Graph DNA: Deep Neighborhood Aware Graph Encoding for Collaborative Filtering

Liwei Wu
University of California, Davis

Hsiang-Fu Yu
Amazon

Nikhil Rao
Amazon

James Sharpnack
University of California, Davis

Cho-Jui Hsieh
University of California, Los Angeles

Abstract

In this paper, we consider recommender systems with side information in the form of graphs. Existing collaborative filtering algorithms mainly utilize only immediate neighborhood information and do not efficiently take advantage of deeper neighborhoods beyond 1-2 hops. The main issue with exploiting deeper graph information is the rapidly growing time and space complexity when incorporating information from these neighborhoods. In this paper, we propose using Graph DNA, a novel Deep Neighborhood Aware graph encoding algorithm, for exploiting multi-hop neighborhood information. DNA encoding computes approximate deep neighborhood information in linear time using Bloom filters, and results in a per-node encoding whose dimension is logarithmic in the number of nodes in the graph. It can be used in conjunction with both feature-based and graph-regularization-based collaborative filtering algorithms. Graph DNA has the advantages of being memory and time efficient and providing additional regularization when compared to directly using higher order graph information. We provide theoretical performance bounds for graph DNA encoding, and experimentally show that graph DNA can be used with 4 popular collaborative filtering algorithms to consistently boost their performances with little computational and memory overhead.

Proceedings of the 23[rd]International Conference on Artificial Intelligence and Statistics (AISTATS) 2020, Palermo, Italy. PMLR: Volume 108. Copyright 2020 by the author(s).

1 Introduction

Recommendation systems are increasingly prevalent due to content delivery platforms, e-commerce websites, and mobile apps Shani et al. ((2008)). Classical collaborative filtering algorithms use matrix factorization to identify latent features that describe the user preferences and item meta-topics from partially observed ratings Koren et al. ((2009)). In addition to rating information, many real-world recommendation datasets also have a wealth of side information in the form of graphs, and incorporating this information often leads to performance gains. For example, Rao et al. ((2015)), Zhou et al. ((2012)) propose to add a graph regularization to the matrix factorization formulation to exploit additional graph structure; and Liang et al. ((2016)) conduct a co-factorization of the graph and rating matrix. However, each of these only utilizes the immediate neighborhood information of each node in the side information graph. More recently, Berg et al. ((2017)) incorporated graph information when learning features with a Graph Convolution Network (GCN) based recommendation algorithm. GCNs Kipf and Welling ((2016)) constitute flexible methods for incorporating graph structure beyond first-order neighborhoods, but their training complexity typically scales rapidly with the depth, even with sub-sampling techniques Chen et al. ((2018)). Intuitively, exploiting higher-order neighborhood information could benefit the generalization performance, especially when the graph is sparse, which is usually the case in practice. The main caveat of exploiting higher-order graph information is the high computational and memory cost when computing higher-order neighbors since the number of t-hop neighbors typically grows exponentially with t.

In this paper, we aim to utilize higher order graph information without introducing much computational and memory overhead. To achieve this goal, we pro-

pose a Graph Deep Neighborhood Aware (Graph DNA) encoding, which approximately captures the higher-order neighborhood information of each node via Bloom filters Bloom ((1970)). Bloom filters encode neighborhood sets as c dimensional 0/1 vectors, where $c = O(\log n)$ for a graph with n nodes, which approximately preserves membership information. This encoding can then be combined with both graph regularized or feature based collaborative filtering algorithms, with little computational and memory overhead. In addition to computational speedups, we find that Graph DNA achieves better performance over competitors, which we hypothesize is due to the unique nature of Graph DNA and its connection to the shortest path length distance. We make this connection precise with theoretical bounds in Section 2.2.

We show that our Graph DNA encoding can be used with several collaborative filtering algorithms: graph-regularized matrix factorization with explicit and implicit feedback Rao et al. ((2015)), Zhou et al. ((2012)), co-factoring Liang et al. ((2016)), and GCN-based recommendation systems Monti et al. ((2017)). In all the cases we tested in this paper, Graph DNA can consistently boost the performance of existing method while having small time and memory overhead.

Related Work : Matrix factorization has been used extensively in recommendation systems with both explicit Koren et al. ((2009)) and implicit Hu et al. ((2008)) feedback. Such methods compute low dimensional user and item representations; their inner product approximates the observed (or to be predicted) entry in the target matrix. To incorporate graph side information in these systems, Rao et al. ((2015)), Zhou et al. ((2012)) used a graph Laplacian based regularization framework that forces a pair of node representations to be similar if they are connected via an edge in the graph. In Yu et al. ((2017)), this was extended to the implicit feedback setting. Liang et al. ((2016)) proposed a method that incorporates first-order information of the rating bipartite graph into the model by considering item co-occurrences. More recently, GC-MC Berg et al. ((2017)) used a GCN approach performing convolutions on the main bipartite graph by treating the first-order side graph information as features, and Monti et al. ((2017)) proposed combining GCNs and RNNs for the same task.

Methods that use higher order graph information are typically based on taking random walks on the graphs Gori et al. ((2007)). Jamali and Ester ((2009)) extended this method to include graph side information in the model. Finally, the PageRank Page et al. ((1999)) algorithm can be seen as computing the steady state distribution of a Markov network, and similar methods for recommender systems was proposed in Abbassi and Mirrokni ((2007)), Xie et al. ((2015)).

For a complete list of related works of representation learning on graphs, we refer the interested user to Hamilton et al. ((2017b)). For the collaborative filtering setting, Berg et al. ((2017)), Monti et al. ((2017)) use Graph Convolutional Neural Networks Defferrard et al. ((2016)), but with some modifications. Standard GCN methods without substantial modifications cannot be directly applied to collaborative filtering rating datasets, including well-known approaches like GCN Kipf and Welling ((2016)) and GraphSage Hamilton et al. ((2017a)), because they are intended to solve semi-supervised classification problem over graphs with nodes' features. PinSage Ying et al. ((2018)) is the GraphSage extension to non-personalized graph-based recommendation algorithm but not meant for collaborative filtering problems. GC-MC Berg et al. ((2017)) extend GCN to collaborative filtering, albeit less scalable than Ying et al. ((2018)). Our Graph DNA scheme can be used to obtain graph features in these extensions. In contrast to the above-mentioned methods involving GCNs, we do not use any loss function to train our graph encoder. This property makes our graph DNA suitable for both transductive as well as inductive problems.

Bloom filters have been used in Machine Learning for multi-label classification Cisse et al. ((2013)), and for hashing deep neural network models representations Courbariaux et al. ((2015)), Han et al. ((2015)), Shi et al. ((2009)). However, to the best of our knowledge, they have not been used to encode graphs, nor has this encoding been applied to recommender systems. So it would be interesting to extend our work to other recommender systems settings, such as Wu et al. ((2019a)) and Wu et al. ((2019b)).

2 Methodology

We consider the problem of recommender system with a partially observed rating matrix R and a Graph that encodes side information G. In this section, we will introduce the Graph DNA algorithm for encoding deep neighborhood information in G. In the next section, we will show how this encoded information can be applied to various graph based recommender systems.

2.1 Bloom Filter

The Bloom filter Bloom ((1970)) is a probabilistic data structure designed to represent a set of elements. Thanks to its space-efficiency and simplicity, Bloom filters are applied in many real-world applications such as database systems Borthakur et al. ((2011)), Chang et al. ((2008)). A Bloom filter $\mathcal{B}$ consists of k indepen-

dent hash functions $h_t(x) \to \{1, \ldots, c\}$. The Bloom filter $\mathcal{B}$ of size c can be represented as a length c bit-array $\boldsymbol{b}$. More details about Bloom filters can be found in Broder and Mitzenmacher ((2004)). Here we highlight a few desirable properties of Bloom filters essential to our graph DNA encoding:

1. Space efficiency: classic Bloom filters use $1.44 \log_2(1/\epsilon)$ of space per inserted key, where ϵ is the false positive rate associated with this Bloom filter.
2. Support for the union operation of two Bloom filters: the Bloom filter for the union of two sets can be obtained by performing bitwise 'OR' operations on the underlying bit-arrays of the two Bloom filters.
3. Size of the Bloom filter can be approximated by the number of nonzeros in the underlying bit array: in particular, given a Bloom filter representation $\mathcal{B}(A)$ of a set A: the number of elements of A can be estimated as $|A| \approx -\frac{c}{k} \log\left(1 - \frac{\text{nnz}(\boldsymbol{b})}{c}\right)$, where $\text{nnz}(\boldsymbol{b})$ is the number of non-zero elements in array $\boldsymbol{b}$. As a result, the number of common nonzero bits of $\mathcal{B}(A_1)$ and $\mathcal{B}(A_2)$ can be used as a proxy for $|A_1 \cap A_2|$.

Algorithm 1 Graph DNA Encoding with Bloom Filters

Input: G: a graph of n nodes, c: the length of codes, k: the number of hash functions, d: the number of iterations, θ: tuning parameter to control the number of elements hashed.

Output: $B \in \{0,1\}^{n \times c}$: a boolean matrix to denote the bipartite relationship between n nodes and c bits.

- $\mathcal{H} \leftarrow \{\mathtt{h_t}(\cdot) : t = 1, \ldots, k\}$ ▷ Pick k hash functions
- **for** $i = 1, \ldots, n$: ▷ *GraphBloom* Initialization
  - $\mathcal{B}^0[\mathtt{i}] \leftarrow \mathtt{BloomFilter}(c, \mathcal{H})$
  - $\mathcal{B}^0[\mathtt{i}].\mathtt{add}(i)$
- **for** $s = 1, \ldots, d$: ▷ d times neighborhood propagations
  - **for** $i = 1, \ldots, n$:
    - **for** $j \in \mathcal{N}_1(i)$: ▷ degree-1 neighbors
      - **if** $|\mathcal{B}^s[\mathtt{i}]| > \theta$: break;
      - $\mathcal{B}^s[\mathtt{i}].\mathtt{union}(\mathcal{B}^{s-1}[\mathtt{j}])$
- $B_{ij} \leftarrow \mathcal{B}^d[\mathtt{i}].\mathtt{b}[\mathtt{j}]$ $\forall (i,j) \in [n] \times [c]$

2.2 Graph DNA Encoding Via Bloom Filters

Now we introduce our Graph DNA encoding. The main idea is to encode the deep (multi-hop) neighborhood aware embedding for each node in the graph approximately using the Bloom filter, which helps avoid performing computationally expensive graph adjacency matrix multiplications. In Graph DNA, we have Bloom filters $\mathcal{B}[\mathtt{i}], i = 1, ..., n$ for the n graph nodes. All the Bloom filters $\mathcal{B}[\mathtt{i}]$ share the same k hash functions. The role of $\mathcal{B}[\mathtt{i}]$ is to store the deep neighborhood information of the i-th node. Taking advantage of the union operations of Bloom filters, one node's neighborhood information can be propagated to its neighbors in an iterative manner using gossip algorithms Shah et al. ((2009)). Initially, each $\mathcal{B}[\mathtt{i}]$ contains only the node itself. At the s-th iteration, $\mathcal{B}[\mathtt{i}]$ is updated by taking union with node i's immediate neighbors' Bloom filters $\mathcal{B}[\mathtt{j}]$. By induction, we see that after the d iterations, $\mathcal{B}[\mathtt{i}]$ represents $\mathcal{N}_d(i) := \{j : \text{distance}_G(i,j) \le d\}$, where $\text{distance}_G(i,j)$ is the shortest path distance between nodes i and j in G. As the last step, we stack array representations of all Bloom filters and form a sparse matrix $B \in \{0,1\}^{n \times c}$, where the i-th row of B is the bit representation of $\mathcal{B}[\mathtt{i}]$. As a practical measure, to prevent over-saturation of Bloom filters for popular nodes in the graph, we add a hyper-parameter θ to control the max saturation level allowed for Bloom filters. This would also prevent hub nodes dominating in graph DNA encoding. The pseudo-code for the proposed encoding algorithm is given in Algorithm 1. We use graph DNA-d to denote our obtained graph encoding after applying Algorithm 1 with s looping from 1 to d. We also give a simple example to illustrate how the graph DNA is encoded into Bloom filter representations in Figure 1. Our usage of Bloom filters is very different from previous works in Pozo et al. ((2016)), Serrà and Karatzoglou ((2017)), Shinde and Savant ((2016)), which use Bloom filter for standard hashing and is unrelated to graph encoding.

2.3 Theoretical Guarantees

It is intuitive that the number of 1-bits in common between two Bloom filters should be closely related to the size of the intersection of their neighborhoods. However, there may also be false positives in the bit-representations. We control the size of such false positives and the number of common bits in Theorem 1, which only applies to Bloom filters without the max saturation threshold θ.

Theorem 1. *Suppose that the Bloom filters have c bits and the k hash functions are independent for all nodes. Consider two nodes $i, j = 1, \ldots, n$, their d-hop neighborhoods $\mathcal{N}_d(i), \mathcal{N}_d(j)$, and their d-depth Bloom filters $\mathcal{B}[\mathtt{i}], \mathcal{B}[\mathtt{j}]$, respectively. Let $Q_{i,j}$ be the number of common 1-bits in the Bloom filters of i, j (i.e. $\langle \mathcal{B}[\mathtt{i}], \mathcal{B}[\mathtt{j}] \rangle$). There exists universal constants C_0, C_1, such that for*

any $\gamma > 0$*, with probability* $1-\gamma$*,*

$$Q_{i,j} \leq \left(1 + \frac{1}{C_0}\ln\frac{C_1}{\gamma}\right) \cdot \left(k^2\frac{|\mathcal{N}_d(i)\triangle\mathcal{N}_d(j)|^2}{4c} + \frac{ck|\mathcal{N}_d(i)\cap\mathcal{N}_d(j)|}{c-1}\right), \quad (1)$$

where $\mathcal{N}_d(i)\triangle\mathcal{N}_d(j)$ *denotes the symmetric difference. Furthermore, for any* $\delta \in (0,1)$ *there exists a constant* $\alpha > 0$ *such that if* $c\alpha > k|\mathcal{N}_d(i)\cap\mathcal{N}_d(j)|$ *then*

$$\mathbb{P}\{Q_{i,j} > (1-\delta)k|\mathcal{N}_d(i)\cap\mathcal{N}_d(j)|\} \geq 1 - e^{-\frac{1}{3}(1-\delta)\delta^2 k|\mathcal{N}_d(i)\cap\mathcal{N}_d(j)|}. \quad (2)$$

This theorem is a corollary of the more precise Theorem 2, which we state below. Proofs for both Theorems 1 and 2 are provided in the Appendix. In order to establish results of Theorem 2, we provide Lemma 1 in the Appendix, which demonstrates that the bits of Bloom filters are negatively associated (basic properties of negative associativity can be found in Dubhashi and Ranjan ((1998)), Joag-Dev et al. ((1983))), and this property is preserved under bitwise 'or' and 'and' operations on independent Bloom filters. As a result, $Q_{i,j}$ enjoys Chernov-Hoeffding bounds, and the result of Theorem 1 follows by analyzing its expectation.

Remark 1. *When the neighborhoods have no intersection,* $|\mathcal{N}_d(i)\cap\mathcal{N}_d(j)| = 0$ *then we have that* $Q_{i,j} = O_P(k^2|\mathcal{N}_d(i)\cup\mathcal{N}_d(j)|^2/c)$ *which is approaching* 0 *when* $k|\mathcal{N}_d(i)\cup\mathcal{N}_d(j)| = o(\sqrt{c})$ *(the number of bits in the Bloom filters are taken to be large enough) by* (1)*.*

Remark 2. *Generally,* (2) *states that when the number of hashed functions for the intersection is large,* $k|\mathcal{N}_d(i)\cap\mathcal{N}_d(j)| \to \infty$*, but dominated by the number of bits,* $k|\mathcal{N}_d(i)\cup\mathcal{N}_d(j)| = o(c)$*, then we have that* $\lim(Q_{i,j}/(k|\mathcal{N}_d(i)\cap\mathcal{N}_d(j)|)) \geq 1$ *almost surely. For fixed neighborhood sizes, we can take* $c \propto \log n$ *and* $k \propto \log\log n$*, and obtain that* $Q_{i,j}/k = O_P(|\mathcal{N}_d(i)\cap\mathcal{N}_d(j)|)$ *by* (1) *and* $Q_{i,j}/k = \Omega_P(|\mathcal{N}_d(i)\cap\mathcal{N}_d(j)|)$ *by* (2)*.*

We now provide the result that will help is prove Theorem 1

Theorem 2. *Let* B_x, B_y *be the length* c *Bloom filter bitarrays for* $N(x), N(y)$ *with independent hash functions for all elements of* $N(x)\cup N(y)$ *and let* $|N(x)\triangle N(y)|$ *be their symmetric difference. Let* Q *be the number of common 1-bits in* B_x, B_y*. Then,*

$$\mathbb{P}\{Q \geq (1+\delta)\mathbb{E}Q\} \leq \left(\frac{e^{\delta}}{(1+\delta)^{(1+\delta)}}\right)^{\mathbb{E}Q},$$

$$\mathbb{P}\{Q \leq (1-\delta)\mathbb{E}Q\} \leq \left(\frac{e^{-\delta}}{(1-\delta)^{(1-\delta)}}\right)^{\mathbb{E}Q},$$

and $\Gamma_0 \leq \mathbb{E}Q \leq \Gamma_1$ *where*

$$\Gamma_0 = c\left(1 - \exp\left\{-k\frac{|N(x)\cap N(y)|}{c}\right\}\right),$$

$$\Gamma_1 = c\left(1 - \exp\left\{-k^2\frac{|N(x)\triangle N(y)|^2}{4c^2} - \frac{k|N(x)\cap N(y)|}{c-1}\right\}\right).$$

Graph DNA encodes deep neighborhood information such that for any two nodes whose shortest path length distance is at most $2d$, we only need to run Algorithm 1 for d iterations. For example, in Figure 2, nodes x and y are 6 hops away on the shortest path, but they will start to share their bits' representations after 3 iterations because the node z's information can be propagated to node x and y after exactly 3 iterations. Theorem 1 and the remarks that follow it demonstrate that by increasing the number of hash functions and the number of bits in the Bloom filter, the number of common 1-bits in these Bloom filters becomes an accurate surrogate for $|\mathcal{N}_d(x)\cap\mathcal{N}_d(y)|$.

The $n \times c$ Bloom filter matrix B can also be viewed as the adjacency matrix of a bipartite graph between the n nodes in the original graph and c meta nodes of Bloom filters. In this way, nodes x and y have a bit in common in their Bloom filter representations if they are both connected to at least one meta node in B. This property saves memory and time required for graph encoding, allowing us to use B instead of the adjacency matrix G in graph Laplacian regularization methods Rao et al. ((2015)), and to use B as side features in graph convolutional network based geometric matrix factorization algorithm Berg et al. ((2017)), Monti et al. ((2017)) with little computational and memory overhead. We elaborate on this in the following section.

3 Collaborative Filtering with Graph DNA

Suppose we are given the sparse rating matrix $R \in \mathbb{R}^{n\times m}$ with n users and m items, and a graph $G \in \mathbb{R}^{n\times n}$ encoding relationships between users. For simplicity, we do not assume a graph on the m items, though including it should be straightforward.

3.1 Graph Regularized Matrix Factorization

Explicit Feedback : The objective function of Graph Regularized Matrix Factorization (GRMF) Cai et al. ((2011)), Rao et al. ((2015)), Zhou et al. ((2012)) is:

$$\min_{U,V} \sum_{(i,j)\in\Omega} \left(R_{i,j} - u_i^\top v_j\right)^2 + \frac{\lambda}{2}(\|U\|_F^2 + \|V\|_F^2) + \mu\,\mathrm{tr}(U^\top \mathrm{Lap}(G)U) \quad (3)$$

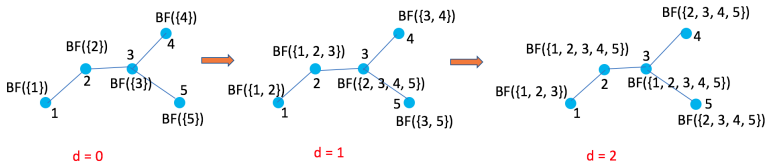

Figure 1: Illustration of Algorithm 1: the graph DNA encoding procedure. The curly brackets at each node indicate the nodes encoded at a particular step. At $d = 0$ each node's Bloom filter only encodes itself, and multi-hop neighbors are included as d increases.

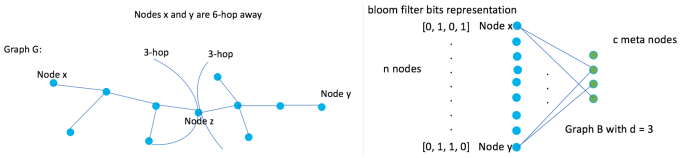

Figure 2: Illustration of our proposed DNA encoding method (DNA-3), with the corresponding bipartite graph representation.

where $U \in \mathbb{R}^{n\times r}, V \in \mathbb{R}^{m\times r}$ are the embeddings associated with users and items respectively, n is the number of users and m is the number of items, $R \in \mathbb{R}^{n\times m}$ is the sparse rating matrix, tr() is the trace operator, λ, μ are tuning coefficients, and Lap$(\cdot)$ is the graph Laplacian operator.

The last term is called graph regularization, which tries to enforce similar nodes (measured by edge weights in G) to have similar embeddings. One naive way Cao et al. ((2015)) to extend this to higher-order graph regularization is to replace the graph G with $\sum_{i=1}^{K} w_i \cdot G^i$ and then use the graph Laplacian of $\sum_{i=1}^{K} w_i \cdot G^i$ to replace G in (3). Computing G^i for even small i is computationally infeasible for most real-world applications, and we will soon lose the sparsity of the graph, leading to memory issues. Sampling or thresholding could mitigate the problem but suffers from performance degradation.

In contrast, our graph DNA obtained from Algorithm 1 does not suffer from any of these issues. Theorem 1 implies that the space complexity of our method is only of order $O(n \log n)$ for a graph with n nodes, instead of $O(n^2)$. The reduced number of non-zero elements using graph DNA leads to a significant speed-up in many cases.

We can easily use graph DNA in GRMF as follows: we treat the c bits as c new pseudo-nodes and add them to the original graph G. We then have $n + c$ nodes in a modified graph $\dot{G}$:

$$\dot{G} = \begin{bmatrix} G \in \mathbb{R}^{n\times n} & B \in \mathbb{R}^{n\times c} \\ B^\top \in \mathbb{R}^{c\times n} & \mathbf{0} \in \mathbb{R}^{c\times c} \end{bmatrix}. \tag{4}$$

To account for the c new nodes, we expand $U \in \mathbb{R}^{n\times r}$ to $\dot{U} \in \mathbb{R}^{(n+c)\times r}$ by appending parameters for the meta-nodes. The objective function for GRMF with Graph DNA with be the same as (3) except replacing U and G with $\dot{U}$ and $\dot{G}$. At the prediction stage, we discard the meta-node embeddings.

Implicit Feedback : For implicit feedback data, when R is a $0/1$ matrix, weighted matrix factorization is a widely used algorithm Hsieh et al. ((2015)), Hu et al. ((2008)). The only difference is that the loss function in (3) is replaced by $\sum_{(i,j):R_{ij}=1}(R_{ij} - u_i^T v_j)^2 + \sum_{(i,j):R_{ij}=0} \rho(R_{ij} - u_i^T v_j)^2$ where $\rho < 1$ is a hyper-parameter reflecting the confidence of zero entries. In this case, we can apply the Graph DNA encoding as before trivially.

3.2 Co-Factorization with Graph Information

Co-Factorization of Rating and Graph Information (Co-Factor) Liang et al. ((2016)), Singh and Gordon ((2008)) is ideologically very different from GRMF and GRWMF, because it does not use graph information as regularization term. Instead it treats the graph adjacency matrix as another rating matrix, sharing one-sided latent factors with the original rating matrix. Co-Factor minimizes the following objective function: $\min_{U,V} \sum_{(i,j)\in\Omega_R} \left(R_{i,j} - u_i^\top v_j\right)^2 + \frac{\lambda}{2}(\|U\|_F^2 + \|V\|_F^2 + \|V'\|_F^2) + \sum_{(i,j)\in\Omega_G} \left(G_{i,j} - u_i^\top v'_j\right)^2$, where $U \in \mathbb{R}^{n\times r}, V \in \mathbb{R}^{m\times r}, V' \in \mathbb{R}^{n\times r}$. We can extend Co-Factor to incorporate our DNA-d by replacing G with B in the equation above, where $B \in \mathbb{R}^{n\times c}$ is the Bloom filter bipartite graph adjacency matrix of n real-user nodes and c pseudo-user nodes, similar to B as in (4). We call the extension Co-Factor_DNA-d.

3.3 Graph Convolutional Matrix Completion

Graph Convolutional Matrix Completion (GC-MC) is a graph convolutional network (GCN) based geometric

matrix completion method Berg et al. ((2017)). In Berg et al. ((2017)), the rating matrix R is treated as adjacency matrix in GCN while side information G is treated as feature matrix for nodes — each user has an n-dimensional 0/1 feature that corresponds to a column of G. The GCN model then performs convolutions of these features on the bipartite rating graph. Convolutions of these features are performed on the bipartite rating graph. We find in our experiments that using these one-hot encodings of the graph as feature is an inferior choice both in terms of performance and speed. To capture higher order side graph information, it is better to use $G + \alpha G^2$ for some constant α and this alternate choice usually gives smaller generalization error than the original GC-MC method. However, it is hard to explicitly calculate $G + \alpha G^2$ and store the entire matrix for a large graph for the same reason described in Section 3.1. Again, we can use graph DNA to efficiently encode and store the higher order information before feeding it into GC-MC. We show in our experiments that this outperforms current state-of-the-art GCN methods Berg et al. ((2017)), Monti et al. ((2017)) as well as GC-MC with graph encoding methods that require training, such as Node2vec Grover and Leskovec ((2016)) and Deepwalk Perozzi et al. ((2014)). Our encoding scheme does not require training and therefore is a lot faster than previous encoding methods. More details are discussed in the experiment section 4.3.

4 Experiments

We show that our Graph DNA encoding technique can improve the performance of 4 popular graph-based recommendation algorithms: graph-regularized matrix factorization, co-factorization, weighted matrix factorization, and GCN-based graph convolution matrix factorization. All experiments except GCN are conducted on a server with Intel Xeon E5-2699 v3 @ 2.30GHz CPU and 256G RAM. The GCN experiments are conducted on Google Cloud with Nvidia V100 GPU.

4.1 Simulation Study

We first simulate a user/item rating dataset with user graph as side information, generate its graph DNA, and use it on a downstream task: matrix factorization.

We randomly generate user and item embeddings from standard Gaussian distributions, and construct an Erdős-Rényi Random graph of users. User embeddings are generated using Algorithm 3 in Appendix: at each propagation step, each user's embedding is updated by an average of its current embedding and its neighbors' embeddings. Based on user and item embeddings after $T = 3$ iterations of propagation, we generate the underlying ratings for each user-item pairs according to the inner product of their embeddings, and then sample a small portion of the dense rating matrix as training and test sets.

We implement our graph DNA encoding algorithm in python using a scalable python library Almeida et al. ((2007)) to generate Bloom filter matrix B. We adapt the GRMF C++ code to solve the objective function of GRMF_DNA-K with our Bloom filter enhanced graph $\dot{G}$. We compare the following variants:

1. MF: classical matrix factorization only with ℓ_2 regularization without graph information.
2. GRMF_G^d: GRMF with ℓ_2 regularization and using G, G^2, ..., G^d Cao et al. ((2015)).
3. GRMF_DNA-d: GRMF with ℓ_2 but using our proposed graph DNA-d.

We report the prediction performance with Root Mean Squared Error (RMSE) on test data. All results are reported on the test set, with all relevant hyperparameters tuned on a held-out validation set. To accurately measure how large the relative gain is from using deeper information, we introduce a new metric called Relative Graph Gain (RGG) for using information X, which is defined as:

$$\text{RGG}(X)\% = \left(\frac{\text{RMSE without Graph} - \text{RMSE with } X}{\text{RMSE without Graph} - \text{RMSE with } G} - 1\right) \times 100, \quad (5)$$

where RMSE is measured for the same method with different graph information. This metric would be 0 if only first order graph information is utilized and is only defined when the denominator is positive.

In Table 1, we can easily see that using a deeper neighborhood helps the recommendation performances on this synthetic dataset. Graph DNA-3's gain is 166% larger than that of using first-order graph G. We can see an increase in performance gain for an increase in depth d when $d \leq 3$. This is expected because we set $T = 3$ during our creation of this dataset.

4.2 Graph Regularized Matrix Factorization for Explicit Feedback

Next, we show that graph DNA can improve the performance of GRMF for explicit feedback. We conduct experiments on two real datasets: Douban Ma et al. ((2011)) and Flixster Zafarani and Liu ((2009)). Both datasets contain explicit feedback with ratings from 1 to 5. There are 129,490 users, 58,541 items in Douban. There are 147,612 users, 48,794 items in Flixster. Both datasets have a graph defined on the respective sets of users.

We pre-processed Douban and Flixster following the

same procedure in Rao et al. ((2015)), Wu et al. ((2017)). The experimental setups and comparisons are almost identical to the synthetic data experiment (see details in section 4.1). Due to the exponentially growing non-zero elements in the graph as we go deeper (see Table 7), we are unable to run full GRMF_G^4 and GRMF_G^5 for these datasets. In fact, GRMF_G^3 itself is too slow so we thresholded G^3 by only considering entries whose values are equal to or larger than 4. For the Bloom filter, we set a false positive rate of 0.1 and use capacity of 500 for Bloom filters, resulting in $c = 4,796$.

We can see from Table 1 that deeper graph information always helps. For Douban, graph DNA-3 is most effective, giving a relative graph gain of 82.79% compared to only 2% gain when using G^2 or G^3 naively. Interestingly for Flixster, using G^2 is better than using G^3. However, Graph DNA-3 and DNA-4 yield 10x and 15x performance improvements respectively, lending credence to the implicit regularization property of graph DNA. For a fixed size Bloom filter, the computational complexity of graph DNA scales linearly with depth d, as compared to exponentially for GRMF_G^d. We measure the speed in Table 2. The memory cost is only a fraction of n^2 after hashing. Such low memory and computational complexity allow us to scale to larger d, compared to baseline methods.

4.3 Co-Factorization with Graph for Explicit Feedback

We show our graph DNA can improve Co-Factor Liang et al. ((2016)), Singh and Gordon ((2008)) as well. The results are in Table 1. We find that applying DNA-3 to the Co-Factor method improves performance on both the datasets, more so for Flixster. This is consistent with our observations for GRMF in Table 1: deep graph information is more helpful for Flixster than Douban. Applying Graph DNA to Co-Factor is detailed in the Appendix.

4.4 Graph Regularized Weighted Matrix Factorization for Implicit Feedback

We follow the same procedure as in Wu et al. ((2018)) to set ratings of 4 and above to 1, and the rest to 0. We compare the baseline graph based weighted matrix factorization Hsieh et al. ((2015)), Hu et al. ((2008)) with our proposed weighted matrix factorization with DNA-3. We do not compare with Bayesian personalized ranking Rendle et al. ((2009)) and the recently proposed SQL-rank Wu et al. ((2018)) as they cannot easily utilize graph information.

The results are summarized in Table 3 with experimental details in the Appendix. Again, using DNA-3 achieves better prediction results over the baseline in terms of every single metric on both Douban and Flixster datasets.

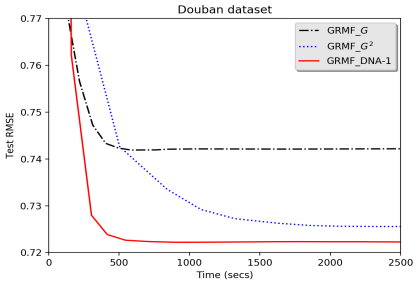

Figure 3: Compare Training Speed of GRMF, with and without Graph DNA.

4.5 Graph Convolutional Matrix Factorization

We can use graph DNA instead to efficiently encode and store the higher order information before feeding it into GC-MC.

We use the same split of three real-world datasets and follow the exact procedures as in Berg et al. ((2017)), Monti et al. ((2017)). We tuned hyperparameters using a validation dataset and obtain the best test results found within 200 epochs using optimal parameters. We repeated the experiments 6 times and report the mean and standard deviation of test RMSE. After some tuning, we use the capacity of 10 Bloom filters for Douban and 60 for Flixster, as the latter has a much denser second-order graph. With a false positive rate of 0.1, this implies that we use 96-bits Bloom filters for Douban and 960 bits for Flixster. We use the resulting bloom filter bitarrays as the node features, and pass that as the input to GC-MC. Using Graph DNA-2, the input feature dimensions are thus reduced from 3000 to 96 and 960, which leads to a significant speed-up. The original GC-MC method did not scale up well beyond 3000 by 3000 rating matrices with the user and the item side graphs as it requires using normalized adjacency matrix as user/item features. PinSage Ying et al. ((2018)), while scalable, does not utilize the user/item side graphs. Furthermore, it is not feasible to have $O(n)$ dimensional features for the nodes, where n is the number of nodes in side graphs. In contrast, our method only requires $O(\log(n))$ dimensional features. We can see from Table 4 that we outperform both GCN-based methods Berg et al. ((2017)) and Monti et al. ((2017)) in terms of performance by a large margin.

Note that another potential way to improve over GC-MC is to use other graph encoding schemes like Node2Vec Grover and Leskovec ((2016)) and DeepWalk

Table 1: Comparison of Graph Regularized Matrix Factorization Variants for Explicit Feedback on Synthetic, Douban and Flixster data. We use rank $r = 10$. RGG is the Relative Graph Gain defined in (5).

	Synthetic		Douban		Flixster	
Dataset	RMSE ($\times 10^{-1}$)	% RGG	RMSE ($\times 10^{-1}$)	% RGG	RMSE ($\times 10^{-1}$)	% RGG
MF	2.9971	-	7.3107	-	8.8111	-
GRMF_G	2.7823	0	7.2398	0	8.8049	0
GRMF_G^2	2.6543	59.5903	7.2381	2.3977	8.7849	322.5806
GRMF_G^3	2.5687	99.4413	7.2432	-4.7954	8.7932	188.7097
GRMF_G^4	2.5562	105.2607	-	-	-	-
GRMF_G^5	2.4853	138.2682	-	-	-	-
GRMF_G^6	2.4852	138.3147	-	-	-	-
GRMF_DNA-1	2.4303	163.8734	7.2191	29.1960	8.8013	58.0645
GRMF_DNA-2	2.4510	154.2365	7.2359	5.5007	8.8007	67.7419
GRMF_DNA-3	**2.4247**	**166.4804**	**7.1811**	**82.7927**	8.7383	1074.1935
GRMF_DNA-4	2.4466	156.2849	7.1971	60.2257	**8.7122**	**1495.1613**
Co-Factor_G	-	-	7.2743	0	8.7957	0
Co-Factor_DNA-3	-	-	**7.2623**	**32.9670**	**8.7354**	**391.5584**

Table 2: Graph DNA (Algorithm 1) Encoding Speed. We set number $c = 500$ and implement Graph DNA using single-core python. We can scale up linearly in terms of depth d for a fixed c.

	Graph Statistics		Graph DNA Encoding Time (secs)			
Dataset	Number of Nodes	Graph Density	DNA-1	DNA-2	DNA-3	DNA-4
Douban	129,490	0.0102%	132.2717	266.3740	403.9747	580.1547
Flixster	147,612	0.0117%	157.3103	317.7706	482.0360	686.8048

Table 3: Comparison of GRWMF Variants for Implicit Feedback on Douban and Flixster datasets. P stands for precision and N stands for NDCG. We use rank $r = 10$ and all results are in %.

Dataset	Methods	MAP	HLU	P@1	P@5	N@1	N@5
Douban	GRWMF_G	8.340	13.033	14.944	10.371	14.944	12.564
	GRWMF_DNA-3	**8.400**	**13.110**	**14.991**	**10.397**	**14.991**	**12.619**
Flixster	GRWMF_G	10.889	14.909	12.303	7.9927	12.303	12.734
	GRWMF_DNA-3	**11.612**	**15.687**	**12.644**	**8.1583**	**12.644**	**13.399**

Table 4: Comparison of GCN Methods for Explicit Feedback on Douban, Flixster and Yahoo Music datasets (3000 by 3000 as in Berg et al. ((2017)), Monti et al. ((2017))). All the methods except GC-MC utilize side graph information.

Dataset	Methods	Test RMSE ($\times 10^{-1}$)	% RGG
Douban	SRGCNN (reported by Berg et al. ((2017)))	-	-
	GC-MC	**7.3109 ± 0.0150**	-
	GC-MC_G	7.3698 ± 0.0737	N/A
	GC-MC_G^2	7.3123 ± 0.0139	N/A
	GC-MC_Node2vec	7.3666 ± 0.0218	N/A
	GC-MC_Deepwalk	7.3394 ± 0.0343	N/A
	GC-MC_DNA-2	7.3117 ± 0.0129	N/A
Flixster	SRGCNN (reported by Berg et al. ((2017)))	9.2600	-
	GC-MC	9.2614 ± 0.0578	-
	GC-MC_G	9.2374 ± 0.1045	0
	GC-MC_G^2	**8.9344 ± 0.0333**	**1262.4999**
	GC-MC_Node2vec	12.0370 ± 1.9474	N/A
	GC-MC_Deepwalk	9.0507 ± 0.1692	777.9167
	GC-MC_DNA-2	8.9536 ± 0.0770	1182.4999
Yahoo Music	SRGCNN (reported by Berg et al. ((2017)))	-	-
	GC-MC	22.6697 ± 0.3530	-
	GC-MC_G	21.3672 ± 0.4190	0
	GC-MC_G^2	20.2189 ± 0.8664	88.1612
	GC-MC_Node2vec	19.8901 ± 0.7948	113.4050
	GC-MC_Deepwalk	20.1603 ± 0.9342	92.6603
	GC-MC_DNA-2	**19.3879 ± 0.2874**	**151.9616**

Perozzi et al. ((2014)) to encode the user-user graph into node features. One clear drawback is that those graph embedding methods are time-consuming. Using the official Node2vec implementation, excluding reading and writing, it takes 416.13 seconds to encode the 3K by 3K subsampled Yahoo-Music item graph and obtain resulting 760-d node embeddings. For our method, it only takes 7.55 seconds to obtain the same 760-d features. Similarly, it takes over 15 mins to run the official C++ codes for DeepWalk Perozzi et al. ((2014)) using the same parameters as Node2Vec to encode the graph. In fact, fast encoding via hashing and bitwise-or that does not require training is one of the main advantages of our method.

Furthermore, even without considering the time overhead, we found our graph DNA encoding outperforms Node2Vec and DeepWalk in terms of test RMSE. Details can be found in Table 4. This could be due to that encoding higher-order information is more important for graph-regularized recommendation tasks, and graph DNA is a better and more direct way to encode higher order information compared with Node2Vec and DeepWalk.

Speed Comparisons Next, we compare the speed-ups obtained by graph DNA-d with GRMF G^d (a naive way to encode higher order information by computing powers of G). Figure 3 suggests that graph DNA-1 (which encodes hop-2 information) scales better than directly computing G^2 in GRMF.

Exploring Effects of Rank Finally, we investigate whether the proposed DNA coding can achieve consistent improvements when varying the rank in the

Table 5: Comparison of GRMF Methods of different ranks for Explicit Feedback on Flixster Dataset.

Rank	methods	test RMSE ($\times 10^{-1}$)	% gain
10	GRMF_G^2	8.7849	-
	GRMF_DNA-3	**8.7383**	**0.8262**
20	GRMF_G^2	8.9179	-
	GRMF_DNA-3	**8.7565**	**1.8098**
30	GRMF_G^2	9.0865	-
	GRMF_DNA-3	**8.9255**	**1.7719**

GRMF algorithm. In Table 5, we compare the proposed GRMF_DNA-3 with GRMF_G^2, which achieves the best RMSE without using DNA coding in the previous tables. The results clearly show that the improvement of the proposed DNA coding is consistent over different ranks and works even better when rank is larger.

5 Conclusion

In this paper, we proposed Graph DNA, a deep neighborhood aware encoding scheme for collaborative filtering with graph information. We make use of Bloom filters to incorporate higher order graph information, without the need to explicitly minimize a loss function. The resulting encoding is extremely space and computationally efficient, and lends itself well to multiple algorithms that make use of graph information, including Graph Convolutional Networks. Experiments show that Graph DNA encoding outperforms several baseline methods on multiple datasets in both speed and performance.

Acknowledgement This work is partially supported by NSF via IIS 1901527.

References

Z. Abbassi and V. S. Mirrokni. A recommender system based on local random walks and spectral methods. In *Proceedings of the 9th WebKDD and 1st SNA-KDD 2007 workshop on Web mining and social network analysis*, pages 102–108. ACM, 2007.

P. S. Almeida, C. Baquero, N. Preguiça, and D. Hutchison. Scalable bloom filters. *Information Processing Letters*, 101(6):255–261, 2007.

R. v. d. Berg, T. N. Kipf, and M. Welling. Graph convolutional matrix completion. *arXiv preprint arXiv:1706.02263*, 2017.

B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

D. Borthakur, J. Gray, J. S. Sarma, K. Muthukkaruppan, N. Spiegelberg, H. Kuang, K. Ranganathan, D. Molkov, A. Menon, S. Rash, et al. Apache hadoop goes realtime at facebook. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 1071–1080. ACM, 2011.

J. S. Breese, D. Heckerman, and C. Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*, pages 43–52. Morgan Kaufmann Publishers Inc., 1998.

A. Broder and M. Mitzenmacher. Network applications of bloom filters: A survey. *Internet mathematics*, 1(4):485–509, 2004.

D. Cai, X. He, J. Han, and T. S. Huang. Graph regularized nonnegative matrix factorization for data representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(8):1548–1560, 2011.

S. Cao, W. Lu, and Q. Xu. Grarep: Learning graph representations with global structural information. In *Proceedings of the 24th ACM international on conference on information and knowledge management*, pages 891–900. ACM, 2015.

F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS)*, 26(2):4, 2008.

J. Chen, J. Zhu, and L. Song. Stochastic training of graph convolutional networks with variance reduction. In *International Conference on Machine Learning*, pages 941–949, 2018.

M. M. Cisse, N. Usunier, T. Artieres, and P. Gallinari. Robust bloom filters for large multilabel classification tasks. In *Advances in Neural Information Processing Systems*, pages 1851–1859, 2013.

M. Courbariaux, Y. Bengio, and J.-P. David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in neural information processing systems*, pages 3123–3131, 2015.

M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*, pages 3844–3852, 2016.

D. Dubhashi and D. Ranjan. Balls and bins: A study in negative dependence. *Random Structures & Algorithms*, 13(2):99–124, 1998.

M. Gori, A. Pucci, V. Roma, and I. Siena. Itemrank: A random-walk based scoring algorithm for recommender engines. In *IJCAI*, volume 7, pages 2766–2771, 2007.

A. Grover and J. Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864. ACM, 2016.

W. Hamilton, Z. Ying, and J. Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pages 1024–1034, 2017a.

W. L. Hamilton, R. Ying, and J. Leskovec. Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584*, 2017b.

S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

C.-J. Hsieh, N. Natarajan, and I. Dhillon. Pu learning for matrix completion. In *International Conference on Machine Learning*, pages 2445–2453, 2015.

Y. Hu, Y. Koren, and C. Volinsky. Collaborative filtering for implicit feedback datasets. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*, pages 263–272. Ieee, 2008.

M. Jamali and M. Ester. Trustwalker: a random walk model for combining trust-based and item-based recommendation. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 397–406. ACM, 2009.

K. Joag-Dev, F. Proschan, et al. Negative association of random variables with applications. *The Annals of Statistics*, 11(1):286–295, 1983.

T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, (8):30–37, 2009.

D. Liang, J. Altosaar, L. Charlin, and D. M. Blei. Factorization meets the item embedding: Regularizing matrix factorization with item co-occurrence. In *Proceedings of the 10th ACM conference on recommender systems*, pages 59–66. ACM, 2016.

H. Ma, D. Zhou, C. Liu, M. R. Lyu, and I. King. Recommender systems with social regularization. In *Proceedings of the fourth ACM international conference on Web search and data mining*, pages 287–296. ACM, 2011.

F. Monti, M. Bronstein, and X. Bresson. Geometric matrix completion with recurrent multi-graph neural networks. In *Advances in Neural Information Processing Systems*, pages 3697–3707, 2017.

L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.

B. Perozzi, R. Al-Rfou, and S. Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710. ACM, 2014.

M. Pozo, R. Chiky, F. Meziane, and E. Métais. An item/user representation for recommender systems based on bloom filters. In *2016 IEEE Tenth International Conference on Research Challenges in Information Science (RCIS)*, pages 1–12. IEEE, 2016.

N. Rao, H.-F. Yu, P. K. Ravikumar, and I. S. Dhillon. Collaborative filtering with graph information: Consistency and scalable methods. In *Advances in neural information processing systems*, pages 2107–2115, 2015.

S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. In *Proceedings of the twenty-fifth conference on uncertainty in artificial intelligence*, pages 452–461. AUAI Press, 2009.

J. Serrà and A. Karatzoglou. Getting deep recommenders fit: Bloom embeddings for sparse binary input/output networks. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*, pages 279–287. ACM, 2017.

D. Shah et al. Gossip algorithms. *Foundations and Trends® in Networking*, 3(1):1–125, 2009.

G. Shani, M. Chickering, and C. Meek. Mining recommendations from the web. In *Proceedings of the 2008 ACM conference on Recommender systems*, pages 35–42. ACM, 2008.

Q. Shi, J. Petterson, G. Dror, J. Langford, A. Smola, and S. Vishwanathan. Hash kernels for structured data. *Journal of Machine Learning Research*, 10(Nov): 2615–2637, 2009.

A. Shinde and I. Savant. User based collaborative filtering using bloom filter with mapreduce. In *Proceedings of International Conference on ICT for Sustainable Development*, pages 115–123. Springer, 2016.

A. P. Singh and G. J. Gordon. Relational learning via collective matrix factorization. In *Proceedings of*

the 14th ACM SIGKDD international conference on Knowledge discovery and data mining, pages 650–658. ACM, 2008.

L. Wu, C.-J. Hsieh, and J. Sharpnack. Large-scale collaborative ranking in near-linear time. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 515–524. ACM, 2017.

L. Wu, C.-J. Hsieh, and J. Sharpnack. Sql-rank: A listwise approach to collaborative ranking. In *Proceedings of Machine Learning Research (35th International Conference on Machine Learning)*, volume 80, 2018.

L. Wu, S. Li, C.-J. Hsieh, and J. Sharpnack. Temporal collaborative ranking via personalized transformer. *arXiv preprint arXiv:1908.05435*, 2019a.

L. Wu, S. Li, C.-J. Hsieh, and J. L. Sharpnack. Stochastic shared embeddings: Data-driven regularization of embedding layers. In *Advances in Neural Information Processing Systems*, pages 24–34, 2019b.

W. Xie, D. Bindel, A. Demers, and J. Gehrke. Edge-weighted personalized pagerank: breaking a decade-old performance barrier. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1325–1334. ACM, 2015.

R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 974–983. ACM, 2018.

H.-F. Yu, H.-Y. Huang, I. S. Dhillon, and C.-J. Lin. A unified algorithm for one-class structured matrix factorization with side information. In *AAAI*, pages 2845–2851, 2017.

R. Zafarani and H. Liu. Social computing data repository at ASU, 2009. URL http://socialcomputing.asu.edu.

T. Zhou, H. Shan, A. Banerjee, and G. Sapiro. Kernelized probabilistic matrix factorization: Exploiting graphs and side information. In *Proceedings of the 2012 SIAM international Conference on Data mining*, pages 403–414. SIAM, 2012.