Fast Non-negative Matrix Factorization under KL-Divergence: A Case for Link Prediction

Payam Siyari, and Hamid R. Rabiee, *Senior Member, IEEE,*

***Abstract*—In this paper, we study the link prediction problem, a fundamental issue that arises in many applications such as recommender systems, social and biological networks. To this end, we cast link prediction as a variant of Non-negative Matrix Factorization (NMF) problem. Specifically, we take a probabilistic approach by assigning a Poisson generative model to the underlying network and utilize maximum likelihood in order to infer the model parameters. We show that maximum likelihood under Poisson modeling, which is equivalent to non-negative matrix factorization problem under generalized KL-divergence, can be efficiently performed in a time linear to the number of nodes on a sparse network. Experiments on real-world networks indicate that under the proposed method, link prediction can be done 15 times faster than previous state-of-the-art methods. We also show that the proposed method can be effectively applied to other applications of NMF by developing a greedy optimization method for minimizing the KL-divergence. In addition, we explore the speed-up in the parallelized version of our greedy NMF method. We test the performance of our greedy and parallel methods on two well-known datasets. The results show that our method is capable of up to 6 times faster minimization of the KL-divergence than the recently proposed methods.**

***Index Terms*—Link Prediction, Non-negative Matrix Factorization, KL-Divergence, Cyclic Block Coordinate Descent, Greedy Coordinate Selection, Parallel Optimization.**

I. Introduction

Ubiquity of networked systems, from social interactions to biological networks, has attracted many researchers from various disciplines during the last decade. Consequently, depending on the application, different challenges may arise in the analysis of networks. Link prediction is one of the fundamental problems which appears in the study of all kinds of networks and has important applications in recommender systems, social and biological networks.

In this paper, we focus on the study of the problem of structural link prediction [1] in which a subgraph consisting of the vertices and a subset of the edges of a network is observed and the goal is to predict the existence of the remaining unobserved edges. Link prediction can also be analyzed in a temporal manner in which full snapshots of a network are given in a number of timestamps and the goal is to predict the absence or presence of the links in the upcoming timestamps.

Scalability is one of the most important challenges in link prediction in most of the real-world large-scale networks. However, sparsity (smaller number of existing links than the number of all possible links) is the other important characteristic of these networks and in this paper, we are aiming to utilize this property in order to achieve a significant speedup.

We present our method in the framework of latent factor models in which the goal is to infer a set of latent factors for the entities (e.g. nodes in a network) from their observed connectivity patterns and simply infer the unobserved links by the similarity of these factors. Matrix Factorization (MF) is a recently emerged dimensionality reduction paradigm for this purpose [2]. In MF, we seek to find an approximation for an input matrix which minimizes a particular loss function. Basically, this approximation is considered to be the product of two rank-k matrices, each representing the latent factors of a set of entities in our system. For example, in a bipartite user-item network where links represent the existence of a rating from a user to a particular item, the rating matrix is approximated by the product of the users' and items' latent factors. In other words, we are looking for latent user and item matrices U^* and V^* for which:

$$(U^*, V^*) = \underset{(U,V)}{arg\ min}\ \ l(R, L(U^T V)) \tag{1}$$

where R, $l(.,.)$ and $L(.)$ are rating matrix, proper loss and link functions, respectively. The most common constraint on factor matrices is the non-negativity of U and V. In this sense, we call (1) a Non-negative Matrix Factorization (NMF) problem.

In this paper, we adopt a probabilistic view of the NMF [3]. To be more specific, by assigning a Poisson generative model as the likelihood for the network graph, we aim to infer the latent factors in this model using Maximum Likelihood (ML) which leads to minimization of KL-divergence as the loss function in Eq. (1). We apply coordinate descent for inference of the latent factors. It is shown that the use of Poisson generative model, instead of other well-known models like Bernoulli-Logistic, allows us to use the sparsity in the network structure in order to boost the inference of model parameters. Consequently, to predict the missing links in a time linear to the number of the network nodes. Evaluations of the proposed algorithm on large real-world datasets demonstrate up to 15 times faster convergence than previous methods.

It is important to note that with straightforward modifications, our algorithm can be used in other areas involving NMF. To this end, we present a greedy version of our proposed method that is suitable for other applications of NMF such as text mining. Experimental results shows significant improvements in speed and scalability of inference. Further, we present a possible extension for the parallelization of the optimization procedure.

P. Siyari and H. R. Rabiee are with the Department of Computer Engineering, Sharif University of Technology (SUT).
e-mail: siyari@ce.sharif.edu, rabiee@sharif.edu
Manuscript received MMM DD, YYYY; revised MMM DD, YYYY.

Overall, the main contributions of this work are:

- Presenting a scalable NMF approach for link prediction by exploiting Poisson generative modeling and the sparse structure of the networks.
- Extending the proposed method for general NMF problems under KL-divergence by presenting greedy and parallel optimization schemes for minimization of KL-divergence.

The rest of the paper is organized as follows. In Section II, we cover the related work specifically those which propose link prediction in latent factor models. Section III presents the proposed method for link prediction and Section IV demonstrates its experimental evaluations. Section V discusses the greedy extension to the proposed method in other NMF applications. In Section V, we also point out the parallel implementation of the proposed method and discuss the issues related to greedy and parallel optimization in the case of link prediction. Finally, Section VI concludes the paper and mentions future directions.

II. Related Work

As a dimensionality reduction method, NMF has drawn a considerable attention during the last decade. Comprehensive surveys of matrix factorization models and methods are presented in [2], [4], [5].

In this section, we review the related work in link prediction with emphasis on those exploiting the latent factor modeling and matrix factorization framework. We also present a quick survey of related problems to link prediction in network analysis.

A. Link Prediction

In general, network graph reconstruction from incomplete data has been widely studied in several fields under many different titles, amongst which link prediction is the most common. As mentioned, this problem has been mainly studied under two scenarios [1]. Structural link prediction is the case where we have the adjacency matrix of an observed network and aim to predict its unobserved links. The other is temporal link prediction where we face a set of full snapshots from the graph and we are looking for the future ones. Despite being different, the methods presented in each of these two categories can be adapted to be used in the other. A simple example is the case where only a single snapshot of the graph is available.

Link prediction can be viewed as a binary classification problem in which presence or absence of links are treated as labels to pairs of nodes. Hence, the methods for link prediction are either unsupervised or supervised. In unsupervised methods, a precomputed similarity score (e.g., number of common neighbors) between unknown pairs is used to predict the existence of links between those pairs. Therefore, these methods do not involve any learning [1]. One of the earliest studies in link prediction is [6] where an array of unsupervised scores are suggested. Many of the unsupervised scores have the advantage of quick calculation for the large graphs. But the varying performance of these scores makes them unable to be used as a reliable method in different datasets.

All other methods for link prediction are supervised methods. See [1] for a comprehensive categorization of these methods. Our proposed method lies in latent feature models category where the goal is to map the nodes to a latent feature space of some dimensionality and to estimate the presence or absence of the links using the inferred features from training data. Methods presented in [1], [7], [8], [9] are based on latent feature models.

In [1], the focus is on optimizing AUC measure (Area under ROC curve) because of high class imbalance between links and non-links. The method is presented in matrix factorization framework and is evaluated under different scenarios, e.g. with and without side information. However, the method does not seem to be scalable to very large graphs as in this method, it is needed to scan over all pairs of links and non-links. This leads to a space and time complexity of $O(|V|^3)$ (where V is the set of nodes) which makes the prediction infeasible for a large dataset. Although data reduction and stochastic gradient descent is employed in order to alleviate this problem, there can be inaccurate and very low-speed convergence in large graphs using these techniques.

[7], [8] propose similar probabilistic approaches for link prediction by considering a Bernoulli-Logistic generative model for the underlying graph. They use Minorization-Maximization (MM) algorithm [10] to infer the parameters. In later sections, it is shown that Bernoulli trial modeling may not scale well to large datasets. Also, we will mention some of the issues regarding the optimization procedure of [8] in experimental evaluations section. In this paper, we show that changing the generative model to a Poisson process leads to scalable inference of parameter models. [11] tries to handle the case of Bernoulli trial and logistic regression over the binary adjacency matrices by matrix factorization. However, the training part uses Markov Chain Monte Carlo (MCMC) techniques which is not efficiently applicable in large-scale problems.

One of the important parameters in latent feature models is the latent space dimension. In the meantime, setting an optimal value for the latent space dimension appears to be a time consuming task. Many papers like [12] try to present non-parametric approaches where the latent space dimension is inferred automatically from the data.

Similar to latent feature models, another related modeling scheme for networks is stochastic blockmodeling [13]. According to this model, occurrence probability of links are independent given their community memberships. Preliminary work in this field consider each node to belong to a single community. Recently, a variant of this modeling, namely "Mixed Membership Stochastic Blockmodel" has been proposed to capture multiple community memberships [9]. Stochastic blockmodeling proposes a Bayesian model of generative processes of the networks under certain priors and conditional distributions. Unfortunately, posterior computation is intractable for most of such Bayesian models and approximation techniques like MCMC are exploited in their inference stage which as mentioned, are not suitable for large real-world data.

In summary, the above methods and many others that have been presented based on the latent feature models, cannot be scaled to very large datasets. We aim to present a scalable method considering the fact that network adjacency matrices and other real-world data usually have an extremely sparse structure.

B. Related Problems

There have also been a number of related studies to link prediction in the area of Network Completion. Basically, network completion has a more general view than link prediction where it is also assumed that there are a number of missing nodes. However, the problem of missing nodes is alleviated by using standard methods for estimating the size of missing nodes in the networks [14]. [15] uses a model-based approach where Kronecker graph [16] is assumed to be the underlying generative model for the graphs. To infer the parameters of this model, the author presents a novel Expectation-Maximization (EM) [17] approach based on Gibbs sampling. As mentioned before, Gibbs sampling and other MCMC methods can hardly achieve scalability and reasonable run-times. [18] explores the problem of network completion when random subsamples from the network are generated by survey sampling: choose a node in the network uniformly at random, and observe the links that node is incident with.

Link prediction may also be viewed as a special case for the Matrix Completion problem [19]. In matrix completion, the aim is to fill the missing entries of an incomplete matrix, however, it is assumed the original matrix is low-rank and there is also no consideration of adjacency matrices, e.g. binary entries, clustered and sparse structure, etc. Recently, there have been several attempts in order to utilize this framework in analysis of networks. [20] uses the sum of a low-rank and a sparse matrix, representing communities and the links between them, respectively, in order to perform clustering on a partially observed graph. However, the method only considers disjoint clusters (as opposed to the findings of [21]) and it is not clear if there are scalable algorithms for it. [22] takes matrix completion and matrix factorization approaches in inference of unknown trust/distrust interactions (i.e. sign inference problem) given a partially observed signed network. It utilizes the notion of weak structural balance in signed networks in order to propose a low-rank model for signed networks.

The problem of Network Inference, mostly in social networks, is also related to link prediction. In network inference, we are interested in inferring the hidden underlying network from the data of the diffusion of some information through the graph [23]. [24] tries to do the same task via Compressed Sensing [25], a sparse recovery framework which is recently exploited as an efficient tool for sparse signal recovery over networks [26], [27].

III. Scalable Link Prediction via Symmetric Matrix Factorization

In this section, first we state our problem formally. Second, our generative process as the likelihood of the network data is presented. Then, we propose our inference scheme. Considering the sparse structure of real networks, it is shown that the proposed scheme can be efficiently computed which makes it appealing for large-scale datasets.

A. Problem Formulation

Consider the observed undirected graph $G = (V, E)$, where V represents the set of nodes and E represents the set of undirected links. Suppose M is the corresponding symmetric adjacency matrix of G. We show the latent feature matrix related to the graph nodes by F, where F_i, i-th column of the latent feature matrix, is the latent feature vector of node i. Let k be the dimension of the latent feature vectors. Given such graph, our goal is to estimate existence of links by inferring the latent feature matrix F^*.

We model the undirected network adjacency matrix M as a Poisson process and set the distribution parameter of M_{ij} to be $F_i^T F_j$:

$$M_{ij} \sim Poisson(F_i^T F_j) \tag{2}$$

Further, we assume the entries of M are conditionally independent given the latent feature vectors. Hence, the likelihood of the adjacency matrix M is:

$$P(M|F) = \prod_{i,j} \frac{(F_i^T F_j)^{M_{ij}} e^{-F_i^T F_j}}{M_{ij}!} \tag{3}$$

This factorization is applicable on undirected networks. For directed networks, as suggested in many papers like [28], we can set M_{ij} to be:

$$M_{ij} \sim Poisson(F_i^T \Lambda F_j) \tag{4}$$

where Λ is an asymmetric matrix.

Poisson distribution is mainly used in Poisson regression (in analogy with Bernoulli trials in logistic regression) in order to model count data and contingency tables [29]. Although adjacency matrices mostly comprise binary entries, they can also be considered as a special kind of counting data. Moreover, we observe that links in a network can be created by the occurrence of special events, e.g., friendship requests, messages, data transfer, etc., where Poisson process is suitable for modeling such events [30]. Furthermore, extensive experimental investigations in [31] show that Poisson model performs the same as Bernoulli model and approximating the modeling of binary graphs by Poisson likelihood does not appear to influence the ability to detect structure. See [31] for a comprehensive investigation about these models.

The idea of Bernoulli trials for the elements of adjacency matrices is explored in [8] and it is shown that the time complexity of inferring the model parameters is linear to the number of observations in each iteration. However, in the next sections, we will show that Bernoulli-Logistic modeling has scalability issues in the optimization procedure when facing with a very large adjacency matrix with millions of entries. While by using a Poisson process as the underlying generative model, we show that inferring the model parameters in each iteration will be possible in a time linear to the number of nodes which is significantly more scalable than

Bernoulli generative model. To the best of our knowledge, no other method has utilized Poisson generative modeling to improve scalability or accuracy in link prediction. In the following sections, we will show that using this generative model, leads to significantly faster prediction and inference of model parameters.

For parameter estimation, we adopt a Maximum Likelihood (ML) approach in which we are seeking to infer the parameters that maximize the likelihood of the observed data. We can write ML for Poisson likelihood in (3) as:

$$\min_{F} \quad lP(M|F) = \sum_{i,j} -M_{ij} \log(F_i^T F_j) + F_i^T F_j + C \quad (5)$$

where lP is the negative log-likelihood and C is a constant independent of F. The problem in (5) is a variant of symmetric matrix factorization problem. Meaning that we want to find F^* where:

$$F^* = \underset{F}{arg\,min}\ l\left(M, F^T F\right) \quad (6)$$

where negative log-likelihood is the corresponding loss $l(.,.)$.

It is important to note that minimizing the negative log-likelihood of Poisson process is equivalent to minimizing generalized KL-divergence [2]. Hence, $l(.,.)$ can also be considered as the generalized KL-divergence.

B. Parameter Inference

After defining the likelihood of the observed graph, now we aim to find the most likely latent factors to the corresponding observed graph.

First note that the domain of the problem in (5) is the set of matrices F for which:

$$\forall i, j \ F_i^T F_j > 0 \quad (7)$$

For simplicity of the feasible region, consider only a subset of the solutions, in which:

$$\forall i \ F_i > 0 \quad (8)$$

In addition, as [2] suggests, in NMF, this constraint helps increase interpretability and sparsity of latent factors.

Problem (5) is not jointly convex regarding all latent feature vectors. But if we fix all latent feature vectors except one, the problem becomes a convex optimization problem. Hence, we apply a block coordinate descent algorithm in order to minimize $lP(M|F)$. By fixing all latent vectors except F_i, let:

$$f(F_i) = lP(M|F) \quad (9)$$

As f is convex and there are no closed form solutions for $\nabla f(F_i) = 0$, the iterative gradient descent update rule for F_i will be:

$$F_i^{(n+1)} = F_i^{(n)} - \eta \nabla f(F_i^{(n)}) \quad (10)$$

where η is the learning rate. $\nabla f(F_i)$ is written as follows:

$$\nabla f(F_i) = 2 \sum_j F_j \left(1 - \frac{M_{ij}}{F_i^T F_j}\right) \quad (11)$$

simplifying to:

$$\nabla f(F_i) = 2 \sum_j F_j + 2 \sum_j -\frac{M_{ij}}{F_i^T F_j} F_j \quad (12)$$

We can proceed further and remove the sentences corresponding to the zero entries of M. Thus, (12) is simplified one more step to:

$$\nabla f(F_i) = 2 \sum_j F_j + \sum_{(i,j)\in E} -\frac{2}{F_i^T F_j} F_j \quad (13)$$

A regular calculation of the gradient vector can be done in a time linear to the number of latent features $|V|$. We now seek for a method which updates F_i more quickly. To this end, we precompute the summation over all factors. Many of the matrix factorization techniques (e.g. [32], [21]) use similar ideas in order to speedup the basic algorithms.

According to Eq. (13), if $\sum_j F_j$ was calculated, the whole gradient calculation would be done in a time linear to the number of known links incident with node i. As we mentioned, many real-world networks have sparse structures, thus, by precomputing $\sum_j F_j$ we can achieve the gradient much faster than the regular gradient calculation. For fast gradient calculation, at the very first iteration (when nothing is inferred and the latent factors are simply initialized), we calculate the sum of the latent factors $\sum_j F_j$ and after each coordinate descent, we will update this summation. In other words, if F_i^{old} and F_i^{new} are the vectors before and after coordinate descent on F_i, we update the sum as:

$$\sum_j F_j \leftarrow \sum_j F_j - F_i^{old} + F_i^{new} \quad (14)$$

This way, for each coordinate, the gradient will be calculated in a time linear to the number of neighbors with the node corresponding to the current coordinate.

We have tested Newton and quasi-Newton methods in order to apply a better coordinate descent, although these methods have the extra complexity of computing the inverse of Hessian matrix. It was observed that in large datasets, the condition number of Hessian matrix turns out to be very large and makes the problem ill-posed. Therefore for line search, we use a combination of projection to ensure the feasibility of our solution and checking Wolfe condition [33] to adaptively change the step size. After each update of F_i, we use the trivial proximity mapping below [33]:

$$F_i^* \leftarrow max\left(0, F_i - \eta \nabla f(F_i)\right) \quad (15)$$

This mapping also leads us to sparser latent feature vectors. For ensuring acceptable descent in each iteration, Wolfe condition is checked. This condition can be evaluated quickly in the same manner as the gradient. A step size η satisfies Wolfe condition if for $0 < \alpha < 1$:

$$f(F_i - \eta \nabla f(F_i)) < f(F_i) - \alpha \eta \|\nabla f(F_i)\|^2 \quad (16)$$

Expanding this condition and performing the simplifications

will result in checking the following inequality:

$$\sum_{(i,j)\in E} -\big(\log((F_i-\eta\nabla f(F_i))^T F_j) - \log(F_i^T F_j)\big) - \eta\nabla f(F_i)^T \sum_j F_j + \alpha\eta\|\nabla f(F_i)\|^2 < 0 \quad (17)$$

Hence, (17) can also be evaluated in a time linear to the number of neighbors of node i. As $\nabla f(F_i)$ and $\sum_j F_j$ are already calculated and we are ensuring the feasibility of the updated F_i with projection, (17) can be evaluated both efficiently and correctly.

A summary of proposed method is shown in Algorithm 1.

Algorithm 1 Scalable Link Prediction using Matrix Factorization (*SLPMF*)

Input: M (Input Matrix), k (Latent Space Dimension)

1: Initialize F `//Initializtion`
2: Calculate $\sum_j F_j$
3: **repeat**
4: **for** i=1 to $|V|$ **do** `//Coordinate Descent`
5: **repeat**
6: Calculate the gradient $\nabla f(F_i)$ from (13)
7: Choose the step size η by checking (17)
8: $F_i^* \leftarrow max\,(0, F_i - \eta\nabla f(F_i))$
9: $\sum_j F_j \leftarrow \sum_j F_j - F_i + F_i^*$
10: **until** Gradient Descent Convergence
11: **end for**
12: **until** Coordinate Descent Convergence

Output: F

So far, we have shown that Poisson generative model leads to efficient gradient calculation and thus, fast gradient descent. For Bernoulli-Logistic model, we can show that such gradient descent approach will not have this property. Bernoulli-Logistic generative model has the following form:

$$P(M|F) = \prod_{i,j} \big(\sigma(F_i^T F_j)\big)^{M_{ij}} \big(1-\sigma(F_i^T F_j)\big)^{1-M_{ij}} \quad (18)$$

where $\sigma(.)$ is sigmoid function. Hence, negative log-likelihood will be:

$$lP(M|F) = \sum_{i,j} -M_{ij}(F_i^T F_j) + \log(1+F_i^T F_j) + C \quad (19)$$

By $f(F_i) = lP(M|F)$ and computing the gradient, we get:

$$\nabla f(F_i) = \sum_j -M_{ij}F_j + \frac{F_j}{1+F_i^T F_j} \quad (20)$$

Obviously, (20) cannot be computed with the technique used in (13) and requires $O(|V|)$ time to get computed.

C. Complexity Analysis

1) **Time Complexity:** Up to line (2) of the Algorithm 1, it takes $O(k|V|)$ time to initialize and compute sum of the latent feature vectors. We assume that c, g and l are the average number of coordinate descent, gradient descent and line search iterations, respectively. For each coordinate, Line (6) takes $O\,(kN)$ time where N is the average number of neighbors of the network nodes. Line (7) takes $O\,(lkN)$ and line (8) and line (9) take $O(k)$ time. Hence, time complexity of the algorithm is:

$$O\,(k|V| + c|V|g(l+1)kN) \quad (21)$$

Due to the sparsity of the networks, we can consider N to be a constant. Also for c, g and l, we observed that very small values were required to achieve acceptable convergence. Thus, the above algorithm runs in $O(|V|)$ time. As shown, in case of using Bernoulli-Logistic model, instead of N, we would have a cost of $O(|V|)$.

Another approach for the analysis of time complexity is to sum up the cost of each iteration of the loop in line (4). In each iteration of the loop in line (4), we iterate over the neighbors of each node. In other words, the i-th iteration of this loop has the cost $O(g(l+1)kN_i)$ where N_i is the number of neighbors adjacent to i-th node. By summing up these costs for all nodes, we will have: $O(g(l+1)k\sum_i N_i) = O(g(l+1)k|E|)$. Hence, a more accurate time complexity of the above algorithm will be:

$$O\,(k|V| + c|E|g(l+1)k) \quad (22)$$

However, due to the sparsity of real networks, we can consider $|E| = O(|V|)$ and therefore this time complexity reveals the same result as in (21).

2) **Space Complexity:** One of the advantages of our algorithm is that its mechanism is based on iterations over the neighbors of nodes. Hence, we can store the input M as adjacency linked list which has the space complexity of $O(|E|)$. For the variable F, we will need $O(k|V|)$ space. Overall, we can write the space complexity of the algorithm as:

$$O(k|V| + |E|) \quad (23)$$

Again, with the sparsity assumption on M ($|E| = O(|V|)$), our algorithm runs in $O(|V|)$ space complexity.

D. Notes about EM Implementation

We can also view link prediction in EM framework. Since we are applying maximum likelihood on the observed data to infer information about the missing ones, we can tie known and unknown links in a unified likelihood and then perform maximum likelihood. In other words, in this setting, we will have a matrix with two sets of entries: First, observed entries which we denote with O and can be either 0 or 1. Second, Unobserved entries denoted with U that the real value of them is not known to us. Therefore, the complete negative log-likelihood of the matrix will be:

$$lP(M|F) = \sum_{(i,j)\in O} -M_{ij}\log(F_i^T F_j) + F_i^T F_j \sum_{(i,j)\in U} -M_{ij}\log(F_i^T F_j) + F_i^T F_j + C \quad (24)$$

This naturally suggests the use of EM algorithm [17]: a general iterative method for ML estimates when the data is incomplete.

Each iteration of EM consists of two stages: E-step and M-step. In E-step, we seek to compute the expected value of the negative log-likelihood with respect to the conditional distribution of U given O and the estimation of the parameters from the previous iteration, $F^{(t)}$ in our case. Thus, we want to compute the following:

$$Q(F|F^{(t)}) = E_{U|O,F^{(t)}}[lP(O,U|F)] \tag{25}$$

To summarize, in E-step of each iteration of EM, instead each missing entry of M, we use their corresponding expected value. However, as in regular test cases, we remove a fraction of the entries of the matrix (e.g. 10% of the entries), EM implementation will have an inefficiency in terms of both space and time complexity. Because we have to store the whole unknown entries and update them per each iteration, although we may have a slight improvement in accuracy of predictions. Hence, applying EM to our problem does not seem to scale well to large datasets.

IV. Experimental Evaluations for Link Prediction

A. Settings

1) **Datasets***:* We use a variety of real-world datasets from different areas that link prediction methods can be applied. As previously mentioned, all datasets we use are undirected. We included networks from various applications. Datasets statistics are listed in Table 1. The sparsity of the datasets is one minus the quotient of the number of ones in the adjacency matrix ($2|E|$) divided by the whole number of entries in the adjacency matrix which is $|V|^2$. The datasets are as follows:

- Among biological networks, we use the well known protein-protein interaction network in budding yeast (denoted *Yeast*) with 2,361 nodes and 6,914 links [34].
- From social networks, a network of Facebook users with 5,793 nodes and 30,753 links (known as *FacebookL1*) who, according to their Facebook profile page, worked in an international IT Corporation that provides products and services to customers around the world [35].
- Graph of Autonomous Systems (AS) peering information inferred from Oregon route-views between March 31 2001 and May 26 2001. This dataset consists of 11,492 nodes and 23,409 links.
- Arxiv *COND-MAT* and *GrQc* in co-authorship networks [36]. *COND-MAT* is from the e-print arXiv and covers scientific collaborations between authors papers submitted to Condense Matter category and consists of 23,133 nodes and 93,497 links. Arxiv *GrQc* collaboration network is from the e-print arXiv and covers scientific collaborations between authors papers submitted to General Relativity and Quantum Cosmology category and has 5,242 nodes and 14,490 links.
- August 24 2002 snapshot from Gnutella peer-to-peer network (denoted *P2P*[1]) which has 26,518 nodes and 65,369 links [36], [37].

[1]Note that P2P dataset is originally a directed network, however we converted it to an undirected one by simply changing unmatched zeroes to ones.

Table 1: Datasets Statistics

Dataset	Nodes	Links	Sparsity	Avg. Degree
Yeast [34]	2,361	6,914	0.99752	2.9
GrQc [36]	5,242	14,490	0.99891	2.8
FacebookL1 [35]	5,793	30,753	0.99817	5.3
AS [36]	11,492	23,409	0.99965	2.0
COND-MAT [36]	23,133	93,497	0.99965	4.0
P2P [36], [37]	26,518	65,369	0.99981	2.5

2) **Competing Methods***:* We refer to our method as *SLPMF*. Methods that we compare our performance to are:

- *Unsupervised Scores:* We picked 3 popular scores: Adamic-Adar (*AA*), Shortest-Path (*SH*) and Preferential-Attachment (*PA*) [38].
- *NMF under KL-Divergence:* There are a variety of methods in this category. We use [32] (denoted *NMF-KL*) as a recent and efficient method. The reason we chose KL-divergence is that by considering the Poisson generative model for the original matrix, the maximum likelihood inference of the parameters leads to the minimization of KL-Divergence [2]. However, the method in [32] is suited for collaborative filtering rather than link prediction. Hence, the factorization in this work is based on equation (1) and the output is not necessarily a symmetric matrix. Also, the mapping to latent feature space corresponds to user-item structure (U is for user features and V is for item features). Our method assumes that the matrix is a graph adjacency matrix rather than a user-item matrix. We will show the effect of different factorizations in next section.
- *GLFM:* This method, presented in [8], uses Bernoulli-Logistic model as its generative model.

As mentioned in earlier sections, there are many methods using latent feature models for link prediction. However, many popular ones are outperformed by methods in [7] and [8]. Hence, we decided to compare our performance to [8] instead of all methods which is the closest to ours in general methodology. Also, the work in [1], as discussed in Section II-A, has a very long run-time that it was infeasible for us to get its results.

3) **Setup***:* For all of the test cases, we randomly removed 10% of the entries of the adjacency matrix in the following way: First we choose a random entry, say (i, j). We add (i, j) and (j, i) to the unknowns list and if (i, j) equals to 1, we set the value of (i, j) and (j, i) to zero. The resulted matrix will be the adjacency matrix of the observed graph G. For evaluation of the methods, we use AUC (Area under ROC curve) as the measure of the quality of predictions because ROC curves are insensitive to changes in class distribution [39].

We chose a random initialization for F. All results show the average values over 10 cross-validation folds in each experiment. For implementation of our method, we used MATLAB. For unsupervised methods, we used *LPmade* package [40] and for the other methods, we used codes provided by the authors. All experiments were run on a PC with Intel Core i7 2.6GHz CPU and 12GB of RAM.

4) **Choosing the dimension of latent space***:* Clearly, the more the dimension of the latent space, the more degrees of freedom we can cover. However, smaller dimensions are

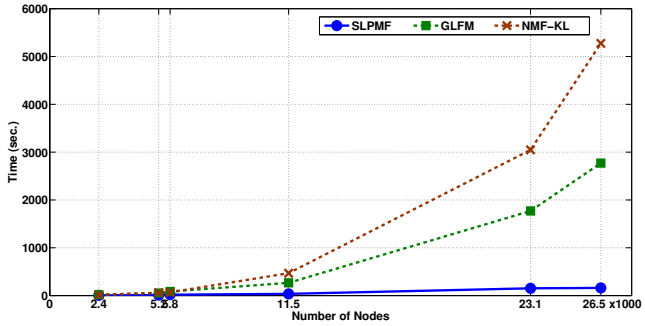


Fig. 1. Run-time Comparison - Each point corresponds to one dataset. *GLFM* and *NMF-KL* run-times become much worse on larger datasets while *SLPMF* run-time increases linearly.

preferable in terms of speed and memory issues. One approach to identify a good number (the same as in [9], [21]) of latent space dimension is to extract a hold out set (e.g. 10% of the observed graph links which is our train set) from the observed graph and run the algorithm on the rest of the dataset. The latent dimension achieving the most prediction will be a suitable one. We repeated this task for all three supervised methods (*NMF-KL*, *GLFM* and *SLPMF*) and chose the dimension that works best for all in each dataset.

B. Accuracy

The results are shown in Table 2. It can be seen that our proposed method outperforms unsupervised scores in all datasets. We also see better AUC for SLPMF comparing to *GLFM* and *NMF-KL*. The reason that *GLFM* cannot achieve the optimal performance of Bernoulli-Logistic generative model as it should, can be investigated in *GLFM* optimization strategy which is one of the main contributions of it [8]. In *GLFM*, an approximation to Hessian matrix through MM method is employed. However, we observed high condition numbers on this approximation (as we had in Newton-based optimizations for our own method) in our datasets which causes the descent algorithm to become slower and more inaccurate [33]. AUC of *NMF-KL* is better than *GLFM* however this method suffers from two issues that puts it behind our method: First, it is not suited for link prediction as we described and loses some of its performance because of this issue. Second, this algorithm as pointed by the authors, runs at $O(nmkd)$ time where n and m are the matrix dimensions, k is the latent space dimension and d is the average number of gradient descent iterations. Hence, *NMF-KL* also converges slower than our method.

C. Scalability

To test for scalability, we compare the run-time of our method with *GLFM* and *NMF-KL*. Figure 1 shows the average run-times under different datasets. The graphs represent the time until convergence (where change in AUC is less than 0.001) with maximum of 10 iterations for each method. It can be observed that *GLFM* and *NMF-KL* run-times become much worse as we test on larger datasets while our method's run-time increases linearly to the number of network nodes, as it was investigated in Section III-C1. As pointed out in the first experiment, high condition numbers in Hessian-approximations and slower optimization procedure add-up to longer run-times of *GLFM*. Also, higher algorithmic order of *NMF-KL* causes a significant drop in its run-time for larger datasets.

Figure 2 shows the resulted AUC from different time-stamps during the run-time on all datasets. Each method was evaluated for maximum of 10 iterations. As it is observed, our method converges faster and in the same running time achieves better AUC than other two methods.

V. Application in Matrix Factorization under KL-Divergence

In this section, we apply our proposed method to other applications of matrix factorization under KL-divergence. The problem we want to solve is:

$$\underset{(U,V)\geq 0}{arg\ min} \sum_{i,j} \log\left(\frac{M_{ij}}{U_i^T V_j}\right) - M_{ij} + U_i^T V_j \qquad (26)$$

where M is an $m \times n$ input matrix and U_i and V_j are i-th and j-th column of U and V, respectively. For this purpose, we perform the same coordinate descent described in previous section by fixing all variables except single columns of U and/or V. Note that we still assume that the datasets we want to apply our method on are sparse which is a reasonable assumption in many real-world networks. To improve the speed of the algorithm, instead of cycling on all coordinates, we choose the more important coordinates to optimize in a greedy manner. Specifically, according to a particular measure in each stage of coordinate descent, we choose the most important coordinate and optimize the objective function towards the chosen coordinate.

In general, we can rewrite the problem in (26) as:

$$\underset{(U,V)\geq 0}{min} \quad f(U_1, ..., U_m, V_1, ..., V_n) \qquad (27)$$

One of the appropriate strategies for identifying important coordinates is to choose the coordinates with the highest gradient norm, expecting that such coordinates cause more descent on objective value when chosen. However, to apply a greedy coordinate descent, we need a proper data structure in order to store and retrieve the data related to the coordinates efficiently.

A. Greedy Coordinate Descent for Matrix Factorization under KL-Divergence

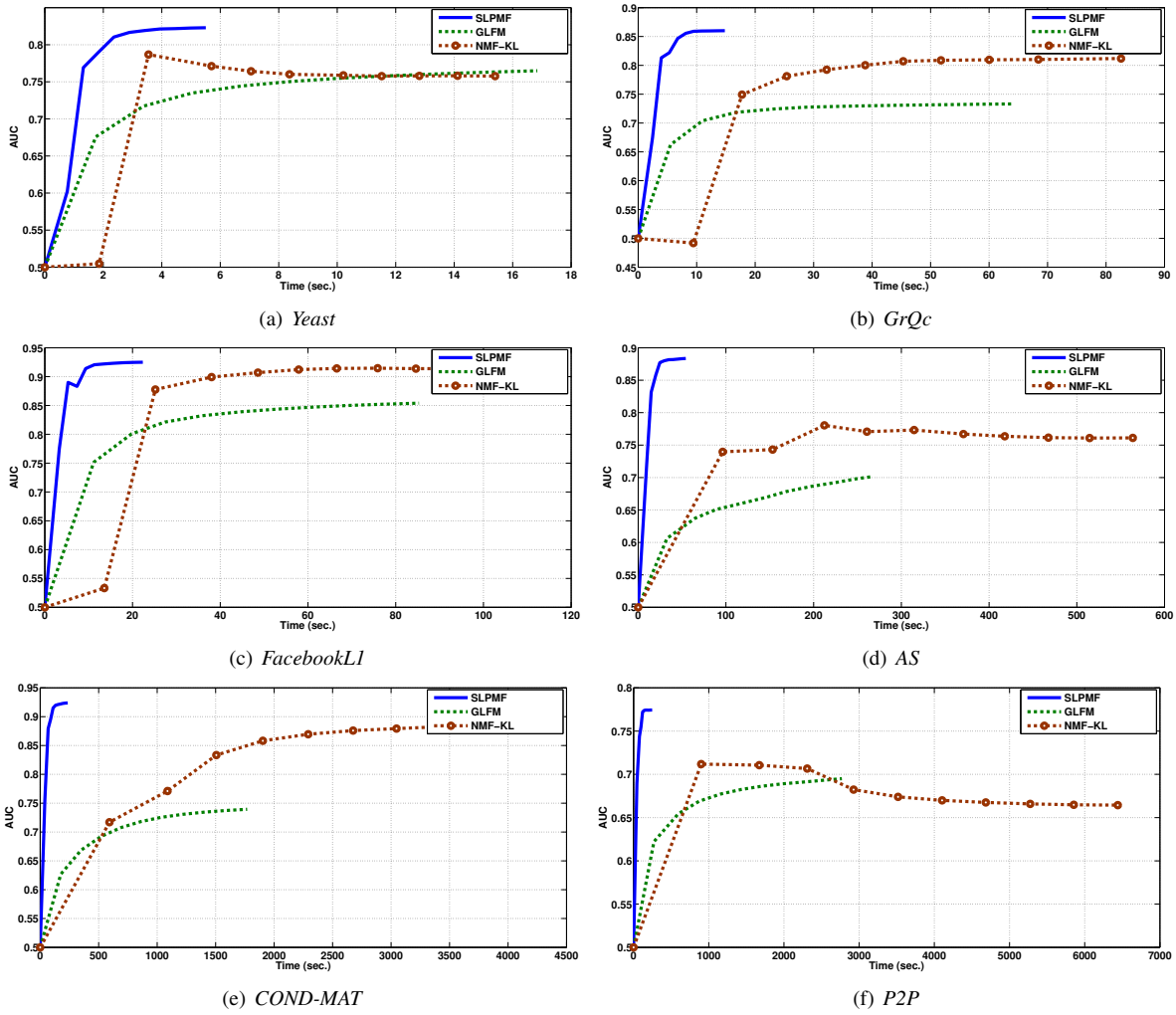
To perform a greedy coordinate descent based on gradient norms, we need a data structure that ables us to perform the procedures below efficiently:

- Data structure initialization
- Choosing the gradient with highest norm
- Updating the dependent data after the optimization of one coordinate
- Deleting the stationary coordinates

We choose heap data structure in order to store and retrieve gradient norms. Initialization and Choosing the highest gradient can be done in $O(|V|)$ and $O(1)$ respectively. Note that

Table 2: AUC Scores Comparison - k is the latent space dimension for supervised methods (*NMF-KL, GLFM, SLPMF*). Bold numbers represent the best results

Network	k	*AA*	*PA*	*SH*	*NMF-KL*	*GLFM*	***SLPMF***
Yeast	10	0.6917 ± 0.005	0.6634 ± 0.024	0.6925 ± 0.028	0.7577 ± 0.011	0.7648 ± 0.010	**0.8230 ± 0.005**
GrQc	10	0.7554 ± 0.026	0.7302 ± 0.014	0.7411 ± 0.003	0.8118 ± 0.017	0.7331 ± 0.021	**0.8600 ± 0.012**
FacebookL1	10	0.8014 ± 0.013	0.7752 ± 0.011	0.7871 ± 0.019	0.9135 ± 0.009	0.8539 ± 0.023	**0.9250 ± 0.008**
AS	15	0.7182 ± 0.010	0.6697 ± 0.016	0.6934 ± 0.011	0.7610 ± 0.005	0.7013 ± 0.021	**0.8835 ± 0.015**
COND-MAT	25	0.7301 ± 0.012	0.7132 ± 0.014	0.7124 ± 0.013	0.8865 ± 0.016	0.7393 ± 0.011	**0.9238 ± 0.017**
P2P	30	0.6111 ± 0.005	0.6201 ± 0.005	0.6249 ± 0.020	0.6644 ± 0.031	0.6950 ± 0.014	**0.7744 ± 0.021**



(a) *Yeast* (b) *GrQc*

(c) *FacebookL1* (d) *AS*

(e) *COND-MAT* (f) *P2P*

Fig. 2. AUC vs. Run-time Comparison - Results are taken for maximum of 10 iterations for each algortihm. The results show that *SLPMF* converges up to 15 times faster and in the same running time, *SLPMF* achieves better solutions.

$|V|$ is the number of network nodes which is also the number of corresponding coordinates.

Before the details of the greedy algorithm, we define one term used in the description of the algorithm steps. We present all formulations and definitions for coordinates of U. All formulations and definitions for V can be derived similarly. Consider the gradient of the objective function towards to coordinate U_i:

$$\nabla f(U_i) = \sum_j V_j + \sum_{j:\ M_{ij}\neq 0} -\frac{M_{ij}}{U_i^T V_j} V_j \tag{28}$$

We define the second term of (28) to be the "Gradient Residue of coordinate U_i":

$$Res\left(\nabla f(U_i)\right) = \sum_{j:\ M_{ij}\neq 0} -\frac{M_{ij}}{U_i^T V_j} V_j \tag{29}$$

A summary of our greedy optimization procedure is shown in Algorithm 2. We refer to this algorithm as *GSCD*. *GSCD* is divided into five major steps: (Assume the iterations to happen on U and the chosen coordinate is U_i. Iterations over V can be derived similarly.)

1) **Initialization:** In the beginning of *GSCD*, we first calculate the two sums $\sum_i U_i$ and $\sum_j V_j$. Then we calculate the gradient residues for all coordinates U

and V and initialize a heap only for U gradient norms using the calculated sum $\sum_j V_j$ and gradient residues of U. For V gradients, we initialize the heap after the termination of U iterations. Due to the calculation of all gradient residues and heap initialization, this step has $O(m + M_{non-zero})$ time complexity where m is the number of columns of U and $M_{non-zero}$ is the number of non-zero elements of the input matrix.

2) **Gradient Descent:** After selecting a U coordinate (e.g. U_i) from the heap, we perform the gradient descent on the chosen coordinate which is done in the order of non-zero entries of the i-th row of the input matrix that is assumed to be sparse. Hence, because of storing the sums and the sparsity of the input matrix, gradient descent can be done efficiently on each coordinate in $O(1)$. Due to updating the heap data structure, this step will have $O(\log m)$ time complexity.
3) **Updating Sums:** After optimizing U_i, we must update $\sum_i U_i$. This step has constant time complexity.
4) **Updating Gradient Residues:** U_i will also affect the gradient residues of V. Specifically, coordinates like V_j for which $M_{ij} \neq 0$ will be affected by optimizing U_i. Hence we should update the gradient residue for all such V_j as follows:

$$Res(\nabla f(V_j)) \leftarrow Res(\nabla f(V_j)) + \frac{M_{ij}}{U_i^{old^T} V_j} V_j - \frac{M_{ij}}{U_i^{new^T} V_j} V_j \quad (30)$$

Due to the sparsity of the input matrix, this step will also have constant time complexity.
5) **Removal of Stationary Coordinates:** If we reach a coordinate which can not be optimized any further, we won't be able to optimize other coordinates because the same stationary coordinate with the highest gradient norm will appear on top of the heap and will be chosen every time. To prevent such deadlocks, we remove these coordinates upon reaching and add them to heap in next iteration over U. This step has $O(\log m)$ time complexity.

Space and time complexity analysis of *GSCD* is almost like the analysis we had for *SLPMF* in previous section. As all iterations of the gradient descents are on the non-zero elements of the input matrix corresponding to the coordinate being optimized, we can observe that *GSCD* has space complexity of $O(M_{non-zero})$. For time complexity, according to the cost mentioned for each step, the overall complexity will be $O(n \log n + m \log m + M_{non-zero})$ where n is the number of columns of V. Note that in the final algorithm, we limit the number of iterations on coordinates to the number of columns in U or V so that we do not exceed the number of iterations we had in the cyclic version.

To evaluate the performance of our greedy approach, we chose two text datasets. The datasets statistics are listed in Table 3.

We compare our method with the cyclic coordinate descent method (*CCD*) that is presented in [32] for matrix factorization under KL-divergence. We use the authors' implementation.

Algorithm 2 Greedy Scalable Coordnate Descent for NMF under KL-Divergence (*GSCD*)

Input: M (Input Matrix),
k (Latent Space Dimension)

1: Initialize U and V `//Initializtion`
2: Calculate $\sum_i U_i$ and $\sum_j V_j$
3: Calculate All $Res(\nabla U_i)$ and $Res(\nabla V_j)$
4: **repeat**
5: Initialize and Heapify $GradientHeap_U$
6: **repeat** `//Coordinate Descent`
7: Pick Highest from $GradientHeap_U$ (e.g. U_i)
8: $U_i^{old} \leftarrow U_i$
9: Perform Gradient Descent on U_i and store in U_i^{new}
10: **if** $\|U_i - U_i^{new}\| \leq \epsilon$ **then**
11: Delete U_i gradient from $GradientHeap_U$
12: continue;
13: **end if**
14: $\sum_i U_i \leftarrow \sum_i U_i - U_i^{old} + U_i^{new}$
15: Recalculate $\nabla f(U_i)$ and update $GradientHeap_U$
16: **for all** V_j where $M_{ij} \neq 0$ **do**
17: $Res(\nabla f(V_j)) \leftarrow Res(\nabla f(V_j)) + \frac{M_{ij}}{U_i^{old^T} V_j} V_j - \frac{M_{ij}}{U_i^{new^T} V_j} V_j$
18: **end for**
19: $U_i \leftarrow U_i^{new}$
20: **until** Convergence or Maximum Iterations Reached
21: Repeat above steps for V analogous to Line $5 - 20$
22: **until** Coordinate Descent Convergence

Output: F

We compare the attained objective value in different time stamps during the runtime of both algorithms. The results are presented in Figure 3.

As seen, *GSCD* achieves a lower objective value in the same running time and converges up to six times faster than *CCD*.

Table 3: Statistics of Text Dataesets, k is the latent space dimension.

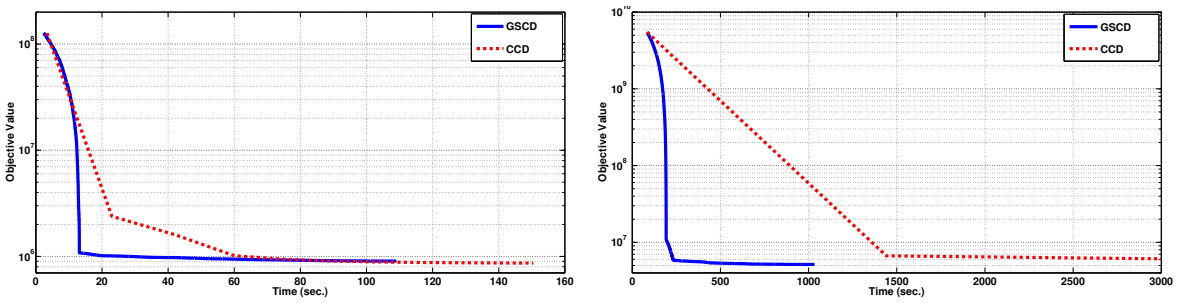
Dataset	k	m	n	Sparsity
Yahoo-News[32]	20	21,839	2,340	0.99316
RCV1 [41]	20	47,236	23,149	0.99839

B. Parallelization of Greedy Coordinate Descent

The first step to parallelize *GSCD* is to identify which data is shared between different stages of coordinate descent. Assuming optimizing U matrix, updates in *GSCD* occur on the following:

- Sum term $\sum_i U_i$
- Heap of U gradient norms
- Gradient residues of V

Clearly, sum term $\sum_i U_i$ and the heap related to U gradients can not be updated in parallel. However for gradient residues, assuming optimizing U_i, only gradient residues V_j for which $M_{ij} \neq 0$ are affected. Hence, by choosing coordinates from U that are mutually independent from each other, meaning



(a) *Yahoo-News* (b) *RCV1*

Fig. 3. Attained objective value comparison between proposed method (GSCD) and CCD [32] on Text Data: GSCD achieves a lower objective value in the same running time and converges up to six times faster than CCD.

those which do not share any common neighbor in bipartite graph representation of the input matrix, we can optimize these coordinates independently, although for $\sum_i U_i$ and heap of gradient norms of U we need sequential updating. This way, the most inner loop of *GSCD* that is the gradient descent on coordinates, is performed in parallel. To choose mutually independent coordinates and to preserve greedy coordinate descent at the same time, we need to choose the coordinates that are mutually independent and have the maximum norm. However, a search over the coordinates with such constraints does not seem to be efficiently applicable. Hence, we use another approach in which we first select the highest gradient from heap and afterwards, we randomly select the needed number of coordinates (depending on the number of parallel threads available). Although by random selection of coordinates we are not choosing coordinates in a perfectly greedy manner, we spend a constant time for choosing the coordinates.

The summary of the parallelization idea can be seen in Algorithm 3.

To see how much parallelization can improve the performance, we tested our algorithm on the same datasets as we had for *GSCD*. We ran the algorithm on different number of parallel threads and tracked the objective value in different iterations. The results are shown in Figure 4.

In Yahoo-News dataset, it can be observed that we need a high number of parallel threads (e.g. 8 parallel threads) in order to see a significant improvement in objective value descent. In RCV1 dataset, even for 8 threads, we can hardly see any improvement in the algorithm performance. The reason can be related to the preprocessing stage on this dataset. As Yahoo-news and RCV1 are text datasets, preprocessing schemes such as stemming (i.e. mapping different forms of a word into their original stem) can cause the dataset to have a very few number of coordinates which are mutually independent.

Algorithm 3 Parallel Greedy Scalable Coordnate Descent for NMF under KL-Divergence (*PGSCD*)

Input: M (Input Matrix),
k (Latent Space Dimension),
p (Number of Parallel Instances)

1: Initialize U and V `//Initializtion`
2: Calculate $\sum_i U_i$ and $\sum_j V_j$
3: Calculate All $Res(\nabla U_i)$ and $Res(\nabla V_j)$
4: **repeat**
5: Initialize and Heapify $GradientHeap_U$
6: **repeat** `//Coordinate Descent`
7: Pick the Highest from $GradientHeap_U$ (e.g. U_i)
8: choose $p-1$ other coordinates from U randomly and check their adjacencies to be independent
9: **Parallel: for all** U_i
10: Perform Gradient Descent on U_i and store in U_i^{new}
11: **for all** V_j where $M_{ij} \neq 0$ **do**
12: $Res(\nabla f(V_j)) \leftarrow Res(\nabla f(V_j)) + \frac{M_{ij}}{U_i^{old\,T} V_j} V_j - \frac{M_{ij}}{U_i^{new\,T} V_j} V_j$
13: **end for**
14: **EndParallel**
15: **for all** Chosen Coordinates U_i **do**
16: **if** $\|U_i - U_i^{new}\| \leq \epsilon$ **then**
17: Delete U_i gradient from $GradientHeap_U$
18: **end if**
19: $\sum_i U_i \leftarrow \sum_i U_i - U_i^{old} + U_i^{new}$
20: Recalculate $\nabla f(U_i)$, update $GradientHeap_U$
21: **end for**
22: **until** Convergence or Maximum Iterations Reached
23: Repeat above steps for V analogous to Line $5-22$
24: **until** Coordinate Descent Convergence

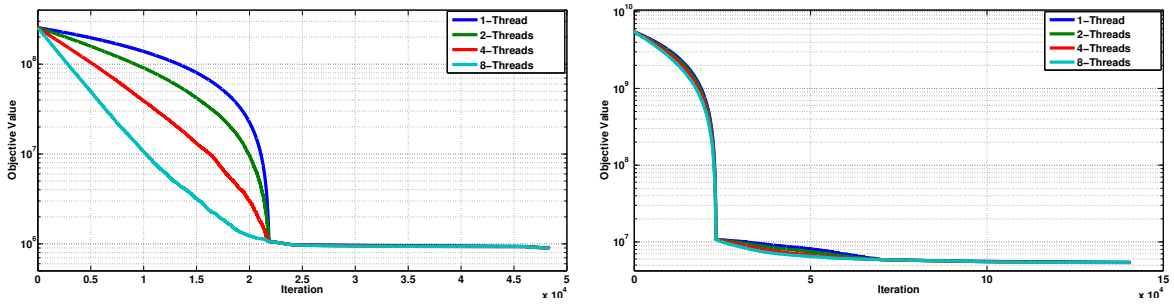
Output: F

C. Is Greedy/Parallel Coordinate Descent Possible for Link Prediction?

Consider the gradient for i-th coordinate in symmetric matrix factorization which is the case for link prediction:

$$\nabla f(F_i) = 2\sum_j F_j + \sum_{(i,j)\in E} -\frac{2}{F_i^T F_j} F_j \qquad (31)$$

Obviously, optimization over one coordinate results in change of gradient for all coordinates since all gradients have the term $\sum_j F_j$. Hence, exact greedy/parallel coordinate descent in symmetric matrix factorization under KL-divergence (that we used for link prediction) is not possible due to the change in gradient of all coordinates after each coordinate optimization. However, we can apply an approximation to the update pro-



(a) *Yahoo-News* (b) *RCV1*

Fig. 4. Effect of Parallelization of *GSCD* on Text Data: It appears that preprocessing on RCV1 makes the parallelization be less effective than Yahoo-News.

cedure such that only a few number of gradients are affected after the optimization of one coordinate.

Let the gradient for i-th coordinate in link prediction be:

$$\nabla f(F_i) = \Sigma + R_i \tag{32}$$

where $\Sigma = 2\sum_j F_j$ and $R_i = \sum_{(i,j)\in E} -\frac{2}{F_i^T F_j} F_j$.

We assume that $\|\Sigma + R_i\|^2$ is stored in a heap data structure. We define the effect of updating i-th coordinate on other coordinates like $\nabla f(F_j)$ to be:

$$\nabla f(F_j) = \Sigma + \Delta_i + R_j + \Delta_{R_j} \tag{33}$$

where $\Delta_i = F_i^{new} - F_i^{old}$ and the effect on gradient residue is $\Delta_{R_j} = \frac{2}{F_i^{old\,T} F_j} F_j - \frac{2}{F_i^{new\,T} F_j} F_j$.

Similar to what we had for *GSCD*, only the gradient residues of neighbors of i-th node will change after updating F_i. Hence for j-th coordinate where $M_{ij} \neq 0$, we can write the new gradient as:

$$\|\nabla f(F_j)^{new}\|^2 = \|\Sigma + R_j\|^2 + 2(\Sigma + R_j)^T(\Delta_i + \Delta_{R_j}) + \|\Delta_i + \Delta_{R_j}\|^2 \tag{34}$$

We can further expand and rearrange the terms of this gradient as:

$$\|\nabla f(F_j)^{new}\|^2 = \|\Sigma + R_j\|^2 + 2\Sigma^T\Delta_i + \|\Delta_i\|^2 + 2R_j^T\Delta_i + 2(\Sigma + R_j)^T\Delta_{R_j} + 2\Delta_i^T\Delta_{R_j} + \|\Delta_{R_j}\|^2 \tag{35}$$

if $M_{ij} = 0$, then Δ_{R_i} terms in (35) will be zero and for such coordinates, we will have:

$$\|\nabla f(F_j)^{new}\|^2 = \|\Sigma + R_j\|^2 + 2\Sigma^T\Delta_i + \|\Delta_i\|^2 + 2R_j^T\Delta_i \tag{36}$$

Overall, for (35) and (36), we have the observations below:

- The two terms $2\Sigma^T\Delta_i + \|\Delta_i\|^2$ are equal for all coordinates. Therefore, they do not change the order of elements stored in a heap data structure and we can omit them from update procedure.
- The terms $2(\Sigma + R_j)^T\Delta_{R_j} + 2\Delta_i^T\Delta_{R_j} + \|\Delta_{R_j}\|^2$ are only effective on gradients of neighbors of i-th node. Hence, updating gradient of all these coordinates will have the time complexity of $O(N_i \log |V|)$ (N_i the number of neighbors of i-th node) for which with sparsity assumption on the network, we will have $O(\log |V|)$. This time complexity will not deteriorate the overall time complexity of our greedy approach and we can tolerate such update procedure.
- The only term that is different for all coordinates is $2R_j\Delta_i$ and updating all coordinates after each iteration of coordinate descent will have $O(|V|)$ time complexity.

According to the above observations, we can propose an approximation to the update procedure by ignoring updating all coordinates by $2R_j\Delta_i$ based on the intuition that these terms will have small values during the optimization procedure and do not change the heap order significantly.

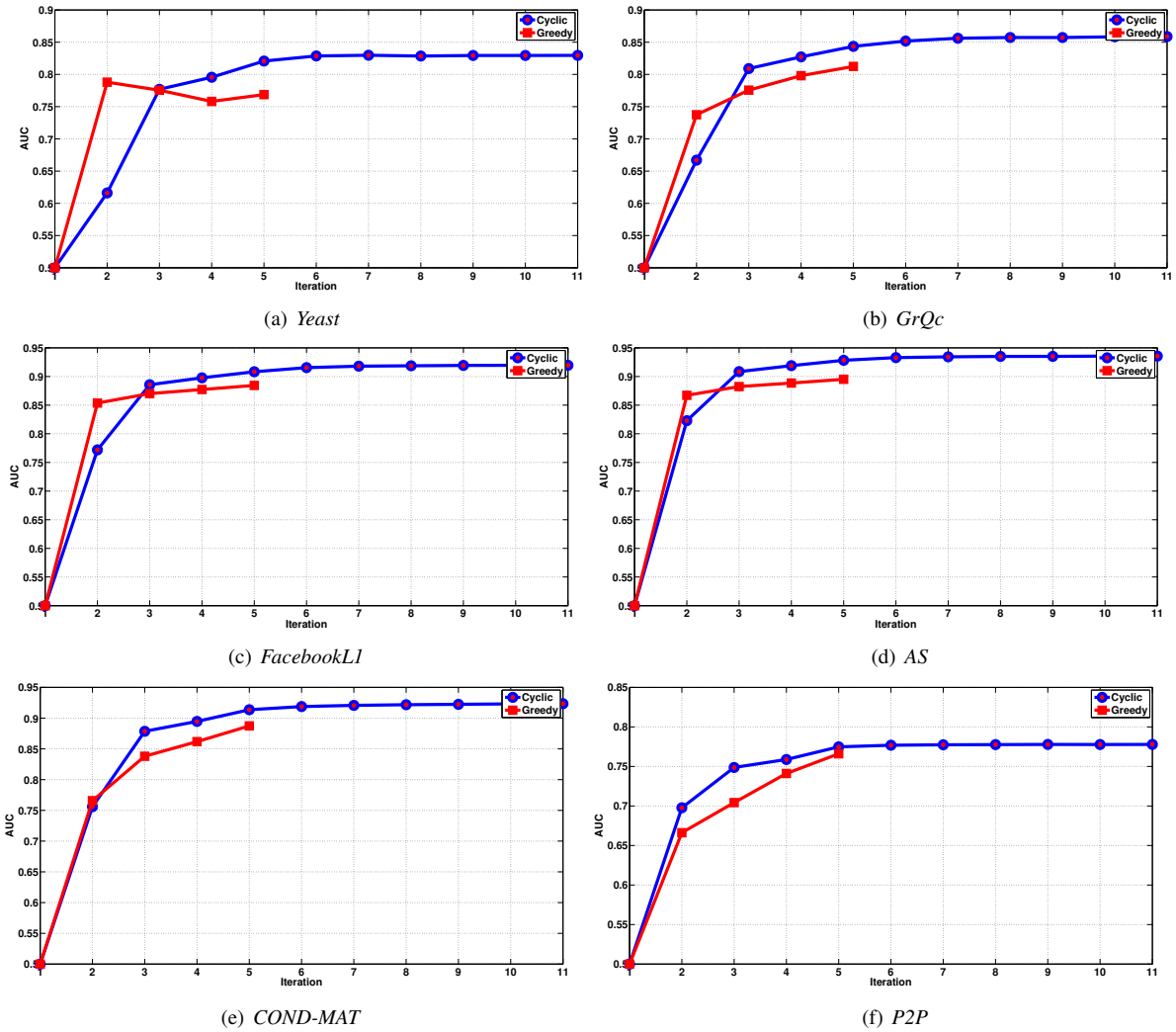
We tested such approximation on the datasets that we presented in Table 1. Figure 5 shows the results for cyclic and greedy method. As experimental evaluations reveal, we observe a faster increase in AUC in the beginning of the greedy algorithm. However, the more the greedy algorithm runs, the more the effect of the proposed approximation will be on the performance.

In sum, our approximation strategy can be effective as a measure of choosing a proper starting point for the cyclic algorithm as greedy optimization terminates faster and has a better starting AUC compared to the cyclic version, although it can not converge to a better AUC in final iterations.

VI. Conclusion

In this paper, we proposed a scalable method for link prediction problem using matrix factorization techniques. We used a Poisson generative model for the networks and utilized maximum likelihood to estimate the parameters. To the best of our knowledge, previous methods for link prediction did not consider such generative model and were introduced in other models like Bernoulli-Logistic. However, according to experimental investigations in recent papers and real-world evidences, it seems networks can be modeled and analyzed as Poisson processes. By considering the sparsity of real-world networks, we showed that our method can scale easily to large-scale datasets. Experiments on networks of various types, e.g. social and biological networks, showed that we can achieve better performance compared to unsupervised and supervised methods. In addition, our method is significantly more scalable.

We also applied our method in matrix factorization problem under KL-divergence. For the first time, we proposed a greedy



(a) *Yeast* (b) *GrQc*

(c) *FacebookL1* (d) *AS*

(e) *COND-MAT* (f) *P2P*

Fig. 5. AUC comparison between greedy and cyclic coordinate descent in link prediction: Greedy coordinate descent acheives a better starting point but converges to a lower AUC than the cyclic coordinate descent.

coordinate descent algorithm and its parallelized version for KL-divergence. Experimental evaluations depicted faster convergence of the proposed methods.

There are several directions for future work. The existence of faster coordinate and gradient descent methods can be investigated, e.g. by adaptive coordinate descent algorithms or by better line search algorithms and exploiting the sparsity of latent features. Also, distributed optimization procedures can help us to speed up the algorithm.

REFERENCES

[1] A. K. Menon and C. Elkan, "Link prediction via matrix factorization," in *Machine Learning and Knowledge Discovery in Databases*, ser. Lecture Notes in Computer Science, vol. 6912. Springer Berlin Heidelberg, 2011, pp. 437–452.

[2] Y.-X. Wang and Y.-J. Zhang, "Nonnegative matrix factorization: A comprehensive review," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 25, no. 6, pp. 1336–1353, 2013.

[3] R. Salakhutdinov and A. Mnih, "Probabilistic matrix factorization," in *Advances in Neural Information Processing Systems*, 2008.

[4] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, no. 8, pp. 30–37, 2009.

[5] A. Cichocki, R. Zdunek, A.-H. Phan, and S. Amari, *Nonnegative Matrix and Tensor Factorizations: Applications to Exploratory Multi-way Data Analysis and Blind Source Separation*. John Wiley and Sons, Ltd, 2009.

[6] D. Liben-Nowell and J. Kleinberg, "The link-prediction problem for social networks," *Journal of the American Society for Information Science and Technology*, vol. 58, no. 7, pp. 1019–1031, May 2007.

[7] S. Gao, L. Denoyer, P. Gallinari, and J. Guo, "Latent factor blockmodel for modelling relational data," in *Proceedings of the 35th European conference on Advances in Information Retrieval*, ser. ECIR'13. Berlin, Heidelberg: Springer-Verlag, 2013, pp. 447–458.

[8] W.-J. Li, D.-Y. Yeung, and Z. Zhang, "Generalized latent factor models for social network analysis," in *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence - Volume Two*, ser. IJCAI'11. AAAI Press, 2011, pp. 1705–1710.

[9] E. M. Airoldi, D. M. Blei, S. E. Fienberg, and E. P. Xing, "Mixed membership stochastic blockmodels," *Journal of Machine Learning Research*, vol. 9, pp. 1981–2014, Jun. 2008.

[10] K. Lange, D. R. Hunter, and I. Yang, "Optimization Transfer Using Surrogate Objective Functions," *Journal of Computational and Graphical Statistics*, vol. 9, no. 1, 2000.

[11] P. D. Hoff, "Multiplicative latent factor models for description and prediction of social networks," *Computational and Mathematical Organization Theory*, vol. 15, no. 4, pp. 261–272, Dec. 2009.

[12] J. Zhu, "Max-margin nonparametric latent feature models for link prediction." in *Proceedings of the 29th International Conference on Machine Learning*, ser. ICML '12. Omnipress, 2012.

[13] J. M. Hofman and C. H. Wiggins, "Bayesian Approach to Network Modularity," *Physical Review Letters*, vol. 100, no. 25, pp. 258 701+, Jun. 2008.

[14] T. H. McCormick, M. J. Salganik, and T. Zheng, "How Many People Do You Know?: Efficiently Estimating Personal Network Size," *Journal*

of the American Statistical Association, vol. 105, no. 489, pp. 59–70, Mar. 2010.
[15] M. Kim and J. Leskovec, "The network completion problem: Inferring missing nodes and edges in networks," in *SDM*. SIAM / Omnipress, 2011, pp. 47–58.
[16] J. Leskovec, D. Chakrabarti, J. M. Kleinberg, C. Faloutsos, and Z. Ghahramani, "Kronecker graphs: An approach to modeling networks," *Journal of Machine Learning Research*, vol. 11, pp. 985–1042, 2010.
[17] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum Likelihood from Incomplete Data via the EM Algorithm," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 39, no. 1, pp. 1–38, 1977.
[18] S. Hanneke and E. P. Xing, "Network completion and survey sampling," *Journal of Machine Learning Research - Proceedings Track*, vol. 5, pp. 209–215, 2009.
[19] E. Candes and B. Recht, "Exact matrix completion via convex optimization," *Foundations of Computational Mathematics*, vol. 9, no. 6, pp. 717–772, 2009.
[20] A. Jalali, Y. Chen, S. Sanghavi, and H. Xu, "Clustering partially observed graphs via convex optimization," in *Proceedings of the 28th International Conference on Machine Learning*, ser. ICML '11. Omnipress, 2011, pp. 1001–1008.
[21] J. Yang and J. Leskovec, "Overlapping community detection at scale: a nonnegative matrix factorization approach," in *Proceedings of the sixth ACM international conference on Web search and data mining*, ser. WSDM '13. ACM, 2013, pp. 587–596.
[22] C.-J. Hsieh, K.-Y. Chiang, and I. S. Dhillon, "Low rank modeling of signed networks." in *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2012, pp. 507–515.
[23] M. Gomez-Rodriguez, J. Leskovec, and A. Krause, "Inferring networks of diffusion and influence," *TKDD*, vol. 5, no. 4, p. 21, 2012.
[24] P. Siyari, H. R. Rabiee, M. Salehi, and M. E. Mehdiabadi, "Network reconstruction under compressive sensing," in *Proceedings of ASE International Conference on Social Informatics*. IEEE Computer Society, 2012, pp. 19–25.
[25] E. Candes, J. Romberg, and T. Tao, "Stable signal recovery from incomplete and inaccurate measurements," *Communications on Pure and Applied Mathematics*, vol. 59, no. 8, p. 15, 2005.
[26] H. Mahyar, H. R. Rabiee, Z. S. Hashemifar, and P. Siyari, "UCS-WN: An unbiased compressive sensing framework for weighted networks," in *Proceedings of 47th Conference on Information Sciences and Systems, CISS '13*, 2013.
[27] H. Mahyar, H. R. Rabiee, and Z. S. Hashemifar, "UCS-NT: An unbiased compressive sensing framework for network tomography," in *Proceedings of 38th International Conference on Acoustics, Speech, and Signal Processing, ICASSP '13*, 2013.
[28] A. K. Menon and C. Elkan, "A log-linear model with latent features for dyadic prediction," in *Proceedings of the 2010 IEEE International Conference on Data Mining*, ser. ICDM '10. IEEE Computer Society, 2010, pp. 364–373.
[29] C. A. Cameron and P. K. Trivedi, *Regression Analysis of Count Data (Econometric Society Monographs)*. Cambridge, United Kingdom: Cambridge University Press, Sep. 1998.
[30] I. Psorakis, S. J. Roberts, M. Ebden, and B. Sheldon, "Overlapping community detection using bayesian nonnegative matrix factorization." *Physical Review E*, vol. 83, no. 6, 2011.
[31] D. Wind and M. Morup, "Link prediction in weighted networks," in *Machine Learning for Signal Processing (MLSP), 2012 IEEE International Workshop on*, 2012, pp. 1–6.
[32] C.-J. Hsieh and I. S. Dhillon, "Fast coordinate descent methods with variable selection for non-negative matrix factorization," in *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, ser. KDD '11. ACM, 2011, pp. 1064–1072.
[33] S. Boyd and L. Vandenberghe, *Convex Optimization*. New York, NY, USA: Cambridge University Press, 2004.
[34] D. Bu, Y. Zhao, L. Cai, H. Xue, X. Zhu, H. Lu, J. Zhang, S. Sun, L. Ling, N. Zhang, G. Li, and R. Chen, "Topological structure analysis of the protein-protein interaction network in budding yeast," *Nucleic Acids Research*, vol. 31, pp. 2443–2450, 2003.
[35] M. Fire, R. Puzis, and Y. Elovici, "Organizational mining using online social networks," *ArXiv Preprint*, 2012.
[36] J. Leskovec, J. M. Kleinberg, and C. Faloutsos, "Graph evolution: Densification and shrinking diameters," *TKDD*, vol. 1, no. 1, 2007.
[37] M. Ripeanu, A. Iamnitchi, and I. T. Foster, "Mapping the gnutella network," *IEEE Internet Computing*, vol. 6, no. 1, pp. 50–57, 2002.
[38] L. Lu and T. Zhou, "Link prediction in complex networks: A survey," *Physica A: Statistical Mechanics and its Applications*, vol. 390, no. 6, pp. 1150 – 1170, 2011.
[39] T. Fawcett, "An introduction to roc analysis," *Pattern Recognition Letters*, vol. 27, no. 8, pp. 861–874, Jun. 2006.
[40] R. N. Lichtenwalter and N. V. Chawla, "Lpmade: Link prediction made easy," *Journal of Machine Learning Research*, vol. 999888, pp. 2489–2492, Nov. 2011.
[41] D. D. Lewis, Y. Yang, T. G. Rose, and F. Li, "Rcv1: A new benchmark collection for text categorization research," *Journal of Machine Learning Research*, vol. 5, no. Apr, pp. 361–397, 2004.

Payam Siyari received his B.Sc. in Computer Science from Shahid Beheshti University, Tehran, Iran, in 2011 and is currently working towards his M.Sc. degree in the Department of Computer Engineering at Sharif University of Technology, Tehran, Iran. His current research interests include Large-scale Optimization Algorithms and Machine Learning especially Mining and Modeling of Social and Information Networks, Signal Processing, Compressive Sensing and its Applications on Network Analysis.

Hamid R. Rabiee (SM07) received his B.S. and M.S. degrees (with great distinction) in Electrical Engineering from CSULB, his EEE in Electrical and Computer Engineering from USC and his Ph.D. in Electrical and Computer Engineering from Purdue University, West Lafayette, in 1996. From 1993 to 1996 he was a Member of Technical Staff at AT&T Bell Laboratories. From 1996 to 1999 he worked as a Senior Software Engineer at Intel Corporation. He was also with PSU, OGI and OSU universities as an adjunct professor of Electrical and Computer Engineering from 1996 to 2000. Since September 2000, he has joined Sharif University of Technology (SUT), Tehran, Iran. He is the founder of Sharif University's Advanced Information and Communication Technology Research Center (AICT), Sharif University Advanced Technologies Incubator (SATI), Sharif Digital Media Laboratory (DML) and Mobile Value Added Services (MVAS) laboratories. He is currently a Professor of Computer Engineering at Sharif University of Technology, and the Director of AICT, DML and MVAS. He has been the initiator and director of national and international level projects in the context of UNDP International Open Source Network (IOSN) and Iran's National ICT Development Plan. He has received numerous awards and honors for his industrial, scientific and academic contributions, and has acted as chairman in a number of national and international conferences, and holds three patents.