo Sapiens. The labels of a node refer to its gene sets and represent biological states. *Wikipedia (**Wiki**)* [9] is a co-occurrence network of words appearing in a sampled set of the Wikipedia dump. The labels represent the part-of-speech tags. ***DBLP*** [38] is a collaboration network capturing the co-authorship of authors. The labels of a node refer to the publication venues of the corresponding author. ***YouTube*** [28] is a social network of users on YouTube. The labels of a node refer to the groups (e.g., anime) that the corresponding user is interested in. Table 1 summarizes the main statistics of those graphs.

4.1.2 Baselines. We compare NodeSketch against a sizable collection of state-of-the-art techniques from three categories: 1) classical graph embedding techniques (preserving cosine similarity), including **DeepWalk** [23], **Node2Vec** [9], **LINE** [26], **VERSE** [29], **GraRep** [4], **HOPE** [22] and **NetMF** [24]; 2) learning-to-hash techniques which are among the best-performing techniques in [18], including Spectral Hashing (**SH**) [32], Iterative quantization (**ITQ**) [8], Scalable Graph Hashing (**SGH**) [14] and **INH-MF** [18]; 3) sketching techniques, including **NetHash** [33], **KatzSketch** (which directly sketches the high-order node proximity matrix using Katz index) , **NodeSketch(NoSLA)** (a variant of our proposed NodeSketch by using the original adjacency matrix rather than the SLA adjacency matrix). *Please refer to the supplemental materials for detailed description on configuration and parameter tuning for individual methods. In all the experiments, we tune the parameters of each method on each task to let it achieve its highest performance.* The dimension of the node embeddings L is set to 128 for all methods.

4.2 Multi-label Node Classi□cation Task

Node classification predicts the most probable label(s) for some nodes based on other labeled nodes. In this experiment, we randomly pick a set of nodes as labeled nodes for training, and use the rest for testing. To fairly compare node embeddings with different similarity measures, we train a one-vs-rest kernel SVM classifier with a pre-computed kernel (cosine or Hamming kernel according to the embedding techniques) to return the most probable labels for each node. We report the average Macro-F1 and Micro-F1 scores from 10 repeated trials, with 90% training ratio on Blogcatalog, PPI, Wiki and DBLP, and 9% training ratio on YouTube. We note that similar results are observed with different training ratios (not shown due to the space limitation).

Table 2 shows the results. Note that on our YouTube dataset (>1M nodes), many baselines run out of memory, marked as "-" (also on other evaluation tasks). More precisely, Node2Vec requires to compute and store a large and non-sparse 2nd-order transition probability matrix for parameterized random walk; GraRep, NetMF, SH, ITQ, INH-MF and KatzSketch involve expensive matrix factorization/inversion/multiplication operations. We also highlight the best-performing technique from each of the three categories.

First, we observe that NodeSketch outperforms all sketching baselines in general. The only exception is on the Wiki dataset, where our proposed baseline KatzSketch is slightly better than NodeSketch. However, as KatzSketch involves expensive matrix multiplication/inversion operations to compute Katz index, NodeSketch is much more efficient than it, showing a 10x speedup on average (see Section 4.4 below). Second, among all the learning-to-hash methods, we find that INH-MF is the best-performing method on small and mid-size datasets (Blog, PPI, Wiki and DBLP), while SGH is the only technique that can handle the large YouTube dataset. However, they still show inferior performance compared to NodeSketch. Finally, among classical graph embedding methods, NetMF achieves the best performance on Blog, Wiki and DBLP, while DeepWalk and VERSE are the best ones on POI and YouTube, respectively. NodeSketch shows comparable performance to these best-performing baselines; it has better results on PPI, Wiki and DBLP, and slightly worse results on Blog and YouTube. However, our NodeSketch is *far* more efficient than these baselines, i.e., 22x, 239x and 59x faster than NetMF, DeepWalk and VERSE, respectively (see Section 4.4 below).

Table 2: Node classi□cation performance using kernel SVM

Methods	Micro-F1 (%)					Macro-F1 (%)				
	Blog	PPI	Wiki	DBLP	YouTube	Blog	PPI	Wiki	DBLP	YouTube
DeepWalk	39.60	**17.24**	46.05	83.46	39.03	21.93	**10.28**	6.62	83.16	27.84
Node2Vec	37.95	16.04	50.32	93.25	-	20.22	9.57	9.86	93.12	-
LINE	35.49	15.01	48.22	86.83	38.00	16.60	8.70	8.47	86.54	25.68
VERSE	39.61	15.90	41.39	92.79	**39.62**	22.85	9.76	4.14	92.66	**29.16**
GraRep	36.21	5.83	56.22	91.41	-	16.91	1.52	12.14	91.25	-
HOPE	31.37	14.69	56.68	91.47	33.91	11.74	8.13	13.30	91.30	19.62
NetMF	**40.04**	15.03	**57.62**	**93.59**	-	**23.43**	8.74	**14.35**	**93.46**	-
SH	17.88	9.03	38.00	66.36	-	4.20	4.04	3.60	60.38	-
ITQ	17.47	6.21	39.97	58.13	-	3.08	2.46	3.14	39.82	-
SGH	21.28	10.95	43.82	68.36	**27.74**	6.60	6.89	5.63	64.98	**11.21**
INH-MF	**36.13**	**15.50**	**45.03**	**93.27**	-	**18.88**	**9.55**	**6.90**	**93.16**	-
NetHash	35.80	18.85	47.57	97.61	38.61	18.72	12.91	8.05	97.57	27.41
KatzSketch	37.96	20.17	**59.09**	98.17	-	21.26	13.83	**17.01**	98.13	-
NodeSketch(NoSLA)	37.10	20.20	59.02	98.23	38.40	20.47	14.36	16.22	98.20	27.08
NodeSketch	**38.16**	**21.04**	59.07	**98.83**	**39.18**	**21.84**	**15.55**	16.31	**98.81**	**28.53**

4.3 Link Prediction Task

Link prediction predicts potential links between nodes in a graph. In this task, we use the same setting as in [22]. Specifically, we randomly sample 20% of the edges from each graph as test data, and use the rest of the graph for training. After learning the node embeddings based on the training graph, we predict the missing edges by generating a ranked list of potential edges. For each pair of nodes, we use the cosine or Hamming similarity (according to the embedding techniques) of their embeddings to generate the ranked list. We randomly sample 0.1% pairs of nodes for evaluation (0.01% on YouTube). We report the average precision@100 and recall@100 on Blog, PPI, Wiki and DBLP, and precision@1000 and recall@1000 on YouTube (due to its large size) from 10 repeated trials.

Table 3 shows the results. First, we observe that NodeSketch consistently outperforms all sketching and learning-to-hash baselines. Moreover, compared to the best performing classical graph embedding baselines (Node2Vec on Blog and Wiki datasets, VERSE on DBLP and YouTube datasets, and NetMF on PPI dataset), NodeSketch shows comparable results (better on PPI and DBLP, and worse on other datasets). However, it significantly outperforms these baselines in terms of runtime efficiency, which we present below.

4.4 Runtime Performance

We investigate the efficiency of both node embedding learning and the application of learnt node embeddings on different evaluation tasks. All the experiments are conducted on a commodity PC (Intel Core i7-6820HQ@2.70GHz, 16GB RAM, Mac OS X). To discount the impact of explicit/implicit multi-threading implementation of individual methods, we use one thread (when applicable) and also report the CPU time for each method. For NodeSketch, as its complexity mainly depends on the order of proximity k (see Section 4.5 below), we set k=5 which is the largest of all the best-performing k values in different evaluation tasks and datasets.

First, Table 4 shows the node embedding learning time. We observe that NodeSketch is highly-efficient, and significantly outperforms all classical graph embedding baselines with 12x-372x speedup, all learning-to-hash baselines with 9x-163x speedup, and other sketching baselines with a 10x speedup. Moreover, compared to its variant NodeSketch(NoSLA), NodeSketch using the SLA adjacency matrix creates negligible computational overhead, but shows significantly improved performance on individual evaluation tasks. We also note that as the runtime complexity of LINE depends mostly on the number of sampled node pairs for learning, we observe a sublinear learning time for LINE as we set the number of samples to 1 billion for Blog, PPI, Wiki and DBLP datasets and 10 billion for YouTube (please refer to supplemental materials for details).

Second, Table 5 shows the end-to-end execution time of node classification (including classifier training and evaluation) and link prediction (including similarity computation, ranking and evaluation). We group the results by their distance measures, i.e., cosine distance for all classical graph embedding techniques, and Hamming distance for all learning-to-hash and sketching techniques. We see that Hamming distance is consistently more efficient than cosine distance, showing 1.19x and 1.68x speedup on average on the node classification and link prediction tasks, respectively. In particular, the link prediction task shows higher speedup, as it heavily involves similarity computations between node embeddings.

4.5 Parameter Sensitivity of NodeSketch

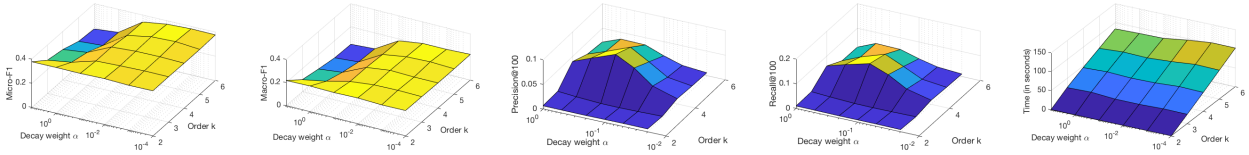
We study the impact of decay weight α and order of proximity k on both graph analysis performance and runtime performance. First, Figure 4 shows the results on the node classification and link prediction tasks using the Blog dataset. We observe a clear convex surface for each metric. Furthermore, we see that the optimal setting varies across different tasks ($\alpha = 0.001$ and $k = 5$ for node classification, and $\alpha = 0.2$ and $k = 4$ for link prediction), which suggests that node embeddings are often task-specific and should be appropriately generated by tuning parameters for individual graph analysis tasks. Second, Figure 4(e) shows the impact on embedding learning time. We see that our embedding learning time linearly increases with k due to our recursive sketching process, and α has little impact on the embedding learning time.

5 DISCUSSION

NodeSketch preserves the normalized min-max similarity between the high-order SLA adjacency vectors of nodes. Due to the nonlinearity of the min-max kernel [17], the resulting node embeddings

Table 3: Link prediction performance

Methods	Precision@100				Precision@1000	Recall@100				Recall@1000
	Blog	PPI	Wiki	DBLP	YouTube	Blog	PPI	Wiki	DBLP	YouTube
DeepWalk	0.0200	0.0159	0.0090	0.0423	0.0001	0.0301	0.2227	0.0493	0.6749	0.0022
Node2Vec	**0.0927**	0.0137	**0.0267**	0.0321	-	**0.1378**	0.1958	**0.1514**	0.5174	-
LINE	0.0070	0.0073	0.0031	0.0392	0.0004	0.0103	0.0923	0.0167	0.6186	0.0513
VERSE	0.0404	**0.0206**	0.0212	**0.0436**	**0.0007**	0.0602	**0.2723**	0.1118	**0.6906**	**0.0783**
GraRep	0.0014	0.0011	0.0054	0.0001	-	0.0020	0.0118	0.0286	0.0011	-
HOPE	0.0023	0.0073	0.0027	0.0248	0.0001	0.0035	0.0960	0.0149	0.4034	0.0062
NetMF	0.0175	0.0174	0.0084	0.0218	-	0.0266	0.2287	0.0474	0.3126	
SH	0.0001	0.0001	0.0003	0.0001	-	0.0001	0.0001	0.0017	0.0018	-
ITQ	0.0014	0.0007	0.0010	0.0002	-	0.0020	0.0099	0.0057	0.0041	-
SGH	0.0148	0.0019	0.0033	0.0002	**0.0001**	0.0222	0.0274	0.0186	0.0022	**0.0278**
INH-MF	**0.0158**	**0.0158**	**0.0084**	**0.0252**	-	**0.0227**	**0.2209**	**0.0454**	**0.4052**	-
NetHash	0.0015	0.0134	0.0020	0.0387	0.0001	0.0022	0.1899	0.0101	0.5958	0.0105
KatzSketch	0.0052	0.0124	0.0025	0.0422	-	0.0078	0.1659	0.0134	0.6705	-
NodeSketch(NoSLA)	0.0376	0.0167	0.0034	0.0433	0.0003	0.0564	0.2255	0.0183	0.7558	0.0447
NodeSketch	**0.0729**	**0.0250**	**0.0176**	**0.0462**	**0.0005**	**0.1080**	**0.3331**	**0.0942**	**0.7595**	**0.0769**



(a) (b) (c) (d) (e)

Figure 4: Impact of k and α on a) Micro-F1 in Node classi□cation, b) Macro-F1 in Node classi□cation, c) Precision@100 in Link prediction, d) Recall@100 in Link prediction, and e) Embedding learning time in seconds.

Table 4: Node embedding learning time (in seconds) and the average speedup of NodeSketch over each baseline.

Methods	Blog	PPI	Wiki	DBLP	YouTube	Speedup
DeepWalk	3375	1273	1369	4665	747060	239x
Node2Vec	1073	383	1265	504	-	51x
LINE	2233	2153	1879	2508	29403	148x
VERSE	1095	203	276	1096	245334	59x
GraRep	3364	323	422	10582	-	372x
HOPE	239	100	78	283	15517	12x
NetMF	487	124	708	213	-	22x
SH	2014	99	202	4259	-	151x
ITQ	2295	111	197	4575	-	163x
SGH	200	106	237	126	6579	9x
INH-MF	509	39	98	378	-	16x
NetHash	721	201	134	35	12708	10x
KatzSketch	213	22	42	264	-	10x
NodeSketch(NoSLA)	71	8	17	8	2456	1.01x
NodeSketch	70	8	17	8	2439	N/A

Table 5: Execution time (in seconds) of the evaluation tasks.

Tasks	Distance Measures	Blog	PPI	Wiki	DBLP	YouTube
Node Classification	Cosine	255.05	55.57	54.16	229.06	170.57
	Hamming	**226.96**	**42.78**	**45.27**	**204.45**	**139.25**
Link Prediction	Cosine	5.57	0.75	1.31	8.35	47.70
	Hamming	**3.51**	**0.42**	**0.65**	**5.46**	**32.17**

cannot be directly used by linear algorithms such as logistic regression, which is widely used in classical graph embedding papers [4, 9, 23, 24, 26, 29] for performing node classification. However, as proposed in [17], the min-max kernel can be easily linearized via a simple transformation scheme, which suggests to store only the lowest b bits of each value in a sketch vector; it has also been shown that $b = 8$ is often sufficient in practice. Using such a transformation scheme, we report the performance of node classification using a one-vs-rest logistic regression classifier in Table 6. We make the same observation as above; NodeSketch outperforms sketching and learning-to-hash baselines in general, and shows a level of performance comparable to classical graph embedding baselines (NodeSketch has better results on Wiki, DBLP and YouTube, and worse results on Blog and PPI).

6 CONCLUSION

This paper introduced NodeSketch, a highly-efficient graph embedding technique preserving high-order node proximity via recursive sketching. Built on top of an efficient consistent weighted sampling technique, NodeSketch generates node embeddings in Hamming space. It starts by sketching the SLA adjacency vector of each node to output low-order node embeddings, and then recursively generates k-order node embeddings based on the SLA adjacency matrix and the (k-1)-order node embeddings. We conducted a thorough empirical evaluation of our technique using five real-world graphs on two graph analysis tasks, and compared NodeSketch against a sizable collection of state-of-the-art techniques. The results show that NodeSketch significantly outperforms learning-to-hash and other sketching techniques, and achieves state-of-the-art performance compared to classical graph embedding techniques. More importantly, NodeSketch is *highly-efficient* in the embedding learning process and significantly outperforms all baselines with 9x-372x speedup. In addition, its node embeddings preserving Hamming

Table 6: Node classi□cation performance using logistic regression

Methods	Micro-F1 (%)					Macro-F1 (%)				
	Blog	PPI	Wiki	DBLP	YouTube	Blog	PPI	Wiki	DBLP	YouTube
DeepWalk	42.84	23.21	50.33	93.19	43.21	28.76	19.48	10.12	93.07	34.74
Node2Vec	41.48	22.30	51.68	93.51	-	27.31	19.14	11.10	93.41	-
LINE	41.91	23.89	53.24	86.49	**43.91**	25.75	18.97	11.21	86.25	**36.16**
VERSE	41.96	19.39	42.88	92.35	40.91	**29.11**	16.54	6.93	92.22	34.36
GraRep	40.53	5.73	58.06	91.81	-	24.26	1.52	12.85	91.64	-
HOPE	35.68	22.96	58.19	91.30	42.37	17.22	18.53	13.52	91.14	32.39
NetMF	**42.98**	**24.95**	**58.24**	**94.02**	-	28.34	**21.07**	**13.99**	**93.94**	-
SH	22.04	12.88	42.38	67.92	-	7.65	9.57	6.64	61.92	-
ITQ	13.49	6.65	38.95	55.88	-	3.77	5.36	4.04	46.33	-
SGH	23.29	12.41	46.66	68.23	**26.60**	11.68	10.85	10.08	64.63	**16.17**
INH-MF	**38.14**	**20.73**	**53.80**	**93.14**	-	**24.80**	**18.03**	**13.31**	**93.04**	-
NetHash	35.04	22.45	45.23	97.81	43.30	21.96	19.27	10.21	97.77	34.33
KatzSketch	36.75	23.12	**58.81**	97.99	-	24.08	20.24	16.21	97.96	-
NodeSketch(NoSLA)	35.82	23.60	57.76	98.07	43.29	22.71	20.56	16.20	98.04	34.36
NodeSketch	**38.06**	**24.61**	**58.81**	**98.34**	**43.99**	**25.85**	**21.47**	**16.23**	**98.32**	**36.23**

distance also lead to improved efficiency on downstream graph analysis tasks, with 1.19x-1.68x speedup over cosine distance.

In the future, we plan to extend NodeSketch on dynamic graphs which evolve over time.

ACKNOWLEDGMENTS

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement 683253/GraphInt), the Swiss National Science Foundation (grant #407540 167320 Tighten-it-All), the STCSM Project (16JC1420400, 18511103104) and the Program for Professor of Special Appointment (Eastern Scholar) at Shanghai Institutions of Higher Learning.

REFERENCES

[1] Alex Arenas, Alberto Fernandez, and Sergio Gomez. 2008. Analysis of the structure of complex networks at different resolution levels. *New Journal of Physics* 10, 5 (2008), 053039.
[2] Andrei Z Broder. 1997. On the resemblance and containment of documents. In *Proceedings of Compression and complexity of sequences 1997*. IEEE, 21–29.
[3] Hongyun Cai, Vincent W Zheng, and Kevin Chang. 2018. A comprehensive survey of graph embedding: problems, techniques and applications. *TKDE* (2018).
[4] Shaosheng Cao, Wei Lu, and Qiongkai Xu. 2015. Grarep: Learning graph representations with global structural information. In *CIKM'15*. ACM, 891–900.
[5] Lianhua Chi, Bin Li, and Xingquan Zhu. 2014. Context-preserving hashing for fast text classification. In *SDM'14*. SIAM, 100–108.
[6] Lianhua Chi and Xingquan Zhu. 2017. Hashing techniques: A survey and taxonomy. *ACM Computing Surveys (CSUR)* 50, 1 (2017), 11.
[7] Aristides Gionis, Piotr Indyk, Rajeev Motwani, et al. 1999. Similarity search in high dimensions via hashing. In *VLDB'99*, Vol. 99. 518–529.
[8] Yunchao Gong and Svetlana Lazebnik. 2011. Iterative quantization: A procrustean approach to learning binary codes. In *CVPR'11*. IEEE, 817–824.
[9] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *KDD'16*. ACM, 855–864.
[10] Bernhard Haeupler, Mark Manasse, and Kunal Talwar. 2014. Consistent weighted sampling made fast, small, and easy. *arXiv preprint arXiv:1410.4266* (2014).
[11] Wassily Hoeffding. 1963. Probability inequalities for sums of bounded random variables. *Journal of the American statistical association* 58, 301 (1963), 13–30.
[12] Rana Hussein, Dingqi Yang, and Philippe Cudré-Mauroux. 2018. Are Meta-Paths Necessary?: Revisiting Heterogeneous Graph Embeddings. In *CIKM'18*. ACM, 437–446.
[13] Sergey Ioffe. 2010. Improved consistent sampling, weighted minhash and l1 sketching. In *ICDM'10*. IEEE, 246–255.
[14] Qing-Yuan Jiang and Wu-Jun Li. 2015. Scalable Graph Hashing with Feature Transformation.. In *IJCAI'15*. 2248–2254.
[15] Bin Li, Xingquan Zhu, Lianhua Chi, and Chengqi Zhang. 2012. Nested subtree hash kernels for large-scale graph classification over streams. In *ICDM'12*. IEEE, 399–408.
[16] Ping Li. 2015. 0-bit consistent weighted sampling. In *KDD'15*. ACM, 665–674.
[17] Ping Li. 2015. Min-max kernels. *arXiv preprint arXiv:1503.01737* (2015).
[18] Defu Lian, Kai Zheng, Vincent W Zheng, Yong Ge, Longbing Cao, Ivor W Tsang, and Xing Xie. 2018. High-order Proximity Preserving Information Network Hashing. In *KDD'18*. ACM, 1744–1753.
[19] Wei Liu, Cun Mu, Sanjiv Kumar, and Shih-Fu Chang. 2014. Discrete graph hashing. In *NIPS'14*. 3419–3427.
[20] Mark Manasse, Frank McSherry, and Kunal Talwar. 2010. Consistent weighted sampling. *Technical Report MSR-TR-2010-73* (2010).
[21] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *NIPS*. 3111–3119.
[22] Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. 2016. Asymmetric transitivity preserving graph embedding. In *KDD'16*. ACM, 1105–1114.
[23] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *KDD'14*. ACM, 701–710.
[24] Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Kuansan Wang, and Jie Tang. 2018. Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec. In *WSDM'18*. ACM, 459–467.
[25] Fumin Shen, Chunhua Shen, Wei Liu, and Heng Tao Shen. 2015. Supervised discrete hashing. In *CVPR'15*. 37–45.
[26] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. Line: Large-scale information network embedding. In *WWW'15*. 1067–1077.
[27] Lei Tang and Huan Liu. 2009. Relational learning via latent social dimensions. In *KDD'09*. ACM, 817–826.
[28] Lei Tang and Huan Liu. 2009. Scalable learning of collective behavior based on sparse social dimensions. In *CIKM'09*. ACM, 1107–1116.
[29] Anton Tsitsulin, Davide Mottin, Panagiotis Karras, and Emmanuel Müller. 2018. VERSE: Versatile Graph Embeddings from Similarity Measures. In *WWW'18*. 539–548.
[30] Daixin Wang, Peng Cui, and Wenwu Zhu. 2016. Structural deep network embedding. In *KDD'16*. ACM, 1225–1234.
[31] Jingdong Wang, Ting Zhang, Song Jingkuan, Nicu Sebe, and Heng Tao Shen. 2018. A survey on learning to hash. *TPAMI* 40, 4 (2018), 769–790.
[32] Yair Weiss, Antonio Torralba, and Rob Fergus. 2009. Spectral hashing. In *NIPS'09*. 1753–1760.
[33] Wei Wu, Bin Li, Ling Chen, and Chengqi Zhang. 2018. Efficient Attributed Network Embedding via Recursive Randomized Hashing.. In *IJCAI'18*. 2861–2867.
[34] Dingqi Yang, Bin Li, and Philippe Cudré-Mauroux. 2016. POIsketch: Semantic Place Labeling over User Activity Streams. In *IJCAI'16*. 2697–2703.
[35] Dingqi Yang, Bin Li, Laura Rettig, and Philippe Cudré-Mauroux. 2017. HistoSketch: Fast Similarity-Preserving Sketching of Streaming Histograms with Concept Drift. In *ICDM'17*. IEEE, 545–554.
[36] Dingqi Yang, Bin Li, Laura Rettig, and Philippe Cudré-Mauroux. 2018. D2HistoSketch: Discriminative and Dynamic Similarity-Preserving Sketching of Streaming Histograms. *TKDE* 1 (2018), 1–14.
[37] Dingqi Yang, Bingqing Qu, Jie Yang, and Philippe Cudre-Mauroux. 2019. Revisiting User Mobility and Social Relationships in LBSNs: A Hypergraph Embedding Approach. In *WWW'19*. ACM, 2147–2157.
[38] Jaewon Yang and Jure Leskovec. 2015. Defining and evaluating network communities based on ground-truth. *KIS* 42, 1 (2015), 181–213.
[39] Dingyuan Zhu, Peng Cui, Daixin Wang, and Wenwu Zhu. 2018. Deep variational network embedding in wasserstein space. In *KDD'18*. ACM, 2827–2836.

SUPPLEMENTAL MATERIALS

Proof of the Uniformity Property

We present a brief proof of the uniformity of the generated samples (using Eq. 3), which states that the probability of selecting i is proportional to V_i. First, as $h_j(i) \sim Uniform(0,1)$ in Eq. 3, by applying the change of variable technique we have $\frac{-\log h_j(i)}{V_i} \sim Exp(V_i)$. We hereafter use $X_i = \frac{-\log h_j(i)}{V_i} \sim Exp(V_i)$ for the sake of notation simplicity. We now investigate the probability distribution of the minimum of $\{X_q | q \neq i\}$:

$$Pr(\min_{q\neq i} X_q > x) = Pr(\bigcap_{q\neq i}\{X_q > x\}) = \prod_{q\neq i} e^{-V_q x} = e^{-(\lambda - V_i)x} \tag{8}$$

where $\lambda = \sum_q V_q$. As X_i are independent of $\{X_q | q \neq i\}$, the conditional probability of sampling i given X_i is:

$$Pr(\min_{q\neq i} X_q > X_i | X_i) = e^{-(\lambda - V_i)X_i s} \tag{9}$$

By integrating over the distribution of $X_i \sim Exp(V_i)$, we obtain the probability of sampling i as:

$$Pr(\operatorname*{argmin}_q X_q = i) = \int_0^\infty V_i e^{-V_i x} e^{-(\lambda - V_i)x} dx = \frac{V_i}{\lambda} = \frac{V_i}{\sum_q V_q} \tag{10}$$

which means that the probability of selecting i is proportional to V_i. This completes the proof.

Proof of the Approximation Error Bound

We present a brief proof of the approximation error bound (Eq. 7). First, let $Y_j = \mathbb{1}_{S_j^a = S_j^b}$, where $\mathbb{1}_{cond}$ is an indicator function which is equal to 1 when *cond* is true and to 0 otherwise. Subsequently, the Hamming similarity between the embeddings of two nodes a and b can be formulated as the empirical mean of variables $Y_j \in [0,1]$:

$$H(S^a, S^b) = \frac{1}{L}\sum_{j=1}^{L} Y_j \tag{11}$$

Based on Hoeffding inequality [11], we then have:

$$Pr[|H - \mathbb{E}(H)| \geq \epsilon] \leq 2\exp(-2L\epsilon^2) \tag{12}$$

As $\mathbb{E}(H) = Sim_{NMM}$, we obtain the approximation error bound:

$$Pr[|H - Sim_{NMM}| \geq \epsilon] \leq 2\exp(-2L\epsilon^2) \tag{13}$$

This completes the proof.

Detailed Settings and Parameter Tuning for Baselines

We compare NodeSketch against a sizable collection of state-of-the-art techniques from three categories, i.e., classical graph embedding techniques (preserving cosine similarity), learning-to-hash techniques and data-independent sketching techniques (preserving Hamming similarity).

First, we consider the following graph embedding techniques:

- **DeepWalk**[5] [23] is a graph-sampling (via random-walk) based graph embedding technique. It first generates node sequences using random walks, and then feeds them to the SkipGram model to output node embeddings. Following the suggestion of the authors, we set the walk length to 40, the number of walks per node to 80, and the context window size to 10.
- **Node2Vec**[6] [9] extends DeepWalk by introducing a parameterized random walk method to balance the breadth-first search (return parameter p) and depth-first search (in-out parameter q) strategies to capture richer graph structures. Following the suggestions from the original paper, we tune p and q with a grid search over $p, q \in \{0.25, 0.05, 1, 2, 4\}$. We keep the other parameters same as for DeepWalk.
- **LINE**[7] [26] directly samples node pairs from a graph to learn graph embeddings; it first learns two sets of $d/2$-dimension node embeddings based on 1st and 2nd-order node proximity, respectively, and then concatenates them together. We set the total number of samples to 1 billion for Blog, PPI, Wiki and DBLP datasets and 10 billion for YouTube.
- **VERSE**[8] [29] directly samples node pairs to learn graph embeddings to preserve the node proximity measured by personalized PageRank. We tune the damping factor α of personalized PageRank using the method suggested by the authors, and leave all other parameters as default.
- **GraRep**[9] [4] factorizes the k-order transition matrix to generate node embeddings. It first separately learns k sets of d/k-dimension node embeddings capturing 1st to kth-order node proximity, respectively, and then concatenate them together. We tune k by searching over $\{1, 2, 3, 4, 5, 6\}$. When d/k is not an integer, we learn the first $k-1$ sets of $\lceil d/k \rceil$-dimension embeddings, and the last set of embeddings of dimension $d - (k-1)\lceil d/k \rceil$.
- **HOPE**[10] [22] factorizes the up-to-k-order node proximity matrix measured by Katz index using a generalized SVD method to learn node embeddings. The proposed generalized SVD method can scale up to large matrices. We search the optimal decay parameter β from 0.1 to 0.9 with a step of 0.1 (further multiplied by the spectral radius of the adjacency matrix of the graph).
- **NetMF**[11] [24] derives the closed form of DeepWalk's implicit matrix, and factorizes this matrix to output node embeddings. Following the suggestion made by the authors, we tune the implicit window size T within $\{1, 10\}$.

Second, we consider the following learning-to-hash techniques, which are among the best-performing techniques in [18].

- **Spectral Hashing (SH)**[12] [32] learns the hash code of the input data by minimizing the product of the similarity between each pair of the input data samples and the Hamming distance between the corresponding pairs of hash code.
- **Iterative quantization (ITQ)**[13] [8] first processes the input data by reducing the dimension using PCA, and then performs quantization to learn the hash code of the input data via alternative optimization.

[5] https://github.com/phanein/deepwalk

[6] https://github.com/snap-stanford/snap/tree/master/examples/node2vec

[7] https://github.com/tangjianpku/LINE

[8] https://github.com/xgfs/verse

[9] https://github.com/ShelsonCao/GraRep

[10] http://git.thumedia.org/embedding/HOPE

[11] https://github.com/xptree/NetMF

[12] //www.cs.huji.ac.il/ yweiss/SpectralHashing/

[13] https://github.com/jfeng10/ITQ-image-retrieval

- **Scalable Graph Hashing (SGH)**[14] [14] learns the hash code of the input data by minimizing the difference between the similarity of input data and the Hamming similarity of the corresponding pairs of hash code. It uses the feature transformation to approximate the similarity matrix without explicitly computing the similarity matrix, in order to make it scalable to large datasets.
- **INH-MF**[15] [18] is the first learning-to-hash technique proposed for the graph embedding problem. It learns the hash code of nodes in a graph via matrix factorization while preserving the high-order node proximity in the graph. It also incorporates a hamming subspace learning process to improve the efficiency of the learning process. We set the ratio for subspace learning to 100%, to let it achieve its optimal performance w.r.t. the quality of the learnt node embeddings.

Finally, we consider the following sketching techniques:

- **NetHash**[16] [33] is the first work applying sketching techniques on *attributed* graph embedding problems. It first builds a parent-pointer-tree for each node in a graph, and then recursively applies minhash on the *attributes* of nodes in the tree from bottom to top. However, it cannot be directly applied to our graphs, as it requires a set of attributes for each node in the graph. To remedy this issue, we use a node's direct neighbors as its attributes, which represents the node's context in the graph. As suggested by the authors, we search the optimal tree depth in $\{1, 2, 3\}$.
- **KatzSketch** is one of our proposed baselines. It first computes the high-order node proximity matrix using Katz index, which is the best performing high-order proximity measure for graph embedding problem as suggested by [22]. Then, it directly generates sketches based on the high-order node proximity vectors of nodes (rows of the obtained proximity matrix). We search the optimal decay parameter β used by Katz index in the same way as for HOPE.
- **NodeSketch(NoSLA)** is a variant of our proposed NodeSketch by using the original adjacency matrix rather than the SLA adjacency matrix. We search the optimal parameters k and α using the same strategy as for our method NodeSketch (see below).
- **NodeSketch**[17] is our proposed technique. We search the optimal order of proximity k up to 6 and the optimal decay parameter α from 0.0001 to 1 on a log scale.

[14] https://github.com/jiangqy/SGH-IJCAI2015
[15] https://github.com/DefuLian/network
[16] https://github.com/williamweiwu/williamweiwu.github.io/tree/master/Graph_Network%20Embedding/NetHash
[17] https://github.com/eXascaleInfolab/NodeSketch