A Parallel and Efficient Algorithm for Learning to Match

Jingbo Shang[1,4], Tianqi Chen[2], Hang Li[3], Zhengdong Lu[3], Yong Yu[4]
[1]*University of Illinois at Urbana Champaign, IL, USA, shang7@illinois.edu*
[2]*University of Washington, WA, USA, tqchen@cs.washington.edu*
[3]*Huawei Noah's Ark Lab, Hong Kong, {hangli.hl, lu.zhengdong}@huawei.com*
[4]*Shanghai Jiao Tong University, Shanghai, China, {shangjingbo, yyu}@apex.sjtu.edu.cn*

***Abstract*—Many tasks in data mining and related fields can be formalized as matching between objects in two heterogeneous domains, including collaborative filtering, link prediction, image tagging, and web search. Machine learning techniques, referred to as learning-to-match in this paper, have been successfully applied to the problems. Among them, a class of state-of-the-art methods, named feature-based matrix factorization, formalize the task as an extension to matrix factorization by incorporating auxiliary features into the model. Unfortunately, making those algorithms scale to real world problems is challenging, and simple parallelization strategies fail due to the complex cross talking patterns between sub-tasks. In this paper, we tackle this challenge with a novel parallel and efficient algorithm. Our algorithm, based on coordinate descent, can easily handle hundreds of millions of instances and features on a single machine. The key recipe of this algorithm is an iterative relaxation of the objective to facilitate parallel updates of parameters, with guaranteed convergence on minimizing the original objective function. Experimental results demonstrate that the proposed method is effective on a wide range of matching problems, with efficiency significantly improved upon the baselines while accuracy retained unchanged.**

I. Introduction

Many tasks in applications can be formalized as matching between objects in two heterogeneous domains, while information on the relations between some objects in the two domains as well as information on objects themselves are supposed to be given. For example, in a recommender system, one manages to find items, e.g. movies or albums, which best match the users by using preference of some users on some items. Another example is image tagging, for which one wants to associate tags with images by utilizing some tagged images. Recent years have observed a great success of employing various machine learning techniques to solve the matching problem. To better understand the generality and specificity of the learning techniques and to make further improvements on the techniques, generalization of the techniques, referred to as *learning to match* in this paper, appears to be very necessary. Formally, we denote $\mathbf{X} = [\mathbf{X}_1, \mathbf{X}_2 \cdots \mathbf{X}_q] \in \mathbb{R}^{n \times q}$ and $\mathbf{Z} = [\mathbf{Z}_1, \mathbf{Z}_2 \cdots \mathbf{Z}_p] \in \mathbb{R}^{m \times p}$, as features in the query and target domains respectively. Our problem is to learn to predict the matching score $\hat{\mathbf{Y}}_{ij} = f(\mathbf{X}_i, \mathbf{Z}_j)$ between each pair of query i and target j, with the matching scores of some query-target pairs $\{\mathbf{Y}_{ij} | (i,j) \in \mathcal{O}\}$ as training data, where $\mathcal{O}$ denotes the set of indices for the given query-target pairs. The objective of this work is to develop a parallel and efficient algorithm to solve the following learning to match task:

$$\mathbf{P}, \mathbf{Q} = \text{argmin}_{\mathbf{P},\mathbf{Q}} \sum_{i,j \in \mathcal{O}} l(\hat{\mathbf{Y}}_{ij}, \mathbf{Y}_{ij}) + \Omega(\mathbf{P}) + \Omega(\mathbf{Q}) \quad (1)$$

where $\hat{\mathbf{Y}}_{ipj} = (\mathbf{P}\mathbf{X}_i)^T \mathbf{Q}\mathbf{Z}_j$, $l(\hat{\mathbf{Y}}_{ij}, \mathbf{Y}_{ij})$ is a strongly convex loss function over $\hat{\mathbf{Y}}_{ij}$ and Ω is a regularization term base on elastic net [28]. $\Omega(\mathbf{P}) = \sum_{s,k}(\alpha|\mathbf{P}_{s,k}| + \frac{\lambda}{2}\mathbf{P}^2_{s,k})$, $\Omega(\mathbf{Q}) = \sum_{s,k}(\alpha|\mathbf{Q}_{s,k}| + \frac{\lambda}{2}\mathbf{Q}^2_{s,k})$. $\mathbf{P} \in \mathbb{R}^{d \times n}$ and $\mathbf{Q} \in \mathbb{R}^{d \times m}$ are transformation matrices that map instances from the feature spaces into the latent space. The model is a generalization of matrix factorization [18] incorporating features in the two domains and has been proved to very effective to produce accurate prediction. Such matching models have been employed to solve different learning to match problems. In collaborative filtering, models have been proposed to utilize user feedback information [9], attribute information [1], and content information [22]. In search Wu et.al [23] have developed a model to learn matching (relevance) scores between queries and documents with term information of queries and documents. In link prediction, Menon et.al [13] have proposed a feature-based latent factor model to give the state-of-art result. A similar model has also been successfully employed in KDD Cup 2012 in the which task is social link prediction [16], [5]. From the viewpoint of learning, the key to the success of these models is full utilization of available information in the data to enhance the accuracies.

Making learning of the matching model scalable and efficient is difficult, however, because of the two obstacles. First, learning of the model requires simultaneous access to the features, making the existing techniques for parallelization of matrix factorization [8], [25] inapplicable. Second, the computation cost will become prohibitively high, when the scale of the problem (the number of nonzero features and the number of observed entries) is large, which necessitates speeding up the learning process. To our best knowledge, little work has been done on parallelization and acceleration of the learning to match task.

There is a 10 pages version on arXiv.org.

1550-4786/14 $31.00 © 2014 IEEE
DOI 10.1109/ICDM.2014.71

Algorithm 1: Plain Coordinate Descent

randomly initialize $\mathbf{P}, \mathbf{Q}$;
$\mathbf{U} \leftarrow \mathbf{PX}$, $\mathbf{V} \leftarrow \mathbf{QZ}$, $\hat{\mathbf{Y}} \leftarrow \mathbf{U}^T\mathbf{V}$ {calculate $\mathbf{U}$, $\mathbf{V}$, $\hat{\mathbf{Y}}$}
while *not converge* **do**
  for $k = 1$ ***to*** d **do**
    for $s = 1$ ***to*** n **do**
      $x \leftarrow \sum_{i,j \in \mathcal{O}} g_{ij} \mathbf{V}_{kj} \mathbf{X}_{is}$
      $y \leftarrow \beta \sum_{i,j \in \mathcal{O}} \mathbf{V}_{kj}^2 \mathbf{X}_{is}^2$
      $\Delta \mathbf{P}_{ks} \leftarrow T(x, y, \mathbf{P}_{ks}, \alpha, \lambda)$
      $\mathbf{P}_{ks} \leftarrow \mathbf{P}_{ks} + \Delta \mathbf{P}_{ks}$
  $\mathbf{U} = \mathbf{PX}$ {recalculate buffered $\mathbf{U}$}
  update $\mathbf{Q}$ in the same way as $\mathbf{P}$

Algorithm 2: Efficient Coordinate Descent

randomly initialize $\mathbf{P}, \mathbf{Q}$
$\mathbf{U} \leftarrow \mathbf{PX}$, $\mathbf{V} \leftarrow \mathbf{QZ}$, $\hat{\mathbf{Y}} \leftarrow \mathbf{U}^T\mathbf{V}$ {calculate $\mathbf{U}$, $\mathbf{V}$, $\hat{\mathbf{Y}}$}
while *not converge* **do**
  for $k = 1$ ***to*** d **do**
    for $i = 1$ ***to*** q **do**
      $G_{ki} \leftarrow \sum_{j \in \mathcal{O}_i} g_{ij} \mathbf{V}_{kj}$
      $H_{ki} \leftarrow \beta \sum_{j \in \mathcal{O}_i} \mathbf{V}_{kj}^2$
    for $s = 1$ ***to*** n **do**
      $\Delta \mathbf{P}_{ks} \leftarrow$
      $T(\sum_i G_{ki} \mathbf{X}_{is}, \sum_i H_{ki} \mathbf{X}_{is}^2, \mathbf{P}_{ks}, \alpha, \lambda)$
      $\mathbf{P}_{ks} \leftarrow \mathbf{P}_{ks} + \Delta \mathbf{P}_{ks}$
      for $i = 1$ ***to*** q **do**
        $G_{ki} \leftarrow G_{ki} + \mathbf{X}_{is} \Delta \mathbf{P}_{ks} H_{ki}$
  $\mathbf{U} = \mathbf{PX}$ {recalculate buffered $\mathbf{U}$}
  update $\mathbf{Q}$ in the same way as $\mathbf{P}$

In this paper, we try to tackle the two challenges by developing a coordinate descent algorithm tailored to the task. The contribution of the paper is as follows. 1) We propose an efficient coordinate descent algorithm which greatly reduces the time complexity of learning a general matching model. 2) We propose a parallel algorithm to further significantly enhance the scalability and efficiency. The proposed algorithm automatically adjusts the rate of parallel update according to the conditions in learning, and we prove that there is a guarantee on the convergence of the algorithm. 3) We empirically demonstrate the effectiveness and efficiency of the proposed algorithm on three learning to match tasks.

The rest of the paper is organized as follows. Related work is introduced in Section II. We explain the efficient coordinate descent algorithm in Section III. Section IV describes the parallel coordinate descent algorithm. Experiments are shown in Section V. Finally, we conclude the paper in Section VI.

Notations We use $\mathbf{U}_i = \mathbf{PX}_i$ and $\mathbf{V}_j = \mathbf{QZ}_j$ to denote the latent vectors of query i and target j. $\hat{\mathbf{Y}}_{ij} = \mathbf{U}_i^T \mathbf{V}_j$ is used to indicate the prediction of matching score. We use $\mathbf{P}_s$ to indicate sth column of $\mathbf{P}$ and $\mathbf{P}_{k,:}$ to indicate kth row of it. $\|\mathbf{X}\|_0$ is used to denote number of nonzero entries in feature matrix $\mathbf{X}$. We emphasize that $\mathbf{X}$ and $\mathbf{Z}$ are sparse and assume all the calculations related to them[1] take this fact into consideration.

II. Related Work

Matrix Factorization (MF) models [10] are arguably the most successful approach to learning-to-match, especially Feature based Matrix Factorization (FMF) models, which achieve state-of-the-art results with different types of features used. In collaborative filtering, user feedback information (SVD++) [9], user attribute information [1], and product attribute information [21] are incorporated into models to further enhance the accuracies in prediction. In web search [23], [24], term vectors of queries and documents are utilized as features to significantly improve relevance ranking. FMF models also give the best results in link prediction in KDD Cup 2012 [13], [3], [4]. These works demonstrate the effectiveness of the learning-to-match models, but also create necessity for parallelization of the learning algorithms.

There has been much effort on parallelizing the process of plain MF. Gemulla et al. [8] propose a method of distributed stochastic gradient descent and Yu et al. [25] introduce a parallel coordinate descent algorithm. An alternating least square method is proposed as well [26]. Recently, many improvements on the efficiency are proposed [27], [12], [7]. However, all these models reply on the fact that the rows and columns can be naturally separated, and thus cannot work on FMF due to the complex feature dependencies.

There is only a little work about acceleration of coordinate descent for FMF. The most recent one scales up coordinate descent by making use of repeating patterns of features [17]. However, it is specialized for the least-squares loss and probit loss while our algorithm takes a completely different approach and can handle any convex loss functions. As general parallelization technique, the Hogwild! algorithm [15] can be applied to our problem but has some repeated calculations. Other algorithms of parallel coordinate descent, such as [6], [14] cannot be directly applied to FMF because of complex feature dependencies. The convergence of existing parallel coordinate descent algorithms for linear regression [2], [19], [20] depends on the spectrum of covariance matrix, which changes in each round in our learning setting (due to the changes in $\mathbf{U}$ and $\mathbf{V}$), and thus the algorithms cannot be directly applied to our problem. Our algorithm makes use of parallel updates and minimizes an upper bound re-estimated each round to ensure convergence, which can also be viewed as a kind of minorization-maximization algorithm [11]. Our algorithm makes use of parallel update to minimize an upper bound re-estimated each round to ensure convergence, which

[1] including matrix operations, and summations over entries.

can also be viewed as a kind of minorization-maximization algorithm [11].

III. Efficient Algorithm for Learning to Match

We propose an efficient coordinate descent algorithm to solve Equation (1). Let $g_{ij} = \frac{\partial}{\partial \hat{\mathbf{Y}}_{ij}} l(\hat{\mathbf{Y}}_{ij}, \mathbf{Y}_{ij})$ be the gradient of each instance over prediction and $\beta = \sup_y \frac{\partial^2}{\partial y^2} l(y, \mathbf{Y}_{ij})$. We can exploit the standard technique to learn the model using coordinate descent (CD), as shown in Algorithm 1. Here, T is defined using the following thresholding function to handle optimization of l_1 norm. The regularization term only affects the result through function T, and thus makes most part of the algorithm independent of regularization. Note that we implicitly assume $\hat{\mathbf{Y}}_{ij}$ is buffered and kept up to date, when g_{ij} is needed in the algorithm.

$$T(x, y, w, \alpha, \lambda) = \begin{cases} \max(-\frac{x+\lambda w+\alpha}{y+\lambda}, -w) & w - \frac{x+\lambda w}{y+\lambda} \geq 0 \\ \min(-\frac{x+\lambda w-\alpha}{y+\lambda}, -w) & w - \frac{x+\lambda w}{y+\lambda} < 0 \end{cases}$$

The time complexity of one update in Algorithm 1 is $O(d|\mathcal{O}|(\frac{\|\mathbf{X}\|_0}{q} + \frac{\|\mathbf{Z}\|_0}{p}))$, where $\|\mathbf{X}\|_0$ and $\|\mathbf{Z}\|_0$ denote numbers of nonzero entries in feature matrices $\mathbf{X}$ and $\mathbf{Z}$, q and p denote numbers of query and target instances, and $(\frac{\|\mathbf{X}\|_0}{q} + \frac{\|\mathbf{Z}\|_0}{p})$ denotes average number of nonzero features for each pair $(i, j) \in \mathcal{O}$. We note that the time complexity is the same as the time complexity of stochastic gradient optimization for Equation (1). From the analysis, we can see that the time complexity of the algorithm is increased by order of $O(d|\mathcal{O}|)$ when average number of nonzero features increases. This can greatly hamper the learning of matching model, when a large number of features are used.

To solve the problem, we give an efficient algorithm for learning-to-match by avoiding the repeated calculations caused by features. For ease of explanation, we introduce two auxiliary variables G_{ki} and H_{ki} calculated by

$$G_{ki} = \sum_{j \in \mathcal{O}_i} g_{ij} \mathbf{V}_{kj}, \; H_{ki} = \beta \sum_{j \in \mathcal{O}_i} \mathbf{V}_{kj}^2$$

where $\mathcal{O}_i = \{j | (i, j) \in \mathcal{O}\}$ is the set of observed target instances associated with query instance i. The key idea of efficient CD is to make use of G_{ki} and H_{ki} to save duplicated summations in Algorithm 1. Since the gradient value g_{ij} is changed after each update, it is not trivial to let G_{ki} unchanged. Our algorithm keeps making G_{ki} updated to ensure the convergence of the algorithm.

The efficient CD is shown in Algorithm 2, the time complexity of the new algorithm is only of $O(d(\|\mathbf{X}\|_0 + \|\mathbf{Z}\|_0 + |\mathcal{O}|))$. It is linear to numbers of nonzero entries of feature matrices and number of observed entries of $\mathbf{Y}$. The *speedup* over Algorithm 1 on $\mathbf{P}$ updates is about $O(|\mathcal{O}|/q)$, which corresponds to average number of observed target instances per query instance. This can be at level of 10 to 100 in problems such as collaborative filtering and link prediction. When $\|\mathbf{X}\|_0 + \|\mathbf{Z}\|_0$ is close to (or smaller than) $|\mathcal{O}|$, our algorithm runs as fast as the algorithm of plain matrix factorization even though it uses extra features.

Algorithm 3: Parallel Algorithm for Learning to Match

```
randomly initialize P, Q
U ← PX, V ← QZ, Ŷ ← U^T V {calculate U, V, Ŷ}
while not converge do
    schedule a partition P of {1, 2, ··· , n}
    for k = 1 to d do
        for i = 1 to q in parallel do
            G_ki ← Σ_{j∈O_i} g_ij V_kj
            H_ki ← β Σ_{j∈O_i} V_kj^2
        for each index set S ∈ P do
            for i = 1 to q in parallel do
                C_ki ← Σ_{s∈S} |X_is|
            for s ∈ S in parallel do
                x ← Σ_i G_ki X_is
                y ← Σ_i H_ki |X_is| C_ki
                ΔP_ks ← T(x, y, P_ks, α, λ)
                P_ks ← P_ks + ΔP_ks
            for i = 1 to q in parallel do
                G_ki ← G_ki + Σ_{s∈S} X_is ΔP_ks H_ki
    U = PX {recalculate buffered U}
    update Q in the same way as P
```

IV. Parallel and Efficient Algorithm for Learning to Match

The statistics calculation and preprocessing steps in Algorithm 2 can be naturally separated into several independent tasks. However, the $\mathbf{P}$ update steps depend on each other, making the parallelization of it a difficult task.

Analysis of Parallel Update Let S be a set of feature indices to be updated in parallel. Assume that the statistics of G_{ki} is up to date as in Algorithm 2 and we want to change $\mathbf{P}_{ks}$ for $s \in S$ in parallel. For simplicity of notation, we use $\Delta\mathbf{P}_{k,:}$ to represent the change in $\mathbf{P}_{k,:}$. After this change $\tilde{L}$ will be

$$\begin{aligned} \tilde{L}(\Delta\mathbf{P}_{k,:}) = & \sum_{i,j \in \mathcal{O}} l(\hat{\mathbf{Y}}_{ij}, \mathbf{Y}_{ij}) \\ & + \sum_{s \in S} (\sum_i G_{ki} \mathbf{X}_{is} \Delta\mathbf{P}_{ks} + \frac{1}{2} \sum_i H_{ki} \mathbf{X}_{is}^2 \Delta\mathbf{P}_{ks}^2) \\ & + \sum_{s \in S} \sum_{t \in S, t \neq s} \sum_i H_{ki} \mathbf{X}_{it} \mathbf{X}_{is} \Delta\mathbf{P}_{kt} \Delta\mathbf{P}_{ks} \end{aligned} \tag{2}$$

In a specific case in which $\mathbf{X}_{it}\mathbf{X}_{is} = 0$ for $s \neq t; s, t \in S$, the third line in Equation (2) becomes zero. This means that the features in the selected set do not appear in the same instance. In such case, the loss can be separated into $|S|$ independent parts and the original update rule can be applied in parallel. Not surprisingly, such condition does not hold in many real world scenarios. We need to remove these troublesome cross terms in the second line, by deriving

Table I
DETAILS OF 4 DATASETS. THE EVALUATION METRIC IS DETERMINED BY THEIR TASKS.

Dataset	$q \times p$	$\|\mathbf{X}\|_0$	$\|\mathbf{Z}\|_0$	$\|\mathcal{O}\|$	Task	Metric	Available Features
Yahoo! Music	$1M \times 0.6M$	$263M$	$1M$	$250M$	Collaborative Filtering	RMSE	User Feedback [9], Taxonomy
Tencent Weibo	$2M \times 5K$	$55M$	$83K$	$93M$	Social Link Prediction	MAP@K	Social Network, User Profile, Taxonomy, Tag
Flickr	$0.65M \times 20K$	$451M$	$20K$	$39M$	Image Tagging	MAP, P@K	Sift Descriptors of Image
Movielens-10M	$69K \times 10K$	$10M$	$10K$	$9M$	Collaborative Filtering	RMSE	User Feedback [9]

an adaptive estimation of the conflicts caused by parallel updates, more specifically, by the inequality:

$$\Delta\mathbf{P}_{kt}\Delta\mathbf{P}_{ks}\mathbf{X}_{it}\mathbf{X}_{is} \leq \frac{1}{2}|\mathbf{X}_{it}\mathbf{X}_{is}|\left(\Delta\mathbf{P}_{kt}^2 + \Delta\mathbf{P}_{ks}^2\right)$$

With the inequality, we can bound $\tilde{L}$ as follows

$$\begin{aligned}\tilde{L}(\Delta\mathbf{P}_{k,:}) \leq & c + \sum_{s\in S}(\sum_i G_{ki}\mathbf{X}_{is}\Delta\mathbf{P}_{ks}) \\ & + \sum_{s\in S}(\frac{1}{2}\sum_i H_{ki}|\mathbf{X}_{is}|(\sum_{t\in S}|\mathbf{X}_{it}|)\Delta\mathbf{P}_{ks}^2) \\ \triangleq & \tilde{L}^p(\Delta\mathbf{P}_{k,:}).\end{aligned}$$

Obviously this new upper bound $\tilde{L}^p(\Delta\mathbf{P}_{k,:})$ can be separated into $|S|$ independent parts and optimized in parallel. Moreover, the sum $\sum_{t\in S}|\mathbf{X}_{it}|$ is common for all features in set S and is only needed to be calculated once. With this result, we give a parallel and efficient algorithm for learning-to-match, shown in Algorithm 3.

Convergence of Parallel Algorithm The relaxation of $\tilde{L}(\Delta\mathbf{P}_{k,:})$ into $\tilde{L}^p(\Delta\mathbf{P}_{k,:})$ is performed iteratively in the optimization, and it still attempts to optimize the original objective as in Equation (1), which is a case much analogous to Expectation-Maximization algorithm in finding a maximum-likelihood solution. Let $\Delta\mathbf{P}^*_{k,:}$ be the change in $\mathbf{P}_{k,:}$ after each parallel update. Since each parallel update optimizes $\tilde{L}^p(\Delta\mathbf{P}_{k,:})$, we have the following inequality

$$\begin{aligned}&\tilde{L}(\Delta\mathbf{P}^*_{k,:}) + \Omega(\mathbf{P}_{k,:} + \Delta\mathbf{P}^*_{k,:}) \\ \leq & \tilde{L}^p(\Delta\mathbf{P}^*_{k,:}) + \Omega(\mathbf{P}_{k,:} + \Delta\mathbf{P}^*_{k,:}) \\ \leq & \tilde{L}^p(\mathbf{0}) + \Omega(\mathbf{P}_{k,:} + \mathbf{0}) = \tilde{L}(\mathbf{0}) + \Omega(\mathbf{P}_{k,:} + \mathbf{0})\end{aligned}$$

It indicates that $\tilde{L}$ decreases after each parallel update. It then follows that the parallel procedure for optimizing the original loss function in Algorithm 3 always converges.

The update rule depends on the statistics $C_{ki} = \sum_{t\in S}|\mathbf{X}_{it}|$. With $\eta_{ks} = \frac{\sum_i H_{ki}\mathbf{X}_{is}^2 + \lambda}{\sum_i H_{ki}C_{ki}|\mathbf{X}_{is}| + \lambda}$, it can be shown that the parallel update of $\Delta\mathbf{P}_{ks}$ is shrunken by η_{ks} compared to sequential update. Intuitively η_{ks} depends on the co-occurrence between features $s \in S$. When features in S rarely co-occur, η_{ks} will be close to one, which means that we can update "aggressively". When features in S co-occur frequently, η_{ks} will get small and we need to update more "conservatively". In an extreme case in which no feature co-occurs with each other, $\eta_{ks} = 1$ and we get perfect parallelization without any loss of update efficiency. In another extreme case in which we have $|S|$ duplicated features ($\mathbf{X}_{is} = \mathbf{X}_{it}$ for all $s, t \in S$), $\eta_{ks} = \frac{1}{|S|}$, which is extremely conservative given the size of S. The advantage of our algorithm is that it *automatically adjusts* its "level of conservativeness" by the condition in learning, and thus it always ensures the convergence of the algorithm regardless of the number of threads and the nature of dataset.

The changes in loss function can be analyzed accordingly. Let us consider the simple case in which $\alpha = 0$ and only l_2 regularization is involved. The change of loss after a parallel update can be bounded by

$$\begin{aligned}&\tilde{L}(\Delta\mathbf{P}^*_{k,:}) + \Omega(\mathbf{P}_{k,:} + \Delta\mathbf{P}^*_{k,:}) - \tilde{L}(\mathbf{0}) - \Omega(\mathbf{P}_{k,:} + \mathbf{0}) \\ \leq & -\frac{1}{4}\sum_{s\in S}\eta_{ks}\frac{(\sum_i G_{ki}\mathbf{X}_{is})^2}{\sum_i H_{ki}\mathbf{X}_{is}^2 + \lambda}\end{aligned}$$

As it indicates, compared to the ideal case in which features do not co-occur, each parallel update's contribution to the loss change is scaled by η_{ks}. The above analysis also intuitively justifies that η_{ks} controls the efficiency of the update.

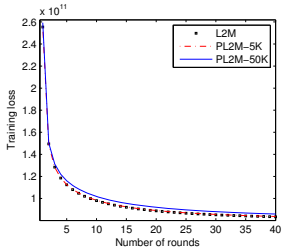
Time Complexity Assuming that we use K threads to run the algorithm, the time complexity for one update is $O(\frac{1}{K}d(\|\mathbf{X}\|_0 + \|\mathbf{Z}\|_0 + |\mathcal{O}|))$. It is due to the fact that all parts of the algorithm are parallelized. In practice, we need to take synchronization cost into consideration, the corresponding time complexity is $O(\frac{1}{K}d((\|\mathbf{X}\|_0 + \|\mathbf{Z}\|_0)(1+\sigma) + |\mathcal{O}|))$, where σ stands for variance of time costs by parallel tasks. Assume that we have K tasks whose time cost are $T_1, T_2, \cdots T_K$. We can define $\sigma = max_i(T_i)/mean_i(T_i)$. In this paper, we fix $|S|$ and randomly select elements from the feature indices to generate S.

V. EXPERIMENTS

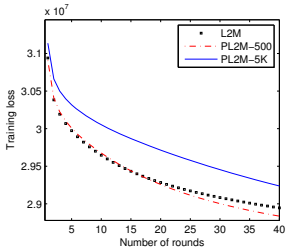
A. Datasets and Settings

Four datasets representing different types of learning to match tasks are summarized in Table I. We have implemented our parallel learning to match algorithm and Hogwild! using OpenMP[2]. *All matrix operations* mentioned in the algorithms take the advantages of data sparsity. All algorithms in experiments share the same codes of elementary operations. The experiments are conducted on a machine that has an Intel Xeon CPU E5-2680 with 8 cores 16 threads support at 2.70GHz and 128GB memory. We utilize up to 15 working threads and leave one thread for scheduling. We empirically set the values of parameters as $\lambda = 1, \alpha = 0.1$, and $d = 64$ for our algorithm. We use PL2M-T to refer to the parallel algorithm for learning-to-match with parallel set $|S| = T$.

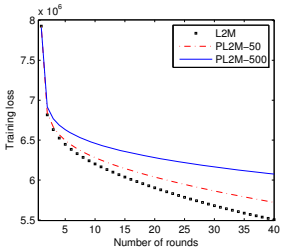
[2]http://www.openmp.org



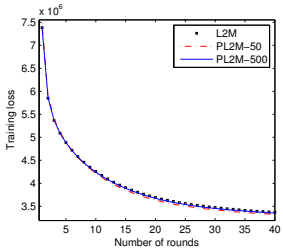
(a) TLC on Yahoo! Music



(b) TLC on Tencent Weibo



(c) TLC on Flickr



(d) TLC on Movielens-10M

Figure 1. Training Loss Convergence(TLC) on Four Datasets.

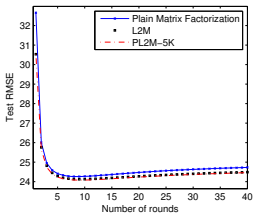


Figure 2. RMSE on Yahoo!

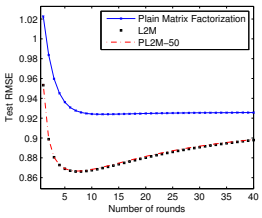


Figure 3. RMSE on ML-10M

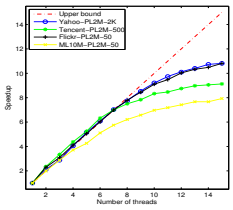


Figure 4. Speedup Curves

Table II
RESULTS OF FLICKR IN PRECISION

Setting	MAP	P@1	P@3
Popularity	3.96%	4.63%	4.08%
L2M	7.18%	11.05%	8.76%
PL2M-50	7.59%	11.86%	9.19%

Table III
RESULTS OF SOCIAL LINK PREDICTION (TENCENT WEIBO) IN MAP

Setting	MAP@1	MAP@3	MAP@5
Popularity	22.54%	34.65%	38.28%
L2M-SNS	24.10%	36.56%	40.19%
PL2M-500-SNS	24.18%	36.65%	40.27%
L2M-ALL	25.44%	38.02%	41.63%
PL2M-500-ALL	25.52%	38.14%	41.75%

Table IV
PL2M VERSUS HOGWILD! (8 THREADS).

Dataset	Method	sec/round	rounds	test error
Tencent Weibo	Hogwild!	104.3	14	24.96%
(MAP@1)	PL2M	70.1	5	25.52%
Flickr	Hogwild!	1117.0	59	7.59%
(MAP)	PL2M	155.0	33	7.59%
Movielens-10M	Hogwild!	162.2	20	0.8756
(RMSE)	PL2M	18.5	7	0.8666

B. *Usefulness of Features*

To investigate the effectiveness of the features, we first compare the convergence of L2M and plain MF. From Figure 2 and 3, we can find that the L2M model with auxiliary features converges faster and can achieve better results than plain matrix factorization. This result is consistent with previous results [9], [16] and confirms the importance of using features in this problem. As shown in Table III and II models incorporating more features always work better than others. Here the suffix ALL stands for using all the available features shown in Table I. We also evaluate the performance when using only social network information, with suffix SNS.

C. *PL2M versus L2M*

As shown in Figure 2,p 3, and Table III, II, PL2M always gives comparable or even better test errors as L2M. However, due to the fact that the loss function is non-convex for $\mathbf{P}$ and $\mathbf{Q}$, after several updates, for example, 20 rounds, the values of $\mathbf{P}$ and $\mathbf{Q}$ may be quite different so that the two methods finally converge to different local minimums.

D. *PL2M versus Hogwild!*

Because Hogwild! is based on stochastic gradient descent, some parameters such as learning rate need to be tuned. After fine tuning the parameters using cross validation on the training set for Hogwild!, we have obtained its performance using 8 threads in Table IV, including test error, running time of one round of training, and number of rounds needed to get the best test error. We can see that the running time for each round of PL2M is much shorter than that of Hogwild!, while their test errors are similar. This is consistent with our theoretical result about the time complexity. Furthermore, Hogwild! needs more rounds than PL2M to achieve its best test errors. Therefore, the total running time for PL2M to get the best performance is much smaller than that of Hogwild!.

E. *Scalability of PL2M*

We test the average running time of PL2M on four datasets with varying numbers of threads and evaluate the improvement in efficiency. As shown in Figure 4, on the first 3 datasets, PL2M can achieve almost linear speedup with less than 8 threads, but the speedup gain slows down with more threads. Because the working threads are still fully occupied with more than 8, we conjecture the reason is that the number of physical cores of the machine is only 8. From the results, we can find that PL2M is able to gain about 9 times speedup using 10 threads, confirming the scalability of the parallel algorithm. Although the running time on Movielens-10M is quite small, the parallel algorithm can also provide accelerations.

VI. CONCLUSION

We have proposed a parallel and efficient algorithm for learning to match. The learning to match model summarized

in the paper is fairly general, subsuming many latent factor models as special cases. The parallel algorithm allows scalable and fast learning of the model with a theoretical guarantee of convergence. Experimental results show that our algorithm is both effective and efficient. As future work, we plan to (1) extend the algorithm to a distributed setting, (2) find better scheduling strategies for making parallel updates, and (3) apply the technique developed in this paper to the parallelization of other learning methods.

REFERENCES

[1] D. Agarwal and B.-C. Chen. Regression-based latent factor models. KDD '09, pages 19–28, New York, NY, USA, 2009. ACM.

[2] J. Bradley, A. Kyrola, D. Bickson, and C. Guestrin. Parallel coordinate descent for l1-regularized loss minimization. In L. Getoor and T. Scheffer, editors, *Proceedings of the 28th International Conference on Machine Learning*, ICML '11, pages 321–328, New York, NY, USA, June 2011. ACM.

[3] K. Chen, T. Chen, G. Zheng, O. Jin, E. Yao, and Y. Yu. Collaborative personalized tweet recommendation. In *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '12, pages 661–670, New York, NY, USA, 2012. ACM.

[4] T. Chen, L. Tang, Q. Liu, D. Yang, S. Xie, X. Cao, C. Wu, E. Yao, Z. Liu, Z. Jiang, et al. Combining factorization model and additive forest for collaborative followee recommendation. *KDD CUP*, 2012.

[5] T. Chen, W. Zhang, Q. Lu, K. Chen, Z. Zheng, and Y. Yu. Svdfeature: A toolkit for feature-based collaborative filtering. *J. Mach. Learn. Res.*, 13(1):3619–3622, Dec. 2012.

[6] M. Collins, R. E. Schapire, and Y. Singer. Logistic regression, adaboost and bregman distances. *Machine Learning*, 48(1-3):253–285, 2002.

[7] A. S. Das, M. Datar, A. Garg, and S. Rajaram. Google news personalization: scalable online collaborative filtering. In *Proceedings of the 16th international conference on World Wide Web*, WWW '07, pages 271–280, New York, NY, USA, 2007. ACM.

[8] R. Gemulla, E. Nijkamp, P. J. Haas, and Y. Sismanis. Large-scale matrix factorization with distributed stochastic gradient descent. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '11, pages 69–77, New York, NY, USA, 2011. ACM.

[9] Y. Koren. Factorization meets the neighborhood: A multifaceted collaborative filtering model. KDD '08, pages 426–434, New York, NY, USA, 2008. ACM.

[10] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, Aug. 2009.

[11] K. Lange, D. R. Hunter, and I. Yang. Optimization transfer using surrogate objective functions. *Journal of computational and graphical statistics*, 9(1):1–20, 2000.

[12] C. Liu, H.-c. Yang, J. Fan, L.-W. He, and Y.-M. Wang. Distributed nonnegative matrix factorization for web-scale dyadic data analysis on mapreduce. In *Proceedings of the 19th international conference on World wide web*, WWW '10, pages 681–690, New York, NY, USA, 2010. ACM.

[13] A. K. Menon and C. Elkan. Link prediction via matrix factorization. In *Proceedings of the 2011 European conference on Machine learning and knowledge discovery in databases - Volume Part II*, ECML PKDD'11, pages 437–452, Berlin, Heidelberg, 2011. Springer-Verlag.

[14] I. Mukherjee, K. Canini, R. Frongillo, and Y. Singer. Parallel boosting with momentum. In *Machine Learning and Knowledge Discovery in Databases*, pages 17–32. Springer, 2013.

[15] B. Recht, C. Re, S. Wright, and F. Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 693–701. 2011.

[16] S. Rendle. Factorization machines with libfm. *ACM Trans. Intell. Syst. Technol.*, 3(3):57:1–57:22, May 2012.

[17] S. Rendle. Scaling factorization machines to relational data. In *Proceedings of the VLDB Endowment*, volume 6, pages 337–348. VLDB Endowment, 2013.

[18] R. Salakhutdinov and A. Mnih. Probabilistic matrix factorization. In *Advances in Neural Information Processing Systems 20*, pages 1257–1264. MIT Press, Cambridge, MA, 2007.

[19] C. Scherrer, M. Halappanavar, A. Tewari, and D. Haglin. Scaling up coordinate descent algorithms for large ℓ_1 regularization problems. In *Proceedings of the 29th International Conference on Machine Learning*, ICML '12, pages 1407–1414, New York, NY, USA, July 2012. Omnipress.

[20] C. Scherrer, A. Tewari, M. Halappanavar, and D. Haglin. Feature clustering for accelerating parallel coordinate descent. In P. Bartlett, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 28–36. 2012.

[21] D. H. Stern, R. Herbrich, and T. Graepel. Matchbox: large scale online bayesian recommendations. In *Proceedings of the 18th international conference on World wide web*, WWW '09, pages 111–120, New York, NY, USA, 2009. ACM.

[22] J. Weston, C. Wang, R. Weiss, and A. Berenzweig. Latent collaborative retrieval. In J. Langford and J. Pineau, editors, *Proceedings of the 29th International Conference on Machine Learning*, ICML '12, pages 9–16, New York, NY, USA, July 2012. Omnipress.

[23] W. Wu, H. Li, and J. Xu. Learning query and document similarities from click-through bipartite graph with metadata. In *Proceedings of the Sixth ACM International Conference on Web Search and Data Mining*, WSDM '13, pages 687–696, New York, NY, USA, 2013. ACM.

[24] W. Wu, Z. Lu, and H. Li. Learning bilinear model for matching queries and documents. *J. Mach. Learn. Res.*, 14(1):2519–2548, Jan. 2013.

[25] H.-F. Yu, C.-J. Hsieh, S. Si, and I. Dhillon. Scalable coordinate descent approaches to parallel matrix factorization for recommender systems. ICDM '12, pages 765–774, Washington, DC, USA, 2012. IEEE Computer Society.

[26] Y. Zhou, D. Wilkinson, R. Schreiber, and R. Pan. Large-scale parallel collaborative filtering for the netflix prize. In *Proceedings of the 4th international conference on Algorithmic Aspects in Information and Management*, AAIM '08, pages 337–348, Berlin, Heidelberg, 2008. Springer-Verlag.

[27] Y. Zhuang, W.-S. Chin, Y.-C. Juan, and C.-J. Lin. A fast parallel sgd for matrix factorization in shared memory systems. In *Proceedings of the 7th ACM conference on Recommender systems*, RecSys '13, pages 249–256, New York, NY, USA, 2013. ACM.

[28] H. Zou and T. Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(2):301–320, 2005.