Matrix Factorization with Side and Higher Order Information*†

Vatsal Shah
University of Texas at Austin
vatsalshah1106@utexas.edu

Nikhil Rao
Amazon
nikhilrao86@gmail.com

Weicong Ding
Amazon
20008005dwc@gmail.com

ABSTRACT

The problem of predicting unobserved entries of a partially observed matrix has found wide applicability in several areas, such as recommender systems, computational biology, and computer vision. Many scalable methods with rigorous theoretical guarantees have been developed for algorithms where the matrix is factored into low-rank components, and embeddings are learned for the row and column entities. While there has been recent research on incorporating explicit side information in the low-rank matrix factorization setting, often implicit information can be gleaned from the data, via higher order interactions among entities. In this paper, we design a method to make use of this implicit information, via random walks on graphs. We show that the problem we intend to solve can be cast as factoring a nonlinear transform of the (partially) observed matrix and develop an efficient coordinate descent based algorithm for the same. Experiments on several datasets show that the method we propose outperforms vanilla matrix factorization, and also those methods that use explicitly available side information.

KEYWORDS

Matrix Factorization, Recommender Systems, Alternating Minimization

ACM Reference format:
Vatsal Shah, Nikhil Rao, and Weicong Ding. 2017. Matrix Factorization with Side and Higher Order Information. In *Proceedings of* 8 pages.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The problem of factorizing a matrix into low-rank components plays a vital role in many applications, such as recommender systems [15], computational biology [2] and computer vision [11]. Standard approaches to matrix factorization involves either explicitly looking for low-rank approximation via non-convex optimization [21, 29] or resorting to convex, nuclear norm based methods and solving it using semi-definite programs [4] or projected gradient methods that involve computing the SVD [3]. Overall, these methods can be viewed as learning dense and low dimensional vector *embeddings* of the row and column entities (for example, users and movies), the inner products of which best approximate entries of the target matrix.

Modern datasets also contain vast amounts of side information, incorporating which typically aids in prediction. For example, one might have access to user social networks, product co-purchasing graphs, gene interaction networks and so on. It is reasonable to posit that side information can be exploited to yield better predictions [5]. Recent work has focussed on developing new models that incorporate this side information [14, 31], and developing highly scalable algorithms for the same [20, 26]

While it seems natural to enforce the learned embeddings to be faithful to the side information present, there also exists implicit information in the data. For instance, two users consistently providing same ratings to the same set of movies *implicitly* suggests the similarity in their preferences irrespective of the explicit links that may or may not exist between them. Indeed, low rank representations inherently look to achieve this by assuming that users (in this case) all lie in a common low dimensional subspace. However, explicitly modeling such an implicit similarity tends to yield better results. Recent advances have demonstrated that exploiting second-order co-occurrences of users and items indeed results in better prediction [17] which further strengthen our theory that the more structured information we have about higher order transitions, the better will be the predictions.

In this paper, we develop a general framework, Higher Order Matrix Factorization (HOMF), that can incorporate both implicit relationships via higher order information and explicit side information present in the data in a natural way to enhance prediction power. To clarify this further, our algorithm is not dependent on the presence of explicit side information, but if present we can efficiently assimilate it to improve our performance. Our method scales gracefully to incorporate information from higher order co-occurrences (beyond 2). Also, we can efficiently incorporate pre-existing graph [1] side information on both row and column entities of the matrix. The ease of scaling, utilization of side information coupled with its strong performance give our algorithm a clear advantage over most existing recommendation algorithms.

The key idea of our approach is to model the target matrix *combined* with side information as a *graph* and then learn low-rank vector embeddings that best approximate the local neighborhood structure of the constructed graph. To be more explicit, consider a movie recommendation system. We construct a graph with all the users and movies as nodes. An (weighted, undirected) edge exists between a user node and movie node if the corresponding entry in the target matrix exists i.e. the user has rated the movie. Side information such as social networks amongst users, when present, are also used to form edges between user nodes. We aim to approximate the *multi-step transition probability*, i.e., the probability of jumping between two nodes within a small, predefined number of hops, via a function proportional to the inner product between

*Produces the permission block, and copyright information
†Work performed while all the authors were at Technicolor Research

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

[1] note that graphs can be constructed via user or item features.

the node embeddings. This objective naturally encodes both the implicit and side information: transition is more likely between users preferring a similar set of items; a dense clique in user social network also increase their probability of co-occurrence within a small number of steps. Noting that we consider co-occurrence within multiple steps, our embedding could account for higher order implicit similarities rather than the standard pair-wise similarity between users or movies, or just second order co-occurrences [17]. Our experiments reinforce our claims that taking higher-order information indeed improves the prediction accuracy on a variety of datasets.

HOMF is closely related to recent advances in representation learning in natural language processing [16, 18] and graphs [19, 23, 24]. [9] provides a comprehensive survey on various graph embedding techniques. While these works also consider co-occurrences of words within a small window in a sentence or nodes within a few steps on graphs, their objective is equivalent to factorizing the logarithm of such local co-occurrences [27]. In contrast, HOMF directly attempts to approximate the multi-step transition probabilities. Moreover, in this paper, we explore various methods to *construct* edge weights of our graph from both rating-like and binary target matrix. Also, unlike the negative sampling and stochastic gradient descent approach in [18, 19], we derive efficient coordinate descent based methods for solving the corresponding optimization problem. We verify that HOMF leads to better results on a number of different problem settings, with and without side information, feedback in standard star-rating or in binary form.

The rest of the paper is organized as follows: In the next section, we formally set up the problem we wish to solve and provide motivation for the algorithm we use. We also summarize related work in Section 2. In Section 3 we provide our algorithm, and comment on its computational complexity, as well as remark on its generality. We provide extensive experimental results in Section 4.1. In Section 5, we provide some theoretical considerations behind our approach and discuss possible future directions. We conclude the paper in Section 6.

2 PROBLEM SETUP

We assume we are given a partially observed target matrix $R \in \mathbb{R}^{m \times n}$. Let the set of observed entries in R be denoted by Ω, where typically $|\Omega| \ll mn$. Given a rank parameter k, the goal in standard Matrix Factorization (MF) is to learn k-dimensional embedding vectors corresponding to the row and column indices of the matrix R. The standard matrix factorization algorithm aims to solve a problem of the form[2]:

$$\hat{U}, \hat{V} = \arg\min_{U,V} \frac{1}{2}\|P_\Omega(R - UV^T)\|_F^2 + \lambda \left(\|U\|_F^2 + \|V\|_F^2\right) \quad (1)$$

where $P_\Omega(\cdot)$ is the projection operator that retains those entries of the matrix that lie in Ω, $U \in \mathbb{R}^{m \times k}$ and $V \in \mathbb{R}^{n \times k}$.

Incorporating Side Information : Following recent works by [20, 31], it is possible to model the relationships between row (and column) entities via graphs. For example, in the case of a recommender system there might exist a graph G_r among users, such as a social network, and a product co-purchasing graph G_c among items. Therefore, it is reasonable to encourage users belonging to the same social community or products often co-purchased together to have similar embeddings. Current state-of-the-art techniques proposed to solve the MF problem with side information encourage the embeddings of the row and column entities to be faithful with respect to the eigenspace of the corresponding Laplacians:

$$\hat{U}, \hat{V} = \arg\min_{U,V} \frac{1}{2}\|P_\Omega(R - UV^T)\|_F^2 + \lambda \left(\mathrm{Tr}(U^T L_r U) + \mathrm{Tr}(V^T L_c V)\right) \quad (2)$$

where L_r and L_c are the graph Laplacians corresponding to G_r and G_c respectively.

Our approach- HOMF: In this paper, we look for a unified approach that can make use of (1) explicit side information provided in the form of graphs and (2) implicit information inherent in the data.

To this end, we propose constructing a (weighted, undirected) graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ with $\mathcal{V}$ containing all the m row entities and n column entities. The edges in the graph are constructed as follows:

- If two row entities i, j are connected in G_r, we form an edge $e_{ij} \in \mathcal{E}$ with $e_{ij} = g_1(G_r(i,j))$. Here $g_1(.)$ is some non-negative, monotonic function of the edge weight in G_r. The same procedure is repeated for column entities that are connected in G_c.
- If i is a row node (e.g., user) and j is a column node (e.g., a movie), we form an edge $e_{ij} = g_2(R(i,j))$ to encode the interactions observed in the data matrix R. $g_2(.)$ should also be a non-negative, monotonic function, and can potentially be the same as $g_1(.)$.
- We then scale the side-information edges with some weight parameter $\alpha \in (0, 1)$. The data-matrix edges are scaled by $1 - \alpha$.

As a result of our graph construction, if a user rates a movie highly, then there will be a large edge weight between that user and movie. Similarly, if two movies are connected to each other via G_c, then again we expect the edge weight to be large. Typical choices for $g_1(.), g_2(.)$ include:

- **Exponential :** $g(x) = \exp(x)$
- **Linear :** $g(x) = c \cdot x, \quad$ for some $c \in \mathbb{R}$
- **Step :** $g(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{else} \end{cases}$

Appropriately arranging the nodes, we get the matrix G which is defined as follows:

$$G = \begin{bmatrix} \alpha g_1(G_r) & (1-\alpha) g_2(R) \\ (1-\alpha) g_2(R)^T & \alpha g_3(G_c) \end{bmatrix} \quad (3)$$

where with some abuse of notation, the functions $g_i(.)$ act elementwise on the arguments (i.e. matrices). We denote by A the *row-normalized* version of this matrix G, so that each row of A sums up to 1. A is thus a transition probability matrix (TPM), with A_{ij} indicating the probability that a random walk starting at node i jumps to node j in one step.

Let

$$f_T(A) := \frac{A + A^2 + \ldots + A^T}{T}$$

[2] bias variables are typically included, but we omit them from the text here for ease of explanation.

for some positive integer T. Let Ω_T be the set of non-zero entries of $f_T(A)$. A simple example of how to create A from R is presented in the appendix.

From a probabilistic point of view, the ij^{th} entry of A^l is the probability of jumping from node i to j in l random steps on $\mathcal{G}$. Thus, $[f_T(A)]_{ij}$ is the probability of jumping from i to j *at least once* in T steps in a random walk on $\mathcal{G}$.
Our objective in this paper is to factorize:

$$\hat{U}, \hat{V} = \arg\min_{U,V} \frac{1}{2}\|P_{\Omega_T}(f_T(A) - UV^T)\|_F^2 + \lambda\left(\|U\|_F^2 + \|V\|_F^2\right) \quad (4)$$

where $U, V \in \mathcal{R}^{(m+n)\times k}$ and $f_T(A)$ encodes higher-order hidden information in the data. For example, two users rating the same set of movies have a higher probability of jumping between each other in a few steps on $\mathcal{G}$. In other words, if two users have rated movies in a similar fashion, then their future interactions will tend to be more alike than not. Similarly, two users with common friends in the side social network also have a higher transition probability to land on the other within a small number of jumps. Letting $\hat{u}_i$ be the i^{th} row of $\hat{U}$, we use $\hat{u}_i\hat{v}_j^T$ as a proxy for the predicted value of the corresponding entry in the target matrix.

We note that the design parameter α controls the overall weights of the side information, and the number of steps T determines how "local" our search space in the graph will be. Overall, (4) encourages nodes with higher co-occurrence probability within a small number of steps to have similar vector representations. The adverse effects of noisy/missing side information can be reduced by appropriately tuning α. When there is no side information, the algorithm forces α to be 0 and ignores the side information matrix. Further motivation for factorizing $f_T(A)$ is provided in Section 5.

2.1 Related Work

While several methods for vanilla matrix factorization (1) have been proposed, incorporating explicit side information is a much more recent phenomenon. Given attributes or features for the row and/or column entities, [13, 26] consider the row and column factors to be a linear combination of the features. This was extended to nonlinear combinations in [22]. If graph information is known beforehand, then methods proposed in [20, 31] can be applied to solve (2). Further, note that one can construct the pairwise relationship between entities from the feature representations. Methods that can implicitly glean relationships between row and/or column entities have not been that forthcoming, an exception being [17].

[8, 25] and references therein view collaborative filtering as functions of random walks on graphs. Indeed, the canonical user-item rating matrix can be seen as a bipartite graph, and several highly scalable methods have been proposed that take this viewpoint [30]. Methods that incorporate existing graph information in this context have also been studied [7, 14]. [6] consider metrics such as average commute times on random walks to automatically figure out the similarity between nodes and apply it to recommender systems, but also note that such methods do not always yield the best results. Similarly, [1] consider local random walks on a user-item graph, and resorts to a PageRank-style algorithm. Furthermore, they require the graph to have a high average degree, something most applications we consider will not have.

Recently, ideas from learning vector representations of words [18] have been used to obtain vector embeddings for nodes in a graph. Specifically, the authors in [19] considered random walks on a graph as equivalent to "sentences" in a text corpus, and applied the standard Word2Vec method on the resulting dataset. [28] showed that such a method is equivalent to factorizing a matrix polynomial after logarithmic transformations. We instead consider factorizing the function $f_T(\cdot)$, which has the interpretation detailed above.

3 OUR PROPOSED ALGORITHM

We now describe the algorithm to solve (4). Note that once the matrix $f_T(A)$ is formed, the problem reduces to standard matrix factorization, for which highly efficient methods exist. However, obtaining $f_T(A)$ is expensive both from a computational and memory point of view. Indeed, regardless of the sparsity of A, A^l for even small l, (say $l \geq 3$) will not be sparse. Figure 1 displays this phenomenon, where we created a random 1000×1000 matrix R, randomly select the observed set Ω with different sparsity levels, and constructed a block diagonal A (eqn. (3)), without side information graphs ($G_r = G_c = 0$). [3]. Even for 1% sparse Ω, the multi-step transition A^T quickly become dense.

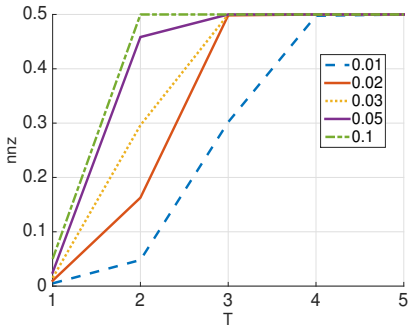

Figure 1: Sparsity of A^T for different power T and sparsity level of observed set Ω for rating matrix R. The sparsity level is defined as the ratio of the size of Ω (i.e., the non-zero entries (nnz)) to the overall size of R. Note that even for 1% sampling, the matrix saturates in terms of sparsity. (best seen in color)

Given the size of modern datasets, storing such A^l will be prohibitive, let alone storing $f_T(A)$. One thus needs an efficient method to compute higher matrix powers. We will see in the sequel that we in fact need not compute $f_T(A)$, but only specific rows or columns of the matrix, which can be done efficiently.

We propose a coordinate descent method to alternatively update U and V. Let $u_i^{(t)}$ (respectively $v_i^{(t)}$) be the i^{th} row (respectively column) of U (respectively V) at iteration t. We describe our method to update V keeping U fixed. The updates for U with V fixed will be analogous. At iteration t, the update equation for V keeping the other entities fixed is as follows:

$$V^{(t)} = \arg\min_V \|P_{\Omega_T}(f_T(A) - U^{(t-1)}V^T)\|_F^2 + \lambda\|V\|_F^2 \quad (5)$$

[3] In this case, the maximal sparsity of A^l, $\forall l$ can be 0.5

Algorithm 1 Recursive method for column sampling

Input: matrix A, column index i, steps T
Initialize: $a_1 = A_i, \quad t = 1$
for $t = 2, 3, \cdots, T$ **do**
  $a_t = a_1 + Aa_{t-1}$
end for
Return: $f_T(A)_{\Omega_T^i} = \frac{1}{T} a_T$

Algorithm 2 Procedure for solving (5)

Input: matrix A, steps T, fixed U, regularizer λ
for $i = 1, 2, \cdots, M$ **in parallel do**
  Column sample $x := f_T(A))_{\Omega_T^i}$ using Algorithm 1
  Solve (6) with x, U, λ as inputs using conjugate gradients
  Set v_i to be solution of above problem
end for
Return: V

This ridge regression problem can be solved column-wise without storing the $f_T(A)$ matrix. The key observation is that each column of $f_T(A)$ can be calculated efficiently as:

$$f_T(A)\mathbf{e}_i = \frac{1}{T} A(\mathbf{e}_i + A(\mathbf{e}_i + A(\dots)\dots))$$

where $\mathbf{e}_i$ is the i^{th} canonical basis vector. This computation requires only the storage of sparse matrix A and can be executed on-the-fly by T sparse matrix-vector multiplications. We outline the recursive details in Algorithm 1. An analogous procedure to the one mentioned in Algorithm 1 can be applied to obtain the row wise updates. The updates for row-wise and column-wise optimization strategy lead us to solve the following problem:

$$v_i(t) = \arg\min_v \frac{1}{2} \|(f_T(A))_{\Omega_T^i} - Uv\|_2^2 + \lambda \|v\|_2^2 \tag{6}$$

where Ω_T^i is the set of nonzeros of the i^{th} column of $f_T(A)$. We note that Eq. (6) can be solved using standard regularized least squares solvers, for example the conjugate gradient method as we adopted in this paper. Moreover, the updates for different columns can be parallelized. The empirical speedup via this parallelization is investigated later in the experiment section. Algorithm 2 details the pseudocode for solving (5). A row-wise analogous version of Algorithm 2 can be used to update $u_i(.)$.

We briefly describe the computational complexity of our method. Each matrix-vector multiplication in Algorithm 1 can be performed in $O(nnz(A)(m+n))$ time. For a given T, the complexity for column sampling is then $O(nnz(A)(m+n)T)$. The main bottleneck in the conjugate gradients procedure to solve (6) is the Hessian-vector multiplication. The Hessian of (6) is $\left(U^T U + \lambda I\right) \in \mathbb{R}^{k \times k}$, and hence the complexity of the conjugate gradient method is $O(k^2(m+n))$. The per-iteration complexity of Algorithm 2 is thus $O((nnz(A)T + k^2)(m+n))$, since the per-column update can be parallelized. Typically k is a small number, so $k^2 \ll nnz(A)$, and hence the complexity of the method is essentially linear in m, n, T and the size of the data, including graph side information, since $nnz(A) = 2nnz(R) + nnz(G_r) + nnz(G_c)$.

3.1 Generality of HOMF

We see that HOMF is highly general, and can efficiently incorporate pre-existing side information via the parameter α. Indeed, the side information graphs can also yield interesting higher order interaction information, as we will see from our experiments. When side information is not present, we can obtain higher order information from only existing data. Furthermore, when $\alpha = 0, T = 1$ effectively ignores higher order information, and is conceptually similar to the standard MF routine. $T = 2$ in the same vein is similar to the co-factor method in [17].

4 EXPERIMENTS

4.1 Computational Complexity

In this section, we test our method against standard matrix factorization and other methods that use graph side information. We have attempted to use a variety of datasets, some with additional graph containing side information, and some without to demonstrate the universality of our algorithm. We also consider a binary dataset with graphs. Details about the datasets are provided in Table 1. MovieLens 1 million (ML-1M) is a standard movie recommendation dataset [4] [12]. The FilmTrust dataset [10] has a similar task as that of ML1m, but also has user-user network given [5]. The Gene dataset is a binary matrix, where the task is to determine what genes are useful for predicting the occurrence of various diseases [6]. There also exist gene-gene interaction network data and a disease co-morbidity graph. For ML-1M and FilmTrust, we generate a random 80/20 Train-Test split of the data. For Gene, we use the split provided online. Since this is a one-class classification problem, we randomly sampled the same number of samples as in the data, to act as the negative class for classification.

4.2 Evaluation Metrics

For the Gene dataset with binary observations, we computed the standard AUC score on the test set. For non-binary data (ML-1M and FilmTrust), we compute Precision, Recall, Mean Average Precision (MAP), and Normalized Discounted Cumulative Gain (NDCG), all @ various thresholds (K). Note that since we are factorizing a nonlinear transformation of the ratings matrix R, the RMSE will not be a useful metric to compute. For the ML-1M dataset, we determine that a movie rated by a user is a true positive if the corresponding rating is 5 while $K = \{5, 10\}$. For the FilmTrust dataset, we determine that a movie is a true positive if the corresponding rating is at least 3 and corresponding values of K are set to 1 and 2 since the dataset is small and it is hard to find many highly rated movies for most users. The K and threshold values are determind by the average number of highly rated movies for each user which is consistent with most available literaute. For the sake of completeness and to prevent confusion, we provide explicit formulas for the metrics used. Let $i_1, \dots, i_{n_u}$ be items in test set rated by user u *sorted* by the predicted score. Let $\mathcal{I}(u, i) = 1$ if user u rated item i as relevant in the ground-truth test data and 0 otherwise. For simplicity, let $\mathcal{I}_u = \sum_{\ell=1}^{n_u} \mathcal{I}(u, i_\ell)$ be the total number of relevant items per user.

[4] https://grouplens.org/datasets/movielens/1m/
[5] http://www.librec.net/datasets.html
[6] http://bigdata.ices.utexas.edu/project/gene-disease/

Table 1: Descriptions about the datasets used in our experiments

Dataset	Type	# rows	# columns	# entries	Graph present?	# links
ML-1M	0-5	6040	3952	1000208	no	-
FilmTrust	0-5	1508	2071	35497	rows	1632
Gene	Binary	1071	1150	2908	rows and columns	7424 (rows)
						437239 (columns)

We first calculate for each user u:

$$\text{Precision@}(K,u) = \sum_{j=1}^{K} \mathcal{I}(u,i_j)/K$$

$$\text{Recall@}(K,u) = \sum_{j=1}^{K} \mathcal{I}(u,i_j)/\mathcal{I}_u$$

$$\text{AP@}(K,u) = \sum_{j=1}^{K} \text{Precision@}(j,u)/\min(\mathcal{I}_u, K)$$

and then average across all the users to get Precision@K, Recall@K, and MAP@K. For NDCG, we first calculate

$$\text{DCG@}(K,u) = \sum_{j=1}^{K} \frac{2^{\mathcal{I}(u,i_j)} - 1}{\log(i+1)}$$

and IDCG@(K,u) based on the ordered ground truth ranking of all the items. We obtain NDCG@(K,u) = DCG@(K,u)/IDCG@(K,u) and average it across all users.

4.3 Baselines, Parameters and Initialization

We fixed the rank (k) for all methods to be 10. For comparing against standard matrix completion and the graph based counterparts (equations (1) and (2)), we used the GRALS method [20][7]. For FilmTrust dataset with side information graph, we also compared against the TrustWalker algorithm [14] which makes recommendations using random walks on graphs.

We implemented our method in python using the multiprocessing framework to parallelize the method in Algorithm 2.[8] Wherever applicable, we varied $\lambda \in \{10^{-4}, 10^{-3}, \ldots, 10^{2}\}$ and cross-validate λ on a fixed validation set (20% of the training data). When the graph side information is available, we also varied $\alpha \in \{0.15, 0.25, 0.5, 0.75\}$. We consider random walks of lengths $\{2, 4, 6, \ldots T_{max}\}$ for all datasets. For ML-1M, FilmTrust and Gene, T_{max} was fixed at 8, 6 and 6 respectively. We initialized the factor matrices U, V such that each entry is an independent uniform $[0, 1]$ random variable.

4.4 Results

Table 2 summarizes the results obtained on the test set in ML-1M and FilmTrust. We see that HOMF consistently outperforms standard matrix factorization, and also the version that uses graph side information. In some cases, the performance gap is significant.

In Table 3, we show the results obtained on the Gene-Disease dataset. In this setting, with rich graph information encoding known relationships between the entities, HOMF significantly outperforms graph based matrix completion. This suggests that there are hidden, higher order interactions present in the data, which the given graphs do not fully capture by themselves. This opens an interesting avenue for further research, especially in domains such as computational biology where obtaining data is hard, and hence hidden information in the data is even more valuable.

4.5 Effect of T, $g_i(.)$ and α

We now study the computational and statistical effect of the parameter T (Figs. 2, 3), the length of walks considered, in HOMF, as well as the weighting parameter α (Fig. 3) when the side information graphs are present. We also consider both the exp(.) and *linear* functions for $g_1(.), g_2(.)$

Figure 2 shows the effect of varying the walk length T on the runtime of the algorithm, on the ML-1M dataset. As expected, the time increases with T. Importantly, note that the time increases nearly linearly with T (with a very small slope), as indicated by the complexity argument in Section 4.1. However, it also has to be noted that the time difference is not much: approximately 15 seconds extra for the method when $T = 8$, compared to $T = 2$ seems to suggest that the method is scalable. From a theoretical perspective, as discussed in Section 5, even though it may not be useful to take very long walks ($T > 20$), it is impressive to note that long walk lengths do not hamper computational performance.

Next, we show that the walk length should be treated as a (hyper)parameter of our method, and should be tuned via cross validation. Figure 3 ((a)-(d)) shows the effect of varying T for the ML-1M dataset. We see that bigger is not necessarily better when it comes to T but for higher orders ($T > 2$), we usually get superior performance in terms of the evaluation metrics.

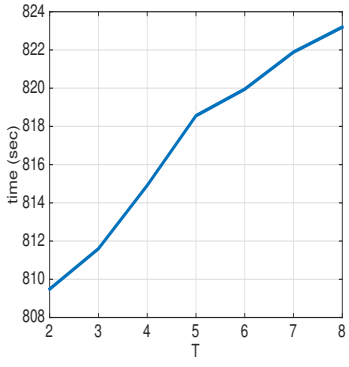

Figure 2: Time taken as a function of walk length

[7] code available online
[8] Our code will be made public after the reviewing procedure.

Table 2: Comparison of various algorithms using Top-N evaluation metrics. HOMF performs comparably to standard matrix factorization with or without side information, and often beats it. Bold values indicate the best result among the methods considered. The exponential activation function (i.e. $g(.)$) was used to obtain the following results.

Data	Algos	Precision		Recall		MAP		NDCG	
		@5	@10	@5	@10	@5	@10	@5	@10
ML-1M	MF	0.316	0.311	**0.598**	0.614	0.462	0.474	0.692	0.707
	HOMF	**0.370**	**0.331**	0.544	**0.662**	**0.499**	**0.529**	**0.744**	**0.749**
		@1	@2	@1	@2	@1	@2	@1	@2
FilmTrust	MF	0.701	0.633	0.345	0.436	0.795	0.744	0.761	0.747
	TrustWalker	0.506	0.497	0.316	0.456	0.598	0.607	0.584	0.568
	GRALS	0.752	0.740	0.365	0.492	0.812	0.801	0.772	0.770
	HOMF	**0.754**	**0.745**	**0.375**	**0.502**	**0.816**	**0.802**	**0.778**	**0.773**

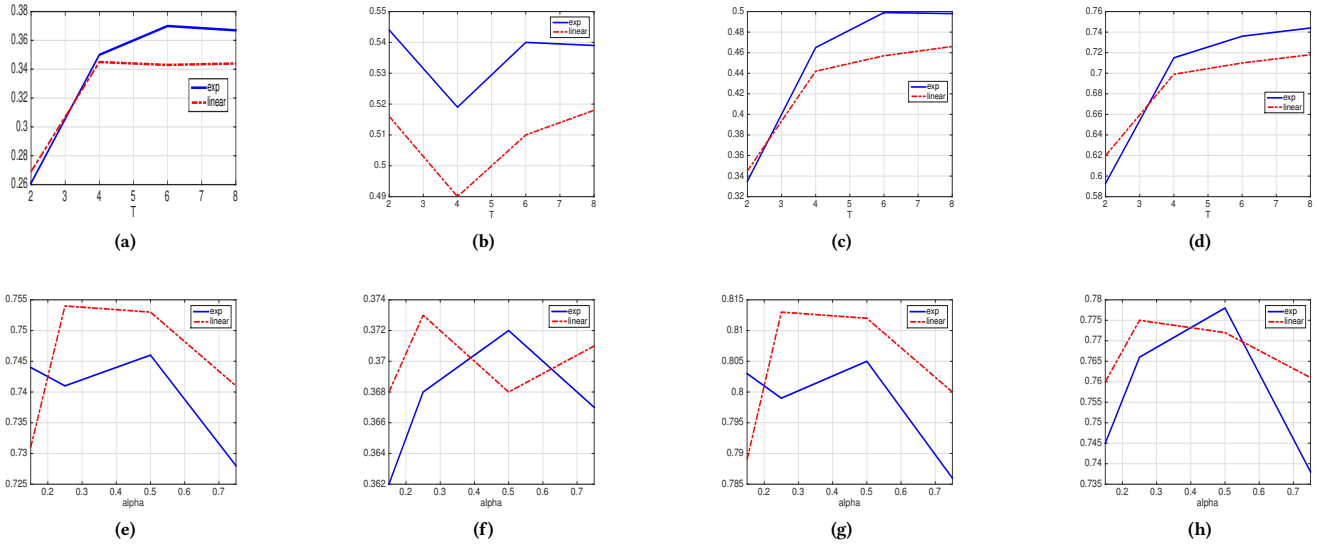

Figure 3: Performance of HOMF on the ML-1M (top) and FilmTrust (bottom) datasets, as T (top) and α (bottom) is varied. [From left to right]: Precision, Recall, MAP and NDCG, for exponential and linear activation functions. We also see that sometimes the exp() function is better and sometimes it is worse. (best seen in color)

Table 3: Comparison of various algorithms for Gene dataset

Data	Method	AUC
Gene	MF	0.546
	GRALS	0.572
	HOMF(exp)	0.623
	HOMF(linear)	**0.630**

We also studied the effect of varying α on the FilmTrust dataset, where we fixed $T = 4$. Figure 3 (e)-(h) again shows that different values of α yield different results, though they are fairly uniform.

4.6 Speedup via Parallelization

Finally, we see from Figure 4 that we obtain near-linear speedups via the parallel procedure in Algorithm 2. We used the ML-1M dataset here and varied the number of processors over which we parallelized the updates. When the number of processors is increased to about 20, we see that the corresponding speedup is nearly 17. For a given number of processors N, we define the speedup to be

$$speedup(N) = \frac{\text{Time taken with 1 processor}}{\text{Time taken with N processors}}$$

5 DISCUSSION AND FUTURE WORK

Our motivation behind transforming the original rating matrix into a Transition Probability Matrix is to find higher order information from the graph. A^l will encode the l^{th} order information. Our aim is to assimilate as much (non-redundant) higher order information as possible. Another key point is that our approach is highly general, and by setting $T = 1$ we can obtain similar performance to standard matrix factorization.

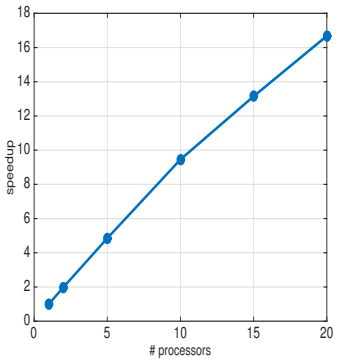

Figure 4: Speedup obtained as a function of # processors

At large l, the matrix A^l might not capture too much information. Clearly, for large l, one might inherently force two nodes very far in the graph to have a similar representation. At low values of l, one can miss out on the implicit higher order relationships that exist between nearby nodes of the graph. Hence, it is necessary to be prudent while choosing l.

Future Work: The development of a fully distributed method remains an open challenge. Given the matrix A, we can solve the problem in a parallel fashion on a multicore machine as we have demonstrated in this paper. However, for extremely large datasets, the matrix A might not fit into memory. When A itself is distributed, computing higher order powers can be a challenging task, since Algorithm 1 requires access to the full matrix A.

From a theoretical perspective, we would like to address the problem posed in Equation 4 and obtain convergence guarantees for the low rank matrix factorization for functions of matrix A without explicitly computing the function. As of now, we are unaware of any literature pertaining to matrix factorization of higher orders of A, say A^p let alone $f_p(A)$ for general matrices A. Another interesting avenue is to compute the minimum number of samples required from A to recover A^p as accurately as possible.

6 CONCLUSION

In this paper, we presented HOMF, a unified method for matrix factorization that can take advantage of both explicit and implicit side information in the data. We developed a scalable method for solving the resulting optimization problem and presented results on several datasets. Our method is one of the first to include both existing side information and implicit higher order information and yields superior results with respect to various evaluation metrics when compared to other methods that only use explicit side information, or no information at all.

REFERENCES

[1] Zeinab Abbassi and Vahab S Mirrokni. 2007. 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*. ACM, 102–108.

[2] Ashis Kumer Biswas, Mingon Kang, Dong-Chul Kim, Chris HQ Ding, Baoju Zhang, Xiaoyong Wu, and Jean X Gao. 2015. Inferring disease associations of the long non-coding RNAs through non-negative matrix factorization. *Network Modeling Analysis in Health Informatics and Bioinformatics* 4, 1 (2015), 1–17.

[3] Jian-Feng Cai, Emmanuel J Candès, and Zuowei Shen. 2010. A singular value thresholding algorithm for matrix completion. *SIAM Journal on Optimization* 20, 4 (2010), 1956–1982.

[4] Emmanuel J Candès and Benjamin Recht. 2009. Exact matrix completion via convex optimization. *Foundations of Computational mathematics* 9, 6 (2009), 717–772.

[5] David Crandall, Dan Cosley, Daniel Huttenlocher, Jon Kleinberg, and Siddharth Suri. 2008. Feedback effects between similarity and social influence in online communities. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 160–168.

[6] Francois Fouss, Alain Pirotte, Jean-Michel Renders, and Marco Saerens. 2007. Random-walk computation of similarities between nodes of a graph with application to collaborative recommendation. *IEEE Transactions on knowledge and data engineering* 19, 3 (2007), 355–369.

[7] Jennifer Ann Golbeck. 2005. Computing and applying trust in web-based social networks. *ACM Digital Library* (2005).

[8] Marco Gori, Augusto Pucci, V Roma, and I Siena. 2007. ItemRank: A Random-Walk Based Scoring Algorithm for Recommender Engines.. In *IJCAI*, Vol. 7. 2766–2771.

[9] Palash Goyal and Emilio Ferrara. 2017. Graph Embedding Techniques, Applications, and Performance: A Survey. *arXiv preprint arXiv:1705.02801* (2017).

[10] G. Guo, J. Zhang, and N. Yorke-Smith. 2013. A Novel Bayesian Similarity Measure for Recommender Systems. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI)*. 2619–2625.

[11] Benjamin Haeffele, Eric Young, and Rene Vidal. 2014. Structured low-rank matrix factorization: Optimality, algorithm, and applications to image processing. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*. 2007–2015.

[12] F Maxwell Harper and Joseph A Konstan. 2016. The movielens datasets: History and context. *ACM Transactions on Interactive Intelligent Systems (TiiS)* 5, 4 (2016), 19.

[13] Prateek Jain and Inderjit S Dhillon. 2013. Provable inductive matrix completion. *arXiv preprint arXiv:1306.0626* (2013).

[14] Mohsen Jamali and Martin Ester. 2009. 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*. ACM, 397–406.

[15] Yehuda Koren, Robert Bell, Chris Volinsky, et al. 2009. Matrix factorization techniques for recommender systems. *Computer* 42, 8 (2009), 30–37.

[16] Omer Levy and Yoav Goldberg. 2014. Neural word embedding as implicit matrix factorization. In *Advances in neural information processing systems*. 2177–2185.

[17] Dawen Liang, Jaan Altosaar, Laurent Charlin, and David M. Blei. 2016. Factorization Meets the Item Embedding: Regularizing Matrix Factorization with Item Co-occurrence. In *Proceedings of the 10th ACM Conference on Recommender Systems (RecSys '16)*. 59–66.

[18] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*. 3111–3119.

[19] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 701–710.

[20] Nikhil Rao, Hsiang-Fu Yu, Pradeep K Ravikumar, and Inderjit S Dhillon. 2015. Collaborative filtering with graph information: Consistency and scalable methods. In *Advances in Neural Information Processing Systems*. 2107–2115.

[21] Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. 2011. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in Neural Information Processing Systems*. 693–701.

[22] Si Si, Kai-Y Chiang, Cho-Jui Hsieh, Nikhil Rao, and Inderjit S Dhillon. 2016. Goal directed inductive matrix completion. *KDD* (2016).

[23] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. Line: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web*. 1067–1077.

[24] Cunchao Tu, Weicheng Zhang, Zhiyuan Liu, and Maosong Sun. 2016. Max-margin DeepWalk: discriminative learning of network representation. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI 2016)*. 3889–3895.

[25] Wenlei Xie, David Bindel, Alan Demers, and Johannes Gehrke. 2015. 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*. ACM, 1325–1334.

[26] Miao Xu, Rong Jin, and Zhi-Hua Zhou. 2013. Speedup matrix completion with side information: Application to multi-label learning. In *Advances in Neural Information Processing Systems*. 2301–2309.

[27] Cheng Yang and Zhiyuan Liu. 2015. Comprehend deepwalk as matrix factorization. *arXiv preprint arXiv:1501.00358* (2015).

[28] Cheng Yang, Zhiyuan Liu, Deli Zhao, Maosong Sun, and Edward Y Chang. 2015. Network representation learning with rich text information. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence, Buenos Aires, Argentina*. 2111–2117.

[29] Hsiang-Fu Yu, Cho-Jui Hsieh, Si Si, and Inderjit Dhillon. 2012. Scalable coordinate descent approaches to parallel matrix factorization for recommender systems. In *2012 IEEE 12th International Conference on Data Mining*. IEEE, 765–774.

[30] Hyokun Yun, Hsiang-Fu Yu, Cho-Jui Hsieh, SVN Vishwanathan, and Inderjit Dhillon. 2014. NOMAD: Non-locking, stOchastic Multi-machine algorithm for Asynchronous and Decentralized matrix completion. *Proceedings of the VLDB Endowment* 7, 11 (2014), 975–986.

[31] Tinghui Zhou, Hanhuai Shan, Arindam Banerjee, and Guillermo Sapiro. 2012. Kernelized Probabilistic Matrix Factorization: Exploiting Graphs and Side Information.. In *SDM*, Vol. 12. SIAM, 403–414.