Graph DNA: Deep Neighborhood Aware Graph Encoding for Collaborative Filtering

Liwei Wu
Department of Statistics
University of California, Davis
Davis, CA 95616
liwu@ucdavis.edu

Hsiang-Fu Yu
Amazon
Palo Alto, CA 94301
rofu.yu@gmail.com

Nikhil Rao
Amazon
Palo Alto, CA 94301
nikhilrao86@gmail.com

James Sharpnack
Department of Statistics
University of California, Davis
Davis, CA 95616
jsharpna@ucdavis.edu

Cho-Jui Hsieh
Department of Computer Science
University of California, Los Angles
Los Angles, CA 90095
chohsieh@cs.ucla.edu

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, leading to a performance boost with little computational and memory overhead.

1 Introduction

Recommendation systems are increasingly prevalent due to content delivery platforms, e-commerce websites, and mobile apps [35]. Classical collaborative filtering algorithms use matrix factorization to identify latent features that describe the user preferences and item meta-topics from partially observed ratings [25]. 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, [31, 45] propose to add a graph regularization to the matrix factorization formulation to exploit additional graph structure; and [26] 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, [3] incorporated graph information when learning features with a Graph Convolution Network (GCN) based recommendation algorithm. GCNs [24] constitute flexible methods for incorporating graph structure beyond first-order

Preprint. Under review.

neighborhoods, but their training complexity typically scales rapidly with the depth, even with sub-sampling techniques [11]. 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 propose a Graph Deep Neighborhood Aware (Graph DNA) encoding, which approximately captures the higher-order neighborhood information of each node via Bloom filters [4]. 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 [31, 45], co-factoring [26], and GCN-based recommendation systems [28]. In some cases, using information from deeper neighborhoods (like 4^{th} order) yields a 15x increase in performance, with graph DNA encoding yielding a 6x speedup compared to directly using the 4^{th} power of the graph adjacency matrix.

Related Work Matrix factorization has been used extensively in recommendation systems with both explicit [25] and implicit [21] 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, [31, 45] 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 [43], this was extended to the implicit feedback setting. [26] 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 [3] used a GCN approach performing convolutions on the main bipartite graph by treating the first-order side graph information as features, and [28] 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 [16]. [22] extended this method to include graph side information in the model. Finally, the PageRank [29] algorithm can be seen as computing the steady state distribution of a Markov network, and similar methods for recommender systems was proposed in [1, 41].

For a complete list of related works of representation learning on graphs, we refer the interested user to [18]. For the collaborative filtering setting, [3, 28] use Graph Convolutional Neural Networks [14], 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 [24] and GraphSage [17], because they are intended to solve semi-supervised classification problem over graphs with nodes' features. PinSage [42] is the GraphSage extension to non-personalized graph-based recommendation algorithm but not meant for collaborative filtering problems. GC-MC [3] extend GCN to collaborative filtering, albeit less scalable than [42]. 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 [12], and for hashing deep neural network models representations [13, 19, 36]. However, to the best of our knowledge, they have not been used to encode graphs, nor has this encoding been applied to recommender systems

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 [4] 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 [5, 10]. A Bloom filter $\mathcal{B}$ consists of k independent 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 [7]. 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 [34]. 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 $\mathcal{N}_d(i) := \{j : \text{distance}_G(i, j) \leq d\}$, 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 [30, 33, 37], which use Bloom filter for standard hashing and is unrelated to graph encoding.

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 precisely the size of such false positives and the number of common bits in the following theorem. The following theorem 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 (the inner product of the vectorized Bloom filters, $\langle \mathcal{B}[\mathtt{i}], \mathcal{B}[\mathtt{j}] \rangle$). There exists universal constants C_0, C_1, such that for any $\gamma > 0$, with probability $1 - \gamma$,*

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

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

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

This theorem is a corollary of the more precise Theorem 2, which is stated in the Appendix. In order to establish these results, we provide Lemma 1, which demonstrates that the bits of Bloom filters are negatively associated (basic properties of negative associativity can be found in [15, 23]), 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 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).*

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 [31], and to use B as side features in graph convolutional network based geometric matrix factorization algorithm [3, 28] 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 is straightforward.

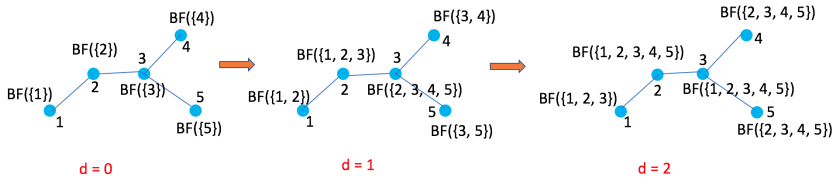

Figure 1: Illustration of Algorithm 1: the graph DNA encoding procedure. The curly brackets at each node indicate the nodes encoded at a particular step. At $d = 0$ each node's Bloom filter only encodes itself, and multi-hop neighbors are included as d increases.

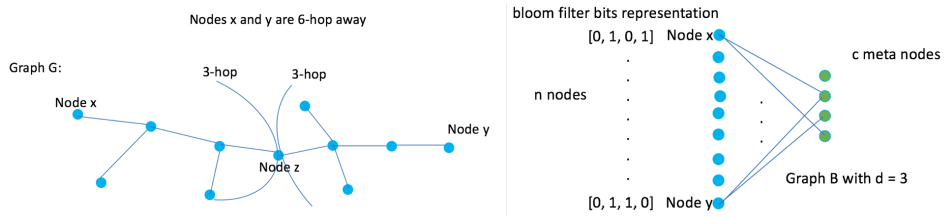

Figure 2: Illustration of our proposed DNA encoding method (DNA-3), with the corresponding bipartite graph representation.

3.1 Graph Regularized Matrix Factorization

The objective function of Graph Regularized Matrix Factorization (GRMF) [8, 31, 45] is:

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

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

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 [9] 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 B from Algorithm 1 does not suffer from any of the 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.

For implicit feedback data, when R is a 0/1 matrix, weighted matrix factorization is a widely used algorithm [20, 21]. 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. We also describe how to apply graph DNA towards Co-Factor [26, 38] and Graph Convolutional Matrix Completion [3] in the Appendix.

4 Experiments

We show that our proposed 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.

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 graphs 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 [2] 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 [9].
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:

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

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.

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 [27] and Flixster [44]. 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 [31, 39]. The experimental setups and comparisons are almost identical to the synthetic data experiment (see details in section 4). Due to the exponentially growing non-zero elements in the graph as we go deeper (see Table 6), 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 $c = 4,796$.

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 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

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.

Co-Factorization with Graph for Explicit Feedback We show our graph DNA can improve Co-Factor [26, 38] 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.

Graph Regularized Weighted Matrix Factorization for Implicit feedback We follow the same procedure as in [40] to set ratings of 4 and above to 1, and the rest to 0. We compare the baseline graph based weighted matrix factorization [20, 21] with our proposed weighted matrix factorization with DNA-3. We do not compare with Bayesian personalized ranking [32] and the recently proposed SQL-rank [40] 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.

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 [3, 28]). All the methods except GC-MC utilize side graph information.

Dataset	Methods	Test RMSE ($\times 10^{-1}$)	Time/epoch (secs)	% RGG	Speedup
Douban	SRGCNN (reported by [3])	8.0100	-	-	-
	GC-MC	7.3109 ± 0.0150	**0.0410**	-	9.72x
	GC-MC_G	7.3698 ± 0.0737	0.3985	N/A	1.00x
	GC-MC_G^2	7.3123 ± 0.0139	0.4221	N/A	0.94x
	GC-MC_DNA-2	$\mathbf{7.3117 \pm 0.0129}$	0.1709	N/A	2.33x
Flixster	SRGCNN (reported by [3])	9.2600	-	-	-
	GC-MC	9.2614 ± 0.0578	**0.0232**	-	**13.65x**
	GC-MC_G	9.2374 ± 0.1045	0.3166	0	1.00x
	GC-MC_G^2	$\mathbf{8.9344 \pm 0.0333}$	0.3291	**1262.4999**	0.96x
	GC-MC_DNA-2	8.9536 ± 0.0770	0.0524	1182.4999	6.04x
Yahoo Music	SRGCNN (reported by [3])	22.4000	-	-	-
	GC-MC	22.6697 ± 0.3530	**0.0684**	-	**1.75x**
	GC-MC_G	21.3672 ± 0.4190	0.1198	0	1.00x
	GC-MC_G^2	20.2189 ± 0.8664	0.1177	88.1612	1.02x
	GC-MC_DNA-2	$\mathbf{19.3879 \pm 0.2874}$	0.0896	**151.9616**	1.34x

Graph Convolutional Matrix Factorization Graph Convolutional Matrix Completion (GC-MC) is a graph convolutional network (GCN) based geometric matrix completion method [3]. In [3], the side graphs over users and items are represented as the adjacency matrices and these one-hot encodings are treated as features for nodes in the 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. Again, we can use graph DNA instead to efficiently encode and store the higher order information before feeding it into GC-MC. The exact means to use Graph DNA is detailed in the Appendix.

We use the same split of three real-world datasets and follow the exact procedures as in [3, 28]. 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. So the feature dimension is reduced from 3000 to 96 and 960 when using our graph DNA-2, 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 [42], 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. By contrast, our method only requires $O(\log(n))$ dimensional features. We can see from Table 4 that we outperform both GCN-based methods [3] and [28] in terms of speed and performance by a large margin.

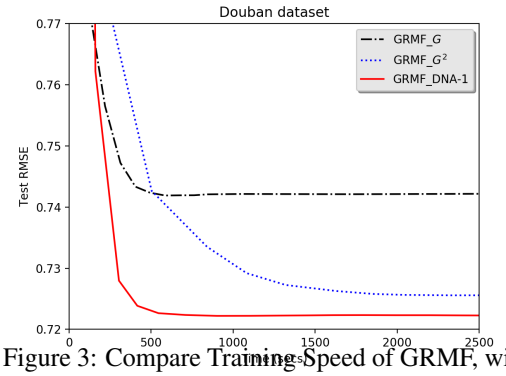

Figure 3: Compare Training Speed of GRMF, with and without Graph DNA.

Speed Comparisons Finally, we compare the speed-ups obtained by graph DNA-d with GRMF G^d. Since both algorithms scale with the number of edges in the constructed graph, we see that the Bloom filter based method scales substantially better compared to computing and using G^2 in Figure 3.

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.

References

[1] Zeinab Abbassi and Vahab 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.

[2] Paulo Sérgio Almeida, Carlos Baquero, Nuno Preguiça, and David Hutchison. Scalable bloom filters. *Information Processing Letters*, 101(6):255–261, 2007.

[3] Rianne van den Berg, Thomas N Kipf, and Max Welling. Graph convolutional matrix completion. *arXiv preprint arXiv:1706.02263*, 2017.

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

[5] Dhruba Borthakur, Jonathan Gray, Joydeep Sen Sarma, Kannan Muthukkaruppan, Nicolas Spiegelberg, Hairong Kuang, Karthik Ranganathan, Dmytro Molkov, Aravind Menon, Samuel 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.

[6] John S Breese, David Heckerman, and Carl 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.

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

[8] Deng Cai, Xiaofei He, Jiawei Han, and Thomas S Huang. Graph regularized nonnegative matrix factorization for data representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(8):1548–1560, 2011.

[9] Shaosheng Cao, Wei Lu, and Qiongkai 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.

[10] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C Hsieh, Deborah A Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E Gruber. Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS)*, 26(2):4, 2008.

[11] Jianfei Chen, Jun Zhu, and Le Song. Stochastic training of graph convolutional networks with variance reduction. In *International Conference on Machine Learning*, pages 941–949, 2018.

[12] Moustapha M Cisse, Nicolas Usunier, Thierry Artieres, and Patrick Gallinari. Robust bloom filters for large multilabel classification tasks. In *Advances in Neural Information Processing Systems*, pages 1851–1859, 2013.

[13] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in neural information processing systems*, pages 3123–3131, 2015.

[14] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*, pages 3844–3852, 2016.

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

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

[17] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pages 1024–1034, 2017.

[18] William L Hamilton, Rex Ying, and Jure Leskovec. Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584*, 2017.

[19] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[20] Cho-Jui Hsieh, Nagarajan Natarajan, and Inderjit Dhillon. Pu learning for matrix completion. In *International Conference on Machine Learning*, pages 2445–2453, 2015.

[21] Yifan Hu, Yehuda Koren, and Chris Volinsky. Collaborative filtering for implicit feedback datasets. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*, pages 263–272. Ieee, 2008.

[22] Mohsen Jamali and Martin 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.

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

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

[25] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, (8):30–37, 2009.

[26] Dawen Liang, Jaan Altosaar, Laurent Charlin, and David 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.

[27] Hao Ma, Dengyong Zhou, Chao Liu, Michael R Lyu, and Irwin 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.

[28] Federico Monti, Michael Bronstein, and Xavier Bresson. Geometric matrix completion with recurrent multi-graph neural networks. In *Advances in Neural Information Processing Systems*, pages 3697–3707, 2017.

[29] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.

[30] Manuel Pozo, Raja Chiky, Farid Meziane, and Elisabeth 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.

[31] Nikhil Rao, Hsiang-Fu Yu, Pradeep K Ravikumar, and Inderjit S Dhillon. Collaborative filtering with graph information: Consistency and scalable methods. In *Advances in neural information processing systems*, pages 2107–2115, 2015.

[32] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars 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.

[33] Joan Serrà and Alexandros 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.

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

[35] Guy Shani, Max Chickering, and Christopher Meek. Mining recommendations from the web. In *Proceedings of the 2008 ACM conference on Recommender systems*, pages 35–42. ACM, 2008.

[36] Qinfeng Shi, James Petterson, Gideon Dror, John Langford, Alex Smola, and SVN Vishwanathan. Hash kernels for structured data. *Journal of Machine Learning Research*, 10(Nov): 2615–2637, 2009.

[37] Anita Shinde and Ila 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.

[38] Ajit P Singh and Geoffrey 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.

[39] Liwei Wu, Cho-Jui Hsieh, and James 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.

[40] Liwei Wu, Cho-Jui Hsieh, and James Sharpnack. Sql-rank: A listwise approach to collaborative ranking. In *Proceedings of Machine Learning Research (35th International Conference on Machine Learning)*, volume 80, 2018.

[41] Wenlei Xie, David Bindel, Alan Demers, and Johannes 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.

[42] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure 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.

[43] Hsiang-Fu Yu, Hsin-Yuan Huang, Inderjit S Dhillon, and Chih-Jen Lin. A unified algorithm for one-class structured matrix factorization with side information. In *AAAI*, pages 2845–2851, 2017.

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

[45] Tinghui Zhou, Hanhuai Shan, Arindam Banerjee, and Guillermo 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.

6 Appendix

6.1 Theory for Bloom Filters

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

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

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

$$\Gamma_0 = c\left(1 - \exp\left\{-k\frac{|N(x) \cap N(y)|}{c}\right\}\right),$$
$$\Gamma_1 = c\left(1 - \exp\left\{-k^2\frac{|N(x) \triangle N(y)|^2}{4c^2} - \frac{k|N(x) \cap N(y)|}{c-1}\right\}\right).$$

We prove Theorem 2 in subsection 6.1.2. For now, we prove Theorem 1 which is in fact a corollary of this main result.

Proof of Theorem 1. We can see that there exist C_0, C_1 such that for any $\delta \geq 0$,

$$\left(\frac{e^{\delta}}{(1+\delta)^{(1+\delta)}}\right)^{\mathbb{E}Q} \leq C_1 e^{-C_0\delta\mathbb{E}Q}.$$

Then we have that with probability $1-\gamma$,

$$Q \leq \left(1 + \frac{1}{C_0}\log\frac{C_1}{\gamma}\right)\mathbb{E}Q \leq \left(1 + \frac{1}{C_0}\log\frac{C_1}{\gamma}\right)\Gamma_1.$$

Note that because $1 - e^{-x} \leq x$,

$$\Gamma_1 \leq k^2\frac{|N(x) \triangle N(y)|^2}{4c} + \frac{ck|N(x) \cap N(y)|}{c-1}.$$

Moreover, for $\delta \in (0,1)$,

$$\left(\frac{e^{-\delta}}{(1-\delta)^{(1-\delta)}}\right)^{\mathbb{E}Q} \leq e^{-\frac{\delta^2}{3}\mathbb{E}Q} \leq e^{-\frac{\delta^2}{3}\Gamma_0}.$$

Hence,

$$\mathbb{P}\{Q \leq (1-\delta)\Gamma_0\} \leq e^{-\frac{\delta^2}{3}\Gamma_0}.$$

Suppose that for some $\alpha \in (0,1)$, $\alpha c > k|N(x) \cap N(y)|$ then we have that

$$\Gamma_0 \geq \frac{(1-e^{-\alpha})}{\alpha}k|N(x) \cap N(y)|.$$

The function $(1-e^{-\alpha})/\alpha$ is decreasing and the limit as $\alpha \to 0$ is 1. Thus, for any $\delta \in (0,1)$, there exists an $\alpha \geq 0$ such that if $\alpha c > k|N(x) \cap N(y)|$ then $\Gamma_0 \geq (1-\delta)k|N(x) \cap N(y)|$. If this is the case then

$$\mathbb{P}\{Q \leq (1-\delta)k|N(x) \cap N(y)|\} \leq e^{-\frac{1}{3}(1-\delta)\delta^2 k|N(x)\cap N(y)|}.$$

□

6.1.1 Negative Associativity of Bloom Filters

First, let us go over the definition of negative associativity. Random variables, $\{q_i\}_{i=1}^c$, are negatively associative (NA), if for any functions f, g, both monotonically increasing or decreasing, and disjoint sets $I, J \subset \{1, \ldots, c\}$,

$$\mathbb{E}[f(q_I)g(q_J)] \leq \mathbb{E}[f(q_I)] \cdot \mathbb{E}[g(q_J)],$$

where q_I, q_J are the variables restricted to these sets.

Lemma 1. **(1)** *Let $\{q_{0,i}\}_{i=1}^c, \{q_{1,i}\}_{i=1}^c \subset \{0,1\}^c$ be two independent random bitarrays that are both NA. Then $q_0|q_1$, the elementwise 'or' operation, and $q_0\&q_1$, the elementwise 'and' operation, are both NA. So NA is closed under elementwise 'or' and 'and' operations.*

(2) *Let q_i be the ith bit in any Bloom filter of the set $\mathcal{N}$ with independent hash functions, then the random bits, $\{q_i\}_{i=1}^c$, are NA.*

Proof. **(1)** We show that NA is closed under both elementwise operations. First, note that the concatenation $\{q_{0,1}, \ldots, q_{0,c}, q_{1,1}, \ldots, q_{1,c}$ is NA, by closure of NA under independent union (Property P7 in [23]). Then on the disjoint sets, $\{q_{0,i}, q_{1,i}\}_{i=1}^{c}$, apply the bit operation to produce the resulting array. Operation 'or' is monotonically increasing because, $q_{0,i}|q_{1,i} = 1\{q_{0,i} + q_{1,i} > 0\}$, 'and' is as well because $q_{0,i}\&q_{1,i} = 1\{q_{0,i} + q_{1,i} > 1\}$. Finally we conclude by closure of NA under monotonic increasing functions on disjoint sets (Property P6 in [23]).

(2) Consider hash function $j \in \{1, \ldots, k\}$ for node $v \in \mathcal{N}$. Let B_v^j be the c-bit Bloom filter bitarray for this vertex and hash function only, then B_v^j has only a single bit that is 1 and the rest are 0. By the 0-1 property for binary bits, we know that B_v^j has NA entries (Lemma 8 in [15]), since $\sum_i B_{v,i}^j = 1$. Then the Bloom filter, B, of $\mathcal{N}$ is $B = |_{j=1}^{k}|_{v \in N(x)} B_v^j$—the 'or' operation applied to all hashes and vertices, and we conclude by property (1). □

6.1.2 Proof of Theorem 2

Consider the partition of $N(x) \cup N(y)$ into $A_1 = N(x) \backslash N(y)$, $A_2 = N(y) \backslash N(x)$, $A_3 = N(x) \cap N(y)$. Let B_x, B_y be the Bloom filter bitarrays for $N(x), N(y)$ and let B_1, B_2, B_3 be those for A_1, A_2, A_3 respectively.

Notice that $B_x \& B_y = B_3 | (B_1 \& B_2)$, where the bit operations are elementwise. If all hash functions are independent, then B_1, B_2, B_3 are independent. Notice that for a given node and hash function the bit selected is random, but unique, which means that the elements of the bitarrays are not necessarily independent for any Bloom filter. However, the bitarray $B_3|(B_1 \& B_2)$ is negatively associative by Lemma 1. Let $q_i = (B_{1,i} \& B_{2,i}) | B_{3,i}$, then we have that,

$$\mathbb{E}q_i = 1 - (1 - \mathbb{E}[B_{1,i}] \cdot \mathbb{E}[B_{2,i}])(1 - \mathbb{E}[B_{3,i}]).$$

The probability that bit i in one of the bitarrays is 0 is

$$1 - \mathbb{E}[B_{j,i}] = \left(1 - \frac{1}{c}\right)^{k|A_j|}, \quad j = 1, 2, 3.$$

This can give us an expression in terms of $c, k, |A_1|, |A_2|, |A_3|$ for the expectation of $Q = \sum_{i=1}^{c} q_i$. We have that by Hoeffding's inequality for negatively associative random variables [15],

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

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

It remains to provide intelligible bounds on $\mathbb{E}Q$. By the inequalities $1 - 1/x \leq \log x \leq x - 1$,

$$-\frac{k|A_3|}{c-1} \leq \log(1 - \mathbb{E}[B_{3,i}]) \leq -\frac{k|A_3|}{c}$$

Also,

$$\mathbb{E}[B_{1,i}] \cdot \mathbb{E}[B_{2,i}] = \left(1 - \left(1 - \frac{1}{c}\right)^{k|A_1|}\right)\left(1 - \left(1 - \frac{1}{c}\right)^{k|A_2|}\right)$$

so by the inequality,

$$\log(1 - (1 - (1-x)^a)(1 - (1-x)^b)) \geq -abx^2, \quad a, b > 0, x \in [0, 1];$$

we have that

$$-k^2 \frac{|A_1||A_2|}{c^2} \leq \log(1 - \mathbb{E}[B_{1,i}] \cdot \mathbb{E}[B_{2,i}]) \leq 0.$$

Furthermore, notice that the LHS is minimized when $|A_1| = |A_2| = |N(x) \triangle N(y)|/2$,

$$-k^2 \frac{|A_1||A_2|}{c^2} \geq -k^2 \frac{|N(x) \triangle N(y)|^2}{4c^2}.$$

We then have that

$$\begin{aligned}\log(c - \mathbb{E}Q) &= \log c + \log(1 - \mathbb{E}[B_{1,i}] \cdot \mathbb{E}[B_{2,i}]) + \log(1 - \mathbb{E}[B_{3,i}]) \\ &\leq \log c - \frac{k|N(x) \cap N(y)|}{c}\end{aligned}$$

and

$$\log(c - \mathbb{E}Q) \geq \log c - k^2 \frac{|N(x) \triangle N(y)|^2}{4c^2} - \frac{k|N(x) \cap N(y)|}{c-1}.$$

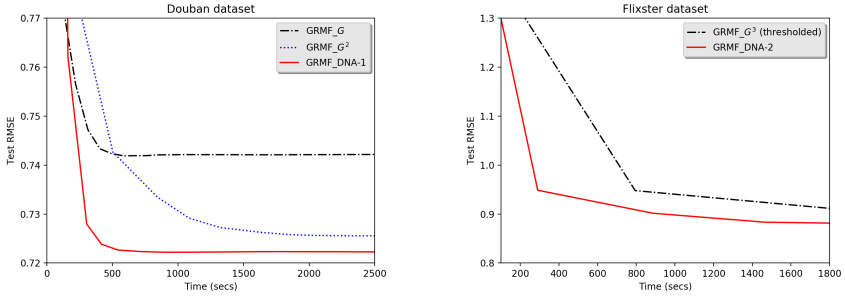

Figure 4: Compare Training Speed of GRMF, with and without Graph DNA.

Algorithm 2 A Standard Bloom Filter

class `BloomFilter`:
  def `constructor`(`self`, c, $\{\mathtt{h_t}(\cdot) : t = 1, \ldots, k\}$):
    $\mathtt{self.b}[i] = 0 \quad \forall i = 1, \ldots, c$
    $\mathtt{self.h_t} = \mathtt{h_t} \quad \forall i = 1, \ldots, k$

  def `add`(`self`, `x`):
    $\mathtt{self.b}[\mathtt{self.h_t}(\mathtt{x})] = 1 \quad \forall t = 1, \ldots, k$

  def `union`(`self`, `bf`):
    $\mathtt{self.b}[i] \leftarrow \mathtt{self.b}[i] \mid \mathtt{bf.b}[i] \quad \forall i = 1, \ldots, c$

  def `size`(`self`):
    return $\left\lceil -\frac{c}{k} \log\left(1 - \frac{\mathtt{nnz(self.b)}}{c}\right)\right\rceil$

6.2 Co-Factorization with Graph Information

Co-Factorization of Rating and Graph Information (Co-Factor) [26, 38] 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:

$$\min_{U,V} \sum_{(i,j)\in\Omega_R} \left(R_{i,j} - u_i^\top v_j\right)^2 + \frac{\lambda}{2}(\|U\|_F^2 + \|V\|_F^2 + \|V'\|_F^2) + \sum_{(i,j)\in\Omega_G} \left(G_{i,j} - u_i^\top v'_j\right)^2, \tag{6}$$

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 (6), 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.

6.3 Graph Convolutional Matrix Completion

Graph Convolutional Matrix Completion (GC-MC) is a graph convolutional network (GCN) based geometric matrix completion method [3]. In [3], the side information graph is represented as the adjacency matrix of the side graph and these one-hot encodings are treated as features for nodes in the 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 [3, 28].

6.4 Simulation Study

In the simulation we carried out, we set the number of users $n = 10,000$ and the number of items $m = 2,000$. We uniformly sample 5% for training and 2% for testing out of the total nm ratings. We choose $T = 3$ so

Table 5: Compare Bloom filters of different depths and sizes an on Synthesis Dataset. Note that the number of bits of Bloom filter is decided by Bloom filter's maximum capacity and tolerable error rate (i.e. false positive error, we use 0.2 as default).

methods	max capacity	c bits	nnz ratio	RMSE ($\times 10^{-3}$)	% Relative Graph Gain
GRMF_G^2	-	-	-	2.6543	59.5903
GRMF_DNA-1	20	135	0.217	2.4303	163.8734
GRMF_DNA-1	50	336	0.093	2.4795	140.9683
GRMF_DNA-2	20	135	0.880	2.4921	135.1024
GRMF_DNA-2	50	336	0.608	2.4937	134.3575
GRMF_DNA-2	100	672	0.381	2.4510	154.2365
GRMF_DNA-2	200	1,341	0.215	2.4541	152.7933
GRMF_DNA-3	200	1,341	0.874	2.4667	146.9274
GRMF_DNA-3	600	4,020	0.525	2.4572	151.3500
GRMF_DNA-3	1,000	6,702	0.364	2.4392	159.7299
GRMF_DNA-3	**1,500**	**10,050**	**0.262**	**2.4247**	**166.4804**
GRMF_DNA-4	2,000	13,401	0.743	2.5532	106.6573
GRMF_DNA-4	4,000	26,799	0.499	2.4466	156.2849

the graph contains at most 6-hop information among n users. We use rank $r = 50$ for both user and item embeddings. We set influence weight $w = 0.6$, i.e. in each propagation step, 60% of one user's preference is decided by its friends (i.e. neighbors in the friendship graph). We set $p = 0.001$, which is the probability for each of the possible edges being chosen in Erdős-Rényi graph G. A small edge probability p, influence weight $w < 1.0$, and a not too-large T is needed, because we don't want that all users become more or less the same after T propagation steps.

6.5 Metrics

We omit the definitions of RMSE, Precision@k, NDCG@k, MAP as those can be easily found online. HLU: Half-Life Utility [6, 35] is defined as:

$$\text{HLU} = \frac{1}{n}\sum_{i=1}^{n}\text{HLU}_i, \tag{7}$$

where n is the number of users and HLU_i is given by:

$$\text{HLU}_i = \sum_{l=1}^{k}\frac{max(R_{i\Pi_{il}} - d, 0)}{2^{(j-1)/(\alpha-1)}}, \tag{8}$$

where $R_{i\Pi_{il}}$ follows previous definition, d is the neural vote (usually the rating average), and α is the viewing halflife. The halflife is the number of the item on the list such that there is a 50-50 chance the user will review that item [6].

Algorithm 3 Simulation of Synthesis Data

Input: n users, m items, rank r, influence weight w, T propagation steps
Output: $R_{\text{tr}} \in \mathbb{R}^{n\times m}$, $R_{\text{te}} \in \mathbb{R}^{n\times m}$, $G \in \mathbb{R}^{n\times n}$

Randomly initialize $U \in \mathbb{R}^{n\times r}, V \in \mathbb{R}^{m\times r}$ from standard normal distribution
Generate a random undirected Erdős-Rényi graph G with each edge being chosen with probability p
for $t = 1, ..., T$ **do**
    for $i = 1, ..., n$ **do**
        $\tilde{U}_i = w \cdot \sum_{j:(i,j)\in G} U_j + (1 - w) \cdot U_i$
    Set $U = \tilde{U}$
Generate rating matrix $R = UV^T$
Random sample observed user/item indices in training and test data: $\Omega_{\text{tr}}, \Omega_{\text{te}}$
Obtain $R_{\text{tr}} = \Omega_{\text{tr}} \circ R$, $R_{\text{te}} = \Omega_{\text{te}} \circ R$
return rating matrices $R_{\text{tr}}, R_{\text{te}}$, user graph G

6.6 Graph Regularized Weighted Matrix Factorization for Implicit feedback

We use the rank $r = 10$, negatives' weight $\rho = 0.01$ and measure the prediction performance with metrics MAP, HLU, Precision@k and NDCG@k (see definitions of metrics in Appendix 6.5).

Table 6: Compare nnz of different methods on Douban and Flixster datasets. GRMF_G^4 and GRMF_DNA-2 are using the same 4-hop information in the graph but in different ways. Note that we do not exclude potential overlapping among columns.

Dataset	methods	R_{tr}	G	G^2	G^3	G^4	B	total nnz
Douban	MF	9,803,098	-	-	-	-	-	9,803,098
	GRMF_G	9,803,098	1,711,780	-	-	-	-	11,514,878
	GRMF_G^2	9,803,098	1,711,780	106,767,776	-	-	-	118,282,654
	GRMF_G^3	9,803,098	1,711,780	106,767,776	2,313,572,544	-	-	2,431,855,198
	GRMF_G^4	**9,803,098**	**1,711,780**	**106,767,776**	**2,313,572,544**	**8,720,553,105**	-	**11,152,408,303**
	GRMF_DNA-1	9,803,098	0	-	-	-	8,834,740	18,637,838
	GRMF_DNA-2	9,803,098	1,711,780	-	-	-	142,897,900	154,412,778
	GRMF_DNA-3	9,803,098	1,711,780	-	-	-	928,159,604	939,674,482
Flixster	MF	3,619,304	-	-	-	-	-	3,619,304
	GRMF_G	3,619,304	2,538,746	-	-	-	-	6,158,050
	GRMF_G^2	3,619,304	2,538,746	130,303,379	-	-	-	136,461,429
	GRMF_G^3	3,619,304	2,538,746	130,303,379	2,793,542,551	-	-	3,060,307,359
	GRMF_G^4	**3,619,304**	**2,538,746**	**130,303,379**	**2,793,542,551**	**12,691,844,513**	-	**15,752,151,872**
	GRMF_DNA-1	3,619,304	0	-	-	-	12,664,952	16,284,256
	GRMF_DNA-2	3,619,304	2,538,746	-	-	-	181,892,883	188,050,933
	GRMF_DNA-3	3,619,304	2,538,746	-	-	-	1,185,535,529	1,191,693,579

Table 7: Compare 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**

We follow the similar procedure to what is done before in GRMF and co-factor: we run all combinations of tuning parameters of $\lambda_l \in \{0.01, 0.1, 1, 10, 100\}$ and $\lambda_g \in \{0.01, 0.1, 1, 10, 100\}$ for each method on validation data for fixed number 40 epochs and choose the best combination as the parameters to use on test data. We then report the best prediction results during first 40 epochs on test data with the chosen parameter combination.

6.7 Explore effects of rank

Next we investigate whether the proposed DNA coding can achieve consistent improvements when varying the rank in the GRMF algorithm. In Table 7, 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.

6.8 Reproducibility

To reproduce results reported in the paper, one need to download data (douban and flixster) and third-party C++ Matrix Factorization library from the link `https://www.csie.ntu.edu.tw/~cjlin/papers/ocmf-side/`. One can simply follow README there to compile the codes in Matlab and run one-class matrix factorization library in different modes (both explicit feedback and implicit feedback works). The advantage of using this library is that the codes support multi-threading and runs quite fast with very efficient memory space allocations. It also supports with graph or other side information. All three methods' baseline can be simply run with the tuning parameters we reported in the Table 9, 10, 11 in Appendix.

To reproduce results of our DNA methods, one need to generate Bloom filter matrix B following Algorithm 1. We will provide our python codes implementing Algorithm 1 and Matlab codes converting into the formats the library requires.

For baselines and our DNA methods, We perform a parameter sweep for $\lambda_l \in \{0.01, 0.1, 1, 10, 100\}$ and $\lambda_g \in \{0.01, 0.1, 1, 10, 100\}$ as well as for $\alpha \in \{0.0001, 0.001, 0.01, 0.1, 0.3, 0.7, 1\}$, for $\beta \in \{0.005, 0.01, 0.03, 0.05, 0.1\}$ when needed. We run all combinations of tuning parameters for each method on validation set for 40 epochs and choose the best combination as the parameters to use on test data. We then report the best test RMSE in first 40 epochs on test data with the chosen parameter combination. We provide all the chosen combinations of tuning parameters that achieves reported optimal results in results tables in the

Table 8: Compare Matrix Factorization for Explicit Feedback on Synthesis Dataset. The synthesis dataset has $10,000$ users and $2,000$ items with user friendship graph of size $10,000 \times 10,000$. Note that the graph only contains at most 6-hop valid information. GRMF_G^6 means GRMF with $G + \alpha \cdot G^2 + \beta \cdot G^3 + \gamma \cdot G^4 + \epsilon \cdot G^5 + \omega \cdot G^6$. GRMF_DNA-$d$ means depth d is used.

methods	test RMSE ($\times 10^{-3}$)	λ_l	λ_g	α	β	γ	ϵ	ω	% gain over baseline
MF	2.9971	0.01	-	-	-	-	-	-	-
GRMF_G	2.7823	0.01	0.01	-	-	-	-	-	7.16693
GRMF_G^2	2.6543	0.01	0.01	0.3	-	-	-	-	11.43772
GRMF_G^3	2.5687	0.01	0.01	0.01	0.05	-	-	-	14.29382
GRMF_G^4	2.5562	0.01	0.01	0.01	0.05	0.1	-	-	14.71088
GRMF_G^5	2.4853	0.01	0.01	0.01	0.05	0.1	0.1	-	17.07651
GRMF_G^6	2.4852	0.01	0.01	0.01	0.05	0.1	0.1	0.01	17.07984
GRMF_DNA-1	2.4303	0.01	0.01	-	-	-	-	-	18.91161
GRMF_DNA-2	2.4510	0.01	0.01	-	-	-	-	-	18.22095
GRMF_DNA-3	**2.4247**	**0.01**	**0.01**	-	-	-	-	-	**19.09846**
GRMF_DNA-4	2.4466	0.01	0.01	-	-	-	-	-	18.36776

Table 9: Compare Matrix Factorization methods for Explicit Feedback on Douban and Flixster data. We use rank $r = 10$.

Dataset	methods	test RMSE ($\times 10^{-1}$)	λ_l	λ_g	α	β	% gain over baseline
Douban	MF	7.3107	1	-	-	-	-
	GRMF_G	7.2398	0.1	100	-	-	0.9698
	GRMF_G^2	7.2381	0.1	100	0.001	-	0.9930
	GRMF_G^3 (full)	7.2432	0.1	100	0.05	0.0005	0.9350
	GRMF_G^3 (thresholded)	7.2382	0.1	100	0.05	0.0005	0.9917
	GRMF_DNA-1	7.2191	0.1	100	-	-	1.2689
	GRMF_DNA-2	7.2359	1	10	-	-	1.0232
	GRMF_DNA-3	**7.2095**	**0.01**	**100**	-	-	**1.3843**
Flixster	MF	8.8111	0.1	1	-	-	-
	GRMF_G	8.8049	0.01	1	-	-	0.0704
	GRMF_G^2	8.7849	0.01	1	0.05	-	0.2974
	GRMF_G^3 (full)	8.7932	0.1	1	0.01	0.1	0.2032
	GRMF_G^3 (thresholded)	8.7920	0.01	1	0.01	0.1	0.2168
	GRMF_DNA-1	8.8013	0.01	1	-	-	0.1112
	GRMF_DNA-2	8.8007	0.1	1	-	-	0.1180
	GRMF_DNA-3	**8.7453**	**0.1**	**100**	-	-	**0.7468**

Table 9, 10, 11 in Appendix. One just need to exactly follow our procedures in Section 3 to construct new $\dot{G}, \dot{U}$ to replace the G, U in baseline methods before feeding into Matlab.

As to simulation study, we will also provide python codes to repeat our Algorithm 3 to generate synthesis dataset. One can easily simulate the data before converting into Matlab data format and running the codes as before. The optimal parameters can be found in Table 8. For all the methods, we select the best parameters λ_l and λ_g from $\{0.01, 0.1, 1, 10, 100\}$. For method GRMF_$G^2$, we tune an additional parameter $\alpha \in \{0.0001, 0.001, 0.01, 0.1, 0.3, 0.7, 1\}$. For the thrid-order method GRMF_G^3, we tune $\beta \in \{0.005, 0.01, 0.03, 0.05, 0.1\}$ in addition to $\lambda_l, \lambda_G, \alpha$. Due to the speed constraint, we are not able to tune a broader range of choices for α and β as it is too time-consuming to do so especially for douban and flixster datasets. For example, it takes takes about 3 weeks using 16-cores CPU to tune both α, β on flixster dataset. We run each method with every possible parameter combination for fixed 80 epochs on the same training data, tune the best parameter combination based on a small predefined validation data and report the best RMSE results on test data with the best tuning parameters during the first 80 epochs. Note that only on the small synthesis dataset, we calculate full G^3 and report the results. On real datasets, there is no way to calculate full G^4 to utilize the complete 4-hop information, because one can easily spot in Table 6 the number of non-zero elements (nnz) is growing exponentially when the hop increases by 1, which makes it impossible for one to utilize complete 3-hop and 4-hop information.

In Table 9, one can compare magnitude of optimal α and β to have a good idea of whether G or G^2 is more useful. G represents shallow graph information and G^2 represents deep graph information. If one already run GRMF_G^2, one can then use this as a preliminary test to decide whether to go deep with DNA-3 ($d = 3$) to capture deep graph information or simply go ahead with DNA-1 ($d = 1$) to fully utilize shallow information. For douban dataset, we have $\alpha = 0.05 > 0.0005 = \beta$, which implies shallow information is important and we should fully utilize it. It explains why DNA-1 is performing well both in terms of performance and speed on douban dataset. It is worth noting that GRMF_DNA-1's Bloom filter matrix B contains much more nnz than that of G in Table 6 though 20% less than that of G^2. On the other hand, for flixster dataset, we have $\alpha = 0.01 < 0.1 = \beta$, which implies in this dataset deeper information is more important and we should go deeper. That explains why here GRMF_DNA-3 (6-hop) achieves about 10 times more gain than using 1-hop GRMF_G.

Table 10: Compare Co-factor Methods for Explicit Feedback on Douban and Flixster Datasets. We use rank $r = 10$ for both methods.

Dataset	methods	test RMSE ($\times 10^{-1}$)	λ_l	% gain over baseline
Douban	co-factor_G	7.2743	1	-
	co-factor_DNA-3	**7.2674**	**1**	**0.5923**
Flixster	co-factor_G	8.7957	0.01	-
	co-factor_DNA-3	**8.7354**	**0.01**	**0.8591**

Table 11: Compare Weighted Matrix Factorization with Graph for Implicit Feedback on Douban and Flixster Datasets. We use rank $r = 10$ for both methods and all metric results are in $\%$.

Dataset	Methods	MAP	HLU	P@1	P@5	NDCG@1	NDCG@5	λ_l	λ_g
Douban	WMF_G	8.340	13.033	14.944	10.371	14.944	12.564	0.01	10
	WMF_DNA-3	**8.400**	**13.110**	**14.991**	**10.397**	**14.991**	**12.619**	**1**	**1**
Flixster	WMF_G	10.889	14.909	12.303	7.9927	12.303	12.734	10	0.1
	WMF_DNA-3	**11.612**	**15.687**	**12.644**	**8.1583**	**12.644**	**13.399**	**1**	**1**

6.9 Code

We will make our code available on Github in the final version of the paper