Scalable Relational Learning for Large Heterogeneous Networks

Ryan A. Rossi and Rong Zhou
Palo Alto Research Center
{rrossi, rzhou}@parc.com

***Abstract*—Relational models for heterogeneous network data are becoming increasingly important for many real-world applications. However, existing relational learning approaches are not parallel, have scalability issues, and thus unable to handle large heterogeneous network data. In this paper, we propose Parallel Collective Matrix Factorization (PCMF) that serves as a fast and flexible framework for joint modeling of large heterogeneous networks. The PCMF learning algorithm solves for a single parameter given the others, leading to a *parallel scheme* that is fast, flexible, and general for a variety of relational learning tasks and heterogeneous data types. The proposed approach is carefully designed to be (a) efficient for large heterogeneous networks (linear in the total number of observations from the set of input matrices), (b) flexible as many components are interchangeable and easily adaptable, and (c) effective for a variety of applications as well as for different types of data. The experiments demonstrate the scalability, flexibility, and effectiveness of PCMF. For instance, we show that PCMF outperforms a recent state-of-the-art parallel approach in runtime, scalability, and prediction quality. Finally, the effectiveness of PCMF is shown on a number of relational learning tasks such as serving predictions in a real-time streaming fashion.**

I. Introduction

Given heterogeneous network data consisting of one or more networks (and/or attribute data), we want to automatically learn a joint model that can be used for a variety of relational learning tasks. More specifically, we require that the learning algorithm be efficient and parallel for large-scale heterogeneous networks, while also flexible and general for modeling a variety of heterogeneous data types. To this end, we propose Parallel Collective Matrix Factorization (PCMF) that serves as a fast and flexible *parallel framework* for jointly modeling large heterogeneous network data. PCMF jointly decomposes a number of matrices into a set of low-rank factors that approximate the original input matrices. Furthermore, PCMF is useful for modeling (a) sparse and dense matrices, (b) heterogeneous networks consisting of multiple node and edge types as well as (c) dense feature and similarity matrices.

At the heart of PCMF lies an efficient *parallel learning algorithm* that analytically solves for one parameter at a time, given the others. The learning algorithm of PCMF is generalized for jointly modeling an arbitrary number of matrices (network data or feature matrices). In addition, we propose a fast parallel learning algorithm that enables PCMF to model extremely large heterogeneous network data. Furthermore, the parallel learning algorithm of PCMF is extremely *flexible* as many components are interchangeable and can be customized for specific relational learning tasks. One important advantage of PCMF lies in the flexibility of choosing when and how parameters are selected and optimized. Our approach also has other benefits such as its ability to handle data that is extremely sparse. Despite the difficulty of this problem, PCMF leverages additional information such as the social network or other known information to improve prediction quality.

The experiments demonstrate the effectiveness of PCMF for jointly modeling heterogeneous network data. In particular, PCMF as well as our single matrix variant PCMF-BASIC outperforms the recent state-of-the-art in terms of the following: (1) runtime, (2) scalability and parallel speedup, and (3) prediction quality. Furthermore, even the most basic PCMF variant called PCMF-BASIC (the single matrix variant) is significantly faster and more scalable than the state-of-the-art parallel approach for recommender systems called CCD++ [1]. That approach works for single matrix factorizations only, and thus cannot handle the types of data that PCMF can handle.

Besides factorizing an arbitrary number of sparse networks, PCMF also allows for a set of attributes/features for each type of node. The set of attributes (for each node type) are represented as a dense matrix, whereas the network data is stored as a sparse matrix (nonzeros only). The generalized PCMF framework is also used for a variety of large-scale heterogeneous relational learning tasks. Due to limitations of existing work, many of these relational learning tasks have only been explored for relatively simple and small network data. Nevertheless, PCMF is shown to be effective and fast for solving a number of these important and novel relational learning tasks for large heterogeneous network data, including: edge role discovery (and collective role discovery), estimating relationship/link strength, collective/heterogeneous link prediction, improving relational classification, and graph-based representation and deep learning.

The main contributions of this work are as follows:

- *Novel algorithm*. We propose PCMF—a fast, parallel relational learning approach for jointly modeling a variety of heterogeneous network data sources. At the heart of PCMF lies a fast parallel optimization scheme that updates a single parameter at a time, given all others.
- *Effectiveness*: The experiments demonstrate the accuracy and predictive quality of PCMF for a variety of network types and predictive modeling tasks.
- *Scalability*: The runtime is linear with respect to the number of nonzero elements in all matrices.
- *Generality*: We demonstrate the generality of PCMF by applying it for a variety of relational learning tasks and network types.

PCMF has many other contributions/novel features which are detailed throughout the paper, including memory and thread layouts, careful memory access patterns to fully utilize available cache lines, the importance of selecting "good" initial matrices, ordering strategies, and other techniques to minimize dynamic load balancing issues, among many others.

II. Related Work

Related research is categorized and discussed below.

Recommender systems. Majority of research in recommender systems focus on analyzing the user-item interactions [2]. Matrix factorization methods have become increasingly popular as they usually outperform traditional approaches such as k-nearest neighbor [3]. Despite this improvement, these methods have many limitations including scalability and sparsity issues, as well as their inability to leverage additional contextual/side-information.

Some recent research has proposed *parallel recommender systems* based on parallelizing traditional matrix factorization techniques that analyze user-item interactions [1], [4]–[9]. PCMF is fundamentally different from that work. In particular, (a) those approaches are focused on recommender systems, (b) they are designed for user-item matrices that are sparse, and therefore do not generalize to the types and characteristics of data handled by PCMF, and (c) they work only for matrix factorization, and thus unable to handle multiple networks as well as the variety of data types that PCMF models. Nevertheless, we show empirically that a single-matrix factorization variant of PCMF called PCMF-BASIC outperforms CCD++ (the recent state-of-the-art parallel method) in runtime, parallel speedup/scalability, and predictive quality.

There is another related body of work in recommender systems, that leverage additional information to improve recommendations [10]–[15]. PCMF is different than this work, as these approaches focus on improving the quality of predictions, and have focused solely on the problem of recommendation systems. In particular, the PCMF learning algorithm analytically solves for each parameter given the others. This gives rise to (a) an extremely flexible learning algorithm (e.g., update order), that is (b) scalable with a runtime that is linear in the total number of observed values, and (c) generalized for solving a number of relational learning tasks and data types. Additionally, PCMF is a completely *parallel scheme* for jointly modeling large-scale heterogeneous networks. Existing work has focused on improving the quality of recommendations, ignoring issues with efficiency and parallel algorithms all together.

Mining heterogeneous networks. A number of approaches have been proposed for mining heterogeneous networks [16], including path-based similarity search [17], and many other [18], [19]. PCMF builds upon this work in numerous ways. Most previous work has focused on quality and other measures of usefulness, while this work focuses on scalability and efficiency issues. Existing work is not parallel, and has mostly focused on quality and various other measures of usefulness. In contrast, PCMF is parallel and scales to large heterogeneous networks, and shown empirically to scale better than recent methods while also resulting in better quality predictions. Moreover, we propose a *parallel learning algorithm* for modeling large heterogeneous networks that is both general and flexible for a variety of applications and data types.

Role discovery. We propose using PCMF for role discovery. Roles have been used to investigate a variety of networks including social, biological, and technological [20]–[22]. In [22], the authors propose feature-based roles for nodes along with a computational framework for computing them. PCMF is different than existing role discovery methods. In particular, (a) PCMF is parallel and able to scale to large networks; (b) PCMF jointly factorizes heterogeneous network data as well as available attributes; (c) PCMF is an edge-based role discovery method.

Link prediction. The existence of links have been predicted for a number of data mining tasks [23]. In [24], the authors predict future friendship links in social networks using topological features. In [25], the authors predict missing and noisy hyperlinks from the web graph to improve the quality of search engines. In [26], the authors predict links to enhance relational representation for relational learning tasks such as classification. There has also been some work on predicting links in heterogeneous networks [27]–[29]. PCMF is different from these approaches. In particular, (1) PCMF is a *parallel* heterogeneous link prediction method; (2) PCMF is scalable (linear in the number of observed links); (3) PCMF learning algorithm analytically solves for each model parameter independently.

III. Parallel Collective Matrix Factorization

Given two matrices $\mathbf{A} \in \mathbb{R}^{m \times n}$ and $\mathbf{B} \in \mathbb{R}^{m \times m}$, one can formulate the collective factorization problem as:

$$\min_{\mathbf{U},\mathbf{V},\mathbf{Z}} \sum_{(i,j)\in\,\Omega_{\mathbf{A}}} (A_{ij} - \mathbf{u}_i^\top \mathbf{v}_j)^2 + \lambda_u \|\mathbf{U}\|_F + \lambda_v \|\mathbf{V}\|_F + \sum_{(i,j)\in\,\Omega_{\mathbf{B}}} (B_{ij} - \mathbf{u}_i^\top \mathbf{z}_j)^2 + \alpha \|\mathbf{Z}\|_F \quad (1)$$

where $\mathbf{U} \in \mathbb{R}^{m \times d}$, $\mathbf{V} \in \mathbb{R}^{n \times d}$, and $\mathbf{Z} \in \mathbb{R}^{m \times d}$ are low-rank factor matrices. Note that $\mathbf{B} \in \mathbb{R}^{m \times m}$ for simplicity, but may also be asymmetric. Further, let $\Omega_i := \{j : (i,j) \in \Omega\}$ be the set of nonzero indices of the i^{th} row whereas $\bar{\Omega}_j := \{i : (i,j) \in \Omega\}$ denotes the set of nonzero indices for the j^{th} column. The goal is to approximate the incomplete matrix $\mathbf{A}$ by $\mathbf{U}\mathbf{V}^\top$ and $\mathbf{B}$ by $\mathbf{U}\mathbf{Z}^\top$. Thus, in this particular case, $\mathbf{U} \in \mathbb{R}^{m \times d}$ represents the *shared* low dimensional *latent feature space*. Each row $\mathbf{u}_i^\top \in \mathbb{R}^d$ of $\mathbf{U}$ can be interpreted as a low dimensional rank-d embedding of the i^{th} row in $\mathbf{A}$. Alternatively, each row $\mathbf{v}_j^\top \in \mathbb{R}^d$ of $\mathbf{V}$ represents a d-dimensional embedding of the j^{th} column in $\mathbf{A}$ using the same low rank-d dimensional space. Also, $\mathbf{u}_k \in \mathbb{R}^m$ is the k^{th} column of $\mathbf{U}$ and similarly $\mathbf{v}_k \in \mathbb{R}^n$ is the k^{th} column of $\mathbf{V}$. Further, let U_{ik} be a scalar (k^{th} element of $\mathbf{u}_i^\top$ or the i^{th} element of $\mathbf{u}_k$). Similarly, let u_{ik} and v_{jk} for $1 < k < d$ denote the k^{th} coordinate of the column vectors $\mathbf{u}_i$ and $\mathbf{v}_j$ (and thus interchangeable with U_{ik} and V_{jk}). For clarity, we also use $\mathbf{U}_{:k}$ to denote the k^{th} column of $\mathbf{U}$ (and $\mathbf{U}_{i:}$ for the i^{th} row of $\mathbf{U}$). Similar notation is used for $\mathbf{Z}$.

To measure the quality of our model, we use a nonzero squared loss: $(A_{ij} - \mathbf{u}_i^\top \mathbf{v}_j)^2$, though any arbitrary separable loss may be used. Regularization terms are also introduced in order to prevent overfitting. While this work mainly uses square-norm regularization, PCMF is easily adapted for various other regularizers.

A. Learning Algorithm

The PCMF learning algorithm optimizes a single parameter at a time, while fixing all others. Moreover, PCMF analytically solves for each model parameter independently. This gives rise to a PCMF learning algorithm that is (a) fast and efficient, (b) general for a variety of data and characteristics, and (c) parallel for modeling large-scale heterogeneous networks. Further, as we shall see, the parallel learning algorithm proposed in Section III-B is extremely flexible as it allows us to solve the parameters in parallel, while not restricting us to solve for the parameters in any specific order.

To start, we randomly initialize $\mathbf{U} \in \mathbb{R}^{m \times d}$, $\mathbf{V} \in \mathbb{R}^{n \times d}$, and $\mathbf{Z} \in \mathbb{R}^{m \times d}$ and apply a sparsification technique. As shown later, this speeds up convergence while also improving the model accuracy by generalizing better (in fewer iterations). For each inner iteration denoted t, we alternatively update $\mathbf{V}_{:k}$ and $\mathbf{Z}_{:k}$ and then use this to update $\mathbf{U}_{:k}$ and repeat. In particular, a single inner iteration updates the k^{th} latent feature of $\mathbf{V}$, $\mathbf{Z}$ and $\mathbf{U}$ in the following order:

$$\overbrace{\underbrace{V_{1k}, V_{2k}, ..., V_{nk}}_{\mathbf{V}_{:k}}, \underbrace{Z_{1k}, Z_{2k}, ..., Z_{mk}}_{\mathbf{Z}_{:k}}, \underbrace{U_{1k}, U_{2k}, ..., U_{mk}}_{\mathbf{U}_{:k}}}^{\text{inner iteration}} \tag{2}$$

and thus, each outer-iteration updates the latent features in the following order:

$$\overbrace{\underbrace{\mathbf{V}_{:1}, \mathbf{Z}_{:1}, \mathbf{U}_{:1}}_{\text{inner iteration}}, ..., \underbrace{\mathbf{V}_{:k}, \mathbf{Z}_{:k}, \mathbf{U}_{:k}}_{\forall t=1,2,...}, ..., \underbrace{\mathbf{V}_{:d}, \mathbf{Z}_{:d}, \mathbf{U}_{:d}}_{\forall t=1,2,...}}^{\tau \text{ outer iteration}} \tag{3}$$

Note that this is in contrast to the common update ordering: $\mathbf{V}_{:1}, \mathbf{V}_{:2}, ..., \mathbf{V}_{:d}$, $\mathbf{Z}_{:1}, \mathbf{Z}_{:2}, ..., \mathbf{Z}_{:d}$, $\mathbf{U}_{:1}, \mathbf{U}_{:2}, ..., \mathbf{U}_{:d}$. We have also experimented with different update strategies such as: $\mathbf{Z}_{:1}, \mathbf{V}_{:1}, \mathbf{U}_{:1}$,..., $\mathbf{Z}_{:d}, \mathbf{V}_{:d}, \mathbf{U}_{:d}$, among more sophisticated adaptive ordering variants.

We now give the update rules for a single element. We allow V_{jk} to change with v and fix all other variables to solve the following one-variable subproblem:

$$\min_{\mathrm{v}} J(\mathrm{v}) = \sum_{i \in \Omega_j^{\mathbf{A}}} \Big(A_{ij} - (\mathbf{U}_{i:}\mathbf{V}_{j:}^\top - U_{ik}V_{jk}) - U_{ik}\mathrm{v} \Big)^2 + \lambda_v \mathrm{v}^2$$

Since $J(\mathrm{v})$ is a univariate quadratic function, the unique solution is:

$$\mathrm{v}^\star = \frac{\sum_{i \in \Omega_j^{\mathbf{A}}} \big(A_{ij} - \mathbf{U}_{i:}\mathbf{V}_{j:}^\top + U_{ik}V_{jk} \big) U_{ik}}{\lambda + \sum_{i \in \Omega_j} U_{ik}U_{ik}}$$

The update is computed efficiently taking $O(|\Omega_j^{\mathbf{A}}|)$ linear time by carefully maintaining the residual matrix $\mathbf{E}^{\mathrm{a}}$:

$$E_{ij}^a \equiv A_{ij} - \mathbf{U}_{i:}\mathbf{V}_{j:}^\top, \quad \forall (i,j) \in \Omega^{\mathbf{A}}$$

Therefore, we can rewrite the above equation in terms of E_{ij}^a, the optimal $\mathrm{v}^\star$ is simply:

$$\mathrm{v}^\star = \frac{\sum_{i \in \Omega_j} \big(E_{ij}^a + U_{ik}V_{jk} \big) U_{ik}}{\lambda + \sum_{i \in \Omega_j} U_{ik}U_{ik}} \tag{4}$$

Now, we update V_{jk} and E_{ij}^a in $O(|\Omega_j|)$ time using

$$E_{ij}^a \leftarrow E_{ij}^a - (\mathrm{v}^\star - V_{jk})U_{ik}, \quad \forall_i \in \Omega_j \tag{5}$$
$$V_{jk} \leftarrow \mathrm{v}^\star \tag{6}$$

Similar update rules for solving a single subproblem in $\mathbf{Z}$ and $\mathbf{U}$ are straightforward to derive and may be computed efficiently using the same trick. Thus, the update rules for the one-variable subproblems V_{jk}, Z_{ik}, and U_{ik} are as follows:

$$\mathrm{v}^\star = \frac{\sum_{i \in \Omega_j^{\mathbf{A}}} \big(E_{ij}^a + U_{ik}V_{jk} \big) U_{ik}}{\lambda_v + \sum_{i \in \Omega_j^{\mathbf{A}}} U_{ik}U_{ik}} \tag{7}$$

$$\mathrm{z}^\star = \frac{\sum_{j \in \Omega_i^{\mathbf{B}}} \big(E_{ij}^b + U_{jk}Z_{ik} \big) U_{jk}}{\alpha + \sum_{j \in \Omega_i^{\mathbf{B}}} U_{jk}U_{jk}} \tag{8}$$

$$\mathrm{u}^\star = \frac{\sum_{j \in \Omega_i^{\mathbf{A}}} \big(E_{ij}^a + U_{ik}V_{jk} \big) V_{jk}}{\lambda_u + \sum_{j \in \Omega_i^{\mathbf{A}}} V_{jk}V_{jk}} + \frac{\sum_{j \in \Omega_i^{\mathbf{B}}} \big(E_{ij}^b + U_{ik}Z_{jk} \big) Z_{jk}}{\alpha + \sum_{j \in \Omega_i^{\mathbf{B}}} Z_{jk}Z_{jk}} \tag{9}$$

These updates rules may be used regardless of the order in which the one-variable subproblems are updated.

Although we update $\mathbf{V}$, $\mathbf{Z}$, and $\mathbf{U}$ via column-wise updates, the order in which the one-variable updates are performed may also impact complexity and convergence properties. The factorization using a column-wise update sequence corresponds to the summation of outer-products: $\mathbf{A} \approx \mathbf{U}\mathbf{V}^\top = \sum_{k=1}^d \mathbf{U}_{:k}\mathbf{V}_{:k}^\top$ and $\mathbf{B} \approx \mathbf{U}\mathbf{Z}^\top = \sum_{k=1}^d \mathbf{U}_{:k}\mathbf{Z}_{:k}^\top$ where $\mathbf{V}_{:k}$, $\mathbf{Z}_{:k}$ and $\mathbf{U}_{:k}$ denote the k^{th} column (or latent feature) of $\mathbf{V}$, $\mathbf{Z}$ and $\mathbf{U}$, respectively. Let $\mathrm{T}_{\text{outer}}$ and $\mathrm{T}_{\text{inner}}$ be the number of inner and outer iterations, respectively. Note $\mathrm{T}_{\text{outer}}$ is the number of times the k^{th} latent feature is updated. $\mathrm{T}_{\text{inner}}$ is the number of times the k^{th} latent feature is updated before updating the $k+1$ latent feature. Furthermore if the update order for each outer iteration is: $\mathbf{V}_{:1}, \mathbf{Z}_{:1}, \mathbf{U}_{:1}, \ldots, \mathbf{V}_{:k}, \mathbf{Z}_{:k}, \mathbf{U}_{:k}, \ldots, \mathbf{V}_{:d}, \mathbf{Z}_{:d}, \mathbf{U}_{:d}$, then each of the d latent features are updated at each outer-iteration (via $\mathrm{T}_{\text{inner}}$ inner-iterations). Since elements in $\mathbf{v}_k$ (or $\mathbf{z}_k$, $\mathbf{z}_k$) can be computed independently, we focus on scalar approximations for each individual element. For now, let us consider column-wise updates where the k^{th} latent feature of $\mathbf{V}$, $\mathbf{Z}$, and $\mathbf{U}$ is selected and updated in arbitrary order. See Alg 1 for a detailed description. Thus, during each inner iteration, we perform the following updates: $\mathbf{V}_{:k} \leftarrow \mathbf{v}^\star$, $\mathbf{Z}_{:k} \leftarrow \mathbf{z}^\star$, $\mathbf{U}_{:k} \leftarrow \mathbf{u}^\star$ where $\mathbf{v}^\star$, $\mathbf{z}^\star$, and $\mathbf{u}^\star$ are obtained via the inner iterations. For efficiency, PCMF performs rank-1 updates in-place so that we always use the current estimate. To obtain the updates above (i.e., $\mathbf{v}^\star$, $\mathbf{z}^\star$, $\mathbf{u}^\star$) one must solve the subproblem:

$$\min_{\mathbf{u} \in \mathbb{R}^m, \mathbf{v} \in \mathbb{R}^n, \mathbf{z} \in \mathbb{R}^m} \Big\{ \sum_{(i,j) \in \Omega^{\mathbf{A}}} \big(E_{ij}^a + U_{ik}^{(\tau)}V_{jk}^{(\tau)} - u_{ik}^\star v_{jk}^\star \big)^2 + \sum_{(i,j) \in \Omega^{\mathbf{B}}} \big(E_{ij}^b + U_{ik}^{(\tau)}Z_{jk}^{(\tau)} - u_i z_j \big)^2 + \lambda_u \|\mathbf{u}\|^2 + \lambda_v \|\mathbf{v}\|^2 + \alpha \|\mathbf{z}\|^2 \Big\} \tag{10}$$

where $\mathbf{E}^{\mathrm{a}} = \mathbf{A} - \mathbf{u}_k\mathbf{v}_k^\top$ and $\mathbf{E}^{\mathrm{b}} = \mathbf{B} - \mathbf{u}_k\mathbf{z}_k^\top$ are the initial sparse residual matrices for $\mathbf{A}$ and $\mathbf{B}$, respectively. The residual term for $\mathbf{A}$ is denoted as $\mathbf{A} - \mathbf{U}\mathbf{V}^\top = \mathbf{A}^{(k)} - \mathbf{u}_k\mathbf{v}_k^\top$ where

the k-residual $\mathbf{A}^{(k)}$ is defined as:

$$\mathbf{A}^{(k)} = \mathbf{A} - \sum_{f \neq k} \mathbf{u}_f \mathbf{v}_f^\top = \mathbf{A} - \mathbf{U}\mathbf{V}^\top + \mathbf{u}_k \mathbf{v}_k^\top, \text{ for } k = 1, 2, ..., d \quad (11)$$

Similarly, the residual term for $\mathbf{B}$ is $\mathbf{B} - \mathbf{U}\mathbf{Z}^\top = \mathbf{B}^{(k)} - \mathbf{u}_k \mathbf{z}_k^\top$ where the k-residual $\mathbf{B}^{(k)}$ is defined as:

$$\mathbf{B}^{(k)} = \mathbf{B} - \sum_{f \neq k} \mathbf{u}_f \mathbf{z}_f^\top = \mathbf{B} - \mathbf{U}\mathbf{Z}^\top + \mathbf{u}_k \mathbf{z}_k^\top, \text{ for } k = 1, 2, ..., d \quad (12)$$

For a single residual entry, let us define $A_{ij}^{(k)}$ and $B_{ij}^{(k)}$ as:

$$A_{ij}^{(k)} = E_{ij}^a + U_{ik}V_{jk}, \ \forall (i,j) \in \Omega^{\mathbf{A}} \quad \text{and} \quad (13)$$

$$B_{ij}^{(k)} = E_{ij}^b + U_{ik}Z_{jk}, \ \forall (i,j) \in \Omega^{\mathbf{B}} \quad (14)$$

Equivalently, let $\mathbf{A}^{(k)} = \mathbf{E}^{\mathrm{a}} + \mathbf{u}_k \mathbf{v}_k^\top$ and $\mathbf{B}^{(k)} = \mathbf{E}^{\mathrm{b}} + \mathbf{u}_k \mathbf{z}_k^\top$. Now a straightforward rewriting of Eq. 10:

$$\min_{\mathbf{u},\mathbf{v},\mathbf{z}} \Big\{ \sum_{(i,j)\in\Omega^{\mathbf{A}}} \left(A_{ij}^{(k)} - u_{ik}v_{jk}\right)^2 + \lambda_v \|\mathbf{v}\|^2 + \sum_{(i,j)\in\Omega^{\mathbf{B}}} \left(B_{ij}^{(k)} - u_{ik}z_{jk}\right)^2 + \lambda_u \|\mathbf{u}\|^2 + \alpha \|\mathbf{z}\|^2 \Big\} \quad (15)$$

Using Eq. 15 gives an approximation by alternating between updating $\mathbf{v}$, $\mathbf{z}$, and $\mathbf{u}$ via column-wise updates. Note that when performing rank-1 updates, a single subproblem can be solved without any further residual maintenance. Thus, each one-variable subproblem may benefit from $\mathrm{T}_{\mathrm{inner}}$ iterations. The inner iterations are fast to compute since we avoid updating the residual matrices while iteratively updating a given k latent factor via a number of inner iterations. As previously mentioned, updates are performed in-place and thus the most recent estimates are leveraged. This also has other important consequences as it reduces storage requirements, memory locality, etc. Furthermore, after the $\mathrm{T}_{\mathrm{inner}}$ inner iterations, we update $\mathbf{E}^{\mathrm{a}}$ and $\mathbf{E}^{\mathrm{b}}$,

$$E_{ij}^a \leftarrow A_{ij}^{(k)} - \mathrm{u}_i^\star \mathrm{v}_j^\star, \ \forall (i,j) \in \Omega^{\mathbf{A}} \quad \text{and} \quad (16)$$

$$E_{ij}^b \leftarrow B_{ij}^{(k)} - \mathrm{u}_i^\star \mathrm{z}_j^\star, \ \forall (i,j) \in \Omega^{\mathbf{B}} \quad (17)$$

For convenience, we also define $\mathbf{E}^{\mathrm{a}} = \mathbf{A}^{(k)} - \mathbf{u}_k \mathbf{v}_k^\top$ and $\mathbf{E}^{\mathrm{b}} = \mathbf{B}^{(k)} - \mathbf{u}_k \mathbf{z}_k^\top$. Finally, the PCMF learning algorithm updates each element independently via the following update rules:

$$V_{jk} = \frac{\sum_{i \in \Omega_j^{\mathbf{A}}} A_{ij}^{(k)} U_{ik}}{\lambda_v + \sum_{i \in \Omega_j^{\mathbf{A}}} U_{ik}U_{ik}}, \ j = 1, 2, ..., n \quad (18)$$

$$Z_{ik} = \frac{\sum_{j \in \Omega_i^{\mathbf{B}}} B_{ij}^{(k)} U_{jk}}{\alpha + \sum_{j \in \Omega_i^{\mathbf{B}}} U_{jk}U_{jk}}, \ i = 1, 2, ..., m \quad (19)$$

$$U_{ik} = \frac{\sum_{j \in \Omega_i^{\mathbf{A}}} A_{ij}^{(k)} V_{jk}}{\lambda_u + \sum_{j \in \Omega_i^{\mathbf{A}}} V_{jk}V_{jk}} + \frac{\sum_{j \in \Omega_i^{\mathbf{B}}} B_{ij}^{(k)} Z_{jk}}{\alpha + \sum_{j \in \Omega_i^{\mathbf{B}}} Z_{jk}Z_{jk}}, i = 1, ..., m \quad (20)$$

Thus, the above update rules perform n element-wise updates on $\mathbf{v}_k$, then we perform m updates on $\mathbf{z}_k$, and finally m updates are performed for $\mathbf{u}_k$. Furthermore that approach does not define an element-wise update strategy (assumes a natural ordering given as input) nor does it allow for partial updates. For instance, one may update a single element in V_{jk}, then Z_{ik}, and U_{ik} and continue rotating until all elements have been updated once.

To avoid overfitting, we leverage the following weighted regularization term that penalizes large parameters: $\lambda_u \sum_{i=1}^m |\Omega_i| \cdot \|\mathbf{u}_i\|^2 + \lambda_v \sum_{j=1}^n |\Omega_j| \cdot \|\mathbf{v}_j\|^2 + \alpha \sum_{i=1}^m |\Omega_i^{\mathbf{B}}| \cdot \|\mathbf{z}_i\|^2$ where $|\cdot|$ is the cardinality of a set (i.e., number of nonzeros in a row or col of $\mathbf{A}$ or $\mathbf{B}$) and $\|\cdot\|^2$ is the L_2 vector norm. Further, a simple yet effective graph sparsifier is used for speeding up the PCMF framework as seen in Section IV. On many problems we have observed faster convergence and consequently better predictions in fewer iterations.

B. Parallel Scheme

Using the proposed PCMF learning algorithm as a basis, we derive a parallel scheme for the PCMF framework that is general for expressing many of the important PCMF variants (such as non-negative PCMF). As shown previously, PCMF optimizes the objective function in (15) one element at a time. A fundamental advantage of this approach is that all such one-variable subproblems of a k latent feature are independent and can be updated simultaneously. Alg. 1 serves as a basis for studying different objective functions/regularizers, ordering strategies, asynchronous updates, among many other variations.

A key advantage of PCMF lies in it's flexibility to choose the order in which updates are performed. This additional flexibility results in a significantly faster parallelization with shorter wait times between respective updates. Let $\pi_v = \{v_1, ..., v_n\}$, $\pi_z = \{z_1, ..., z_m\}$, $\pi_u = \{u_1, ..., u_m\}$ denote an ordering of the rows in $\mathbf{V}$, $\mathbf{Z}$, and $\mathbf{U}$, respectively. For now, π_v, π_z, and π_u are assumed to be independently permuted in an arbitrary manner (largest degree, k-core, max error, etc.). From this, let us denote Π as:

$$\Pi = \{\underbrace{v_1, \cdots, v_n}_{\pi_v^{(t)}}, \underbrace{z_1, \cdots, z_m}_{\pi_z^{(t)}}, \underbrace{u_1, \cdots, u_m}_{\pi_u^{(t)}}\} \quad (21)$$

where t denotes the inner iteration. This implies that one-variable updates are performed in the order given by π_v, then π_z, and

Algorithm 1 PCMF Framework

```
Initialize U, V, and Z uniformly at random
Obtain an initial ordering Π
Set E^a = A and E^b = B
repeat                                   ▷ outer iterations τ = 1, 2, ..., T_outer
    for k = 1, 2, ..., d do
        Compute A^(k) and B^(k) in parallel asynch. via (13), (14)
        for t = 1, 2, ..., T_inner do
            parallel for next b jobs in Π in order do
                while worker w has updates to perform in local queue do
                    Obtain parameter to update (dequeue job)
                    Perform update via Eq. (18), (19), (20)
                end while
            end parallel
            Recompute ordering Π (if adaptive ordering strategy)
        end for
        Update E^a and E^b in parallel asynch. via (16), (17)
    end for
until stopping criterion is reached
return factor matrices V, Z, and U
```

so on. Hence, the next b row indices (i.e., vertices) to update are selected in a dynamic fashion from the Π ordering above.

More generally, PCMF is flexible for updating individual elements in any order and is not restricted to updating all elements in π_v first (or the k^{th} factor of $\mathbf{v}$) before updating π_z, and thus one can select the elements to update at a finer granularity. This is possible since we are focused on the approximation between a single element $a_{ij}^{(k)}$ in the k-residual matrix and the multiplication of U_{ik} and V_{jk} and similarly we are interested in the approximation between $b_{ij}^{(k)}$ and the multiplication of U_{ik} and Z_{jk}. Therefore, let us redefine Π as an arbitrary permutation of the set $\{v_1, ..., v_n, z_1, ..., z_m, u_1, ..., u_m\}$. The ordering may also change at each inner iteration (adaptive approaches) and is not required to be static (Line 14). For instance, the order may adapt based on the error from the previous inner iteration. Further, we may choose to update only the top-x elements with largest error in the previous iteration. This ensures we focus on the variables that are most likely to improve the objective function and also ensures work is not wasted on fruitless updates that are unlikely to lead to a significant decrease in error.

In PCMF, each of the p workers are initially assigned a disjoint set of b vertices (i.e., row indices) to update. After a worker w completes its jobs (e.g., updating all vertices in its assigned queue/vertex set), Line 8 in Alg 1 dynamically assigns the next set of b rows to update (in order of Π). Thus, we assign jobs dynamically based on the availability of a worker. More formally, each worker $w \in \{1, ..., p\}$ has a local concurrent queue which contains a set of $(j, \mathbf{v}_j)$ pairs to process where $\mathbf{v}_j \in \mathbb{R}^k$. Further, every such pair $(j, \mathbf{v}_j)$ is known as a job and corresponds to a one-variable update (from Section III). Thus a worker w pops the next job off its local queue (Line 10), performs one or more updates (Line 11), and repeats. At the start, each worker $w \in \{1, ..., p\}$ initially pushes b job pairs onto its own queue using the ordering Π. If a worker w's local queue becomes empty, an additional set of b jobs are dynamically pushed into the queue of that worker. The specific set of jobs assigned to the w worker is exactly the next b jobs from the ordering Π (i.e., starting from the job last assigned to the p workers). Note that these jobs correspond to the rows that have yet to be updated in the current (inner) iteration (which are given by the ordering defined above). This dynamic scheduling strategy effectively balances the load as it significantly reduces wait times between updates. Furthermore it has been shown to be significantly faster than the previous state-of-the-art for real-world networks with skewed degree and triangle distributions. The number b of such $(j, \mathbf{v}_j)$ job pairs assigned to a worker w is parameterized in PCMF so that the number of jobs assigned at a given time may be adapted automatically or set by the user.

To store the result from a single update, we index directly into the proper vertex/row position in the array and store the updated value. This has two implications. First we avoid additional storage requirements needed to store the intermediate results as done with previous approaches. Second the result from the update is stored using the same array and thus the current estimate may be used immediately. Let us note that CCD++ requires additional arrays of length m and n to store the intermediate results computed at each iteration. The results are then copied back afterwards.

A key advantage of the proposed PCMF framework is the ability to perform updates asynchronously without synchronization barriers between each rank-one update. Previous work in traditional matrix factorization usually requires that updates are completed in full for $\mathbf{U}_{:k}$ before moving on to $\mathbf{V}_{:k}$ and is explicitly implemented using barriers to ensure synchronization between rank-1 updates. For instance, CCD++ has multiple synchronization barriers at each inner iteration. However, this may cause most workers to wait for long periods while waiting for another worker that is either slow or assigned an extremely skewed work load. We relax such a requirement to allow for the inner iterates to be completely asynchronous. The rank one updates for $\mathbf{V}$, $\mathbf{Z}$, and $\mathbf{U}$ are completely asynchronously, and when there is no more jobs to be assigned to a worker w, that worker immediately grabs the next set of b jobs from the ordering Π. Our approach avoids the inner synchronization all together by dynamically assigning the next b jobs in the ordering Π to the next available worker w regardless of the rank-one update. Thus, workers do not have to remain idle while waiting for other workers to finish. A work-stealing strategy is used so that workers do not remain idle waiting for the slowest to finish. In particular, jobs are pushed into each of the local worker queues, and once a worker completes the assigned jobs, we pop the last k jobs from the queue of the slowest worker and assign them to an available worker. Clearly this parallel approach significantly improves CCD++ as workers are never idle and always making progress by partitioning the jobs assigned to the slowest worker.

A summary of the key advantages over the state-of-the-art:

- *Computations performed asynchronously whenever appropriate*. The inner-iteration is completely asynchronous. We also update the residual matrices in a non-blocking asynchronous fashion and carry out many other computations in a similar manner whenever possible.
- *Flexible ordering strategy allows finer granularity*. Our parallel scheme for PCMF improves load balancing by allowing a flexible ordering strategy to be used. Note that prior work is unable to order the updates at this level of granularity. Adaptive techniques based on max error from the previous iteration are also proposed.
- *Updates performed in-place and current estimates utilized*. Furthermore updates are performed in-place and thus the current estimates are immediately utilized. This also reduces the storage requirements.
- *Memory and thread layout optimized for NUMA architecture*. In addition, we also have optimized PCMF for NUMA architecture [30] using interleaved memory allocations in a round robin fashion between processors. Results are discussed in Section IV-C.
- *Column-wise memory optimization*. An optimized memory scheme for our particular update pattern is used where the latent factor matrices are stored and accessed as a $k \times m$ contiguous block of memory. This scheme exploits memory locality while avoiding caching ping pong and other issues by utilizing memory assignments that carefully align with cache lines.

C. Complexity Analysis

Let $|\Omega^{\mathbf{A}}|$ and $|\Omega^{\mathbf{B}}|$ denote the number of nonzeros in $\mathbf{A}$ and $\mathbf{B}$, respectively (e.g., the user-by-item matrix $\mathbf{A}$ and the user-interaction matrix $\mathbf{B}$). As previously mentioned, d is the number

of latent features in the factorization. For a single iteration, PCMF takes $\mathrm{O}\left(d\left(|\Omega^{\mathbf{A}}|+|\Omega^{\mathbf{B}}|\right)\right)$, and therefore linear in the number of nonzeros in $\mathbf{A}$ and $\mathbf{B}$. Note that in the case that $\mathbf{A}$ and $\mathbf{B}$ represent user-item ratings and social network information, then $|\Omega^{\mathbf{A}}|$ usually dominates $|\Omega^{\mathbf{B}}|$ as it is usually more dense than the social network matrix $\mathbf{B}$. Observe that each iteration that updates $\mathbf{U}$, $\mathbf{V}$, and $\mathbf{Z}$ takes $O\big(d(|\Omega^{\mathbf{A}}|+|\Omega^{\mathbf{B}}|)\big)$, $O(|\Omega^{\mathbf{A}}|d)$, and $O(|\Omega^{\mathbf{B}}|d)$ time, respectively. Hence, the total computational complexity in one iteration is $O\big(d(|\Omega^{\mathbf{A}}|+|\Omega^{\mathbf{B}}|)\big)$. Therefore the runtime of PCMF is linear with respect to the number of nonzero elements in the collection of matrices (network data) given as input. This approach is clearly fast for large heterogeneous networks.

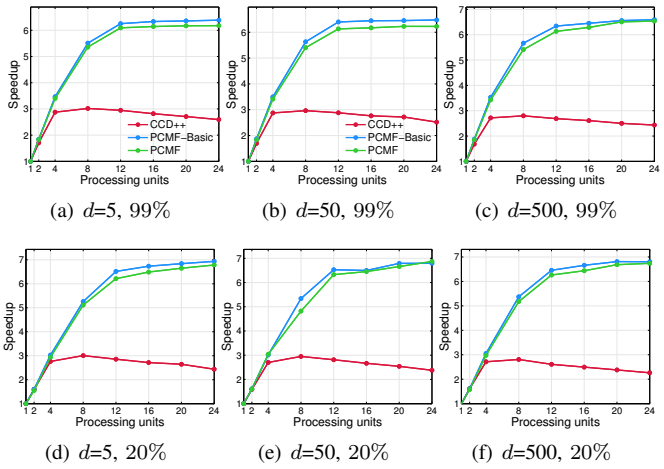
D. A Fast Nonparametric Model

Existing work has one or more parameters that need to be manually selected by the user, which is both time-consuming and expensive for big data. These problems limit the applicability of these techniques in many real-world applications. To overcome these limitations, this work proposes a nonparametric variant of PCMF, that is:

(a) Data-driven, completely automatic, requiring no manual selection/tuning by the user.
(b) Fast relaxation method to efficiently search over the space of models, taking a fraction of the time that PCMF takes.

To search over the space of models from PCMF, one must first choose a model selection criterion. In this work, we primarily used information criterion of Akaike (AIC) [31], though other model selection criterion may also be used in PCMF such as Minimum Description Length (MDL) and many others. The AIC value is $C_{\texttt{AIC}} = 2x - 2\ln(\mathcal{L})$ where x is the number of parameters in the model, that is $x = d(m+n)$, and $\mathcal{L}$ is the maximized likelihood. Thus for a given $\mathbf{U} \in \mathbb{R}^{m\times d}$ and $\mathbf{V} \in \mathbb{R}^{n\times d}$ computed using PCMF with d-dimensions gives $\ln(\mathcal{L}) = -\frac{1}{2\sigma^2}\|\mathbf{A} - \mathbf{U}\mathbf{V}^\top\|_F^2$ where σ^2 is the variance of the error. This criterion balances the trade-off between goodness of fit and the complexity of the model (number of free parameters). Hence, the goodness of fit of a model is penalized by the number of estimated parameters and thus discourages overfitting. The model selected is the one that minimizes the information criterion.

Since efficiency is of fundamental importance, the proposed method avoids precomputing a set of (full PCMF) models (for selection via AIC), and instead leverages the fast relaxation method to effectively search the space of models that are likely to maximize the selection criterion. Hence, we begin computing models using low parameter estimates and gradually increase each parameter until we discover a model that leads to a larger $C_{\texttt{AIC}}$ than found thus far. At this point, one may terminate or continue searching the next few models and terminate if a better model is not found in those few attempts. However, if a better model is found, then the number of failed trials is reset, and the search continues. This last step is to provide more certainty that a global optimum was reached. Note that the method is space-efficient as well as fast, since the relaxation method only requires us to store the best model found thus far. Furthermore, we avoid computing $\|\mathbf{A} - \mathbf{U}\mathbf{V}^\top\|_F^2$ by using the residual matrix $\mathbf{E}^{\mathrm{a}}$ from PCMF.



(a) d=5, 99% (b) d=50, 99% (c) d=500, 99%

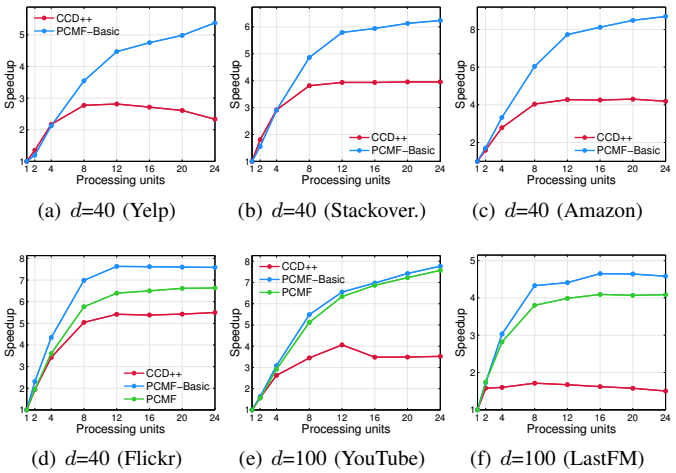
(d) d=5, 20% (e) d=50, 20% (f) d=500, 20%

Fig. 1. **Varying amount of training data and dimensionality of models (epinions).**

Searching over the space of models is expensive and in certain settings may be impractical, even using the relatively fast search technique above. For this reason, we develop a fast relaxation variant of PCMF. Intuitively, the relaxation method performs a few iterations to obtain a fast rough approximation. The method reuses data structures and leverages intermediate computations to improve efficiency when searching the space of models. Furthermore the relaxation method serves as a basis for exploring the space of models defined by the PCMF framework (from Section III). Most importantly, it is shown to be strikingly fast and scalable for large complex heterogeneous networks, yet effective, obtaining near-optimal estimates on the parameters.

Models are learned using the fast relaxation method and our model search technique is used to find the "best performing model" from the infinite model space. We also compared to a naive method that is essentially the vanilla PCMF from Section III using the early termination strategies. Overall, the fast relaxation method is typically 20+ times faster than the naive method, yet is only marginally more accurate than our fast relaxation. For instance, using the eachmovie data, we automatically found a nearly optimal model in only 9.5 seconds compared to 258 seconds using the naive approach. The fast non-parametric relaxation automatically identified a model with $d = 65$ whereas the slower but more accurate approach found $d = 75$. Nevertheless, once the parameters were learned automatically, we then used the corresponding models to predict the unknown test instances and used RMSE to measure the quality of the models. The difference was insignificant as the fast relaxation had 1.108 whereas the much slower approach gave 1.107 and thus the difference in RMSE between the two is insignificant. In a few instances, the model learned from the fast relaxation using AIC was of better quality (lower testing error).

Let us note that in previous work the model is typically selected arbitrarily and typically varies for each dataset [1]. Instead, we perform model selection automatically using AIC and perform a fast relaxation method for computing a rough approximation. As previously mentioned, this data-driven nonparametric PCMF arose from necessity as it is essential for many practical situations and real-time systems (i.e., tuning by users are not possible or expensive and efficiency is critical



(a) d=40 (Yelp) (b) d=40 (Stackover.) (c) d=40 (Amazon)

(d) d=40 (Flickr) (e) d=100 (YouTube) (f) d=100 (LastFM)

Fig. 2. **Speedup of methods from a variety of network types.**

due to the sheer size of the streaming graph data). Note that if external attributes are available, then another approach is to search the space of models and select the one that gives rise to the best performance (i.e., accuracy or application-specific metric). Finally, it is also straightforward to adapt the information criterion above for an arbitrary number of matrices and alternate objective functions.

IV. Experiments

A. Experimental Setup and Evaluation

Platform: For the experiments, we use a 2-processor Intel Xeon X5680 3.33GHz CPU. Each processor has 6 cores (12 hardware threads) with 12MB of L3 cache and each core has 256KB of L2 cache. The server also has 96GB of memory in a NUMA architecture. The PCMF framework is written in C++ and deployed in our high-performance RECommendation PACKage called RECPACK. To compare with the state-of-the-art (CCD++), we used both a single matrix variant called PCMF-BASIC as well as the more general PCMF approach.

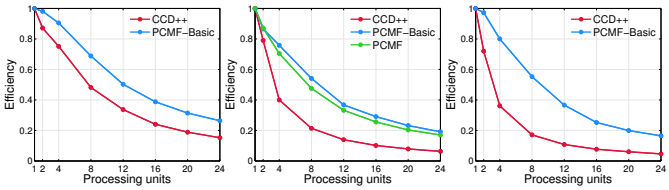
Data: All methods are evaluated on a large diverse collection of data with different properties and types. A complete summary of the data, its semantics, and statistics are provided in the supplementary material[1]. The data used for evaluation is from network repository [32] and accessible online[2,3].

Evaluation Metrics & Comparison: Let Ω^{test} denote the instances in the test set. Given a row i and column j from the test set $(i,j) \in \Omega^{\text{test}}$ (of either $\mathbf{A}$ or $\mathbf{B}$), we predict the value (e.g., rating, group membership, friendship tie) for the $(i,j)^{\text{th}}$ entry in $\mathbf{A}$ (or $\mathbf{B}$) as $\langle \mathbf{u}_i, \mathbf{v}_j \rangle$ where $\langle \cdot, \cdot \rangle$ is the inner Euclidean product of the row and column vectors (e.g., user and item vectors), respectively. To measure the prediction quality (error), we use root mean squared error (RMSE):

$$\sqrt{\frac{\sum_{(i,j)\in\Omega^{\text{test}}} \left(A_{ij} - \langle \mathbf{u}_i, \mathbf{v}_j \rangle\right)^2}{|\Omega^{\text{test}}|}}$$

[1] www.ryanrossi.com/pcmf/supp.pdf
[2] www.networkrepository.com
[3] www.networkrepository.com/graphML

In addition, we also used mean absolute error (MAE) and Normalized RMSE. We set $\lambda_v = \lambda_u = 0.1$ and $\mathrm{T}_{\text{outer}} = \mathrm{T}_{\text{inner}} = 5$, unless otherwise noted. For comparison, we used the exact code from [1].



(a) Dating (b) LastFM (c) EachMovie

Fig. 3. **Efficiency of various methods across a variety of different types of data** ($d = 100$).

B. Parallel Scaling

Figure 1 investigates the scalability of the proposed methods when varying the amount of training data and number of latent dimensions (d). Overall, PCMF and PCMF-BASIC scale significantly better than the recent state-of-the-art, which is consistent across various amounts of training data and parameter settings (Figure 1). Similar results are found using other networks with different characteristics and types (Figure 2).

Figure 3 compares the efficiency of the methods across a variety of network types with different characteristics. Just as before, PCMF variants outperform the others, and in particular, the workers are found to be more effectively utilized (due to improved load balancing, and caching by careful ordering, etc.). We have also used many other benchmark networks in our comparison and in all cases found that PCMF converged faster to the desired error. For instance, in the dating agency data (users-rate-users) it took us 95 seconds compared to 138 given by the best state-of-the-art method, whereas for yelp (users-rate-businesses) it took us 2 seconds compared to 5, and similar times were observed for eachmovie (users-rate-movies). We also experimented with numerous other variants with different update strategies. Regardless of the variant and parameters, the best scalability arises from using one of the PCMF variants.

For comparing the algorithms across a wide range of problems, we utilize performance profiles [33]. Just like ROC curves, the best results lie towards the upper left. For a variety of network problems, PCMF is shown to be significantly faster than the state-of-the-art (Figure 4).

C. Memory and Thread Layout

Since the local memory of a processor can be accessed faster than other memory accesses, we explored two main memory layouts including bounded where memory allocated to the local processor (socket) and interleaving memory allocations in a round robin fashion between processors. Overall, we found the layout of memory to processors had a large effect on scalability and in particular, the best scalability arises from interleaved memory (Figure 5). Interleaved memory typically outperforms the bounded memory layout, regardless of the thread layout strategy used. Additionally, this finding is consistent across various types of data and dimensionality settings. Therefore, PCMF leverages the interleaved memory layout, unless specified

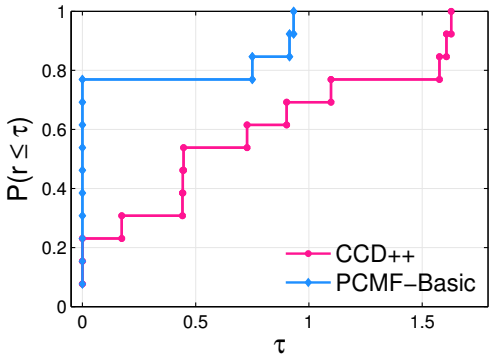
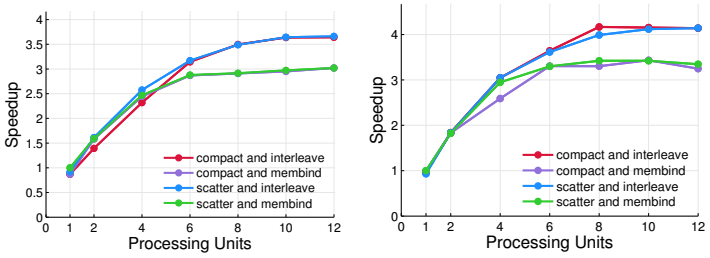
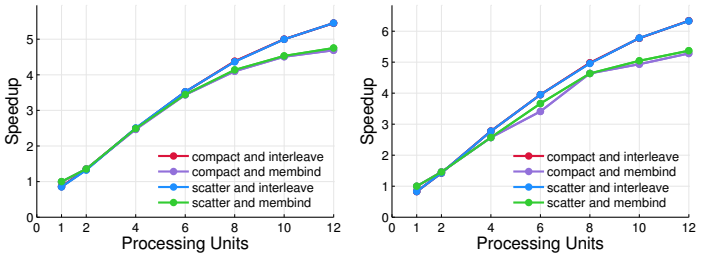


Fig. 4. **Performance profile comparing methods on a number of problem instances.**



(a) LastFM ($d = 40$) (b) EachMovie ($d = 200$)



(c) YouTube ($d = 40$) (d) Amazon ($d = 100$)

Fig. 5. **Comparing the speedup of different memory and thread layouts.**

otherwise. Note that we also experimented with thread layout and found no significant difference.

D. Predictive Quality

Results are summarized in Figure 6 for a variety of different settings and network types. In all cases, we find that PCMF outperforms the baseline as expected. Intuitively, this indicates that modeling the additional matrix collectively is useful and includes useful information that is leveraged in PCMF. Similar results are found when α and λ vary as shown later in Section IV-E. Overall, we find that PCMF improves the accuracy of predictions and the improvement is statistically significant. To understand the accuracy gain with PCMF, we also evaluated variations of PCMF that performed the rank-1 updates in a different order, used different regularization, and a variant that updated entries in a biased manner (top-k entries with largest error). Similar results were observed in majority of cases (plots removed for brevity). Figure 7 investigates the impact of accuracy and runtime of PCMF and PCMF-BASIC when varying the number of inner and outer iterations used in the learning.

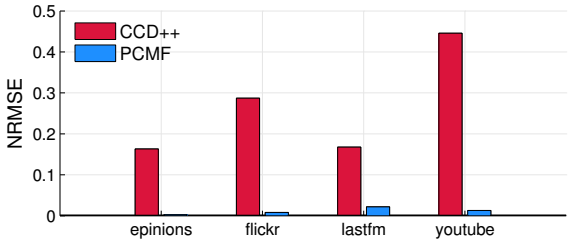
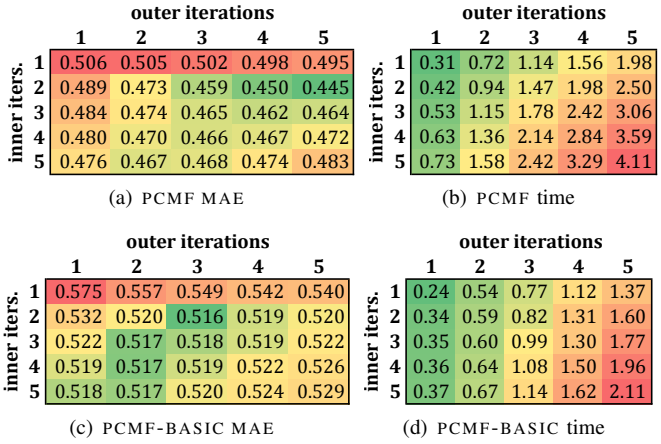


Fig. 6. **Comparing quality of the models learned.**



(a) PCMF MAE (b) PCMF time

(c) PCMF-BASIC MAE (d) PCMF-BASIC time

Fig. 7. **Comparing the effectiveness of inner and outer iterations.**

E. Impact of α

PCMF seamlessly allows for additional information to be used in the factorization. In these experiments, we vary the α parameter that controls the influence/weight given to the additional matrices (e.g., social interaction matrix). In particular, if $\alpha = 0$ then the additional information (e.g., social interactions) are ignored and only the initial target matrix of interest is used in the factorization (e.g., the user-item matrix). Conversely, if $\alpha = \infty$ (or becomes large enough), then only the social interactions are used (as these dominate). We find that α significantly impacts performance as illustrated in Figure 8, and thus incorporating additional information (e.g., social network, group membership) significantly improves the quality of the network model for prediction. Importantly, the optimal accuracy is achieved when all data sources are used in the factorization.

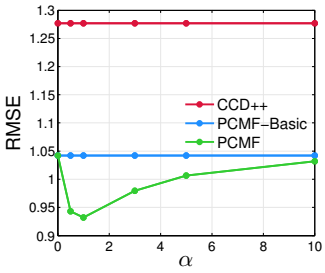


Fig. 8. **Varying** α (epinions 20% with $d = 10$).

Figure 9 investigates the impact of α when λ changes and vice-versa. The best model is found when $\alpha = 1$ with $\lambda = 0.1$, emphasizing the utility of the proposed approach that collectively

factorizing multiple types of network data simultaneously.

α \ λ	0	0.1	0.5	1	5	100
0	1.16	1.04	1.07	1.33	4.7	4.74
0.1	3.4	2.1	0.95	1.01	2.5	4.71
0.5	3.5	0.94	0.98	1.13	2.4	4.64
1	2.6	0.93	1.02	1.21	3.8	4.69
5	1.23	1	1.09	1.35	4.69	4.71
100	1.2	1.04	1.08	1.34	4.69	4.71

Fig. 9. **Exploring the impact of the parameter space on the predictive quality of the models**

F. Impact of the Latent Dimension

For this experiment, we investigate the impact on accuracy and scalability of PCMF when learning models with different dimensions. Using a smaller number of dimensions d leads to faster convergence as updates are linear to d as noted in Section III. Alternatively, as the dimensionality parameter d increases, the model parameter space expands capturing weaker signals in the data at the risk of overfitting. The scalability of both PCMF and PCMF-BASIC does not significantly change as we vary the number of latent dimensions learned. One example of this behavior is shown in Figure 1 for $d \in \{5, 50, 500\}$ and for each of the different amounts of training data available for learning. Similar results are found for other parameter settings as well.

G. Serving Predictions in Real-time Streams

We also proposed fast and effective methods for serving predictions in a streaming fashion as shown in Table I. In particular, we investigate PCMF-based methods in a streaming setting where user requests are unbounded and continuously arriving over time.

TABLE I. EVALUATION OF STREAMING PREDICTION METHODS

	requests served per second			*(items)*	*(dims)*
Dataset	PCMF-BASIC	PCMF-NN	PCMF-SOC	n	d
Epinions	1482	122	5468	755k	5
MovieLens	3783	398	N/A	65k	40
Yelp	4505	3347	N/A	11k	200

H. Heterogeneous Social Link Prediction

The PCMF framework is also effective for predicting the existence of links in large-scale heterogeneous social networks. Existing work in social network link prediction has largely used only the past friendship graphs [24], whereas this work leverages PCMF to jointly model heterogeneous social network data. To evaluate the effectiveness of the heterogeneous link prediction approach, we use the LiveJournal data collected by Mislove *et al.*, see [34]. In particular, we use the social network (user-friends-user) and the group membership bipartite graph (user-joined-group) which indicates the set of groups each user is involved. Note that PCMF may also be used to predict the groups a user is likely to join in the future (heterogeneous link prediction, link between multiple node types) as well as links in a homogeneous context such as friendships. PCMF is used to predict a set of held-out links[4] (i.e., future/missing) and use the NRMSE evaluation metric for comparison. Overall, PCMF consistently resulted in significantly lower error than the baseline. In particular, we observed a 17.5% reduction in the error when PCMF is used instead of the base model that uses only the social network for link prediction. This reveals the importance of carefully leveraging multiple heterogeneous networks for link prediction.

[4]Note that these are known actual relationships in the social network, but are not used for learning.

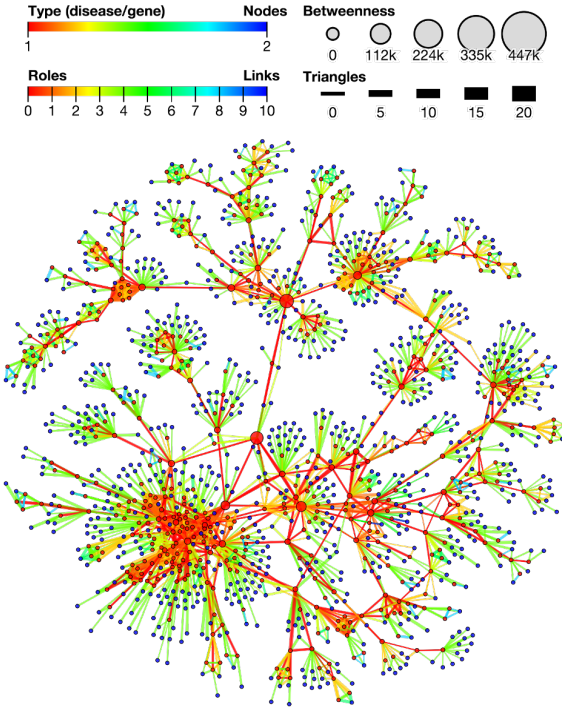


Fig. 10. **Discovering edge roles.** Edges are colored by the role with largest membership. We visualize the diseasome biological network. Node color indicates the class label (disease/gene).

I. Discovering Edge Roles

Role discovery is becoming increasingly popular [20]. However, existing work focuses on discovering roles of nodes, and has ignored the task of discovering edge roles. In this work, we investigate edge-centric roles using a non-negative factorization variant of PCMF. Following the idea of feature-based roles proposed in [22], we systematically discover an edge-based feature representation. As initial features, we use a variety of edge-based graphlet features of size 2,3, and 4. From these initial features, more complicated features are discovered using the algorithm proposed in [22]. Given this large edge-by-feature matrix, PCMF is used to learn edge-role memberships. Importantly, PCMF provides a fast and parallel method for collective role discovery in large heterogeneous networks. Figure 10 demonstrates the effectiveness of PCMF by visualizing the edge roles learned from a biological network. The edge roles discovered by PCMF are clearly correlated with the class label of the node, and make sense as they capture the structural behavior surrounding each edge in the network.

J. Improving Relational Classification

PCMF may also be used to learn a more effective relational representation for a variety of relational learning tasks. In

particular, Figure 11 shows the impact on the network structure when PCMF is used. Strikingly, PCMF creates edges between nodes of the same class, making them significantly closer compared to the original relational data (see Fig. 10).

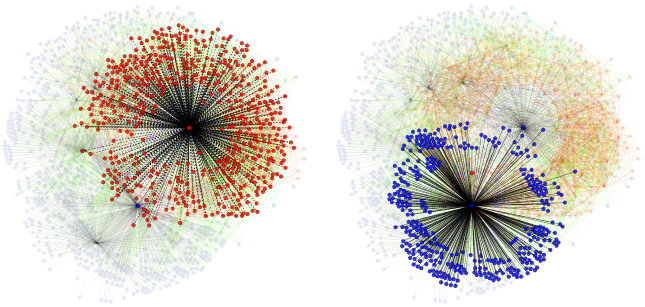


Fig. 11. PCMF improves relational/collective classification by automatically connecting up nodes of the same label. This not only benefits relational classification, but may significantly improve collective approaches that use label propagation by reducing noise and improving the quality of messages passed between such nodes. Nodes are colored by class label (disease/gene).

V. Conclusion

This paper proposed PCMF — a fast *parallel* approach for jointly modeling *large* heterogeneous networks. Unlike existing approaches that are either inefficient or sequential, PCMF is parallel, non-parametric, flexible, and fast for large heterogeneous networks, with a runtime that is linear in the number of observations from the input data. Compared to the recent state-of-the-art parallel method, both PCMF and PCMF-basic (the proposed single matrix factorization variant) were shown to be significantly more scalable, while also providing better quality models for relational learning tasks. Moreover, PCMF is flexible as many components are interchangeable (update order/strategy, loss, regularization, etc.), as well as non-parametric/data-driven (requiring no user-defined parameters), and thus, well-suited for many real-world applications. In addition, PCMF was shown to be effective for a variety of relational learning and modeling tasks across a wide range of network data. A main strength of PCMF lies in its generality as it naturally handles a large class of matrices (i.e., contextual/side information), from sparse weighted single typed networks (i.e., social friendship/comm. networks, web graphs) and multi-typed networks (user-group memberships, word-document matrix) to dense matrices representing node/edge attributes as well as dense similarity matrices.

References

[1] H.-F. Yu, C.-J. Hsieh, S. Si, and I. S. Dhillon, "Parallel matrix factorization for recommender systems," *KIS*, pp. 1–27, 2013.

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

[3] R. Salakhutdinov and A. Mnih, "Probabilistic matrix factorization." in *NIPS*, vol. 1, no. 1, 2007, pp. 2–1.

[4] M. Zinkevich, M. Weimer, A. J. Smola, and L. Li, "Parallelized stochastic gradient descent," in *NIPS*, vol. 4, no. 1, 2010, p. 4.

[5] F. Niu, B. Recht, C. Ré, and S. J. Wright, "Hogwild!: A lock-free approach to parallelizing stochastic gradient descent," *NIPS*, vol. 24, pp. 693–701, 2011.

[6] M. Vorontsov, G. Carhart, and J. Ricklin, "Adaptive phase-distortion correction based on parallel gradient-descent optimization," *Optics letters*, vol. 22, no. 12, pp. 907–909, 1997.

[7] R. Gemulla, E. Nijkamp, P. J. Haas, and Y. Sismanis, "Large-scale matrix factorization with distributed stochastic gradient descent," in *SIGKDD*, 2011, pp. 69–77.

[8] B. Recht and C. Ré, "Parallel stochastic gradient algorithms for large-scale matrix completion," *Math. Prog. Comp.*, vol. 5, no. 2, pp. 201–226, 2013.

[9] Y. Zhou, D. Wilkinson, R. Schreiber, and R. Pan, "Large-scale parallel collaborative filtering for the netflix prize," in *Algorithmic Aspects in Information and Management*. Springer, 2008, pp. 337–348.

[10] H. Ma, H. Yang, M. R. Lyu, and I. King, "Sorec: social recommendation using probabilistic matrix factorization," in *CIKM*, 2008, pp. 931–940.

[11] J. Tang, X. Hu, and H. Liu, "Social recommendation: a review," *SNAM*, vol. 3, no. 4, pp. 1113–1133, 2013.

[12] X. Yang, Y. Guo, Y. Liu, and H. Steck, "A survey of collaborative filtering based social recommender systems," *Computer Comm.*, 2013.

[13] P. Bonhard and M. Sasse, "knowing me, knowing youusing profiles and social networking to improve recommender systems," *BT Technology Journal*, vol. 24, no. 3, pp. 84–98, 2006.

[14] S.-H. Yang, B. Long, A. Smola, N. Sadagopan, Z. Zheng, and H. Zha, "Like like alike: joint friendship and interest propagation in social networks," in *WWW*, 2011, pp. 537–546.

[15] M. Jamali and M. Ester, "A matrix factorization technique with trust propagation for recommendation in social networks," in *RecSys*, 2010, pp. 135–142.

[16] Y. Sun, J. Han, X. Yan, and P. S. Yu, "Mining knowledge from interconnected data: a heterogeneous information network analysis approach," *VLDB*, vol. 5, no. 12, pp. 2022–2023, 2012.

[17] Y. Sun, J. Han, X. Yan, P. S. Yu, and T. Wu, "Pathsim: Meta path-based top-k similarity search in heterogeneous information networks," *VLDB*, 2011.

[18] D. Zhou, S. A. Orshanskiy, H. Zha, and C. L. Giles, "Co-ranking authors and documents in a heterogeneous network," in *ICDM*, 2007.

[19] Y. Li and J. C. Patra, "Genome-wide inferring gene–phenotype relationship by walking on the heterogeneous network," *Bioinformatics*, vol. 26, no. 9, pp. 1219–1224, 2010.

[20] S. P. Borgatti, M. G. Everett, and J. C. Johnson, *Analyzing social networks*. SAGE Publications Limited, 2013.

[21] E. M. Airoldi, D. M. Blei, S. E. Fienberg, and E. P. Xing, "Mixed membership stochastic blockmodels," in *NIPS*, 2009, pp. 33–40.

[22] R. A. Rossi and N. K. Ahmed, "Role discovery in networks," *TKDE*, vol. 26, no. 7, pp. 1–20, 2014.

[23] L. Lü and T. Zhou, "Link prediction in complex networks: A survey," *Physica A*, vol. 390, no. 6, pp. 1150–1170, 2011.

[24] D. Liben-Nowell and J. Kleinberg, "The link-prediction problem for social networks," *JASIST*, vol. 58, no. 7, pp. 1019–1031, 2007.

[25] J.-L. Lassez, R. Rossi, and K. Jeev, "Ranking links on the web: Search and surf engines," *New Frontiers in App. AI*, pp. 199–208, 2008.

[26] R. A. Rossi, L. K. McDowell, D. W. Aha, and J. Neville, "Transforming graph data for statistical relational learning," *JAIR*, vol. 45, no. 1, pp. 363–441, 2012.

[27] D. Davis, R. Lichtenwalter, and N. V. Chawla, "Multi-relational link prediction in heterogeneous information networks," in *ASONAM*, 2011.

[28] X. Kong, J. Zhang, and P. S. Yu, "Inferring anchor links across multiple heterogeneous social networks," in *CIKM*, 2013, pp. 179–188.

[29] B. Cao, N. N. Liu, and Q. Yang, "Transfer learning for collective link prediction in multiple heterogenous domains," in *ICML*, 2010.

[30] Y. Yasui, K. Fujisawa, and K. Goto, "Numa-optimized parallel breadth-first search on multicore single-node system," in *Big Data*, 2013, pp. 394–402.

[31] H. Akaike, "A new look at the statistical model identification," *Transactions on Automatic Control*, vol. 19, no. 6, pp. 716–723, 1974.

[32] R. A. Rossi and N. K. Ahmed, "The network data repository with interactive graph analytics and visualization," in *AAAI*, 2015, pp. 4292–4293. [Online]. Available: http://networkrepository.com

[33] E. D. Dolan and J. J. Moré, "Benchmarking optimization software with performance profiles," *Math. Prog.*, vol. 91, no. 2, pp. 201–213, 2002.

[34] A. Mislove, M. Marcon, K. Gummadi, P. Druschel, and B. Bhattacharjee, "Measurement and analysis of online social networks," in *SIGCOMM*, 2007, pp. 29–42.