High-order Proximity Preserving Information Network Hashing

Defu Lian, Kai Zheng*
School of Computer Science and Engineering, University of Electronic Science and Technology of China
{dove,zhengkai}@uestc.edu.cn

Vincent W. Zheng
Advanced Digital Sciences Center, Singapore
vincent.zheng@adsc-create.edu.sg

Yong Ge
Management Information Systems, University of Arizona
yongge@email.arizona.edu

Longbing Cao
Advanced Analytics Institute, University of Technology Sydney
longbing.cao@uts.edu.au

Ivor W. Tsang
Centre for Artificial Intelligence, University of Technology Sydney
Ivor.Tsang@uts.edu.au

Xing Xie
Microsoft Research Asia
xingx@microsoft.com

ABSTRACT

Information network embedding is an effective way for efficient graph analytics. However, it still faces with computational challenges in problems such as link prediction and node recommendation, particularly with increasing scale of networks. Hashing is a promising approach for accelerating these problems by orders of magnitude. However, no prior studies have been focused on seeking binary codes for information networks to preserve high-order proximity. Since matrix factorization (MF) unifies and outperforms several well-known embedding methods with high-order proximity preserved, we propose a MF-based Information Network Hashing (INH-MF) algorithm, to learn binary codes which can preserve high-order proximity. We also suggest Hamming subspace learning, which only updates partial binary codes each time, to scale up INH-MF. We finally evaluate INH-MF on four real-world information network datasets with respect to the tasks of node classification and node recommendation. The results demonstrate that INH-MF can perform significantly better than competing learning to hash baselines in both tasks, and surprisingly outperforms network embedding methods, including DeepWalk, LINE and NetMF, in the task of node recommendation. The source code of INH-MF is available online[1].

CCS CONCEPTS

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

KEYWORDS

Information Network Hashing, Matrix Factorization, Hamming Subspace Learning

*The corresponding author
[1]https://github.com/DefuLian/network

KDD'18, August 19–23, 2018, London, United Kingdom

ACM ISBN 978-1-4503-5552-0/18/08…$15.00
https://doi.org/10.1145/3219819.3220034

ACM Reference Format:
Defu Lian, Kai Zheng, Vincent W. Zheng, Yong Ge, Longbing Cao, Ivor W. Tsang, and Xing Xie. 2018. High-order Proximity Preserving Information Network Hashing. In *KDD'18: The 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, August 19–23, 2018, London, United Kingdom.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3219819.3220034

1 INTRODUCTION

Information networks are ubiquitous in a wide diversity of real-world scenarios, such as social network, road network, communication network, and the World Wide Web. Effective network analytics can benefit a lot of applications, ranging from classifying the role of a protein [13] to recommending new friends to a user [2]. Network embedding, learning low-dimensional real-valued vector representation for networks, becomes an effective way for subsequent efficient network analytics, such as node classification/clustering/recommendation and link prediction. Recently, lots of efforts have been devoted to improving the capability of preserving high-order proximity within the networks in an explicit or implicit way, such as DeepWalk [27], LINE (second-order) [35], node2vec [12], Grarep [7], NetMF [28], and HOPE [26].

However, it still faces with computational challenges in problems like node recommendation and link prediction. For example, recommending a new friend for all users in a social network of size n costs $O(n^2d)$, if d is dimension of representation. Since real-world social networks may contains hundreds of millions of nodes, this computational cost will be extremely high in practice. Moreover, as social networks keep evolving due to addition/deletion of new edges, network embedding algorithms update representation frequently, and list of recommended friends accordingly. The key of these problems is k-nearest neighbor (knn) search, i.e., seeking the top-k most "similar" nodes for a given "query" node. It is well-known that hashing is a promising approach for fast similarity search [37]. This not only attributes to replacement of dot-product similarity computation with fast Hamming distance calculation, but also to the usage of index structures, like multi-index hashing [25], which may have sub-linear run-time behavior. However, to the best of our knowledge, no prior studies have been focused on seeking binary representation (dubbed binary codes) for information networks to preserve high-order proximity.

The key obstacles that hinder direct exploitation of graph hashing [17, 20, 22, 23, 32] or other existing learning to hash methods,

including spectral hashing (SH) [38], inductive manifold hashing (IMH) [31] and interactive quantization (ITQ) [11], to learn binary codes are two-fold. First, high-order proximity is not preserved even in graph hashing methods, but plays a tremendously important role in network embedding. Note that graph hashing methods are usually applied on constructed graphs from non-relational data. Second, adjacency matrices, for representing information networks, are usually very sparse. The methods like ITQ and SH cannot work with sparse matrices, since they inevitably depend on PCA, which would convert sparse matrices to dense due to zero-centering. Hashing methods for collaborative filtering [39] are also different, since they handle bipartite graphs and necessarily hash both types of nodes. In contrast, we concentrate on homogeneous graphs, in which all nodes are of the same type but each node plays two roles: the node itself and a "context" of other nodes [35]. Hence, it is unnecessary to hash the "context" role of nodes. Moreover, high-order proximity preserving has been almost ignored in hashing methods for collaborative filtering.

To this end, we study learning to hash information networks, which has not been well investigated yet. Within it, information networks are embedded into a low-dimensional binary Hamming space while the sparsity challenge is addressed and high-order proximity between nodes is preserved. According to recent study [28], matrix factorization (MF) unifies several important embedding algorithms with high-order proximity preserved, including DeepWalk, LINE, and node2vec, and shows superior performance to them. Therefore, we propose a MF-based Information Network Hashing (INH-MF) algorithm to simultaneously alleviate sparsity and preserve high-order proximity. It is well-known that obtaining optimal binary codes for nodes is generally NP-hard due to binary constraints [14]. Therefore, we resort to alternating optimization to directly tackle the challenging mixed-integer problem. We find that orthonormal constraints on representation of the "context" role of nodes will lead to a closed form for updating binary codes. For scaling up INH-MF, we further suggest Hamming subspace learning, which only updates parts of binary codes each time. We finally evaluate both effectiveness and efficiency of INH-MF on four real-world information network datasets with respect to the tasks of node classification and node recommendation. Note that node classification is commonly used to evaluate effectiveness of network embedding. The results show that INH-MF performs significantly better than several competing learning to hash algorithms, and surprisingly outperforms network embedding algorithms, including LINE, DeepWalk, and NetMF, in the task of node recommendation.

To summarize, we make the following contributions:

- We study learning to hash information networks for the first time with the aim of dramatically accelerating knn-dependent network analytics, and present its unique characteristics and challenges.
- We propose an information network hashing algorithm based on matrix factorization (INH-MF), which can *address the sparsity challenge* and *preserve high-order proximity*, and develop an efficient parameter learning algorithm based on alternating optimization, which can update parameters in close forms.
- We suggest Hamming subspace learning *to deal with growing density of networks due to high-order proximity preserving*. The evaluation results show it can speed up INH-MF dramatically with a little sacrifice of performance of node classification.
- We extensively evaluate INH-MF on four real-world information network datasets. The results not only show INH-MF outperforms the competing learning to hash baselines, but also demonstrates that *binary representation is surprisingly better at node recommendation than real-valued.*

2 RELATED WORK

This paper is about unsupervised network hashing, so we mainly review unsupervised leaning to hash algorithms, particularly graph hashing, followed by recent advance of network embedding.

2.1 Learning to Hash

Learning to hash algorithms can fall into two categories [37]: two stage approaches and discrete hashing. Two stage approaches first derive real-valued feature learning and then apply quantization methods to obtain binary codes. For example, Salakhutdinov and Hinton [29] proposed semantic hashing (SH) to learn binary codes via Restricted Boltzmann Machine for fast searching similar documents. Weiss et al. applied spectral analysis techniques to a constructed similarity graph between data points, and embedded it into a low-dimensional space [38]. Moreover, this work has introduce balanced and uncorrelated constraints for deriving compact binary codes. Liu et al. proposed anchor graph hashing to scale up spectral analysis [23]. Shen et al. proposed inductive manifold hashing (IMH) for exploiting manifold structures [31]. Noticing variance of the data in each PCA direction is different, Gong et al. proposed iterative quantization to rotate representation for deriving more effective binary codes [11]. A similar idea was also studied in [19] with the goal of isotropic variances. Note that methods except IMH and SH inevitably depend on PCA, which cannot be applicable for large sparse matrices due to zero-centering. Discrete hashing directly learns binary codes. For example, Discrete Graph Hashing [22], Supervised Discrete Hashing [30], Asymmetric Discrete Graph Hashing [32] and Scalable Graph Hashing [17] were proposed for joint optimization quantization losses and intrinsic objective functions. The idea of discrete hashing has also been applied for hashing based recommender system (RS) [21, 39, 40] recently. However, rating data in RS can be organized as bipartite graph, being different from homogeneous networks, as aforementioned.

2.2 Network Embedding

Network embedding is also different from graph embedding such IsoMap and MDS, as discussed in [8, 35]. Hence, pioneer work is DeepWalk [27] and Graph Factorization [1]. Graph Factorization directly factorizes adjacency matrix to obtain a low-dimensional vector. DeepWalk first adopts a truncated random walk on a network to generate a set of walk sequences, and applies Skip-Gram to obtain embeddings of nodes. Similar to DeepWalk, node2vec [12] also optimizes embeddings to encode the statistics of random walks, but designs a second-order random walk strategy to sample neighbor nodes. LINE [35] is proposed for large scale network embedding, and can preserve the first and second order proximity. Grarep [7] demonstrates that k-order proximity should also be captured when

constructing the global representations of nodes. Recently, a general matrix factorization framework is proposed for unifying DeepWalk, node2vec and LINE, within which LINE is a special case of DeepWalk. HOPE [26] summarizes four measurements of high-order proximity in a general formulation, and then applies generalized SVD to get node embedding. Neighborhood aggregation is another framework for embedding, including graph convolutional network [18] and structure2vec [9]. For more literatures about network embedding, refer to recent survey papers, such as [4, 8].

3 A GENERAL FRAMEWORK OF INFORMATION NETWORK HASHING

3.1 Problem Statement

An information network is defined as $G = \langle V, E \rangle$, where V is the set of vertexes/nodes of size $n = |V|$ and E is the set of edges between nodes, representing a relationship. G can be represented by an adjacency matrix $\boldsymbol{A}$, where $A_{i,j}$ is edge weight between node i and node j, indicating the strength of relationship. The high-order proximity matrix between nodes is represented by another matrix $\boldsymbol{S}$, which should be preserved by network embedding. The second-order proximity matrix is directly related to the adjacency matrix and the k-th order proximity matrix can be constructed by the (k-1)-th order proximity matrix. Several ways of construction can be referred in [7, 28]. With these notations, we then define the information network hashing problem.

Definition 3.1 (Information Network Hashing). Given an information network $G = \langle V, E \rangle$, the problem of information network hashing aims to represent each node $v \in V$ into a low-dimensional Hamming space $\{\pm 1\}^d$, i.e., learning a hash function $h_G : V \to \{\pm 1\}^d$, where $d \ll n$. In this space $\{\pm 1\}^d$, not only high-order proximity between nodes should be preserved, but also bit uncorrelation and balance conditions should be satisfied as much as possible.

Bit uncorrelation and balance conditions have been introduced by [38] to approximate the code balance condition: the number of data items mapped to each binary code is the same. Bit balance means that each bit has almost equal chance of being 1 or -1, equivalent to maximizing the entropy of each bit. Bit uncorrelation means that different bits are uncorrelated, being usually implemented by orthogonal constraints.

3.2 Overview

Information network hashing is based on learning to hash algorithms with discrete hashing, i.e., treating binary codes as parameters and learning these codes directly. Binary code for node i is denoted as a column vector $\boldsymbol{b}_i \in \{\pm 1\}^d$, which only capture the role of node itself. The "context" role of the node i is represented by a column vector $\boldsymbol{q}_i \in R^d$, maybe with some constraints. We also denote $\boldsymbol{B} = [\boldsymbol{b}_1, \cdots, \boldsymbol{b}_n]^T$, a matrix of size $n \times d$, and $\boldsymbol{Q} = [\boldsymbol{q}_1, \cdots, \boldsymbol{q}_n]^T$, a matrix of size $n \times d$. Then the bit uncorrelation and balance is implemented by $\boldsymbol{B}^T\boldsymbol{B} = n\boldsymbol{I}_d$ and $\boldsymbol{B}^T\boldsymbol{1}_n = \boldsymbol{0}$, respectively, where $\boldsymbol{1}_n$ is a vector of length n with all ones. Note that these two conditions may be not strictly satisfied. For example, if n is odd, $\boldsymbol{B}^T\boldsymbol{1}_n \neq \boldsymbol{0}$. Therefore, we only ensure that they are satisfied as much as possible. Similarly, it is pretty possible for constraints to be imposed on $\boldsymbol{Q}$, such as spectral norm $\|\boldsymbol{Q}\|_2 \leq 1$ or $\boldsymbol{Q}^T\boldsymbol{Q} = \boldsymbol{I}_d$. We denote $\mathcal{Q}$ as a set of $\boldsymbol{Q}$ with these constraints. Based on $\boldsymbol{B}$ and $\boldsymbol{Q}$, the proximity between node i and j is estimated by

$$\hat{S}_{i,j} = \boldsymbol{b}_i^T \boldsymbol{q}_j. \tag{1}$$

Therefore the loss function for information network hashing can be defined as

$$\begin{aligned} &\min_{\boldsymbol{B},\boldsymbol{Q}} \mathcal{L}_G(\boldsymbol{B},\boldsymbol{Q}) = \sum_{(i,j)\in O} \ell(S_{i,j}, \hat{S}_{i,j}), \\ &\text{s.t. } \boldsymbol{Q} \in \mathcal{Q}, \boldsymbol{B}^T\boldsymbol{1}_n = 0, \boldsymbol{B}^T\boldsymbol{B} = n\boldsymbol{I}_k \end{aligned} \tag{2}$$

where O denotes the edge set, whose edges are necessarily taken into account to preserve proximity, and $\ell(x, y)$ is the loss function, which can be squared loss $\ell(x, y) = (x - y)^2$, absolute loss $\ell(x, y) = |x - y|$. It can also be replaced with ranking based loss function for ranking-based information network hashing.

4 INH-MF: INFORMATION NETWORK HASHING BASED ON MATRIX FACTORIZATION

In this section, based on the general framework for information network hashing, we propose one of its instantiations – Information Network Hashing based on Matrix Factorization (INH-MF).

4.1 Preliminary

Matrix factorization is used for hashing information networks in this paper, since it has been proved to unify DeepWalk, node2vec and LINE, and shows superior performance to them according to [28]. In particular, DeepWalk implicitly approximates and factorizes the following matrix

$$\boldsymbol{S} = \log\left(\mathrm{vol}(G)\left(\frac{1}{T}\sum_{r=1}^{T}(\boldsymbol{D}^{-1}\boldsymbol{A})^r\right)\boldsymbol{D}^{-1}\right) - \log b, \tag{3}$$

where $\mathrm{vol}(G) = \sum_i \sum_j A_{i,j}$ is volume of an information network G, $\boldsymbol{D} = \mathrm{diag}([d_1, \cdots, d_n])$ such that d_i represents generalized degree of node i, T the context widow size ($T + 1$ is order of proximity), b the number of negative samples in skip-gram. LINE(2nd) is a special case of DeepWalk, by setting $T = 1$, that is

$$\boldsymbol{S} = \log(\boldsymbol{D}^{-1}\boldsymbol{A}\boldsymbol{D}^{-1}) + \log \mathrm{vol}(G) - \log b \tag{4}$$

Therefore, different from Graph Factorization, LINE normalizes weight of each edge by the generalized degree of both connected nodes. Such a normalization step plays an important role in network embedding according to experimental results of LINE [35]. There are also other methods for constructing high-order proximity matrix [7, 26], but they are not discussed any more.

To obtain network embedding with k-th order proximity preserved, truncated Singular Value Decomposition (SVD) is suggested for factorizing the proximity matrix $\boldsymbol{S}$ in Eq (3) by setting $T = k - 1$. In particular, $\boldsymbol{S} \approx \boldsymbol{U}_d\boldsymbol{\Sigma}_d\boldsymbol{V}_d^T$, then embedding of a node is suggested as the corresponding row of $\boldsymbol{U}_d\boldsymbol{\Sigma}_d^{\frac{1}{2}}$ according to [7, 28].

4.2 Model Description

A straightforward way to derive binary codes for nodes is applying the sign function to the obtained network embedding, that is

$\boldsymbol{B} = \text{sign}(\boldsymbol{U}_d \Sigma_d^{\frac{1}{2}})$. This way of deriving binary codes is called two-stage approaches. However, according to [22, 30, 39], two-stage approaches may incur large quantization loss. They show that discrete hashing, to treat hash codes as parameters for learning, is a more promising alternative. Next, we first derive the loss function for discrete hashing.

Truncated Singular Value Decomposition $\boldsymbol{S}_d = \boldsymbol{U}_d \Sigma_d \boldsymbol{V}_d^T$ is the best rank-d approximation of $\boldsymbol{S}$ [3], that is, $\|\boldsymbol{S} - \boldsymbol{S}_d\|_F \leq \|\boldsymbol{S} - \boldsymbol{B}\|_F$, where $\boldsymbol{B}$ is any matrix of rank at most d. Hence, we can express truncated SVD as the following optimization problem

$$\boldsymbol{S}_d = \arg\min_{\boldsymbol{B} \in R^{n \times n}} \|\boldsymbol{S} - \boldsymbol{B}\|_F, \text{ s.t. } \text{rank}(B) \leq d. \tag{5}$$

Since every finite-dimensional matrix has a rank factorization, $\boldsymbol{B} = \boldsymbol{P}\boldsymbol{Q}^T$, where $\boldsymbol{P}, \boldsymbol{Q} \in R^{n \times d}$, this optimization can be re-written as

$$\min_{\boldsymbol{P}, \boldsymbol{Q} \in R^{n \times d}} \|\boldsymbol{S} - \boldsymbol{P}\boldsymbol{Q}^T\|_F^2. \tag{6}$$

Here we assume $\boldsymbol{P}$ and $\boldsymbol{Q}$ corresponds to the role of node itself and "context" of other nodes, respectively. This loss function is not convex with respect to $(\boldsymbol{P}, \boldsymbol{Q})$, but convex with respect to $\boldsymbol{P}$ or $\boldsymbol{Q}$ given the other fixed. And all local minima of this objective function are global [33]. Hence, alternating optimization is suggested, and column orthonormal constraint can be imposed on $\boldsymbol{Q}$, i.e., $\boldsymbol{Q}^T\boldsymbol{Q} = \boldsymbol{I}_k$, for faster descending. When fixing $\boldsymbol{Q}$, $\boldsymbol{P} = \boldsymbol{S}\boldsymbol{Q}$ is optimal; when fixing $\boldsymbol{P}$, $\min_{\boldsymbol{Q}} \|\boldsymbol{S} - \boldsymbol{P}\boldsymbol{Q}^T\|$ is equivalent to $\max_{\boldsymbol{Q}} \text{trace}(\boldsymbol{Q}^T\boldsymbol{S}^T\boldsymbol{P})$, which also has a closed form of updating rule. The latter optimization will be elaborated in the next subsection. Note that when $\boldsymbol{Q} = \boldsymbol{V}_d$ and $\boldsymbol{P} = \boldsymbol{S}\boldsymbol{Q}$, the gradient with respect to $(\boldsymbol{P}, \boldsymbol{Q})$ vanishes. Hence, $(\boldsymbol{S}\boldsymbol{V}_d, \boldsymbol{V}_d)$ is a critical point of this objective function.

Since $\boldsymbol{P}$ corresponds to embedding of nodes, it is better to obtain binary codes $\boldsymbol{B}$ by applying thresholding function on a rotated $\hat{\boldsymbol{P}} = \boldsymbol{P}\boldsymbol{W}$, where $\boldsymbol{W} \in R^{k \times k}$ is a orthonormal matrix [11]. Then $\boldsymbol{P}\boldsymbol{Q}^T = \hat{\boldsymbol{P}}(\boldsymbol{Q}\boldsymbol{W})^T$. For the sake of discrete hashing, $\hat{\boldsymbol{P}}$ is replaced with $\boldsymbol{B}$, to be directly learned from $\boldsymbol{S}$. Since $(\boldsymbol{Q}\boldsymbol{W})^T\boldsymbol{Q}\boldsymbol{W} = \boldsymbol{I}_d$, by denoting $\hat{\boldsymbol{Q}} = \boldsymbol{Q}\boldsymbol{W}$, the objective function for learning to hash information networks is formulated as follows:

$$\begin{aligned} &\min_{\boldsymbol{B}, \hat{\boldsymbol{Q}}} \mathcal{L}_G(\boldsymbol{B}, \hat{\boldsymbol{Q}}) = \|\boldsymbol{S} - \boldsymbol{B}\hat{\boldsymbol{Q}}^T\|_F^2 \\ &\text{s.t. } \hat{\boldsymbol{Q}}^T\hat{\boldsymbol{Q}} = \boldsymbol{I}_d, \boldsymbol{B}^T\boldsymbol{1}_n = 0, \boldsymbol{B}^T\boldsymbol{B} = n\boldsymbol{I}_d \end{aligned} \tag{7}$$

where the bit balance and uncorrelation constraints are imposed. And $\hat{\boldsymbol{Q}}$ plays two roles: dimension reduction and rotation of basis. Subsequently, the hat above $\boldsymbol{Q}$ will be dropped for convenience.

4.3 Model Optimization

Due to binary constraints, obtaining optimal binary codes is NP-hard. Therefore, we resort to alternating optimization, which takes turns in updating each parameter given others fixed. Next, we derive the updating rules for $\boldsymbol{B}$ and $\boldsymbol{Q}$.

4.3.1 Learning $\boldsymbol{Q}$. When fixing $\boldsymbol{B}$, via simple algebra, the optimization problem for $\boldsymbol{Q}$ becomes as follows:

$$\max_{\boldsymbol{Q}} \text{trace}(\boldsymbol{Q}^T\boldsymbol{S}^T\boldsymbol{B}), \text{ s.t. } \boldsymbol{Q}^T\boldsymbol{Q} = \boldsymbol{I}_d \tag{8}$$

This is also equivalent to

$$\min_{\boldsymbol{Q} \in Q} \|\boldsymbol{Q} - \boldsymbol{S}^T\boldsymbol{B}\|_F^2, \tag{9}$$

where $Q = \{\boldsymbol{Q} \in R^{n \times d} | \boldsymbol{Q}^T\boldsymbol{Q} = \boldsymbol{I}_d\}$ is Stiefel manifold [10]. Actually Eq (9) corresponds to projecting $\boldsymbol{S}^T\boldsymbol{B}$ to the Stiefel manifold, which can be solved analytically. In particular, according to Von Neumann's trace inequality [15], $\text{trace}(\boldsymbol{Q}^T\boldsymbol{S}^T\boldsymbol{B}) \leq \sum_i \sigma_i(\boldsymbol{Q})\sigma_i(\boldsymbol{S}^T\boldsymbol{B}) = \sum_{i=1}^d \sigma_i(\boldsymbol{S}^T\boldsymbol{B})$, where $\sigma_i(\boldsymbol{A})$ is the i-th largest singular value of $\boldsymbol{A}$ and $\sigma_i(\boldsymbol{Q}) = 1, \forall i \in \{1, \cdots, d\}$. Assume $\boldsymbol{S}^T\boldsymbol{B} = \boldsymbol{X}\Sigma\boldsymbol{Y}^T$ is the thin SVD of $\boldsymbol{S}^T\boldsymbol{B}$, then the optimal solution for Eq (8) is

$$\boldsymbol{Q}^* = \boldsymbol{X}\boldsymbol{Y}^T \tag{10}$$

Below we denote $\mathcal{P}_\perp(\boldsymbol{A})$ the projection of $\boldsymbol{A}$ to the Stiefel manifold of the same size as $\boldsymbol{A}$ for brevity, then $\boldsymbol{Q}^* = \mathcal{P}_\perp(\boldsymbol{S}^T\boldsymbol{B})$.

4.3.2 Learning $\boldsymbol{B}$. When fixing $\boldsymbol{Q}$, via simple algebra, the optimization problem for $\boldsymbol{B}$ is formulated as:

$$\max_{\boldsymbol{B}} \text{trace}(\boldsymbol{Q}^T\boldsymbol{S}^T\boldsymbol{B}), \text{ s.t. } \boldsymbol{B}^T\boldsymbol{B} = n\boldsymbol{I}_d \text{ and } \boldsymbol{B}^T\boldsymbol{1} = \boldsymbol{0} \tag{11}$$

This is also equivalent to

$$\min_{\boldsymbol{B} \in \mathcal{B}_0 \cap \mathcal{B}_\perp} \|\boldsymbol{B} - \boldsymbol{S}\boldsymbol{Q}\|_F^2, \tag{12}$$

which can also be viewed projecting $\boldsymbol{S}\boldsymbol{Q}$ to the intersection of a set $\mathcal{B}_0 = \{\boldsymbol{B} \in \{\pm 1\}^{n \times k} | \boldsymbol{B}^T\boldsymbol{1} = \boldsymbol{0}\}$ and a set $\mathcal{B}_\perp = \{\boldsymbol{B} \in \{\pm 1\}^{n \times k} | \boldsymbol{B}^T\boldsymbol{B} = n\boldsymbol{I}_d\}$. Here the bit balance and uncorrelation are split and handled separately, since we observe the bit balance condition can be easier to deal with. And note that it is not necessary to multiply $\boldsymbol{S}\boldsymbol{Q}$ with an orthonormal matrix $\boldsymbol{W}$ for rotation, since $\boldsymbol{Q}$ has already taken it into account.

Due to the difficulty of handling the uncorrelation constraint, we approximate it by introducing $\mathcal{Z} = \{\boldsymbol{Z} \in R^{n \times d} | \boldsymbol{Z}^T\boldsymbol{Z} = n\boldsymbol{I}_d\}$ and adding a penalty that a feasible $\boldsymbol{B}$ deviates a lot from the set $\mathcal{Z}$. Then the objective function for optimizing $\boldsymbol{B}$ becomes

$$\min_{\boldsymbol{B} \in \mathcal{B}_0, \boldsymbol{Z} \in \mathcal{Z}} \|\boldsymbol{B} - \boldsymbol{S}\boldsymbol{Q}\|_F^2 + \gamma\|\boldsymbol{B} - \boldsymbol{Z}\|_F^2, \tag{13}$$

When $\boldsymbol{Z}$ fixed, the optimization problem for $\boldsymbol{B}$ is reformulated as

$$\min_{\boldsymbol{B} \in \mathcal{B}_0} \|\boldsymbol{B} - (\boldsymbol{S}\boldsymbol{Q} + \gamma\boldsymbol{Z})\|_F^2, \tag{14}$$

This is also called projecting $\boldsymbol{S}\boldsymbol{Q} + \gamma\boldsymbol{Z}$ onto a balanced Hamming space. Denoting $\boldsymbol{\Phi} = \boldsymbol{S}\boldsymbol{Q} + \gamma\boldsymbol{Z}$, and it is easy to see that the optimal $\boldsymbol{B}$ is obtained by

$$\boldsymbol{B}^* = \text{sign}(\boldsymbol{\Phi} - \boldsymbol{1}_n^T\boldsymbol{\lambda}), \tag{15}$$

where $\boldsymbol{\lambda} = \text{median}(\boldsymbol{\Phi})$ of length d is column median of $\boldsymbol{\Phi}$ and is also considered a multiplier of the bit balance constraint.

Due to introducing a new delegate variable $\boldsymbol{Z}$, it is also necessary to update it when $\boldsymbol{B}$ is updated. The updating rule can be easily derived, since it is very similar to projection onto Stiefel manifold. In particular, if $\mathcal{P}_\perp(\boldsymbol{B})$ is denoted the projection of $\boldsymbol{B}$ onto the corresponding Stiefel manifold, $\boldsymbol{Z}$ can be updated according to

$$\boldsymbol{Z}^* = \sqrt{n}\mathcal{P}_\perp(\boldsymbol{B}). \tag{16}$$

4.4 Hamming Subspace Learning

Since high-order proximity is usually preserved in network embedding, proximity matrix $\boldsymbol{S}$ may become much denser than $\boldsymbol{A}$, so much more time is cost for obtaining network embedding with high-order proximity preserved. In order to scaling up information network hashing algorithms, we suggest Hamming subspace learning, to update partial binary codes each time, similar to [24]. Next, we present how to design a loss function to select a subset of bits

for updating. Assuming $\boldsymbol{e} \in \{0,1\}^d$ a bit-selection variable, if the i-th bit of binary code is selected, $e_i = 1$. Then we optimize the following function w.r.t $\boldsymbol{e}$

$$\max_{\boldsymbol{e}:|\boldsymbol{e}|<l} \mathcal{L}_G(\boldsymbol{e}) = \|S - \boldsymbol{B}\text{diag}(\boldsymbol{e})\boldsymbol{Q}^T\|_F^2 \propto -\sum_{i=1}^{d} e_i \boldsymbol{q}_i S^T \boldsymbol{b}_i \quad (17)$$

where l is the number of selected bits. This optimization problem aims to find worst-fit subspaces to update the corresponding bits of binary codes, similar to [34]. Obviously, this optimization has an optimal solution. That is, we first score the i-th bit by $\boldsymbol{q}_i S^T \boldsymbol{b}_i$, and select l bits with smallest scores.

4.5 Initialization

As discussed before, obtaining optimal binary codes is NP-hard, so a good initialization of binary codes is important. Truncated SVD on sparse matrices may not be efficient and even cannot return convergent singular vectors, as evidenced by runtime errors of svds routine of scipy on 50% of the Flickr dataset in the experiment. Therefore, we solve the problem in Eq (6) via alternating optimization to obtain $(\boldsymbol{P}, \boldsymbol{Q})$. Then $\boldsymbol{B}$ is initialized via Eq (15) by setting $\boldsymbol{\Phi} = \boldsymbol{P}$. It is also possible to solve a relaxed problem of Eq (7) by replacing $\boldsymbol{B}$ with a real-valued matrix, but it will be left for future work.

4.6 Convergence and Complexity

Convergence Due to the way of handling uncorrelation, the final objective function for optimization is

$$\min_{\boldsymbol{B}\in\mathcal{B}_0, \boldsymbol{Q}\in\mathcal{Q}, \boldsymbol{Z}\in\mathcal{Z}} \mathcal{L}_G(\boldsymbol{B}, \boldsymbol{Q}, \boldsymbol{Z}) = \|S - \boldsymbol{B}\boldsymbol{Q}^T\|_F^2 + \gamma\|\boldsymbol{B} - \boldsymbol{Z}\|_F^2 \quad (18)$$

The overall algorithm for optimizing this objective function is presented in Algorithm 1, which can be convergent according to Theorem 4.1.

Algorithm 1: INH-MF

```
Initialize B_0 ;
cache Ψ = S^T B_0;
Q_0 ← P_⊥(Ψ); // Eq (10)
Z_0 ← √n P_⊥(B_0); // Eq (16)
t ← 0 ;
repeat
    e ← bit-selection(S, B_{t-1}, Q_{t-1}); // O(nd)
    B_t ← solve Eq(13) with e; // O(‖S‖_0 l)
    update Ψ with B_t and e; // O(‖S‖_0 l)
    Q_t ← P_⊥(Ψ); // O(nd^2)
    Z_t ← √n P_⊥(B_t); // O(nd^2)
    t ← t + 1 ;
until convergent;
```

Theorem 4.1. *$\mathcal{L}_G(\boldsymbol{B}, \boldsymbol{Q}, \boldsymbol{Z})$ is non-increasing based on optimization in Algorithm 1.*

Proof. Eq (13) is based on Frobenius norm, so it can be divided into two parts according to $\boldsymbol{e}$, by splitting $\boldsymbol{B}$ by columns. The part includes submatrix of $\boldsymbol{B}$ with selected columns (bits) is minimized and the other part keeps fixed. Hence, $\mathcal{L}_G(\boldsymbol{B}_{t-1}, \boldsymbol{Q}_{t-1}, \boldsymbol{Z}_{t-1}) \geq \mathcal{L}_G(\boldsymbol{B}_t, \boldsymbol{Q}_{t-1}, \boldsymbol{Z}_{t-1})$. Since $\boldsymbol{Q}_t$ minimizes $\|S - \boldsymbol{B}_t\boldsymbol{Q}^T\|_F^2$ s.t. $\boldsymbol{Q} \in \mathcal{Q}$ and $\boldsymbol{Z}_t$ minimizes $\gamma\|\boldsymbol{B}_t - \boldsymbol{Z}\|_F^2$ s.t. $\boldsymbol{Z} \in \mathcal{Z}$, $\mathcal{L}_G(\boldsymbol{B}_t, \boldsymbol{Q}_{t-1}, \boldsymbol{Z}_{t-1}) \geq \mathcal{L}_G(\boldsymbol{B}_t, \boldsymbol{Q}_t, \boldsymbol{Z}_{t-1}) \geq \mathcal{L}_G(\boldsymbol{B}_t, \boldsymbol{Q}_t, \boldsymbol{Z}_t)$. Putting them together, we have $\mathcal{L}_G(\boldsymbol{B}_{t-1}, \boldsymbol{Q}_{t-1}, \boldsymbol{Z}_{t-1}) \geq \mathcal{L}_G(\boldsymbol{B}_t, \boldsymbol{Q}_t, \boldsymbol{Z}_t)$. □

Complexity Computing $\boldsymbol{e}$ is achieved by element-wise multiplication between $\boldsymbol{Q}$ and $\boldsymbol{\Psi}$, and subsequent search of the top-k smallest values, so it costs $O(nd)$. Updating $\boldsymbol{B}$ with $\boldsymbol{e}$ according to Eq (15) relies on multiplication of $\boldsymbol{S}$ and $\boldsymbol{W}$ with selected columns and on computation of column median. Hence, it costs $O(\|S\|_0 l)$, where $\|S\|_0$ denotes the number of non-zeros in $\boldsymbol{S}$ and $|\boldsymbol{e}| = l$. Updating $\boldsymbol{Q}$ and $\boldsymbol{Z}$ only depends on thin SVD and thus costs $O(nd^2)$. Hence, overall complexity of each round is $O(nd^2 + \|S\|_0 l)$.

5 EXPERIMENTS

In this section, we evaluate INH-MF with respect to the commonly-used tasks of node classification and node recommendation [8]. These two tasks are vital to evaluating the effectiveness of network embedding. The latter task is also useful for evaluating efficiency improvement due to hashing.

5.1 Experimental Setup

5.1.1 Datasets. We evaluate the algorithms with four widely-used real-world network datasets. 1) BlogCatalog [36], a network of social relationships of bloggers in the BlogCatalog website, whose labels represents interests of bloggers. 2) Protein-Protein Interactions (PPI) [12], a subgraph of the PPI network for Homo Sapiens, whose labels represent biological states. 3) Wikipedia is a co-occurrence network of words appearing in the first million bytes of the Wikipedia dump. The labels are Part-of-Speech (POS) tags inferred using the Stanford POS-Tagger. 4) Flickr [36], a network of contacts between Flickr users, whose labels represent the user's interest group. The statistics of these datasets are shown in Table 1.

Table 1: Statistics of Datasets

Dataset	Blogcatalog	PPI	Wikipedia	Flickr
|V|	10,312	3,890	4,777	80,513
|E|	333,983	76,584	184,812	5,899,882
#Label	39	50	40	195
Density	3.14e-3	5.06e-3	8.10e-3	9.10e-4

5.1.2 Compared Algorithms. We compare the proposed algorithms with LINE(2nd) [35] (number of samples 10B, number of negative samples 5, initial learning rate 0.025), DeepWalk [27](window size 10, walk length 40, and the number of walks 80), and NetMF [28] (b=1,T=1 for the Wikipedia dataset and T=10 for others). The dimension of representation is set 128.

We also compare INH-MF with Spectral Hashing (SH) [38], Hashing with Graphs with one layer (AGH-1) and two layers (AGH-2) (anchor number 1000) [23], Inductive Hashing on Manifold with LE manifold learning (IMH-LE) [31], Iterative Quantization (ITQ) [11], Discrete Collaborative Filtering (DCF), which take adjacency matrices as input [39]. Their code length is also set 128. Each node, represented by a row in the adjacency matrix of networks, is considered data points for hashing. Note that SH and ITQ depend on PCA, which does not scale up and can't be applied for the Flickr dataset

(80,513×80,153 dense matrix) due to zero-centering. Therefore, we ignore zero-centering on the Flickr dataset and directly apply SVD to get real-valued low-dimensional representation. INH-MF-i, an initialization algorithm of INH-MF, is also included into baselines.

5.1.3 Settings and Metrics. For node classification, we exactly follow the same experimental procedure and treatment in DeepWalk. In particular, we randomly sample a portion of labeled nodes for training and use the rest for testing. For BlogCatalog, PPI and Wiki datasets, the training ratio ranges from 10% to 90% with a step 20%. For Flickr, the training ratio ranges from 1% to 9% with a step 2%. Using the one-vs-the rest logistic regression, we repeat the prediction procedure 10 times and evaluate the performance of all methods in terms of Micro-F1 and Macro-F1.

For node recommendation, we randomly sample 90% neighbors of each node for training and use the rest for testing. We also repeat the recommendation procedure 10 times and evaluate the performance of all methods in terms of NDCG@50, AUC (Area under ROC Curve) and MPR (Mean Percentile Ranking) [16]. MPR is a common evaluation metric for implicit feedback recommendation. In contrast to NDCG and AUC, the smaller MPR is, the better node recommendation is. For easy differentiation, we use ↑ to annotate NDCG and AUC, and ↓ to annotate MPR, as shown in Table 4.

5.2 Quantitative Results

5.2.1 Node Classification. The results of node classification are shown in Table 2 and Table 3. We have the following observations.

First, INH-MF significantly outperforms all learning to hash algorithms on the BlogCatalog, PPI and Flickr dataset in terms of both Micro-F1 and Macro-F1, indicating the effective approach of INH-MF to handle sparsity and to preserve high-order proximity. The relative improvements in terms of Micro-F1 on the BlogCatalog, PPI and Flickr dataset are at least 43.3%, 22.8% and 6.6%, respectively and the relative improvements in terms of Macro-F1 are at least 80.7%, 23.4% and 25.6%, respectively. And it is worth noting that only second-order proximity is considered on the Flickr dataset due to out of memory issues when $T = 10$, so that its performance can be improved further. ITQ performs best among all learning to hash methods and is even comparable to INH-MF on the Wikipedia dataset, so the rotation of basis to balance variance of different dimensions is effective for deriving more compact binary codes. The main reason that ITQ and INH-MF are comparable on the Wikipedia dataset is because the second-order proximity ($T = 1$ in Eq (3)) is superior to higher-order, so that the advantage of preserving high-order proximity in INH-MF cannot be leveraged. Moreover, the Wikipedia dataset is the densest among all datasets according to Table 1.

Second, INH-MF outperforms LINE (2nd) significantly in the most cases, indicating the importance of preserving high-order proximity. And INH-MF is a little worse than or sometimes even comparable to DeepWalk and NetMF. Therefore, transforming network embedding from real-valued representation to binary will incur information loss, but not much. This benefits from the reasonable design of the proposed algorithm. Moreover, with the increase of training data, the margin between them becomes smaller.

Third, INH-MF is almost significantly better than INH-MF-i, indicating effectiveness of the proposed algorithm and optimization procedure, though the improvements on the Wikipedia dataset are not so large.

Finally, DCF does not perform as well as expected, is almost worst among them. This demonstrates that imposing binary constraints on representation of the "context" role of nodes further incurs information loss and that high order proximity preserving plays an important role in network embedding and hashing. These two points also distinguish homogeneous graphs hashing from bipartite graph hashing.

5.2.2 Node Recommendation. The results of node recommendation are reported in Table 4. Observations are listed as follows.

First, INH-MF not only outperforms all learning to hash algorithms significantly, but also LINE, DeepWalk and NetMF on both BlogCatalog and PPI datasets. This observation also holds on the Flickr dataset in terms of NDCG. The relative improvements of NDCG on the BlogCatalog, PPI and Flickr dataset are at least 73.4%, 53.1% and 23.4%, respectively. This is surprising but reasonable. Network embedding methods are trained with high-order proximity preserved but tested with 1-hop neighbors of each node, so they may overfit in the task of node recommendation. Converting them into binary representation leads to information loss, but it also alleviates the over-fitting problem. This also implies that network embedding being suitable for node classification may not fit for node recommendation.

Second, DCF is still worst of all in terms of NDCG. According to AUC, DCF is only slightly better than random guess. This confirms that homogeneous graph hashing is different from bipartite graph hashing, so that hashing methods for recommender systems cannot be directly applied. Moreover, the input of DCF in recommender system is a user-item rating matrix, whose rating distribution is different from edge weight distribution in information networks.

Third, ITQ performs well on the Flickr dataset, but this is a variant of ITQ. As aforementioned, ITQ cannot be directly applied due to zero-centering in PCA, so we discard zero-centering and perform SVD to obtain the representation. This is quite similar to INH-MF due to $T = 1$ in this case, except an extra transformation of adjacency matrix according to Eq (4). When high-order proximity is preserved, the performance of INH-MF can be further improved.

Fourth, AUC of many learning to hash algorithms and even LINE on some datasets is even lower than random guess. This, on one hand, may be because they do not capture sufficient network information; on the other hand, their goal for embedding does not align with the goal of node recommendation.

Finally, the results on the Wikipedia dataset show AGH-2 performs best rather than INH-MF. This mainly depends on the characteristics of the Wikipedia dataset, as analyzed in node classification.

In addition to superior performance of INH-MF in terms of NDCG, AUC and MPR, hashing also dramatically accelerates node recommendation. The speedup of 200-nn search via hamming distance compared to dot-product is shown in Fig. 1(a). The speedup achieves up to 10 on the Flickr dataset, and round 4 on the other three datasets. Hence, networks with a larger size will gain more speedup. If multi-index hashing [25] is exploited, node recommendation may be accelerated by two orders of magnitude, since it has sub-linear run-time behavior for uniformly distributed codes.

Table 2: Micro/Macro-F1 (%) of node classification on the BlogCatalog and PPI dataset. ** 0.01 & * 0.05 level, paired t-test.

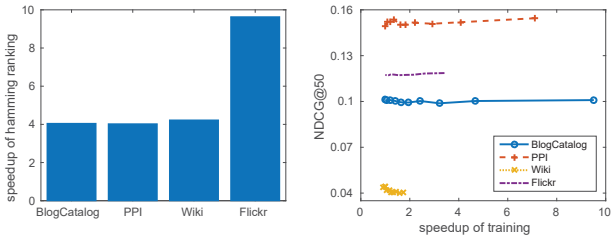
Metric	ALG	BlogCatalog					PPI				
		10%	30%	50%	70%	90%	10%	30%	50%	70%	90%
Micro-F1	LINE	25.35	32.05	35.16	36.61	37.35	11.70	14.20	16.00	17.82	19.59
	DeepWalk	35.85	39.91	41.62	42.45	42.90	16.06	19.37	21.26	22.63	24.36
	NetMF	**38.33**	**41.43**	**42.67**	**43.34**	**43.15**	**18.05**	**21.80**	**23.10**	**24.40**	**25.96**
	DCF	14.23	15.54	16.07	16.58	16.94	7.27	7.49	7.51	8.10	7.86
	AGH-1	15.20	16.15	16.93	17.51	17.93	9.79	11.12	11.76	12.18	12.32
	AGH-2	15.87	16.42	16.91	17.23	17.21	9.28	10.37	10.69	10.85	11.17
	IMH-LE	15.28	15.76	16.10	16.45	16.83	8.94	9.56	9.85	9.88	10.40
	SH	17.20	20.20	20.95	21.37	21.34	9.04	10.50	11.58	12.42	12.82
	ITQ	19.23	24.03	25.87	26.98	27.71	10.97	13.34	14.99	16.23	17.34
	INH-MF-i	25.63	33.91	36.53	37.66	37.81	11.67	14.21	16.44	18.09	19.78
	INH-MF	**29.29****	**35.58****	**38.06****	**39.29****	**39.72****	**13.61****	**16.38****	**18.59****	**20.16****	**22.02****
Macro-F1	LINE	14.38	19.11	21.36	22.25	22.62	9.50	12.15	13.82	15.35	15.92
	DeepWalk	21.16	25.59	27.58	28.47	**28.66**	12.89	16.71	18.25	19.48	20.36
	NetMF	**23.12**	**26.70**	**28.31**	**28.91**	28.44	**13.97**	**18.29**	**19.94**	**21.01**	**21.45**
	DCF	5.04	4.85	4.67	4.68	4.59	4.73	4.84	4.84	5.38	4.90
	AGH-1	5.35	5.29	5.26	5.35	5.27	7.50	8.78	9.45	9.75	9.95
	AGH-2	3.71	3.60	3.65	3.71	3.57	5.80	7.37	7.89	8.10	8.40
	IMH-LE	4.05	3.96	3.93	3.86	4.09	5.56	6.87	7.57	7.46	7.46
	SH	7.48	7.88	7.74	7.83	7.34	7.12	8.55	9.30	9.92	9.62
	ITQ	10.29	12.17	12.61	12.91	13.10	9.17	11.69	13.03	14.08	14.36
	INH-MF-i	16.82	22.29	24.11	24.96	25.30	9.82	12.57	14.30	15.50	16.76
	INH-MF	**18.60****	**23.09****	**25.10****	**26.04****	**26.35****	**11.59****	**14.47****	**16.11****	**17.38****	**18.82****

Table 3: Micro/Macro-F1(%) of node classification on the Wikipedia and Flickr dataset. ** 0.01 & * 0.05 level, paired t-test.

Metric	ALG	Wikipedia					Flickr				
		10%	30%	50%	70%	90%	1%	3%	5%	7%	9%
Micro-F1	LINE	41.30	48.35	51.89	53.57	54.86	25.30	28.64	30.07	31.28	32.34
	DeepWalk	42.32	47.02	48.65	49.80	50.35	**32.06**	**35.89**	**37.46**	**38.29**	**38.84**
	NetMF	**50.10**	**55.81**	**57.26**	**58.47**	**59.13**	31.97	35.07	36.24	36.82	37.19
	DCF	27.74	35.99	39.33	40.82	42.11	15.71	16.20	16.46	16.59	16.69
	AGH-1	35.37	40.71	43.02	44.47	45.88	21.50	23.23	24.02	24.60	24.97
	AGH-2	**40.14****	43.66	44.70	45.22	46.30	19.91	21.18	21.73	22.15	22.35
	IMH-LE	37.50	42.05	43.89	44.64	45.55	19.12	20.05	20.39	20.75	20.90
	SH	**40.81****	41.82	41.83	42.29	42.46	18.85	20.82	21.60	22.08	22.38
	ITQ	39.92	44.74	46.84	47.91	49.45	24.00	26.29	28.09	29.37	30.30
	INH-MF-i	39.03	48.15	51.31	52.87	**54.24****	21.87	26.77	28.88	30.41	31.53
	INH-MF	39.50	**48.95***	**52.74****	**54.42****	**54.82****	**25.58****	**29.77****	**31.66****	**32.85****	**33.74****
Macro-F1	LINE	8.51	10.53	12.63	13.40	13.16	9.01	13.42	15.77	17.44	18.68
	DeepWalk	7.26	9.02	9.71	10.03	9.92	**13.36**	**19.45**	**22.21**	**23.94**	**25.07**
	NetMF	**9.04**	**12.44**	**13.67**	**14.04**	**14.91**	12.32	17.32	19.42	20.68	21.57
	DCF	5.19	6.17	6.68	6.62	6.53	1.10	1.45	1.58	1.65	1.75
	AGH-1	7.82	9.19	9.74	9.38	9.89	5.98	7.11	7.58	7.96	8.19
	AGH-2	7.95	8.97	9.77	9.05	9.63	4.68	5.77	6.24	6.50	6.70
	IMH-LE	7.94	9.36	10.16	9.87	9.79	3.45	4.25	4.59	4.73	4.88
	SH	5.54	6.61	7.31	8.06	7.03	4.74	6.36	7.21	7.61	7.92
	ITQ	**9.98****	**11.97****	**13.46****	13.91	**13.19****	10.38	12.65	13.91	14.77	15.34
	INH-MF-i	8.78	10.98	12.48	13.72	**13.40****	9.25	13.57	15.47	16.94	17.94
	INH-MF	9.18	**11.60****	**13.41****	**14.72***	**14.14****	**13.04****	**17.41****	**19.05****	**20.09****	**20.86****

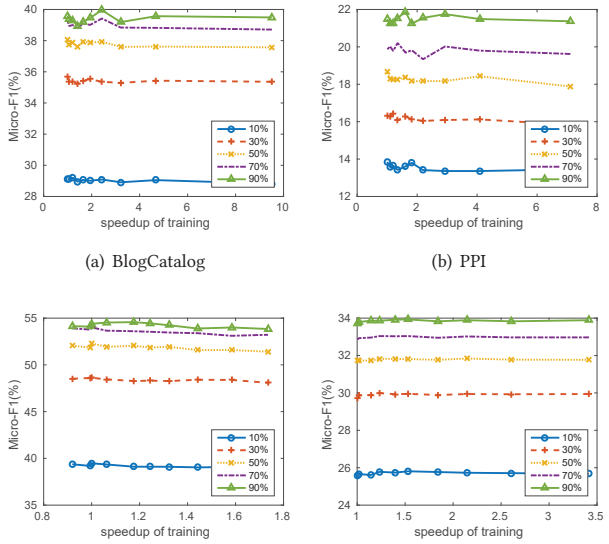
Table 4: NDCG@50, AUC and Mean Percentile Ranking (Section 5.1.3) of node recommendation. ** 0.01 level, paired t-test.

ALG	BlogCatalog			PPI			Wikipedia			Flickr		
	NDCG↑	AUC↑	MPR↓	NDCG↑	AUC↑	MPR↓	NDCG↑	AUC↑	MPR↓	NDCG↑	AUC↑	MPR↓
LINE	0.0417	0.3804	0.6196	0.0874	0.5652	0.4347	0.0339	0.3913	0.6086	**0.0623**	0.6254	0.3746
DeepWalk	**0.0426**	0.6329	0.3671	0.0690	0.6967	0.3033	0.0535	0.5551	0.4448	0.0597	0.8319	0.1682
NetMF	0.0388	**0.6802**	**0.3199**	**0.0981**	**0.7318**	**0.2683**	**0.1695**	**0.6732**	**0.3269**	0.0366	**0.8631**	**0.1370**
DCF	0.0006	0.5294	0.4705	0.0007	0.5365	0.4634	0.0141	0.5445	0.4555	0.0002	0.5677	0.4323
AGH-1	0.0339	0.2759	0.7240	0.0579	0.3373	0.6625	0.0178	0.3986	0.6013	0.0551	0.4285	0.5715
AGH-2	0.0277	0.1770	0.8228	0.0384	0.3022	0.6975	**0.1825****	0.5389	0.4610	0.0274	0.3672	0.6328
IMH-LE	0.0295	0.1785	0.8214	0.0481	0.3259	0.6739	0.0180	0.2650	0.7348	0.0392	0.4209	0.5792
SH	0.0117	0.1573	0.8424	0.0245	0.2749	0.7248	0.0266	0.2203	0.7794	0.0464	0.2983	0.7017
ITQ	0.0593	0.6203	0.3799	0.0968	0.7092	0.2909	0.0344	**0.6581****	**0.3419****	0.0947	**0.8626****	**0.1375****
INH-MF-i	0.0857	0.6560	0.3441	0.1273	0.6895	0.3106	0.0397	0.6163	0.3838	0.1019	0.7583	0.2417
INH-MF	**0.1021****	**0.7519****	**0.2483****	**0.1502****	**0.7596****	**0.2406****	0.0449	0.6135	0.3866	**0.1169****	0.8308	0.1693



(a) Speedup of 200-NN search (b) NDCG@50 v.s. training speedup

Figure 1: (a) speedup of 200-nearest neighbor search via Hamming distance compared to widely-used dot-product. (b) NDCG@50 of node recommendation v.s. speedup of training due to subspace learning;



(a) BlogCatalog (b) PPI

(c) Wiki (d) Flickr

Figure 2: Micro-F1 score(%) of node classification v.s. speedup of training due to subspace learning

5.3 Parameter Sensitivity

In this part, we study the effect of subspace learning, and sensitivity of code length, training size and uncorrelation coefficient.

5.3.1 Hamming Subspace Learning. We vary the ratio of selected bits ($ratio = l/d$) from 0.1 to 1 with a step 0.1, and show the performance of node classification and node recommendation. The running time of INH-MF is also recorded in order to compute the speedup of training due to subspace learning. The results are shown in Fig. 1(b) and Fig. 2. We observe that the performance of both node classification and node recommendation almost do not drop with fewer bits selected. Moreover, the speedup of training can achieve up to 9-10, particularly when high-order proximity is taken into account, such as on the BlogCatalog dataset. The speedup on the Flickr and Wikipedia dataset is a little smaller, since only second-order proximity is considered and the thin SVD of $n \times d$ matrices dominates computation time. Therefore, subspace learning is really useful for improving the efficiency of optimization in information network hashing with a little sacrifice of performance.

5.3.2 Code Length and Training Data Size. Due to space limitation, only the results on the Flickr dataset are reported. The results with varying code length are shown in Fig. 3. In the task of node classification, the performance of network embedding improves first and then become stable when code length increases from 16 to 512. In contrast, when code length is too large, the performance of network hashing drops. In the task of node recommendation, hashing is always better than embedding, and their performances show continuous improvement with the increase of dimension/code length. The results with varying data size (the number of edges) are shown in Fig. 4. Both node classification and node recommendation via INH-MF improve with the increase of training data size. NetMF shows similar trends, but reports runtime errors when network is sparse due to failure of SVD.

5.3.3 Uncorrelation. The bit balance condition is naturally incorporated into INH-MF without any parameter tuning, but the bit

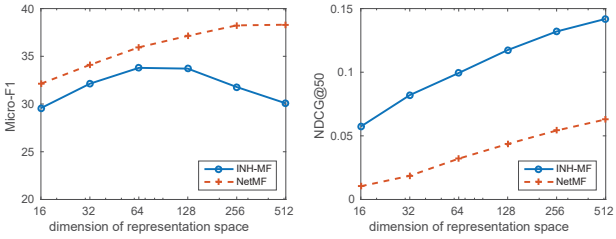


Figure 3: Sensitivity w.r.t code length

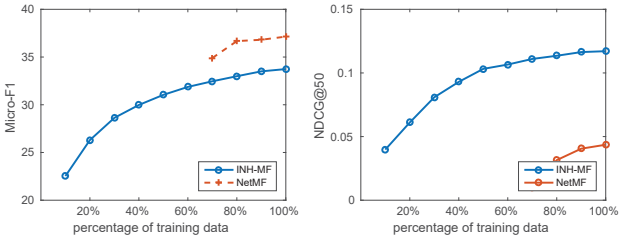


Figure 4: Sensitivity w.r.t training data size.

uncorrelation is achieved by a penalty term with a coefficient γ. The results of node classification and node recommendation with the change of γ from 0.01 to 5.12 are shown in Fig. 5. We can see that uncorrelation does not play an important role, except node recommendation on the Flickr dataset. This may lie in the effect of orthogonal constraints of $\boldsymbol{Q}$, so that different bits are not much correlated [20].

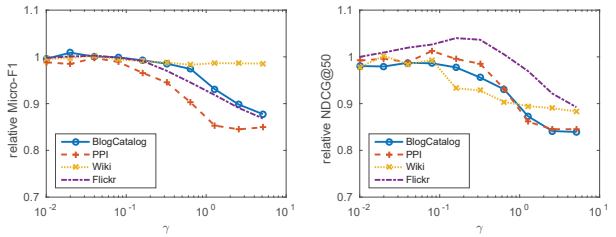


Figure 5: Sensitivity w.r.t coefficient of uncorrelation.

5.4 Scalability

We finally investigate the scalability of INH-MF with the increase of code length and training data size, and show the results in Fig. 6, where INH-MF(20%) selects 20% of bits for subspace learning. INH-MF(20%) is comparably efficient to ITQ, and is much more efficient than AGH-1. Though not as efficient as INH-MF(20%), INH-MF(100%) also scales linearly with training data size, and almost quadratically with code length, aligning with complexity analysis.

6 CONCLUSIONS

In this paper we studied learning to hash information network problems, and proposed a MF-based information network hashing

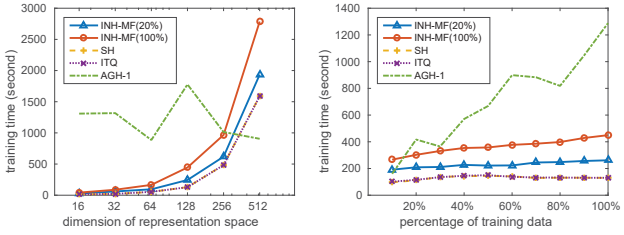


Figure 6: Training time w.r.t code length (left) and training data size (right).

algorithm which can preserve high-order proximity. We developed an efficient alternating optimization algorithm for learning parameters and suggested Hamming subspace learning for scaling up. We extensively evaluated the proposed algorithms (INH-MF) on four real-world network datasets with respect to node classification and node recommendation. In both tasks, INH-MF significantly outperformed competing learning to hash algorithms on three datasets. INH-MF can surprisingly outperform all network embedding algorithms in most cases with respect to node recommendation. And in the task of node classification, INH-MF was almost better than to LINE, but generally a little worse than or sometimes comparable to NetMF and DeepWalk. Moreover, INH-MF was shown dramatical speedup of node recommendation and of parameter learning due to subspace learning.

ACKNOWLEDGMENTS

Defu Lian is supported by the National Natural Science Foundation of China (Grant No. 61502077 and 61631005) and the Fundamental Research Funds for the Central Universities (Grant No. ZYGX2016J087), Kai Zheng is supported by the National Natural Science Foundation of China (Grant No. 61532018 and 61502324). Ivor Tsang is supported by the Australian Research Council (Grant No. FT130100746, DP180100106 and LP150100671). Vincent Zheng is supported by National Research Foundation, Prime Minister's Office, Singapore under its Campus for Research Excellence and Technological Enterprise (CREATE) programme, and Alibaba Innovative Research program.

REFERENCES

[1] Amr Ahmed, Nino Shervashidze, Shravan Narayanamurthy, Vanja Josifovski, and Alexander J Smola. 2013. Distributed large-scale natural graph factorization. In *Proceedings of WWW'13*. ACM, 37–48.
[2] Lars Backstrom and Jure Leskovec. 2011. Supervised random walks: predicting and recommending links in social networks. In *Proceedings of WSDM'11*. ACM, 635–644.
[3] Avrim Blum, John Hopcroft, and Ravindran Kannan. 2016. Foundations of data science. *Vorabversion eines Lehrbuchs* (2016).
[4] Hongyun Cai, Vincent W Zheng, and Kevin Chen-Chuan Chang. 2017. A Comprehensive Survey of Graph Embedding: Problems, Techniques and Applications. *arXiv preprint arXiv:1709.07604* (2017).
[5] Jie Cao, Zhiang Wu, Youquan Wang, and Yi Zhuang. 2013. Hybrid collaborative filtering algorithm for bidirectional web service recommendation. *Knowledge and information systems* 36, 3 (2013), 607–627.
[6] Jie Cao, Zhiang Wu, Junjie Wu, and Hui Xiong. 2013. SAIL: Summation-based incremental learning for information-theoretic text clustering. *IEEE Transactions on Cybernetics* 43, 2 (2013), 570–584.
[7] Shaosheng Cao, Wei Lu, and Qiongkai Xu. 2015. Grarep: Learning graph representations with global structural information. In *Proceedings of CIKM'15*. ACM, 891–900.

[8] Peng Cui, Xiao Wang, Jian Pei, and Wenwu Zhu. 2017. A Survey on Network Embedding. *arXiv preprint arXiv:1711.08752* (2017).

[9] Hanjun Dai, Bo Dai, and Le Song. 2016. Discriminative embeddings of latent variable models for structured data. In *Proceedings of ICML'16*. 2702–2711.

[10] Lars Eldén and Haesun Park. 1999. A Procrustes problem on the Stiefel manifold. *Numer. Math.* 82, 4 (1999), 599–619.

[11] Yunchao Gong, Svetlana Lazebnik, Albert Gordo, and Florent Perronnin. 2013. Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval. *IEEE Trans. Pattern Anal. Mach. Intell.* 35, 12 (2013).

[12] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of KDD'16*. ACM, 855–864.

[13] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *Proceedings of NIPS'17*. 1025–1035.

[14] Johan Håstad. 2001. Some optimal inapproximability results. *Journal of the ACM (JACM)* 48, 4 (2001), 798–859.

[15] Roger A Horn and Charles R Johnson. 1990. *Matrix analysis*. Cambridge press.

[16] Y. Hu, Y. Koren, and C. Volinsky. 2008. Collaborative filtering for implicit feedback datasets. In *Proceedings of ICDM'08*. IEEE, 263–272.

[17] Qing-Yuan Jiang and Wu-Jun Li. 2015. Scalable Graph Hashing with Feature Transformation.. In *Proceedings of IJCAI'15*. 2248–2254.

[18] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[19] Weihao Kong and Wu-Jun Li. 2012. Isotropic hashing. In *Proceedings of NIPS'12*. 1646–1654.

[20] Xuelong Li, Di Hu, and Feiping Nie. 2017. Large Graph Hashing with Spectral Rotation.. In *Proceedings of AAAI'17*. 2203–2209.

[21] Defu Lian, Rui Liu, Yong Ge, Kai Zheng, Xing Xie, and Longbing Cao. 2017. Discrete Content-aware Matrix Factorization. In *Proceedings of KDD'17*. 325–334.

[22] Wei Liu, Cun Mu, Sanjiv Kumar, and Shih-Fu Chang. 2014. Discrete graph hashing. In *Proceedings of NIPS'14*. 3419–3427.

[23] Wei Liu, Jun Wang, Sanjiv Kumar, and Shih-Fu Chang. 2011. Hashing with graphs. In *Proceedings of ICML'11*. 1–8.

[24] Chao Ma, Ivor W Tsang, Furong Peng, and Chuancai Liu. 2017. Partial hash update via hamming subspace learning. *IEEE Transactions on Image Processing* 26, 4 (2017), 1939–1951.

[25] Mohammad Norouzi, Ali Punjani, and David J Fleet. 2012. Fast search in hamming space with multi-index hashing. In *Proceedings of CVPR'12*. IEEE, 3108–3115.

[26] Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. 2016. Asymmetric transitivity preserving graph embedding. In *Proceedings of KDD'16*. ACM, 1105–1114.

[27] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *Proceedings of KDD'14*. ACM, 701–710.

[28] Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Kuansan Wang, and Jie Tang. 2018. Network Embedding as Matrix Factorization: UnifyingDeepWalk, LINE, PTE, and node2vec. In *Proceedings of WSDM'18*. ACM.

[29] Ruslan Salakhutdinov and Geoffrey Hinton. 2009. Semantic hashing. *International Journal of Approximate Reasoning* 50, 7 (2009), 969–978.

[30] Fumin Shen, Chunhua Shen, Wei Liu, and Heng Tao Shen. 2015. Supervised discrete hashing. In *Proceedings of CVPR'15*. 37–45.

[31] Fumin Shen, Chunhua Shen, Qinfeng Shi, Anton Van Den Hengel, and Zhenmin Tang. 2013. Inductive hashing on manifolds. In *Proceedings of CVPR'13*. IEEE, 1562–1569.

[32] Xiaoshuang Shi, Fuyong Xing, Kaidi Xu, Manish Sapkota, and Lin Yang. 2017. Asymmetric Discrete Graph Hashing.. In *Proceedings of AAAI'17*. 2541–2547.

[33] N. Srebro and T. Jaakkola. 2003. Weighted low-rank approximations. In *Proceedings of ICML'03*. 720–727.

[34] Mingkui Tan, Ivor W. Tsang, and Li Wang. 2014. Towards Ultrahigh Dimensional Feature Selection for Big Data. *Journal of Machine Learning Research* 15 (2014).

[35] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. Line: Large-scale information network embedding. In *Proceedings of WWW'15*. 1067–1077.

[36] Lei Tang and Huan Liu. 2009. Relational learning via latent social dimensions. In *Proceedings of KDD'09*. ACM, 817–826.

[37] Jingdong Wang, Ting Zhang, Nicu Sebe, Heng Tao Shen, et al. 2017. A survey on learning to hash. *IEEE Trans. Pattern Anal. Mach. Intell.* (2017).

[38] Yair Weiss, Antonio Torralba, and Rob Fergus. 2009. Spectral hashing. In *Proceedings of NIPS'09*. 1753–1760.

[39] Hanwang Zhang, Fumin Shen, Wei Liu, Xiangnan He, Huanbo Luan, and Tat-Seng Chua. 2016. Discrete collaborative filtering. In *Proceedings of SIGIR'16*. ACM, 325–334.

[40] Yan Zhang, Defu Lian, and Guowu Yang. 2017. Discrete Personalized Ranking for Fast Collaborative Filtering from Implicit Feedback. In *Proceedings of AAAI'17*. 1669–1675.