Search Efficient Binary Network Embedding

DAOKUN ZHANG and JIE YIN, The University of Sydney
XINGQUAN ZHU, Florida Atlantic University
CHENGQI ZHANG, University of Technology Sydney

Traditional network embedding primarily focuses on learning a continuous vector representation for each node, preserving network structure and/or node content information, such that off-the-shelf machine learning algorithms can be easily applied to the vector-format node representations for network analysis. However, the learned continuous vector representations are inefficient for large-scale similarity search, which often involves finding nearest neighbors measured by distance or similarity in a continuous vector space. In this article, we propose a search efficient binary network embedding algorithm called BinaryNE to learn a binary code for each node, by simultaneously modeling node context relations and node attribute relations through a three-layer neural network. BinaryNE learns binary node representations using a stochastic gradient descent-based online learning algorithm. The learned binary encoding not only reduces memory usage to represent each node, but also allows fast bit-wise comparisons to support faster node similarity search than using Euclidean or other distance measures. Extensive experiments and comparisons demonstrate that BinaryNE not only delivers more than 25 times faster search speed, but also provides comparable or better search quality than traditional continuous vector based network embedding methods. The binary codes learned by BinaryNE also render competitive performance on node classification and node clustering tasks. The source code of the BinaryNE algorithm is available at https://github.com/daokunzhang/BinaryNE.

CCS Concepts: • **Information systems** → **Nearest-neighbor search**; *Data mining*;

Additional Key Words and Phrases: Network embedding, binary coding, similarity search, efficiency

ACM Reference format:
Daokun Zhang, Jie Yin, Xingquan Zhu, and Chengqi Zhang. 2021. Search Efficient Binary Network Embedding. *ACM Trans. Knowl. Discov. Data* 15, 4, Article 61 (May 2021), 27 pages.
https://doi.org/10.1145/3436892

1 INTRODUCTION

Networks offer a natural way to capture intrinsic relationships between entities—social interactions among people, collaborations between co-workers, biological interactions among proteins,

This work is supported by a joint CRP research fund between the University of Sydney and Data61, CSIRO, the US National Science Foundation (NSF) through grants IIS-1763452 and CNS-1828181, and the Australian Research Council (ARC) through grants LP160100630 and DP180100966.
Authors' addresses: D. Zhang and J. Yin, Discipline of Business Analytics, The University of Sydney, City Road, Camperdown, NSW 2006, Australia; emails: {daokun.zhang, jie.yin}@sydney.edu.au; X. Zhu, Department of Computer & Electrical Engineering and Computer Science, Florida Atlantic University, Glades Road, Boca Raton, FL 33431; email: xzhu3@fau.edu; C. Zhang, Australian Artificial Intelligence Institute, Faculty of Engineering and Information Technology, University of Technology Sydney, Broadway, Ultimo, NSW 2007, Australia; email: chengqi.zhang@uts.edu.au.

1556-4681/2021/05-ART61 $15.00
https://doi.org/10.1145/3436892

flow-of-funds between financial transactions, and so on. Networks can be modeled as a graph, where nodes indicate entities and edges indicate pairwise relationships between entities. Searching similar nodes in networks is an essential network analytic task, which directly benefits many real-world applications. For social security, potential terrorists can be detected by searching people with the same organization associations in the communication networks or online social networks. On e-commerce platforms, personalized recommendations can be effectively delivered by searching users with similar interests among users' social relations. In social networks, social actors with important structural roles, such as the center of a star or the hole spanner, can be discovered by searching nodes with the same properties among the whole network. Searching similar nodes can also benefits other tasks, such as web page retrieval in the World Wide Web [14], link prediction in social networks [43], and identity resolution in bibliographic collaboration networks [66].

To enable similarity search over networks, structural properties including common neighbors and structural context have been leveraged to estimate the similarity between nodes. Representative algorithms include Personalized PageRank [14], SimRank [18], P-Rank [68], TopSim [25], and Panther [66]. However, these methods suffer from the following major drawback:

— **Incapable of capturing node content similarity.** In addition to network structure, network nodes are often associated with rich content, such as user profiles in social networks, texts in web page networks. Node content contains crucial information that provides direct evidence to measure node similarity. The structure based similarity search methods fail to leverage the similarity measured by node content, leading to suboptimal search results.

Recently, network embedding [35, 46, 58, 64] has been proposed to facilitate network analytic tasks, which aims to embed network nodes into a low-dimensional continuous vector space, by preserving network structure and/or node content information. After learning new node representations, network analytic tasks can be easily carried out by applying off-the-shelf machine learning algorithms to the new embedding space. However, such a machine learning-driven network embedding paradigm often results in node representations that are inefficient for large-scale similarity search in terms of both time and memory. Consider a network with 10 million nodes, if we learn 200-dimensional continuous vector-format representations for each node, it requires 15GB memory to accommodate these representations using standard double precision numbers, which is prohibitively intractable for general computing devices. Given a query node, if we want to find its similar nodes among the whole network using Euclidean distance in the continuous embedding space, it requires 2 billion times of floating-point product operation and 2 billion times of floating-point addition operation, to measure distance between the query node and all other nodes in the network. The high computational cost makes it unsuitable for real-time retrieval systems that require responsive solutions. In summary, searching similar nodes with continuous node representations inevitably incurs high time and memory cost, resulting in unsatisfactory performance on large-scale networks.

As an alternative way of nearest neighbor search, locality-sensitive hashing techniques [9, 38, 39, 67] have been proposed to improve search efficiency. They transform the numeric vector data into binary (or integer) codes that preserve the similarity in the original space. As a consequence, data can be stored with low memory cost and similarity search can be conducted efficiently by calculating the Hamming distance, e.g., between node binary codes with bit-wise operations. Borrowing the idea of hashing, we propose to learn binary representations for network nodes, i.e., transforming network nodes into binary codes rather than numeric vectors, such that the memory and time efficiency for similarity search can be significantly improved. Despite its potential, the binary node representation learning is confronted with the following two challenges:

— **Heterogeneity.** To guarantee search accuracy, binary node representations are expected to capture the information from both network structure and node content, i.e., preserving node similarity at both structure level and node content level. However, network structure and node content are not always consistent or correlated with each other. How to fuse information from these two heterogeneous sources into binary codes and make them complement rather than deteriorate each other is a big challenge.
— **Scalability.** To learn binary node representations that well preserve network structure and node content, we need to jointly optimize the respective objectives, such as random walk context node prediction [35] and node attribute reconstruction [59], which are usually nonlinear functions of node representations. With the constraint that each dimension should take value from $\{0, 1\}$, finding the optimal solution to node binary representations that optimizes the nonlinear objective function is a nonlinear integer programming problem, which has been proved NP-hard [15]. When it comes to large-scale networks with millions or billions of nodes/edges, and high-dimensional node content features, it is impossible to find the exact optimal solutions in an efficient way. To make the learning highly scalable, in the promise of assuring the quality of solutions, approximation techniques together with online or parallel learning strategies need to be developed.

An intuitive solution to binary network embedding is to first learn continuous node representations and then binarize them into binary codes with conventional hashing techniques. However, because converting continuous embeddings into binary codes inevitably causes information loss, the learned binary codes cannot accurately capture node similarity at both structure and content level. As a result, as demonstrated later in our experiments, this two-step learning strategy usually results in suboptimal search accuracy.

The binary network embedding problem can also be addressed by supervised hash learning algorithms [7, 69] originally designed for non-relational data, with the required node similarity labels estimated from network structure and/or node attributes in advance. However, directly applying supervised hash learning algorithms [7, 69] to large-scale networks is confronted with the following three key challenges: (1) the high complexity for calculating all pairwise node similarity values from network structure with the objective of preserving high-order proximities as well as node attributes that are usually high dimensional; (2) the information loss caused by quantizing the continuous similarity values into binary similarity labels; and (3) most seriously, the imbalanced similarity relations, i.e., the number of dissimilar node pairs is overwhelmingly larger than the number of similar node pairs on graphs.

In this article, we propose a novel Binary Network Embedding algorithm, called BinaryNE, to learn binary node representations directly from both network structure and node content features for efficient similarity search. BinaryNE learns binary node representations by simultaneously modeling node context and node attribute relations through a three-layer neural network, with the objective of capturing node similarity in both network structure and node content. To obtain binary codes, the *sign* function $\text{sgn}(\cdot)$ is employed as the activation function in the hidden layer. However, as the gradient of the *sign* function is zero almost everywhere, traditional gradient decent based optimization strategies are infeasible for learning parameters, which is known as the *ill-posed gradient* problem. To address this problem, we adopt the state-of-the-art continuation technique [2, 7] and develop an online stochastic gradient descent algorithm to learn parameters, which guarantees the great scalability of BinaryNE. In the learning process, node binary representations are sequentially updated with regards to the given node-context or node-attribute occurrence pair, by maximizing the occurrence probability of the context node or node attribute conditioned on the given node. Reflected on the binary representation space, nodes are dragged by their

context nodes and attributes back-and-forth until an equilibrium status. In this way, nodes sharing overlapping sets of context nodes and attributes are finally represented closely in the binary embedding space. Our learning strategy can effectively capture the similarity in network structure and node attributes, and avoid the learning process to be dominated by the overwhelmingly large quantities of dissimilar node pairs to the most extent, which seriously challenges traditional supervised hash learning algorithms [7, 69]. Experiments on six real-world networks show that BinaryNE exhibits much lower memory usage and quicker search speed than the state-of-the-art network embedding methods, while achieving appealing search accuracy. BinaryNE's binary node representations also deliver competitive results on node classification and node clustering tasks.

The main contribution of this article is threefold:

- We analyze the feasibility and advantage of learning binary node representations as a solution to efficient node similarity search over large-scale networks involving node attributes.
- We propose a new algorithm called BinaryNE to effectively learn high-quality binary node representations from both network structure and node features, together with an efficient stochastic gradient descent based solution.
- Extensive experiments on six real-world networks validate the superiority of BinaryNE on similarity search in terms of search accuracy, memory usage and efficiency, as well as its competitive performance on node classification and node clustering.

The remainder of this article is organized as follows. In Section 2, we review the related work, including network embedding and node similarity search. In Section 3, we give a formal definition of binary network embedding and review the DeepWalk algorithm as preliminaries. The proposed BinaryNE algorithm is described in Section 4, followed by experiments presented in Section 5. Finally, we conclude this article in Section 6.

2 RELATED WORK

In this section, we review two lines of related work: network embedding that aims to learn node vector-format representations, and node similarity search that is realized by directly estimating node similarity from network structure.

2.1 Network Embedding

According to whether the learned node representations take continuous or discrete values, the network embedding techniques can be divided into two groups: continuous network embedding and discrete network embedding.

2.1.1 Continuous Network Embedding. Depending on whether node content features are leveraged, continuous network embedding techniques can be divided into two groups: *structure preserving network embedding* and *attributed network embedding*.

Structure preserving network embedding learns node representations from only network structure. DeepWalk [35] first encodes network structure into a set of random walk sequences, and then employs Skip-Gram [31] to learn node representations that capture structural context similarity. node2vec [11] extends DeepWalk to better balance the local structure preserving and global structure preserving objective by leveraging biased random walks. LINE [46] learns node representations through directly modeling the first-order proximity (the proximity between connect nodes) and the second-order proximity (the proximity between nodes sharing direct neighbors). Although DeepWalk, node2vec, and LINE are all based on the Skip-Gram model explicitly or implicitly, [37] proves their equivalence to a unified matrix factorization formulation. GraRep [5] further extends LINE [46] to capture high-order proximities through the matrix factorization

version of Skip-Gram [26]. M-NMF [55] complements the local structure proximity with the intra-community proximity to learn community-aware node representations. DNGR [6] first obtains high-dimensional structure preserving node representations through the proposed random surfing method, and then utilizes the *stacked denoising autoencoder* [52] to learn low-dimensional representations. SNDE [53] employs deep autoencoder to learn deep nonlinear node representations, by reconstructing node adjacent matrix representations to preserve the second-order proximity and penalizing the representation difference of connected nodes to preserve the first-order proximity.

Attributed network embedding learns node representations by coupling node attributes with network structure. TADW [58] first proves the equivalence between DeepWalk [35] and a matrix factorization formulation, and then proposes to incorporate rich node text features into network embedding through inductive matrix factorization [33]. Through penalizing the distance of connected nodes in the embedding space, HSCA [62] enforces TADW with the first-order proximity to obtain more informative node representations. UPP-SNE [63] learns node representations by performing a structure-aware non-linear mapping on node content features. CANE [49] learns context-aware node embeddings by applying the mutual attention mechanism on the attributes of connected nodes. MVC-DNE [59] applies deep multi-view learning technique to fuse information from network structure and node content into node representations. GraphSAGE [13] first takes node content features as node representations, and then iteratively updates node representations by aggregating representations of neighboring nodes. AANE [16] employs symmetric matrix factorization [22] to obtain node representations that capture attribute affinity, and simultaneously penalizes the representation difference between connected nodes. SINE [65] learns node representations for large-scale incomplete attributed networks by using node representations to simultaneously predict context nodes and node attributes. These network embedding algorithms learn task-general node representations in an unsupervised setting, where node class labels are not provided.

Recently, several supervised network embedding algorithms have also been proposed, such as DMF [61], TriDNR [34], DDRW [28], MMDW [50], and LANE [17], with the objective of learning discriminative node representations by exerting the power of available node labels. Built upon the success of deep neural networks on grid-structured data (e.g., images), graph neural networks, such as graph convolutional networks (GCN) [21] and graph attention networks [51], generalize to learning node embeddings on graphs by passing, transforming, and aggregating node features from the local neighborhood. The generated node embeddings can then be used as input to any differentiable prediction layer, e.g., for node classification or link prediction.

All of the above network embedding techniques primarily consider embedding network nodes into a continuous Euclidean space, which could be favored by node classification tasks to achieve excellent performance. However, calculating pairwise similarity between nodes in a continuous space is computationally expensive, making it infeasible to perform node similarity search on large-scale networks.

2.1.2 Discrete Network Embedding. Graph-based hashing [30, 56] has been proposed to learn binary codes for traditional non-relational data, such as images. This kind of hashing techniques work on the graphs constructed from data points by calculating their pairwise similarity. The constructed graph reflects the geographical data distribution in their original feature space, and helps learn similar binary codes for closely located data points. The graphs considered by graph based hashing are artificially constructed from the input data, which are essentially different from real-world graphs where edges are naturally formed resulting from interactions between entities (nodes). In real-world natural graphs, connected nodes unnecessarily exhibit similar features [3, 45]. This breaks the assumption made by graph-based hashing [30, 56] that connected nodes are

located closely in the Euclidean feature space. Therefore, directly applying graph based hashing to natural graphs may result in unsatisfactory performance.

Very recently, several embedding algorithms have been proposed to learn discrete node representations for real-world networks. For efficient node retrieval, Bernoulli Network Embedding [32] learns binary node representations by modeling the generation of each dimension as a Bernoulli random test. KD-coding [8] is proposed to learn discrete word embeddings with the Skip-Gram [31] model, which can also be coupled with DeepWalk [35] to learn discrete node representations. Through explicitly capturing high-order structure proximity, INH-MF [29] learns binary codes for network nodes with matrix factorization. DNE [40] learns binary node representations to speed up node classification. However, the above methods cannot support accurate search, because node content features are simply ignored. In addition, DNE is a supervised binary network embedding algorithm that requires node labels to be provided, which is different from our research that aims to learn binary node representations in an unsupervised setting.

NetHash [57] is the first algorithm proposed to generate discrete node representations that encode both network structure and node content features. It applies the MinHash technique [4] to the union of tree-structured neighboring node features. As the learned discrete embeddings do not take binary values, similarity search with such embeddings tends to be inefficient. In this work, we aim to learn binary node representations that are directly optimized with binarization to enable similarity search efficacy and efficiency. BANE [60] is another algorithm that learns binary node representations from network structure and node attributes. BANE first constructs the Weisfeiler–Lehman proximity matrix [41] which enables neighboring node attribute vector aggregation and then performs binary factorization on the proximity matrix to obtain binary codes for each node. The Weisfeiler–Lehman matrix representation heavily relies on the homophily assumption and tends to be less informative when neighboring nodes have discrepant attributes, which is, however, very common in real-world networks. Hence, BANE is less effective in retrieving similar nodes, as demonstrated later in the experiments. DGCN-BinCF [54] is proposed to learn binary node representations for bipartite networks formed by users and items together with their implicit feedback, with the purpose of binary collaborative filtering. To capture user-item relations implied by implicit feedback, GCN [21] is employed to learn their network embeddings. The structure information carried by network embeddings of users and items are then distilled into their binary codes to minimize the rank loss for recommendation. DGCN-BinCF is performed on the heterogeneous bipartite networks under the supervised recommendation setting, which is different from our problem of unsupervised binary node representation learning on homogeneous attributed networks.

2.2 Node Similarity Search

To enable similarity search over networks, various metrics have been proposed to measure the structural relatedness between nodes. Bibliographic Coupling [20] and Co-citation [42] measure node similarity by counting the number of common neighbors. Other common neighbor-based metrics include Jaccard's coefficient, Salton's coefficient, the Adamic/Acar coefficient [1], and the like. This type of metrics are incapable of capturing the similarity between nodes sharing no common neighbors. SimRank [18] estimates node similarity recursively with the principle that two nodes are similar if they have connections with similar nodes. Because calculating SimRank similarity is computationally expensive, other algorithms, like TopSim [25] and [23], are proposed to reduce its time complexity. P-Rank [68] enhances SimRank by jointly modeling both in- and out-link relationships for node structural similarity estimation. VertexSim [48] represents each node as a convex combination of anchor nodes by optimizing a geometric objective, and then measures node similarity with the new representations. The above metrics only capture the similarity

relying on the connectivity among the local neighborhood, but neglect the *structural equivalence* between nodes sharing similar structural roles while being distantly located. [19] justifies a series of axiomatic properties that should be satisfied by a role similarity measure, and proposes RoleSim, a role similarity measure, which is calculated in an iterative way and is proved to satisfy all the justified properties. Panther [66] estimates local structural similarity between pairwise nodes through their co-occurrence frequencies in randomly sampled paths. Panther++ [66] augments Panther with structural role similarity by measuring the difference in neighboring node co-occurrence frequency distributions.

Calculating the aforementioned structure similarity metrics between all pairwise nodes, which is necessary for exact node similarity search, is time-consuming, with a time complexity at least quadratic to the number of nodes. Moreover, the above structure similarity metrics fail to capture the similarity measured by node features. The two limitations make the existing structural similarity estimation based search methods unsuitable for large-scale networks with rich node features.

3 PROBLEM DEFINITION AND PRELIMINARIES

In this section, we give a formal definition of the binary network embedding problem, followed by a review on the preliminaries of DeepWalk.

3.1 Problem Definition

Assume we are given a network $G = (\mathcal{V}, \mathcal{E}, \mathcal{A}, X)$, where $\mathcal{V}$ is the set of nodes, $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of edges, and $\mathcal{A}$ is the set of attributes. $X \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{A}|}$ is the node feature matrix, with each element $X_{ij} \geq 0$ indicating the occurrence times/weights of attribute $a_j \in \mathcal{A}$ at node $v_i \in \mathcal{V}$. Here, we assume node attributes take the discrete values, which is natural for text features in web page/citation networks and most user profile features in social networks, like gender, affiliation, and education type. In case of the numeric node attributes like age, we can easily discretize them into discrete/categorical values through transforming them into interval or bin based features.

The BinaryNE algorithm aims to learn binary representations for network nodes, i.e., learning a mapping function $\Phi : v_i \in \mathcal{V} \mapsto \{+1, -1\}^d$, where d is the dimension of the embedding space. The learned binary node representations $\Phi(v_i)$ are expected to satisfy the following two properties: (1) *low-dimensional*: the dimension d should be much smaller than the dimension of adjacent-matrix node representations, $|\mathcal{V}|$, the original high-dimensional node representations, for the sake of search efficiency; (2) *informative*: the learned binary node representations should capture node similarity measured by both network structure and node content features to guarantee the quality of node similarity search.

3.2 Preliminaries: DeepWalk

Borrowing the idea of Skip-Gram model [31], which learns word representations by preserving context similarity, DeepWalk leverages random walks to generate node context and represents nodes sharing similar context closely in the new embedding space. Given a random walk with length L, $\{v_{r_1}, v_{r_2}, \ldots, v_{r_i}, \ldots v_{r_L}\}$, for each node v_{r_i}, DeepWalk learns its representation by using it to predict its context nodes, which is realized by maximizing the occurrence probability of context nodes conditioned on this node:

$$\min_{\Phi} -\log \mathrm{P}(\{v_{r_{i-t}}, \ldots, v_{r_{i+t}}\} \setminus v_{r_i} | v_{r_i}), \tag{1}$$

where $\{v_{r_{i-t}}, \ldots, v_{r_{i+t}}\} \setminus v_{r_i}$ are the context nodes of v_{r_i} within a window size of t. Here, the window size refers to the number of context nodes to be collected, which are located before or after the given target node in the given random walk.

Using the conditional independence assumption, the probability $\mathrm{P}(\{v_{r_{i-t}}, \ldots, v_{r_{i+t}}\} \setminus v_{r_i} | v_{r_i})$ can be calculated as

$$\mathrm{P}(\{v_{r_{i-t}}, \ldots, v_{r_{i+t}}\} \setminus v_{r_i} | v_{r_i}) = \prod_{j=i-t, j\neq i}^{i+t} \mathrm{P}(v_{r_j} | v_{r_i}). \tag{2}$$

Following [63], after a set of random walks are generated, we can formulate the overall optimization problem as

$$\min_{\Phi} -\sum_{i=1}^{|\mathcal{V}|} \sum_{j=1}^{|\mathcal{V}|} n(v_i, v_j) \log \mathrm{P}(v_j | v_i), \tag{3}$$

where $n(v_i, v_j)$ is the occurrence time of node context pair (v_i, v_j) collected from all random walks with t window size and $\mathrm{P}(v_j|v_i)$ is modeled by softmax:

$$\mathrm{P}(v_j | v_i) = \frac{\exp(\Phi(v_i) \cdot \Psi(v_j))}{\sum_{k=1}^{|\mathcal{V}|} \exp(\Phi(v_i) \cdot \Psi(v_k))},$$

where $\Psi(\cdot)$ is the node embedding vector when the node act as a context node.

The overall optimization problem can be solved by iteratively sampling a node context pair (v_i, v_j) and minimizing the following partial objective:

$$O_{ij}^{s} = -\log \mathrm{P}(v_j | v_i). \tag{4}$$

4 BINARY NETWORK EMBEDDING

This section details the optimization problem that we formulate for the binary network embedding, followed by the solution on how to solve it efficiently.

4.1 The Optimization Problem

Our objective is to learn informative binary network embeddings, with both network structure and node content features well preserved. The idea of DeepWalk can be borrowed for capturing network structure. To capture node content level similarity, we try to represent nodes sharing similar attributes closely in the low-dimensional space. To achieve this goal, we borrow the idea of Paragraph Vector [24], which learns document representations through modeling document-word relations with the Skip-Gram model. This ensures that similar representations can be learned for documents with similar words. Here, to encode node attributes into binary node representations, we apply the idea of Skip-Gram [31] again, by using each node to predict its attributes. For each node attribute co-occurrence pair (v_i, a_j), we minimize the following objective:

$$O_{ij}^{a} = -\log \mathrm{P}(a_j | v_i). \tag{5}$$

We illustrate the architecture of the proposed BinaryNE algorithm in Figure 1, which is a three-layer neural network: the first layer is the one-hot representation for each node v_i, the hidden layer is the binary node representation $\Phi(v_i) \in \{+1, -1\}^d$ constructed from the input layer, and the output layer is the softmax conditional probability $\mathrm{P}(v_j|v_i)$ and $\mathrm{P}(a_j|v_i)$ for each context node v_j and each attribute a_j, modeled through node binary representations in the hidden layer.

Given node v_i's one-hot representation $\mathbf{p}^i \in \mathbb{R}^{|\mathcal{V}|}$ with $\mathbf{p}^i_k = 1$ for $k = i$, and $\mathbf{p}^i_k = 0$ for $k \neq i$. The binary node representation $\Phi(v_i)$ in the hidden layer is constructed by performing a linear transformation on $\mathbf{p}^i$ and activating it with the *sign* function:

$$\begin{aligned} \Phi(v_i) &= \left[\mathrm{sgn}(\mathbf{p}^i \cdot W^{in}_{:1}), \mathrm{sgn}(\mathbf{p}^i \cdot W^{in}_{:2}), \ldots, \mathrm{sgn}(\mathbf{p}^i \cdot W^{in}_{:d})\right]^{\mathrm{T}} \\ &= \left[\mathrm{sgn}(W^{in}_{i1}), \mathrm{sgn}(W^{in}_{i2}), \ldots, \mathrm{sgn}(W^{in}_{id})\right]^{\mathrm{T}}, \end{aligned} \tag{6}$$

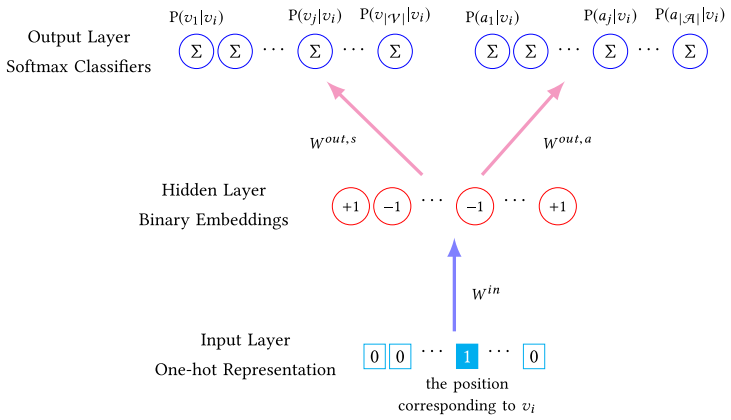


Fig. 1. The model architecture of BinaryNE. For each node v_i, BinaryNE learns its binary representation by using it to predict its context node v_j and its attribute a_j.

where $W^{in}_{:k}$ is the k-th column of $W^{in} \in \mathbb{R}^{|\mathcal{V}|\times d}$ (the weight matrix from the input layer to the hidden layer) and $\text{sgn}(\cdot)$ is the *sign* function, which is defined as

$$\text{sgn}(x) = \begin{cases} +1, & \text{if } x \geqslant 0, \\ -1, & \text{otherwise.} \end{cases}$$

In the output layer, for the node context pair (v_i, v_j), we model the probability $\text{P}(v_j|v_i)$ with softmax:

$$\text{P}(v_j|v_i) = \frac{\exp(\Phi(v_i) \cdot W^{out,s}_{:j})}{\sum_{k=1}^{|\mathcal{V}|} \exp(\Phi(v_i) \cdot W^{out,s}_{:k})},$$

where $W^{out,s}_{:j}$ is the j-th column of $W^{out,s} \in \mathbb{R}^{d\times|\mathcal{V}|}$ (the weight matrix from the hidden layer to the output layer for predicting context node). Similarly, for the node attribute co-occurrence pair (v_i, a_j), we model the probability $\text{P}(a_j|v_i)$ as

$$\text{P}(a_j|v_i) = \frac{\exp(\Phi(v_i) \cdot W^{out,a}_{:j})}{\sum_{k=1}^{|\mathcal{A}|} \exp(\Phi(v_i) \cdot W^{out,a}_{:k})},$$

where $W^{out,a}_{:j}$ is the j-th column of $W^{out,a} \in \mathbb{R}^{d\times|\mathcal{A}|}$ (the weight matrix from the hidden layer to the output layer for predicting node attribute).

To learn informative binary node embeddings, we integrate the structure proximity preserving objective in Equation (4) with the node attribute similarity preserving objective in Equation (5), and obtain the following overall optimization problem:

$$\min_{\Phi} O, \tag{7}$$

where

$$O = -\alpha_1 \sum_{i=1}^{|\mathcal{V}|}\sum_{j=1}^{|\mathcal{V}|} n(v_i, v_j)\text{log}\text{P}(v_j|v_i) - \alpha_2 \sum_{i=1}^{|\mathcal{V}|}\sum_{j=1}^{|\mathcal{A}|} X_{ij}\text{log}\text{P}(a_j|v_i). \tag{8}$$

Here, α_1 and α_2 are the trade-off parameters to balance the contribution of the structure preserving objective and the node content preserving objective. Specifically, we set α_1 and α_2 to the reciprocal

of the number of node context pairs and the number of observed node attribute co-occurrence pairs, respectively:

$$\alpha_1 = \frac{1}{\sum_{i=1}^{|\mathcal{V}|}\sum_{j=1}^{|\mathcal{V}|} n(v_i, v_j)}, \quad \alpha_2 = \frac{1}{\sum_{i=1}^{|\mathcal{V}|}\sum_{j=1}^{|\mathcal{A}|} X_{ij}},$$

which essentially makes the objective Equation (7) perform minimization over the averaged $-\log \mathrm{P}(v_j|v_i)$ and $-\log \mathrm{P}(a_j|v_i)$. In Equation (8), only the non-zero entries of $n(v_i, v_j)$ and X_{ij} are considered, whose numbers are much smaller than $|\mathcal{V}| \times |\mathcal{V}|$ and $|\mathcal{V}| \times |\mathcal{A}|$, respectively.

4.2 Solving the Optimization Problem

As the derivative of the *sign* function used to construct binary codes is zero almost everywhere, solving the optimization problem (7) with gradient descent is *ill-posed*. Following [7], we approximate the non-smooth *sign* function $\mathrm{sgn}(x)$ with its smooth proxy $\tanh(\beta x)$, which satisfies the following property:

$$\lim_{\beta \to \infty} \tanh(\beta x) = \mathrm{sgn}(x).$$

Using tanh, node representation $\Phi(v_i)$ in Equation (6) is constructed as

$$\Phi(v_i) = \left[\tanh\left(\beta W_{i1}^{in}\right), \tanh\left(\beta W_{i2}^{in}\right), \ldots, \tanh\left(\beta W_{id}^{in}\right)\right]^{\mathrm{T}}. \tag{9}$$

With this continuous approximation, we can solve the optimization problem (7) with stochastic gradient descent. At each iteration, we randomly select a node context pair (v_i, v_j) according to the distribution of $n(v_i, v_j)$ or a node attribute co-occurrence pair (v_i, a_j) according to the distribution of X_{ij}, and then update parameters towards minimizing the corresponding partial objective O_{ij}^s in Equation (4) or O_{ij}^a in Equation (5).

Given a sampled node context pair (v_i, v_j), for training efficiency, we adopt negative sampling [12] to approximate the partial objective O_{ij}^s in Equation (4) as

$$O_{ij}^s = -\log \sigma\left(\Phi(v_i) \cdot W_{:j}^{out,s}\right) - \sum_{k:v_k \in \mathcal{V}_{neg}} \log \sigma\left(-\Phi(v_i) \cdot W_{:k}^{out,s}\right), \tag{10}$$

where $\mathcal{V}_{neg}$ is the set of sampled negative nodes and $\sigma(\cdot)$ is the sigmoid function. Then, we update the parameters with gradient descent:

$$\begin{aligned}
W_{i:}^{in} &= W_{i:}^{in} - \eta \frac{\partial O_{ij}^s}{\partial W_{i:}^{in}}, \\
W_{:j}^{out,s} &= W_{:j}^{out,s} - \eta \frac{\partial O_{ij}^s}{\partial W_{:j}^{out,s}}, \\
W_{:k}^{out,s} &= W_{:k}^{out,s} - \eta \frac{\partial O_{ij}^s}{\partial W_{:k}^{out,s}}, \text{ for } v_k \in \mathcal{V}_{neg},
\end{aligned} \tag{11}$$

where η is the learning rate. The gradients are calculated as

$$\begin{aligned}
\frac{\partial O_{ij}^s}{\partial W_{ir}^{in}} &= \beta\Big[1-\tanh\Big(\beta W_{ir}^{in}\Big)^2\Big]\Big[\sigma\Big(\Phi(v_i)\cdot W_{:j}^{out,s}\Big)-1\Big]W_{rj}^{out,s} \\
&+ \beta\Big[1-\tanh\Big(\beta W_{ir}^{in}\Big)^2\Big]\sum_{k:v_k\in\mathcal{V}_{neg}}\sigma\Big(\Phi(v_i)\cdot W_{:k}^{out,s}\Big)W_{rk}^{out,s}, \\
\frac{\partial O_{ij}^s}{W_{:j}^{out,s}} &= \Big[\sigma\Big(\Phi(v_i)\cdot W_{:j}^{out,s}\Big)-1\Big]\Phi(v_i), \\
\frac{\partial O_{ij}^s}{W_{:k}^{out,s}} &= \sigma\Big(\Phi(v_i)\cdot W_{:k}^{out,s}\Big)\Phi(v_i),\ \text{for } v_k\in\mathcal{V}_{neg}.
\end{aligned}$$

Similarly, after a node attribute co-occurrence pair (v_i, a_j) is sampled, with negative sampling [12], the partial objective O_{ij}^a in Equation (5) is approximated as

$$O_{ij}^a = -\log\sigma\Big(\Phi(v_i)\cdot W_{:j}^{out,a}\Big) - \sum_{k:v_k\in\mathcal{A}_{neg}}\log\sigma\Big(-\Phi(v_i)\cdot W_{:k}^{out,a}\Big), \tag{12}$$

where $\mathcal{A}_{neg}$ is the set of sampled negative attributes. We then update the parameters with gradient descent

$$\begin{aligned}
W_{i:}^{in} &= W_{i:}^{in} - \eta\frac{\partial O_{ij}^a}{\partial W_{i:}^{in}}, \\
W_{:j}^{out,a} &= W_{:j}^{out,a} - \eta\frac{\partial O_{ij}^a}{\partial W_{:j}^{out,a}}, \\
W_{:k}^{out,a} &= W_{:k}^{out,a} - \eta\frac{\partial O_{ij}^a}{\partial W_{:k}^{out,a}},\ \text{for } a_k\in\mathcal{A}_{neg}.
\end{aligned} \tag{13}$$

The gradients are calculated as

$$\begin{aligned}
\frac{\partial O_{ij}^a}{\partial W_{ir}^{in}} &= \beta\Big[1-\tanh\Big(\beta W_{ir}^{in}\Big)^2\Big]\Big[\sigma\Big(\Phi(v_i)\cdot W_{:j}^{out,a}\Big)-1\Big]W_{rj}^{out,a} \\
&+ \beta\Big[1-\tanh\Big(\beta W_{ir}^{in}\Big)^2\Big]\sum_{k:v_k\in\mathcal{A}_{neg}}\sigma\Big(\Phi(v_i)\cdot W_{:k}^{out,a}\Big)W_{rk}^{out,a}, \\
\frac{\partial O_{ij}^a}{W_{:j}^{out,a}} &= \Big[\sigma\Big(\Phi(v_i)\cdot W_{:j}^{out,a}\Big)-1\Big]\Phi(v_i), \\
\frac{\partial O_{ij}^a}{W_{:k}^{out,a}} &= \sigma\Big(\Phi(v_i)\cdot W_{:k}^{out,a}\Big)\Phi(v_i),\ \text{for } a_k\in\mathcal{A}_{neg}.
\end{aligned}$$

After the parameters are learned, for node $v_i\in\mathcal{V}$, we construct its embedding $\Phi(v_i)$ as

$$\Phi(v_i)_r = \begin{cases} +1, & \text{if } \tanh(\beta W_{ir}^{in}) \geqslant 0, \\ -1, & \text{if } \tanh(\beta W_{ir}^{in}) < 0. \end{cases} \tag{14}$$

To obtain binary codes for efficient Hamming distance calculation, we store the -1 value of $\Phi(v_i)_r$ as 0 instead.

Algorithm 1 provides the pseudocode of the proposed BinaryNE algorithm. At Step 1, a set of random walks with length L are generated by starting random walks at each node $v_i\in\mathcal{V}$ for γ

ALGORITHM 1: BinaryNE: Binary Network Embedding

Input:
A given network $G = (\mathcal{V}, \mathcal{E}, \mathcal{A}, X)$;
Output:
Binary node embedding $\Phi(\cdot)$ for each $v_i \in \mathcal{V}$;
1: $\mathbb{S} \leftarrow$ generate a set of random walks on G;
2: $n(v_i, v_j) \leftarrow$ count the frequency of node context pairs (v_i, v_j) in $\mathbb{S}$;
3: $(W^{in}, W^{out,s}, W^{out,a}) \leftarrow$ initialization;
4: **repeat**
5: draw a random number $\delta \in (0, 1)$;
6: **if** $\delta \leqslant 0.5$ **then**
7: $(v_i, v_j) \leftarrow$ sample a node context pair according to the distribution of $n(v_i, v_j)$;
8: $\mathcal{V}_{neg} \leftarrow$ draw K negative nodes;
9: $(W^{in}, W^{out,s}) \leftarrow$ update parameters with $(v_i, v_j, \mathcal{V}_{neg})$ according to Equation (11);
10: **else**
11: $(v_i, a_j) \leftarrow$ sample a node attribute pair according to the distribution of X_{ij};
12: $\mathcal{A}_{neg} \leftarrow$ draw K negative attributes;
13: $(W^{in}, W^{out,a}) \leftarrow$ update parameters with $(v_i, a_j, \mathcal{A}_{neg})$ according to Equation (13);
14: **end if**
15: **until** maximum number of iterations expire;
16: construct node embedding $\Phi(\cdot)$ with W^{in} and Equation (14);
17: **return** $\Phi(\cdot)$;

times. At Step 2, on the generated random walks, with t window size, BinaryNE collects node context pairs (v_i, v_j) and counts their occurrence frequencies $n(v_i, v_j)$. At Step 3, W^{in} is initialized with random numbers sampled from a uniform distribution in the range of $[-\frac{1}{2d}, \frac{1}{2d}]$, and $W^{out,s}$ and $W^{out,a}$ are initialized with zero. At Steps 4–15, the parameters are updated with stochastic gradient descent. Each iteration starts from drawing a random switch variable $\delta \in (0, 1)$ to determine which partial objective to be optimized. To optimize the structure preserving partial objective, BinaryNE randomly draws a node context pair (v_i, v_j) according to the distribution of $n(v_i, v_j)$, and draws K negative nodes, forming $\mathcal{V}_{neg}$, then updates the parameters with Equation (11). To optimize the attribute preserving objective, BinaryNE draws a node attribute co-occurrence pair (v_i, a_j) according the distribution of X_{ij} and draws a negative attribute set $\mathcal{A}_{neg}$ with size K, then updates the parameters with Equation (13). For efficient node context pair and node attribute pair sampling, BinaryNE adopts the alias table [27] method, which takes only $O(1)$ time at each sampling. Finally, BinaryNE constructs binary node representations $\Phi(\cdot)$ with W^{in} according to Equation (14).

The time complexity of BinaryNE is determined by only the dimension of node embeddings d and the maximum number of iterations. The scale of the maximum number of iterations is $O(\max(nnz(X), |\mathcal{V}|))$, where $nnz(X)$ is the number of non-zero entries of X, and $|\mathcal{V}|$ is the scale of node context pairs collected via random walks. BinaryNE has a time complexity of $O(d \cdot \max(nnz(X), |\mathcal{V}|))$, which guarantees its ability to scale up to large-scale graphs.

5 EXPERIMENTS

In this section, we conduct experiments on six real-world networks to evaluate the effectiveness of binary node representations learned by BinaryNE for node similarity search, including search precision, response time, and memory usage, as well as node classification and node clustering. The detailed experimental settings are given in the Appendix.

Table 1. Summary of Six Real-world Networks

	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{E}\rvert$	$\lvert\mathcal{A}\rvert$	$nnz(X)$	# of Class
Cora	2,708	5,278	1,433	49,216	7
Citeseer	3,312	4,732	3,703	105,165	6
BlogCatalog	5,196	171,743	8,189	369,435	6
Flickr	7,575	239,738	12,047	182,517	9
DBLP(Subgraph)	18,448	45,611	2,476	103,130	4
DBLP(Full)	1,632,442	2,327,450	154,309	10,413,178	N/A

5.1 Datasets

Six real-world networks are used in the experiments, with the details as follows:

- **Cora**[1] [36]. The Cora network is composed of 2,708 machine learning publications and their citation relationships. Theses publications are categorized into seven groups. Each publication is represented by a 1,433-dimensional binary vector, with each dimension denoting the presence/absence of the corresponding word.
- **Citeseer**[1] [36]. Citeseer is another citation network with 3,312 papers and 4,732 citation relations. There are six classes among papers. According to the occurrence of the corresponding word, each paper is described by a 3,703-dimensional binary vector.
- **BlogCatalog**[2] [16]. The BlogCatalog network is an online social network formed by BlogCatalog, a blogger community. The BlogCatalog network contains 5,196 users and 171,743 follower–followee relations. Users' groups are defined as the categories of their blogs. The keywords of users' blogs are used to construct users' feature vectors. Here, binary feature vectors are constructed, with only the keyword occurrence state concerned.
- **Flickr**[2] [16]. Flickr is an online photo sharing platform. The Flickr network includes 7,575 users and 239,738 follower–followee relations. These users join in nine predefined groups. Users' features are described by the tags of their images. Each user is represented by 12,047-dimensional binary vector, according to the occurrence/absence of the corresponding tag.
- **DBLP(Subgraph)** and **DBLP(Full)**. The DBLP(Full) network is formed by the papers, paper titles, and paper citations of the DBLP bibliographic network[3] [47]. In DBLP(Full), there are in total 1,632,442 papers and 2,327,450 citations. The DBLP(Subgraph) is a subgraph of the DBLP(Full) network, constructed by papers from four research areas: *Database, Data Mining, Artificial Intelligence, and Computer Version*, which also act as paper labels. DBLP(Subgraph) contains 18,448 papers and 45,611 citation relations. For DBLP(Full) and DBLP(Subgraph), papers' titles are used to construct binary bag-of-words feature vectors.

Table 1 summarizes the statistics of the networks. For each network, the direction of links is ignored. *Cora*, *Citeseer*, *BlogCatalog*, *Flickr*, and *DBLP(Subgraph)* are used to evaluate the performance of the binary node representations learned by BinaryNE on node similarity search, including search precision, query time, and memory usage, as well as on node classification and node clustering. *DBLP(Full)* is used to investigate the scalability of node similarity search with BinaryNE binary codes.

[1] https://linqs.soe.ucsc.edu/data.
[2] https://www4.comp.polyu.edu.hk/~xiaohuang/Code.html.
[3] https://aminer.org/citation (Version 3 is used).

5.2 Baseline Methods

BinaryNE is compared with two groups of the state-of-the-art methods:

- **Continuous embeddings measured by Euclidean distance**:
  - **DeepWalk/node2vec** [11, 35] preserves the similarity between nodes sharing similar context in random walks. node2vec is equivalent to DeepWalk with the default parameter setting $p = q = 1$.
  - **LINE1** [46] denotes the version of LINE that captures the first-order proximity.
  - **LINE2** [46] represents the version of LINE that models the second-order proximity.
  - **SDNE** [53] learns deep non-linear node representations via a semi-supervised deep autoencoder.
  - **TADW** [58] learns node embeddings that capture both network structure and node content similarity via inductive matrix factorization [33].
  - **UPP-SNE** [63] performs a non-linear mapping on node content features to learn node embeddings that preserve both network structure and node content features.
  - **MVC-DNE** [59] fuses network structure and node content features into node embeddings through deep cross-view learning.
  - **SINE** [65] learns node representations by using node representations to simultaneously predict context nodes and node attributes.
  - **Feature**. Node raw content feature is also used as a baseline for similarity search. For each node $v_i \in \mathcal{V}$, its feature vector is $X_{i:}$, with $X_{i:}$ being the i-th row of X.
- **Discrete embeddings measured by Hamming distance**:
  - **Quantized Continuous Embeddings**. To obtain binary node representations, one naive way is to quantize the continuous node embeddings into binary codes. As a baseline, we binarize the continuous embeddings learned by above baseline methods with the state-of-the-art hash learning algorithm: Iterative Quantization [10], and denote these methods as *DeepWalk+Q*, *LINE1+Q*, *LINE2+Q*, *SDNE+Q*, *TADW+Q*, *UPP-SNE+Q*, *MVC-DNE+Q*, *SINE+Q*, and *Feature+Q*, respectively.
  - **BANE** [60] learns binary node representations by performing binary factorization on the Weisfeiler-Lehman proximity matrix [41] that carries both network structure and node content information.
  - **KD-coding** [8] learns K-way D-dimensional discrete node representations from the continuous node representations learned by DeepWalk. To obtain 128-dimensional binary codes, we set $K = 2$ and $D = 128$, respectively.
  - **DNE** [40] learns binary node representations through discrete matrix factorization in a supervised manner. To enable DNE to work under our unsupervised setting, we set the weight of its supervised learning component to 0 in the optimization objective.
  - **NetHash** [57] generates discrete node embeddings by randomly sampling the union of neighboring node attributes. As the learned discrete node representations do not take binary values, the Hamming distance cannot be efficiently calculated with bit-wise operations.
  - **BinaryNE_S** is the ablated version of BinaryNE that uses only network structure to learn binary node representations.
  - **BinaryNE_C** is the ablated version of BinaryNE that uses only node content attributes to learn binary node codes.
  - **BinaryNE_S+C** concatenates the binary codes learned via BinaryNE_S and BinaryNE_C, which is another ablated version of BinaryNE that respectively encodes network structure and node content attributes into seperated dimensions of binary node representa-

tions. The binary codes learned via BinaryNE_S and BinaryNE_C have half dimensions of BinaryNE's binary codes for fair comparisons.

5.3 Evaluation Metrics

For all methods, we use the entire network to learn continuous or binary node representations for all nodes and evaluate the node similarity search performance by searching similar nodes for every node in the network from the remaining nodes.

For each node in a network, we in turn query its top-K similar nodes. K is set to 100, 200, and 500, respectively. We adopt averaged *precision* and *MAP (Mean Averaged Precision)* as evaluation metrics.

For querying nodes similar to node v_i, the $precision@K(v_i)$ is defined as

$$precision@K(v_i) = \frac{|\{v_j | rank(v_j) \leq K, C(v_i) = C(v_j)\}|}{K},$$

where $rank(v_j)$ is the position of v_j in the rank list of nodes similar to v_i. $C(v_i) = C(v_j)$ indicates that node v_i and v_j have the same class label, with $C(\cdot)$ denoting node class label. As we in turn take all nodes in $\mathcal{V}$ as query nodes, we report the averaged $precision@K$ as final results.

Mean Average Precision (MAP) is an information retrieval metric with good discrimination and stability. Different from precision, MAP takes into account the order in which relevant nodes are placed in the returned rank list. When we vary the query node v_i over $\mathcal{V}$, the MAP value is calculated as

$$AP@K(v_i) = \frac{\sum_{k=1}^{K} precision@k(v_i) \cdot relavant@k(v_i)}{|\{v_j | C(v_j) = C(v_i), v_j \in \mathcal{V}\}|},$$

$$MAP@K = \frac{\sum_{i=1}^{|\mathcal{V}|} AP@K(v_i)}{|\mathcal{V}|},$$

where $relavant@k(v_i)$ is an indicator function equaling 1 if the k-th retrieved node is relevant to v_i and 0 otherwise.

5.4 Similarity Search Results

Tables 2–6 give similarity search results on Cora, Citeseer, BlogCatalog, Flickr, and DBLP(Subgraph). For query time, we only consider the time consumed by calculating the distance between the query node and all remaining nodes, which contributes to the main computational overhead of similarity search, and report the time averaged over all query nodes (in milliseconds). We provide the search speedup of BinaryNE compared with baselines. We also perform paired t-tests between the search precisions delivered by BinaryNE and baseline methods, where we use • (∘) to indicate that BinaryNE is significantly better (worse) than the compared baseline methods at 95% significance level. For all methods, the best and second best performers are highlighted by **bold** and underline, respectively.

From Tables 2–6, we can see that BinaryNE generally achieves significantly better precision and MAP than other discrete embedding methods on Cora, Citeseer, Flickr, and DBLP(Subgraph), except that for MAP@100, MAP@200 on Cora, and MAP@500 on DBLP(Subgraph) it lags slightly behind the best performers. On BlogCatalog, among the discrete embedding methods, TADW+Q performs best while BinaryNE follows with comparable performance. From the above observations, we can see that our BinaryNE achieves the best overall similarity search performance on all datasets among the discrete embedding methods. What also deserves to be noticed is that BinaryNE still delivers competitive or even better performance compared with the continuous network embedding methods. Through using node binary representations to predict the existence of context

Table 2. Similarity Search Results on Cora

Metric	Method	precision@100	MAP@100	precision@200	MAP@200	precision@500	MAP@500	Query time (ms)	Speedup
Euclidean	DeepWalk	0.5621 ●	0.1149 ○	0.4782 ●	0.1771 ○	0.3464 ●	0.2616 ●	1.62	32.5 ×
	LINE1	0.4042 ●	0.0678 ●	0.3002 ●	0.0828 ●	0.2270 ●	0.1110 ●	1.62	32.5 ×
	LINE2	0.3425 ●	0.0482 ●	0.2868 ●	0.0645 ●	0.2411 ●	0.0993 ●	1.90	37.9 ×
	SDNE	0.3595 ●	0.0565 ●	0.2901 ●	0.0730 ●	0.2370 ●	0.1058 ●	1.62	32.5 ×
	TADW	0.4508 ●	0.0871 ●	0.3621 ●	0.1227 ●	0.2607 ●	0.1682 ●	1.62	32.5 ×
	UPP-SNE	**0.6032** ○	**0.1204** ○	**0.5242** ○	**0.1890** ○	0.3990 ●	**0.2947** ○	1.62	32.5 ×
	MVC-DNE	0.3559 ●	0.0408 ●	0.3190 ●	0.0626 ●	0.2738 ●	0.1094 ●	1.62	32.5 ×
	SINE	0.4309 ●	0.0701 ●	0.3703 ●	0.1026 ●	0.2968 ●	0.1611 ●	1.62	32.5 ×
	Feature	0.2240 ●	0.0166 ●	0.2060 ●	0.0252 ●	0.2189 ●	0.0551 ●	19.23	384.5 ×
Hamming	DeepWalk+Q	0.5748	0.1139 ○	0.4912 ●	0.1749 ○	0.3703 ●	0.2666 ●	0.05	1.0 ×
	LINE1+Q	0.3916 ●	0.0688 ●	0.2922 ●	0.0833 ●	0.2160 ●	0.1087 ●	0.05	1.0 ×
	LINE2+Q	0.3880 ●	0.0606 ●	0.3203 ●	0.0834 ●	0.2516 ●	0.1219 ●	0.05	1.0 ×
	SDNE+Q	0.4522 ●	0.0838 ●	0.3635 ●	0.1146 ●	0.2666 ●	0.1552 ●	0.05	1.0 ×
	TADW+Q	0.5621 ●	0.1138 ○	0.4570 ●	0.1623 ●	0.3310 ●	0.2282 ●	0.06	1.2 ×
	UPP-SNE+Q	0.5520 ●	0.1031 ●	0.4732 ●	0.1572 ●	0.3606 ●	0.2421 ●	0.05	1.0 ×
	MVC-DNE+Q	0.3323 ●	0.0348 ●	0.3013 ●	0.0546 ●	0.2609 ●	0.0976 ●	0.07	1.4 ×
	SINE+Q	0.4564 ●	0.0715 ●	0.3909 ●	0.1055 ●	0.3114 ●	0.1667 ●	0.06	1.2 ×
	Feature+Q	0.3701 ●	0.0459 ●	0.3285 ●	0.0702 ●	0.2740 ●	0.1179 ●	0.06	1.2 ×
	BANE	0.2225 ●	0.0168 ●	0.2086 ●	0.0260 ●	0.1960 ●	0.0499 ●	0.05	1.0 ×
	DNE	0.4781 ●	0.0969 ●	0.3906 ●	0.1397 ●	0.2838 ●	0.1886 ●	0.05	1.0 ×
	KD-coding	0.5092 ●	0.0874 ●	0.4640 ●	0.1460 ●	0.3795 ●	0.2570 ●	0.05	1.0 ×
	NetHash	0.4546 ●	0.0757 ●	0.3852 ●	0.1097 ●	0.2993 ●	0.1656 ●	1.17	23.4 ×
	BinaryNE_S	0.5598 ●	0.1072 ○	0.4916 ●	0.1693	0.3748 ●	0.2603 ●	0.05	1.0 ×
	BinaryNE_C	0.3553 ●	0.0391 ●	0.3217 ●	0.0623 ●	0.2768 ●	0.1114 ●	0.05	1.0 ×
	BinaryNE_S+C	0.5483 ●	0.1009 ●	0.4811 ●	0.1592 ●	0.3738 ●	0.2543 ●	0.05	1.0 ×
	BinaryNE	0.5723	0.1050	0.5091	0.1694	**0.4070**	0.2848	0.05	

nodes, nodes sharing similar structural context are embedded closely in the binary space. Similar binary representations are also learned for nodes sharing similar node attributes with the same mechanism, by using node binary representations to predict node attributes. The seamless integration between embedding binarization and embedding learning empowers BinaryNE to effectively capture both network structure and node content features, which makes the learned binary codes informative enough to measure node similarity accurately.

On the other hand, BinaryNE remarkably improves search efficiency, providing more than 25 times faster search speed than continuous network embedding methods, and more than 20 times than NetHash, which learns non-binary discrete representations. Compared with the Euclidean distance in the continuous embedding space and the Hamming distance in the non-binary discrete embedding space, the Hamming distance measured by binary representations can be calculated far more efficiently with the bit-wise operations.

Among the continuous network embedding baselines, the attributed network embedding (TADW, UPP-SNE, or SINE) achieves the best search precisions. On the five networks, raw node content features consistently fail to achieve satisfactory precisions. By integrating network structure and node content in measuring node similarity, attributed network embedding is superior to structure preserving network embedding and raw node content features.

With the state-of-the-art quantization technique, the binarized continuous network embeddings are able to achieve satisfactory search precision in some cases. However, they still tend to be inferior when they are compared with our BinaryNE algorithm. The results demonstrate that it

Table 3. Similarity Search Results on Citeseer

Metric	Method	precision@100	MAP@100	precision@200	MAP@200	precision@500	MAP@500	Query time (ms)	Speedup
	DeepWalk	0.3806 •	0.0419 •	0.3165 •	0.0597 •	0.2534 •	0.0923 •	1.99	33.1 ×
	LINE1	0.2875 •	0.0282 •	0.2360 •	0.0372 •	0.1997 •	0.0573 •	1.99	33.1 ×
	LINE2	0.2493 •	0.0210 •	0.2171 •	0.0286 •	0.1943 •	0.0473 •	1.99	33.1 ×
	SDNE	0.2436 •	0.0210 •	0.2091 •	0.0282 •	0.1866 •	0.0469 •	1.99	33.1 ×
Euclidean	TADW	0.3728 •	0.0381 •	0.3213 •	0.0572 •	0.2609 •	0.0944 •	1.99	33.1 ×
	UPP-SNE	**0.4951** ○	**0.0591** ○	**0.4504** ○	**0.1000** ○	0.3714 •	0.1801	1.99	33.1 ×
	MVC-DNE	0.3478 •	0.0293 •	0.3143 •	0.0465 •	0.2724 •	0.0847 •	1.99	33.1 ×
	SINE	0.3831 •	0.0376 •	0.3418 •	0.0590 •	0.2882 •	0.1023 •	1.99	33.1 ×
	feature	0.2532 •	0.0140 •	0.2471 •	0.0249 •	0.2320 •	0.0530 •	54.65	910.8 ×
	DeepWalk+Q	0.3854 •	0.0417 •	0.3384 •	0.0647 •	0.2771 •	0.1073 •	0.07	1.2 ×
	LINE1+Q	0.2785 •	0.0266 •	0.2317 •	0.0350 •	0.1967 •	0.0539 •	0.06	1.0 ×
	LINE2+Q	0.2777 •	0.0244 •	0.2384 •	0.0339 •	0.2063 •	0.0553 •	0.07	1.2 ×
	SDNE+Q	0.3232 •	0.0327 •	0.2724 •	0.0462 •	0.2249 •	0.0722 •	0.06	1.0 ×
	TADW+Q	0.4212 •	0.0461 •	0.3605 •	0.0692 •	0.2877 •	0.1112 •	0.07	1.2 ×
	UPP-SNE+Q	0.4768 •	0.0560 •	0.4332 •	0.0942 •	0.3578 •	0.1687 •	0.07	1.2 ×
	MVC-DNE+Q	0.3165 •	0.0239 •	0.2900 •	0.0386 •	0.2574 •	0.0730 •	0.07	1.2 ×
	SINE+Q	0.3435 •	0.0292 •	0.3098 •	0.0460 •	0.2682 •	0.0833 •	0.07	1.2 ×
Hamming	Feature+Q	0.3701 •	0.0331 •	0.3343 •	0.0532 •	0.2841 •	0.0952 •	0.07	1.2 ×
	BANE	0.2300 •	0.0136 •	0.2162 •	0.0219 •	0.2007 •	0.0426 •	0.06	1.0 ×
	DNE	0.3030 •	0.0294 •	0.2675 •	0.0445 •	0.2271 •	0.0732 •	0.07	1.2 ×
	KD-coding	0.2572 •	0.0199 •	0.2496 •	0.0397 •	0.2538 •	0.1033 •	0.06	1.0 ×
	NetHash	0.3866 •	0.0378 •	0.3417 •	0.0583 •	0.2851 •	0.0999 •	1.35	22.5 ×
	BinaryNE_S	0.3975 •	0.0440 •	0.3641 •	0.0741 •	0.3072 •	0.1350 •	0.06	1.0 ×
	BinaryNE_C	0.3528 •	0.0286 •	0.3241 •	0.0473 •	0.2847 •	0.0896 •	0.06	1.0 ×
	BinaryNE_S+C	0.4369 •	0.0485 •	0.4003 •	0.0820 •	0.3393 •	0.1507 •	0.06	1.0 ×
	BinaryNE	0.4846	0.0572	0.4454	0.0981	**0.3758**	**0.1814**	0.06	

is suboptimal to separately learn continuous network embeddings and quantize them into binary codes. In comparison, BinaryNE directly encodes network structure and node content features into binary node representations, achieving superior search precisions.

BANE achieves relatively good performance on BlogCatalog, Flick, and DBLP(Subgraph), but yields unsatisfactory performance on Cora and Citeseer. This is because, BANE performs neighboring node attribute vector aggregation to integrate network structure and node attributes into a unified binary node representation. When the inconsistency between network structure and attributes occurs, that is, linked nodes have discrepant attributes, the similarity measured by network structure and node attributes tends to heavily deteriorate each other.

DNE and KD-coding fail to achieve satisfactory performance in most cases. They learn binary node representations from only network structure and do not leverage the essential information on node attributes. Without the supervision of node labels, DNE cannot learn discriminative binary codes, preventing it from performing well for similarity search.

NetHash constructs discrete node representations by randomly sampling the IDs of node features aggregated from local neighborhood. With both network structure and node features leveraged, NetHash achieves relatively good performance on Cora, Citeseer, and DBLP(Subgraph). As the discrete embeddings do not take binary values, bit-wise operations cannot be performed to calculate Hamming distance. As a result, its query speedup over continuous network embedding is limited.

By comparing BinaryNE with its three ablated versions, BinaryNE_S and BinaryNE_C with only network structure or only node content preserved, as well as BinaryNE_S+C with only half

Table 4. Similarity Search Results on BlogCatalog

Metric	Method	precision@100	MAP@100	precision@200	MAP@200	precision@500	MAP@500	Query time (ms)	Speedup
	DeepWalk	0.4349 •	0.0327 •	0.3818 •	0.0521 •	0.3005 •	0.0859 •	3.12	31.2 ×
	LINE1	0.3850 •	0.0275 •	0.3160 •	0.0411 •	0.2375 •	0.0616 •	3.12	31.2 ×
	LINE2	0.2412 •	0.0140 •	0.2239 •	0.0226 •	0.2029 •	0.0420 •	2.60	26.0 ×
	SDNE	0.3145 •	0.0201 •	0.2795 •	0.0316 •	0.2366 •	0.0548 •	3.12	31.2 ×
Euclidean	TADW	0.7431 ○	0.0758 ○	0.6923 ○	0.1373 ○	0.5675 •	0.2621 ○	2.60	26.0 ×
	UPP-SNE	0.5173 •	0.0417 •	0.4735 •	0.0707 •	0.3931 •	0.1281 •	2.60	26.0 ×
	MVC-DNE	0.5616 •	0.0463 •	0.5008 •	0.0761 •	0.4109 •	0.1356 •	3.12	31.2 ×
	SINE	0.3502 •	0.0216 •	0.3067 •	0.0326 •	0.2599 •	0.0567 •	3.12	31.2 ×
	Feature	0.2424 •	0.0113 •	0.2239 •	0.0177 •	0.2023 •	0.0333 •	184.98	1849.8 ×
	DeepWalk+Q	0.3950 •	0.0266 •	0.3545 •	0.0432 •	0.2883 •	0.0728 •	0.11	1.1 ×
	LINE1+Q	0.4021 •	0.0275 •	0.3529 •	0.0436 •	0.2779 •	0.0701 •	0.11	1.1 ×
	LINE2+Q	0.2958 •	0.0146 •	0.2720 •	0.0236 •	0.2402 •	0.0440 •	0.10	1.0 ×
	SDNE+Q	0.3538 •	0.0219 •	0.3127 •	0.0342 •	0.2607 •	0.0578 •	0.12	1.2 ×
	TADW+Q	**0.7448** ○	**0.0776** ○	**0.7081** ○	**0.1453** ○	**0.6147** ○	**0.3028** ○	0.10	1.0 ×
	UPP-SNE+Q	0.4830 •	0.0372 •	0.4416 •	0.0628 •	0.3655 •	0.1123 •	0.10	1.0 ×
	MVC-DNE+Q	0.5117 •	0.0393 •	0.4568 •	0.0642 •	0.3804 •	0.1154 •	0.11	1.1 ×
	SINE+Q	0.3745 •	0.0225 •	0.3363 •	0.0356 •	0.2903 •	0.0650 •	0.10	1.0 ×
Hamming	Feature+Q	0.4921 •	0.0396 •	0.4473 •	0.0666 •	0.3798 •	0.1244 •	0.10	1.0 ×
	BANE	0.4892 •	0.0371 •	0.4572 •	0.0655 •	0.4023 •	0.1308 •	0.11	1.1 ×
	DNE	0.1689 •	0.0044 •	0.1689 •	0.0080 •	0.1692 •	0.0191 •	0.09	0.9 ×
	KD-coding	0.3754 •	0.0242 •	0.3455 •	0.0409 •	0.2887 •	0.0724 •	0.08	0.8 ×
	NetHash	0.3811 •	0.0246 •	0.3388 •	0.0385 •	0.2882 •	0.0684 •	3.14	31.4 ×
	BinaryNE_S	0.4729 •	0.0352 •	0.4347 •	0.0604 •	0.3672 •	0.1126 •	0.09	0.9 ×
	BinaryNE_C	0.5571 •	0.0466 •	0.5181 •	0.0820 •	0.4531 •	0.1647 •	0.10	1.0 ×
	BinaryNE_S+C	0.6500 •	0.0601 •	0.5846 •	0.1024 •	0.4769 •	0.1870 •	0.10	1.0 ×
	BinaryNE	0.7081	0.0687	0.6637	0.1239	0.5828	0.2547	0.10	

dimensions encoded with network structure/node content, we can find that BinaryNE consistently outperforms the three counterparts in all cases. This verifies BinaryNE's advantage of integrating both network structure and node content to learn informative binary node representations over the counterparts using only network structure or node content, as well as the naive combination of their respective binary node representations. Apart from Cora and Flickr, where network structure and node content, respectively, dominate similarity search, BinaryNE_S+C indeed outperforms BinaryNE_S and BinaryNE_C on Citeseer, BlogCatalog and DBLP(Subgraph), with more information leveraged. However, as BinaryNE_S+C encodes network structure/node content into only half dimensions of binary node representations, it is inferior to BinaryNE that preserves network structure and node content at each dimension.

5.5 Experiments on Node Classification and Node Clustering

We further evaluate the effectiveness of the binary codes learned by BinaryNE through node classification and node clustering. Although the binary codes learned by BinaryNE aim to serve efficient node similarity search, they can also speed up other tasks, like node classification, link prediction and node clustering. Here, with the guaranteed efficiency, we select node classification and node clustering to check whether BinaryNE is able to deliver satisfactory results for other downstream tasks, like the competitive continuous network embedding methods.

We perform node classification experiments on DBLP(Subgraph). By taking the binary codes learned by BinaryNE and node representations learned by other continuous network embedding

Table 5. Similarity Search Results on Flickr

Metric	Method	precision@100	MAP@100	precision@200	MAP@200	precision@500	MAP@500	Query time (ms)	Speedup
Euclidean	DeepWalk	0.2194 •	0.0101 •	0.2014 •	0.0162 •	0.1762 •	0.0287 •	4.54	32.5 ×
	LINE1	0.2318 •	0.0125 •	0.2075 •	0.0198 •	0.1747 •	0.0333 •	4.54	32.5 ×
	LINE2	0.1573 •	0.0074 •	0.1464 •	0.0122 •	0.1357 •	0.0239 •	3.79	27.1 ×
	SDNE	0.1693 •	0.0082 •	0.1548 •	0.0131 •	0.1408 •	0.0251 •	3.79	27.1 ×
	TADW	0.3644 •	0.0278 •	0.3104 •	0.0421 •	0.2368 •	0.0649 •	3.79	27.1 ×
	UPP-SNE	0.3917 •	0.0309 •	0.3594 •	0.0525 •	0.3142 •	0.0990 •	4.54	32.5 ×
	MVC-DNE	0.3047 •	0.0176 •	0.2723 •	0.0275 •	0.2329 •	0.0489 •	4.54	32.5 ×
	SINE	0.3053 •	0.0180 •	0.2630 •	0.0266 •	0.2124 •	0.0436 •	3.79	27.1 ×
	Feature	0.1379 •	0.0055 •	0.1275 •	0.0082 •	0.1190 •	0.0152 •	415.11	2965.1 ×
Hamming	DeepWalk+Q	0.2348 •	0.0119 •	0.2155 •	0.0193 •	0.1855 •	0.0334 •	0.14	1.0 ×
	LINE1+Q	0.2505 •	0.0129 •	0.2244 •	0.0203 •	0.1875 •	0.0338 •	0.16	1.1 ×
	LINE2+Q	0.2135 •	0.0083 •	0.1974 •	0.0135 •	0.1764 •	0.0253 •	0.15	1.1 ×
	SDNE+Q	0.2352 •	0.0109 •	0.2127 •	0.0171 •	0.1835 •	0.0298 •	0.15	1.1 ×
	TADW+Q	0.3952 •	0.0313 •	0.3601 •	0.0522 •	0.3008 •	0.0927 •	0.15	1.1 ×
	UPP-SNE+Q	0.4939 •	0.0434 •	0.4642 •	0.0785 •	0.4028 •	0.1573 •	0.16	1.1 ×
	MVC-DNE+Q	0.3028 •	0.0167 •	0.2711 •	0.0263 •	0.2317 •	0.0468 •	0.14	1.0 ×
	SINE+Q	0.3660 •	0.0248 •	0.3154 •	0.0366 •	0.2575 •	0.0605 •	0.15	1.1 ×
	Feature+Q	0.4733 •	0.0430 •	0.4063 •	0.0662 •	0.3107 •	0.1042 •	0.17	1.2 ×
	BANE	0.3165 •	0.0217 •	0.2887 •	0.0369 •	0.2467 •	0.0692 •	0.17	1.2 ×
	DNE	0.1163 •	0.0023 •	0.1160 •	0.0041 •	0.1157 •	0.0091 •	0.13	0.9 ×
	KD-coding	0.1923 •	0.0076 •	0.1817 •	0.0127 •	0.1655 •	0.0242 •	0.13	0.9 ×
	NetHash	0.2035 •	0.0090 •	0.1814 •	0.0133 •	0.1594 •	0.0232 •	4.28	30.5 ×
	BinaryNE_S	0.2219 •	0.0104 •	0.2071 •	0.0175 •	0.1874 •	0.0339 •	0.13	0.9 ×
	BinaryNE_C	0.5469 •	0.0486 •	0.4897 •	0.0805 •	0.4010 •	0.1434 •	0.14	1.0 ×
	BinaryNE_S+C	0.4293 •	0.0322 •	0.3813 •	0.0523 •	0.3125 •	0.0918 •	0.14	1.0 ×
	BinaryNE	**0.5865**	**0.0548**	**0.5340**	**0.0938**	**0.4467**	**0.1747**	0.14	

algorithms as features, we evaluate their classification effectiveness with a training-test split. We vary the training ratio from 10% to 30%, 50%, and 70%. Node classification accuracy values are reported in Table 7.[4] As shown in Table 7, SINE performs the best while BinaryNE achieves comparable performance.

On Cora, we conduct node clustering experiments on the learned node representations with the *k-means* algorithm. For the node representations learned by continuous network embedding algorithms, we use the Euclidean distance metric, while hamming distance is used for binary codes learned by BinaryNE, which is much more efficient than the Euclidean distance. Table 8[4] reports the node clustering results on Accuracy, Fvalue, and NMI [44]. From Table 8, we can see that BinaryNE achieves the best clustering performance, significantly outperforming other continuous network embedding methods.

The excellent performance on node classification and node clustering proves that the binary codes learned by BinaryNE are informative enough to well represent network nodes. The BinaryNE binary codes can not only serve fast node similarity search with high search precision, but also underpin the fast operation of other network analytic tasks with competitive performance.

[4]In Tables 7 and 8, the best and the second best performers are highlighted by **bold** and underline, respectively, and • (◦) indicates that BinaryNE is significantly better (worse) than the compared baseline methods at 95% significance level.

Table 6. Similarity Search Results on DBLP(Subgraph)

Metric	Method	precision@100	MAP@100	precision@200	MAP@200	precision@500	MAP@500	Query time (ms)	Speedup
	DeepWalk	0.7109 •	0.0113 •	0.6941 •	0.0214 •	0.6648 •	0.0488 •	11.07	31.6 ×
	LINE1	0.6852 •	0.0106 •	0.6421 •	0.0190 •	0.5338 •	0.0350 •	11.07	31.6 ×
	LINE2	0.6302 •	0.0091 •	0.5720 •	0.0151 •	0.4800 •	0.0273 •	11.07	31.6 ×
	SDNE	0.6464 •	0.0096 •	0.6033 •	0.0172 •	0.5149 •	0.0332 •	11.07	31.6 ×
Euclidean	TADW	0.7156 •	0.0107 •	0.6951 •	0.0199 •	0.6581 •	0.0441 •	11.07	31.6 ×
	UPP-SNE	0.7164 •	0.0115 •	0.7098 •	0.0224 •	0.6950 •	0.0535 •	11.07	31.6 ×
	MVC-DNE	0.5575 •	0.0067 •	0.5285 •	0.0118 •	0.4908 •	0.0245 •	11.07	31.6 ×
	SINE	0.7364 •	0.0118 •	0.7130 •	0.0219 •	0.6714 •	0.0480 •	11.07	31.6 ×
	Feature	0.4555 •	0.0043 •	0.4408 •	0.0077 •	0.4289 •	0.0172 •	202.93	579.8 ×
	DeepWalk+Q	0.7261 •	0.0117 •	0.7155 •	0.0225 •	0.6960 •	0.0528 •	0.35	1.0 ×
	LINE1+Q	0.6881 •	0.0106 •	0.6528 •	0.0193 •	0.5555 •	0.0366 •	0.36	1.0 ×
	LINE2+Q	0.6682 •	0.0099 •	0.6278 •	0.0173 •	0.5553 •	0.0340 •	0.38	1.1 ×
	SDNE+Q	0.6922 •	0.0103 •	0.6740 •	0.0195 •	0.6256 •	0.0425 •	0.35	1.0 ×
	TADW+Q	0.6964 •	0.0111 •	0.6751 •	0.0206 •	0.6329 •	0.0444 •	0.35	1.0 ×
	UPP-SNE+Q	0.7291 •	0.0120 •	0.7256 •	0.0238 •	**0.7182**	**0.0582** ○	0.36	1.0 ×
	MVC-DNE+Q	0.5234 •	0.0057 •	0.5005 •	0.0101 •	0.4710 •	0.0217 •	0.36	1.0 ×
	SINE+Q	0.7395 •	0.0120 •	0.7207 •	0.0227 •	0.6867 •	0.0510 •	0.38	1.1 ×
Hamming	Feature+Q	0.5489 •	0.0067 •	0.5304 •	0.0122 •	0.4983 •	0.0263 •	0.36	1.0 ×
	BANE	0.7285 •	0.0116 •	0.7032 •	0.0214 •	0.6608 •	0.0465 •	0.38	1.1 ×
	DNE	0.6237 •	0.0085 •	0.5980 •	0.0155 •	0.5695 •	0.0348 •	0.36	1.0 ×
	KD-coding	0.6437 •	0.0085 •	0.6683 •	0.0182 •	0.6586 •	0.0453 •	0.29	0.8 ×
	NetHash	0.6606 •	0.0097 •	0.6242 •	0.0171 •	0.5750 •	0.0357 •	7.31	20.9 ×
	BinaryNE_S	0.7195 •	0.0115 •	0.7069 •	0.0222 •	0.6830 •	0.0518 •	0.32	0.9 ×
	BinaryNE_C	0.5500 •	0.0064 •	0.5329 •	0.0118 •	0.5088 •	0.0261 •	0.35	1.0 ×
	BinaryNE_S+C	0.7353 •	0.0119 •	0.7203 •	0.0228 •	0.6934 •	0.0520 •	0.37	1.1 ×
	BinaryNE	**0.7528**	**0.0125**	**0.7403**	**0.0242**	0.7155	0.0563	0.35	

Table 7. Node Classification Accuracy on DBLP(Subgraph)

Training Ratio	10%	30%	50%	70%
DeepWalk	0.7869 •	0.7997 •	0.8024 •	0.8036 •
LINE1	0.7320 •	0.7572 •	0.7620 •	0.7639 •
LINE2	0.6567 •	0.6941 •	0.7026 •	0.7089 •
SDNE	0.6391 •	0.6676 •	0.6782 •	0.6838 •
TADW	0.7903 •	0.8125 •	0.8186 •	0.8217
UPP-SNE	0.7881 •	0.7965 •	0.7977 •	0.7993 •
MVC-DNE	0.6984 •	0.7478 •	0.7589 •	0.7650 •
SINE	**0.8054** ○	**0.8270** ○	**0.8319** ○	**0.8327** ○
BinaryNE	0.8005	0.8193	0.8216	0.8237

Table 8. Node Clustering Results on Cora

Metric	Accuracy	Fvalue	NMI
DeepWalk	0.6263 •	0.6096 •	0.4306 •
LINE1	0.3510 •	0.3110 •	0.1023 •
LINE2	0.4170 •	0.3797 •	0.1763 •
SDNE	0.4061 •	0.3804 •	0.1939 •
TADW	0.3705 •	0.3556 •	0.1487 •
UPP-SNE	0.6337 •	0.6209 •	0.4412 •
MVC-DNE	0.6095 •	0.5885 •	0.3510 •
SINE	0.6345 •	0.6239 •	0.4540 •
BinaryNE	**0.6729**	**0.6656**	**0.4810**

5.6 A Case Study on Relevant Paper Search

In this subsection, we conduct a case study on relevant paper search on the DBLP network. We select the paper "Learning Classifiers from Only Positive and Unlabeled Data" published on KDD-2008 as the query paper, which is a highly cited paper on the topic of "Positive Unlabeled Learning". We retrieve the top five similar papers with the node representations learned by DeepWalk+Q, Feature+Q, TADW+Q, NetHash, and BinaryNE, by calculating the Hamming distance between the

Table 9. Top Five Relevant Paper Search on DBLP

Query: **Learning Classifiers from Only Positive and Unlabeled Data**
DeepWalk+Q:
1. Finding Transport Proteins in a General Protein Database
2. A Bayesian Network Framework for Reject Inference
3. Making Generative Classifiers Robust to Selection Bias
4. Building Text Classifiers Using **Positive and Unlabeled** Examples ✓
5. Audience Selection for On-line Brand Advertising: Privacy-friendly Social Network Targeting
Feature+Q:
1. Learning Coordination Classifiers
2. Learning from Little: Comparison of Classifiers Given Little Training
3. Learning a Two-stage SVM/CRF Sequence Classifier
4. Delegating Classifiers
5. On the Chance Accuracies of Large Collections of Classifiers
TADW+Q:
1. Efficient Learning of Naive Bayes Classifiers under Class-conditional Classification Noise
2. Learning to Classify Texts Using **Positive and Unlabeled** Data ✓
3. Semi-Supervised Learning with Very Few Labeled Training Examples
4. Calculation of the Learning Curve of Bayes Optimal Classification Algorithm for Learning a Perceptron With Noise
5. How To Use What You Know
NetHash:
1. Making Generative Classifiers Robust to Selection bias
2. A Bayesian Network Framework for Reject Inference
3. Building Text Classifiers Using **Positive and Unlabeled** Examples ✓
4. Finding Transport Proteins in a General Protein Database
5. Active Learning in Partially Supervised Classification
BinaryNE:
1. Learning to Classify Texts Using **Positive and Unlabeled** Data ✓
2. Learning the Common Structure of Data
3. Enhancing Supervised Learning with Unlabeled Data
4. Learning from Multiple Sources
5. Text Classification from **Positive and Unlabeled** Documents ✓

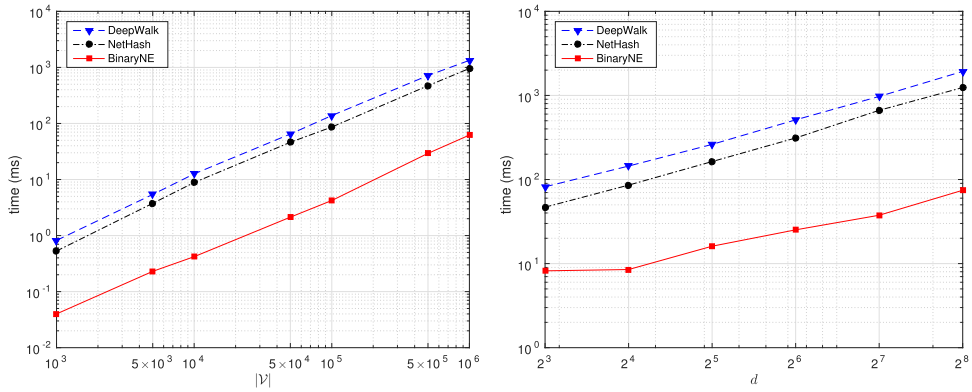
query paper and candidate papers. Table 9 reports the search results. As can be seen, DeepWalk+Q, Feature+Q, TADW+Q, and NetHash only retrieve one relevant paper, and no relevant papers are discovered by Feature+Q. By contrast, our BinaryNE algorithm achieves the best search results, with two relevant papers (1 and 5) discovered. It is worth noting that, as different algorithms use various information sources to measure node similarity, the top five relevant papers retrieved by different algorithms may have few intersections.

5.7 Comparison of Memory Usage

In Table 10, we compare the memory used for accommodating the continuous node representations learned by DeepWalk, the non-binary discrete node representations learned by NetHash, and the binary codes learned by BinaryNE. Compared with DeepWalk and NetHash, with the same dimension, the binary representations learned by BinaryNE significantly reduce the mem-

Table 10. The Memory Usage of DeepWalk, NetHash, and BinaryNE Embeddings

Dataset	DeepWalk		NetHash		BinaryNE
	Memory	Reduction	Memory	Reduction	Memory
Cora	2.64MB	64×	1.32MB	32×	42.32KB
Citeseer	3.23MB	64×	1.62MB	32×	51.75KB
BlogCatalog	5.07MB	64×	2.54MB	32×	81.19KB
Flickr	7.40MB	64×	3.70MB	32×	118.36KB
DBLP(Subgraph)	18.02MB	64×	9.01MB	32×	288.25KB
DBLP(Full)	1.56GB	64×	797.09MB	32×	24.91MB



(a) $|\mathcal{V}|$ (b) d

Fig. 2. Query time with varying $|\mathcal{V}|$ and d.

ory consumption by 64 and 32 times, respectively. For the DBLP(Full) network with more than 1 million nodes, the memory used for storing the continuous node representations is more than 1.5GB, which is intractable for computing devices with low memory configuration to perform node similarity search. By contrast, the binary node representations learned by BinaryNE only consume 25MB memory for the DBLP(Full) network, which is more practical for general devices. The low memory consumption makes BinaryNE more desirable for real-world applications.

5.8 Experiments on Search Scalability

We also conduct experiments on the large DBLP(Full) network to test the search scalability of different types of network embeddings with respect to network size $|\mathcal{V}|$ and embedding dimension d. We compare the binary embeddings generated by BinaryNE with those by DeepWalk and NetHash, which respectively take continuous numeric values and non-binary discrete values.

To study the search scalability on network size $|\mathcal{V}|$, we first learn 128-dimensional embeddings with DeepWalk, NetHash, and BinaryNE on the whole DBLP(Full) network, and then randomly sample a series of node subsets with increasing sizes. Among each node subset, we randomly select 1,000 nodes as query nodes and search similar nodes with the learned node representations. Figure 2(a) shows query time (in milliseconds) with regard to different network sizes, where both query time (in milliseconds) and $|\mathcal{V}|$ are in logarithmic scales. As is shown, node similarity search with different embedding methods scales linearly with the increase of network size, whereas BinaryNE provides more than 10 times faster query speed than Deepwalk and NetHash.

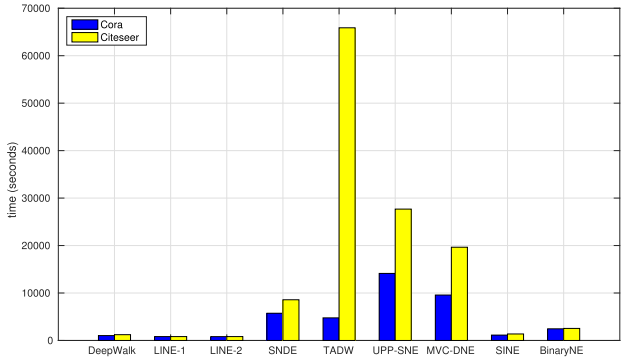


Fig. 3. The time consumed by different network embedding methods for learning node representations.

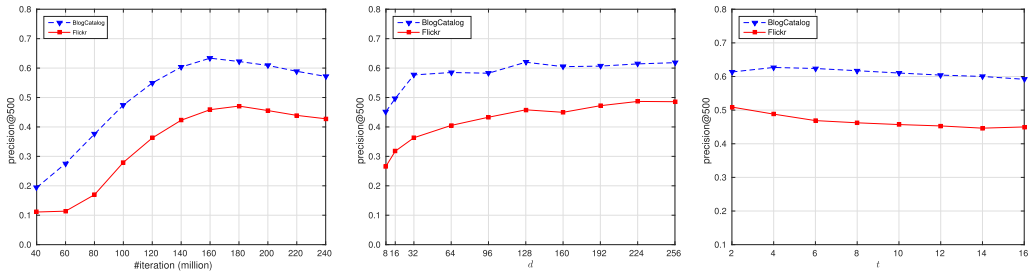
To study the search scalability in terms of embedding dimension d, we learn DeepWalk, NetHash and BinaryNE embeddings with varying dimensions (8, 16, 32, 64, 128, and 256). We randomly select 100 nodes as query nodes, and search similar nodes across the whole DBLP(Full) network. Figure 2(b) shows query time (in milliseconds) with varying embedding dimensions, with both axes in logarithmic scale. We can see that, in general, similarity search with three methods scales almost linearly with regards to embedding dimension, but BinaryNE is consistently more efficient than DeepWalk and Nethash (with more than 10 times search speedup in most cases).

5.9 Comparison of Embedding Learning Time

We now select the Cora and Citeseer network to evaluate the efficiency of learning node representations with different network embedding methods. Figure 3 compares the CPU time (in seconds) consumed by different network embedding methods. As shown in the figure, BinaryNE is far more efficient in learning node representations than SDNE, TADW, UPP-SNE, and MVC-DNE, and its efficiency is comparable to that of DeepWalk, LINE1, LINE2, and SINE, which have been demonstrated to be efficient on large-scale networks. This proves the ability of BinaryNE to scale to large-scale networks for learning node representations, like DeepWalk, LINE, and SINE.

5.10 Experiments on Parameter Sensitivity

Lastly, we perform a case study on BlogCatalog and Flickr to investigate the sensitivity of BinaryNE to three important parameters: the number of iterations, the dimension of learned embeddings d, and the window size t used for collecting node context pairs. We take turns to fix any two parameters and study the effect of the remaining parameter on the search performance measured by precision@500. Figure 4 shows the performance of BinaryNE with respect to varying parameters. As the number of iterations increases, the performance of BinaryNE gradually increases and then declines slightly. This indicates that, in general, more iterations would be helpful for BinaryNE to find the local minimal solution, but excessive iterations tend to make the model parameters deviate from the local minima. However, to guarantee good performance, the number of iterations should scale linearly to the network scale. In practice, the number of iterations should be set to $500 \sim 1000$ times the sum of the number of edges and the number of node-attribute occurrence times. When the embedding dimension d increases, the performance of BinaryNE increases and stabilizes later. This shows that, embeddings with higher dimensions provide more information to measure node similarity. Interestingly, when the window size t increases from 2 to 16, the search precision drops slightly. This is probably because a larger window size imports broader contextual structure, but may introduce more noise to measure node similarity.



(a) #iteration (b) d (c) t

Fig. 4. The sensitivity of BinaryNE with parameters: the number of iterations, the dimension of learned embeddings d, and the window size t.

6 CONCLUSION

Learning binary node representations is a desirable solution to similarity search over large-scale networks, due to its advantage in efficient bit-wise Hamming distance calculation and low memory usage. In this article, we proposed a BinaryNE algorithm to embed network nodes into a binary space, with well preserved network structure and node content features. Through a three-layer neural network, BinaryNE learns binary node representations by modeling node structural context and node attribute relations. The *sign* function is adopted as the activation function in the hidden layer to obtain binary node representations. To deal with the *ill-posed gradient* problem caused by the non-smoothness of the *sign* activation function, the state-of-the-art continuation technique [2, 7] is employed. Model parameters are efficiently learned through an online stochastic gradient descent algorithm, which ensures the low time complexity and great scalability of BinaryNE. Extensive experiments on six real-world networks show that BinaryNE exhibits much lower memory usage and computational cost than continuous network embedding algorithms, but with comparable or even better search precision. The binary node representations learned by BinaryNE also achieve competitive results on other network analytic tasks such as node classification and node clustering.

APPENDIX: EXPERIMENTAL SETTINGS

All reported experiments are conducted on a Mac laptop with an Inter Core i5, 2.3 GHz processor and 8 GB memory, with no GPU or other accelerators used.

For all embedding learning methods used in our experiments, we set the dimension of embeddings $d = 128$. For DeepWalk, UPP-SNE, SINE, and BinaryNE, we set the length of random walks $L = 100$, the number of random walks starting from per node $\gamma = 40$, and the window size $t = 10$.

For fair comparisons, we use the same strategy to train DeepWalk, UPP-SNE, SINE, and BinaryNE: we first collect node context pairs from the generated random walks, and update parameters with stochastic gradient descent by sampling node context pairs. We uniformly set the number of sampled negative context nodes K to 5. For DeepWalk, LINE, UPP-SNE, SINE, and BinaryNE, we set the maximum number of iterations to 100 million for Cora and Citeseer, 200 million for BlogCatalog, Flickr and DBLP(Subgraph). For DeepWalk and BinaryNE, we set the maximum number of iterations to 1 billion for DBLP. For DeepWalk, LINE, UPP-SNE, SINE, and BinaryNE, we gradually decrease the learning rate η from 0.025 to 2.5×10^{-6}. We use the author provided LINE,[5] UPP-SNE,[6] and SINE[7] implementations. We implement DeepWalk based on the SINE implementation.

[5] https://github.com/tangjianpku/LINE.
[6] https://github.com/daokunzhang/UPPSNE.
[7] https://github.com/daokunzhang/SINE.

We use the SDNE implementation[8] provided by authors and implement MVC-DNE based on the SDNE implementation. For SDNE, we find the best hyperparameter setting for α, ν, and β via a grid search from $\{0.01, 0.1\} \times \{0.01, 0.1\} \times \{10, 50\}$ by evaluating on randomly selected 10% nodes for each network. The number of neurons at each layer is set to 2708-512-128, 3312-512-128, 5,196-512-128, 7,575-512-128, and 18,448-512-128 for Cora, Citeseer, BlogCatalog, Flickr, and DBLP(Subgraph), respectively. For MVC-DNE, on Cora, Citeseer, BlogCatalog, Flickr, and DBLP(Subgraph), the number of neurons at each layer in the structure view is set to 2708-512-64, 3312-512-64, 5,196-512-64, 7,575-512-64, and 18,448-512-64, respectively, and the number of neurons at each layer in the node content feature view is set to 1,433-512-64, 3,703-512-64, 8,189-512-64, 12,047-512-64, and 2,476-512-64, respectively. When setting the number of neurons at each layer for SDNE and MVC-DNE, we gradually decrease the number of neurons from the high-dimensional input layer to 128 neurons in the final node representation layer, so as to hierarchically extract deeper and more abstract latent features, as is operated in SDNE [53] and MVC-DNE [59]. For SDNE and MVC-DNE, 500 default epochs are used for pre-training and parameter fine-tuning, respectively. Other parameters of SDNE and MVC-DNE are set according to [59].

As the content feature dimension of BlogCatalog, Flickr, and DBLP(Subgraph) is too large for TADW, before running TADW on them, we reduce the dimension of their node content features to 200 with SVD. Default settings are used to train NetHash and BANE. For BinaryNE, we gradually increase the parameter β from 0.01 to 1. We use the author provided TADW,[9] BANE,[10] and NetHash[11] implementations. We implement the KD-coding for network embedding based on the original KD-coding implementation for word embedding.[12] We implement the DNE algorithm according to the DNE paper [40].

REFERENCES

[1] Lada A. Adamic and Eytan Adar. 2003. Friends and neighbors on the web. *Social Networks* 25, 3 (2003), 211–230.

[2] Eugene L. Allgower and Kurt Georg. 2012. *Numerical Continuation Methods: An Introduction*. Vol. 13. Springer Science & Business Media.

[3] Ginestra Bianconi, Paolo Pin, and Matteo Marsili. 2009. Assessing the relevance of node features for network structure. *Proceedings of the National Academy of Sciences* 106, 28 (2009), 11433–11438.

[4] Andrei Z. Broder, Moses Charikar, Alan M. Frieze, and Michael Mitzenmacher. 2000. Min-wise independent permutations. *Journal of Computer and System Sciences* 60, 3 (2000), 630–659.

[5] Shaosheng Cao, Wei Lu, and Qiongkai Xu. 2015. GraRep: Learning graph representations with global structural information. In *CIKM*. ACM, 891–900.

[6] Shaosheng Cao, Wei Lu, and Qiongkai Xu. 2016. Deep neural networks for learning graph representations. In *AAAI*. AAAI Press, 1145–1152.

[7] Zhangjie Cao, Mingsheng Long, Jianmin Wang, and S. Yu Philip. 2017. HashNet: Deep learning to hash by continuation. In *ICCV*. 5609–5618.

[8] Ting Chen, Martin Renqiang Min, and Yizhou Sun. 2018. Learning k-way d-dimensional discrete codes for compact embedding representations. In *ICML*. 854–863.

[9] Lianhua Chi and Xingquan Zhu. 2017. Hashing techniques: A survey and taxonomy. *ACM Computing Surveys* 50, 1 (2017), 11.

[10] Yunchao Gong, Svetlana Lazebnik, Albert Gordo, and Florent Perronnin. 2012. Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35, 12 (2012), 2916–2929.

[11] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *SIGKDD*. ACM, 855–864.

[8] https://github.com/suanrong/SDNE.

[9] https://github.com/benedekrozemberczki/TADW.

[10] https://github.com/shiruipan/BANE.

[11] https://github.com/williamweiwu/williamweiwu.github.io/tree/master/Graph_Network%20Embedding/NetHash.

[12] https://github.com/chentingpc/kdcode-lm.

[12] Michael U. Gutmann and Aapo Hyvärinen. 2012. Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics. *Journal of Machine Learning Research* 13, 1 (2012), 307–361.
[13] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *NIPS*. 1024–1034.
[14] Taher H. Haveliwala. 2002. Topic-sensitive pagerank. In *WWW*. ACM, 517–526.
[15] Raymond Hemmecke, Matthias Köppe, Jon Lee, and Robert Weismantel. 2010. Nonlinear integer programming. In *50 Years of Integer Programming 1958-2008*. Springer, 561–618.
[16] Xiao Huang, Jundong Li, and Xia Hu. 2017. Accelerated attributed network embedding. In *SDM*. SIAM, 633–641.
[17] Xiao Huang, Jundong Li, and Xia Hu. 2017. Label informed attributed network embedding. In *WSDM*. ACM, 731–739.
[18] Glen Jeh and Jennifer Widom. 2002. SimRank: A measure of structural-context similarity. In *SIGKDD*. ACM, 538–543.
[19] Ruoming Jin, Victor E. Lee, and Hui Hong. 2011. Axiomatic ranking of network role similarity. In *SIGKDD*. ACM, 922–930.
[20] Maxwell Mirton Kessler. 1963. Bibliographic coupling between scientific papers. *American Documentation* 14, 1 (1963), 10–25.
[21] Thomas N. Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. In *ICLR*.
[22] Da Kuang, Chris Ding, and Haesun Park. 2012. Symmetric nonnegative matrix factorization for graph clustering. In *SDM*. SIAM, 106–117.
[23] Mitsuru Kusumoto, Takanori Maehara, and Ken-ichi Kawarabayashi. 2014. Scalable similarity search for SimRank. In *SIGMOD*. ACM, 325–336.
[24] Quoc Le and Tomas Mikolov. 2014. Distributed representations of sentences and documents. In *ICML*. 1188–1196.
[25] Pei Lee, Laks V.S. Lakshmanan, and Jeffrey Xu Yu. 2012. On top-k structural similarity search. In *ICDE*. IEEE, 774–785.
[26] Omer Levy and Yoav Goldberg. 2014. Neural word embedding as implicit matrix factorization. In *NIPS*. 2177–2185.
[27] Aaron Q. Li, Amr Ahmed, Sujith Ravi, and Alexander J. Smola. 2014. Reducing the sampling complexity of topic models. In *SIGKDD*. ACM, 891–900.
[28] Juzheng Li, Jun Zhu, and Bo Zhang. 2016. Discriminative deep random walk for network classification. In *ACL*. Vol. 1. 1004–1013.
[29] 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*. 1744–1753.
[30] Wei Liu, Cun Mu, Sanjiv Kumar, and Shih-Fu Chang. 2014. Discrete graph hashing. In *NIPS*. 3419–3427.
[31] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013).
[32] Vinith Misra and Sumit Bhatia. 2018. Bernoulli embeddings for graphs. In *AAAI*. 3812–3819.
[33] Nagarajan Natarajan and Inderjit S. Dhillon. 2014. Inductive matrix completion for predicting gene–disease associations. *Bioinformatics* 30, 12 (2014), i60–i68.
[34] Shirui Pan, Jia Wu, Xingquan Zhu, Chengqi Zhang, and Yang Wang. 2016. Tri-party deep network representation. In *IJCAI*. 1895–1901.
[35] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. DeepWalk: Online learning of social representations. In *SIGKDD*. ACM, 701–710.
[36] Lise Getoor Qing Lu. 2003. Link-based classification. In *ICML*. 496–503.
[37] 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*. 459–467.
[38] D. A. Rachkovskij. 2017. Binary vectors for fast distance and similarity estimation. *Cybernetics and Systems Analysis* 53, 1 (2017), 138–156.
[39] Anand Rajaraman and Jeffrey David Ullman. 2011. *Mining of Massive Datasets*. Cambridge University Press.
[40] Xiaobo Shen, Shirui Pan, Weiwei Liu, Yew-Soon Ong, and Quan-Sen Sun. 2018. Discrete network embedding. In *IJCAI*. 3549–3555.
[41] Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M. Borgwardt. 2011. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research* 12,9 (2011), 2539–2561.
[42] Henry Small. 1973. Co-citation in the scientific literature: A new measure of the relationship between two documents. *Journal of the American Society for information Science* 24, 4 (1973), 265–269.
[43] Pulipati Srilatha and Ramakrishnan Manjula. 2016. Similarity index based link prediction algorithms in social networks: A survey. *Journal of Telecommunications and Information Technology* 2 (2016), 87–94.
[44] A. Strehl and J. Ghosh. 2003. Cluster ensembles – A knowledge reuse framework for combining multiple partitions. *Journal of Machine Learning Research* 3, 12 (2003), 583–617.
[45] Karthika Subbaraj and Bose Sundan. 2015. What happens next? Prediction of disastrous links in covert networks. *Disaster Advances* 8, 4 (2015), 53–60.

[46] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. LINE: Large-scale information network embedding. In *WWW*. ACM, 1067–1077.
[47] Jie Tang, Jing Zhang, Limin Yao, Juanzi Li, Li Zhang, and Zhong Su. 2008. Arnetminer: Extraction and mining of academic social networks. In *KDD*. 990–998.
[48] Charalampos E. Tsourakakis. 2014. Toward quantifying vertex similarity in networks. *Internet Mathematics* 10, 3-4 (2014), 263–286.
[49] Cunchao Tu, Han Liu, Zhiyuan Liu, and Maosong Sun. 2017. CANE: Context-aware network embedding for relation modeling. In *ACL*, Vol. 1. 1722–1731.
[50] Cunchao Tu, Weicheng Zhang, Zhiyuan Liu, Maosong Sun, et al. 2016. Max-margin DeepWalk: Discriminative learning of network representation. In *IJCAI*. 3889–3895.
[51] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. In *ICLR*.
[52] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. 2010. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research* 11, 12 (2010), 3371–3408.
[53] Daixin Wang, Peng Cui, and Wenwu Zhu. 2016. Structural deep network embedding. In *SIGKDD*. ACM, 1225–1234.
[54] Haoyu Wang, Defu Lian, and Yong Ge. 2019. Binarized collaborative filtering with distilling graph convolutional networks. In *IJCAI*. 4802–4808.
[55] Xiao Wang, Peng Cui, Jing Wang, Jian Pei, Wenwu Zhu, and Shiqiang Yang. 2017. Community preserving network embedding. In *AAAI*. 203–209.
[56] Yair Weiss, Antonio Torralba, and Rob Fergus. 2009. Spectral hashing. In *NIPS*. 1753–1760.
[57] Wei Wu, Bin Li, Ling Chen, and Chengqi Zhang. 2018. Efficient attributed network embedding via recursive randomized hashing. In *IJCAI*. 2861–2867.
[58] Cheng Yang, Zhiyuan Liu, Deli Zhao, Maosong Sun, and Edward Y Chang. 2015. Network representation learning with rich text information. In *IJCAI*. 2111–2117.
[59] Dejian Yang, Senzhang Wang, Chaozhuo Li, Xiaoming Zhang, and Zhoujun Li. 2017. From properties to links: Deep network embedding on incomplete graphs. In *CIKM*. ACM, 367–376.
[60] Hong Yang, Shirui Pan, Peng Zhang, Ling Chen, Defu Lian, and Chengqi Zhang. 2018. Binarized attributed network embedding. In *ICDM*. IEEE, 1476–1481.
[61] Daokun Zhang, Jie Yin, Xingquan Zhu, and Chengqi Zhang. 2016. Collective classification via discriminative matrix factorization on sparsely labeled networks. In *CIKM*. ACM, 1563–1572.
[62] Daokun Zhang, Jie Yin, Xingquan Zhu, and Chengqi Zhang. 2016. Homophily, structure, and content augmented network representation learning. In *ICDM*. IEEE, 609–618.
[63] Daokun Zhang, Jie Yin, Xingquan Zhu, and Chengqi Zhang. 2017. User profile preserving social network embedding. In *IJCAI*. 3378–3384.
[64] Daokun Zhang, Jie Yin, Xingquan Zhu, and Chengqi Zhang. 2020. Network representation learning: A survey. *IEEE Transactions on Big Data* 6, 1 (2020), 3–28.
[65] Daokun Zhang, Jie Yin, Xingquan Zhu, and Chengqi Zhang. 2018. SINE: Scalable incomplete network embedding. In *ICDM*. IEEE.
[66] Jing Zhang, Jie Tang, Cong Ma, Hanghang Tong, Yu Jing, and Juanzi Li. 2015. Panther: Fast top-k similarity search on large networks. In *SIGKDD*. ACM, 1445–1454.
[67] Kang Zhao, Hongtao Lu, and Jincheng Mei. 2014. Locality preserving hashing. In *AAAI*.
[68] Peixiang Zhao, Jiawei Han, and Yizhou Sun. 2009. P-Rank: A comprehensive structural similarity measure over information networks. In *CIKM*. ACM, 553–562.
[69] Han Zhu, Mingsheng Long, Jianmin Wang, and Yue Cao. 2016. Deep hashing network for efficient similarity retrieval. In *AAAI*. 2415–2421.

Received September 2019; revised September 2020; accepted November 2020