A Parallel and Efficient Algorithm for Learning to Match

Jingbo Shang\textsuperscript{1,4}, Tianqi Chen\textsuperscript{2}, Hang Li\textsuperscript{3}, Zhengdong Lu\textsuperscript{3}, Yong Yu\textsuperscript{4}
\textsuperscript{1}University of Illinois at Urbana Champaign, IL, USA, shang7@illinois.edu
\textsuperscript{2}University of Washington, WA, USA, tqchen@cs.washington.edu
\textsuperscript{3}Huawei Noah’s Ark Lab, Hong Kong, \{hangli.hl, lu.zhengdong\}@huawei.com
\textsuperscript{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 preference of some users on some items. 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 $X = [X_1, X_2, \ldots, X_q] \in \mathbb{R}^{n \times q}$ and $Z = [Z_1, Z_2, \ldots, Z_p] \in \mathbb{R}^{m \times p}$, as features in the query and target domains respectively. Our problem is to learn to predict the matching score $Y_{ij} = f(X_i, Z_j)$ between each pair of query $i$ and target $j$, with the matching scores of some query-target pairs \{$(Y_{ij})_{(i,j) \in \mathcal{O}}$\} 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:

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

where $\hat{Y}_{ij} = (PX_i)^TQZ_j$, $l(\hat{Y}_{ij}, Y_{ij})$ is a strongly convex loss function over $\hat{Y}_{ij}$ and $\Omega$ is a regularization term base on elastic net [28]. $\Omega(P) = \sum_{s,k}(\alpha |P_{s,k}| + \frac{1}{2} P_{s,k}^2)$, $\Omega(Q) = \sum_{s,k}(\alpha |Q_{s,k}| + \frac{1}{2} Q_{s,k}^2)$. $P \in \mathbb{R}^{d \times n}$ and $Q \in \mathbb{R}^{d \times m}$ 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.
We prove that there is a guarantee on the convergence of the SVD++ \cite{9}, user attribute information \cite{1}, and product information. The proposed algorithm automatically adjusts the rate of convergence depending on the complexity of the data. We assume all the calculations related to them can be done in parallel.

Feature based Matrix Factorization (FMF) models, which are arguably the most successful approach to learning-to-match, especially in the context of collaborative filtering, user feedback information achieve state-of-the-art results with different types of features. Notations

$X$: feature matrix

$P$: coordinate descent algorithm introduced in Section II. We explain the efficient coordinate descent algorithm in the same way as \cite{11}.

Algorithm 1: Plain Coordinate Descent

randomly initialize $P$, $Q$

\[ U \leftarrow PX, V \leftarrow QZ, \hat{Y} \leftarrow U^T V \{\text{calculate } U, V, \hat{Y}\} \]

\[ \text{for } k = 1 \text{ to } d \text{ do} \]

\[ \text{for } s = 1 \text{ to } n \text{ do} \]

\[ x \leftarrow \sum_{i,j \in O} g_{ij} V_{kj} X_{is} \]

\[ y \leftarrow \beta \sum_{i,j \in O} V_{kj}^2 X_{is}^2 \]

\[ \Delta P_{ks} \leftarrow T(x, y, P_{ks}, \alpha, \lambda) \]

\[ \hat{P}_{ks} \leftarrow P_{ks} + \Delta P_{ks} \]

\[ U = PX \{\text{recalculate buffered } U}\]  

update $Q$ in the same way as $P$.

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 $U_i = PX_i$ and $V_j = QZ_j$ to denote the latent vectors of query $i$ and target $j$. $Y_{ij} = U_i^T V_j$ is used to indicate the prediction of matching score. We use $P_s$ to indicate $s$th column of $P$ and $P_{k,:}$ to indicate $k$th row of it. $\|X\|_0$ is used to denote number of nonzero entries in feature matrix $X$. We emphasize that $X$ and $Z$ are sparse and assume all the calculations related to them take this fact into consideration.

Algorithm 2: Efficient Coordinate Descent

randomly initialize $P$, $Q$

\[ U \leftarrow PX, V \leftarrow QZ, \hat{Y} \leftarrow U^T V \{\text{calculate } U, V, \hat{Y}\} \]

\[ \text{for } k = 1 \text{ to } d \text{ do} \]

\[ \text{for } i = 1 \text{ to } q \text{ do} \]

\[ G_{ki} \leftarrow \sum_{j \in O} g_{ij} V_{kj} \]

\[ H_{ki} \leftarrow \beta \sum_{j \in O} V_{kj}^2 \]

\[ \Delta P_{ks} \leftarrow T(\sum_i G_{ki} X_{is}, \sum_i H_{ki} X_{is}^2, P_{ks}, \alpha, \lambda) \]

\[ P_{ks} \leftarrow P_{ks} + \Delta P_{ks} \]

\[ \text{for } i = 1 \text{ to } q \text{ do} \]

\[ G_{ki} \leftarrow G_{ki} + X_{is} \Delta P_{ks} H_{ki} \]

\[ U = PX \{\text{recalculate buffered } U}\]  

update $Q$ in the same way as $P$.

There has been much effort on parallelizing the process of plain MF. Gemulla et al. \cite{8} propose a method of distributed stochastic gradient descent and Yu et al. \cite{25} introduce a parallel coordinate descent algorithm. An alternating least squares method is proposed as well \cite{26}. Recently, many improvements on the efficiency are proposed \cite{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 \cite{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 \cite{15} can be applied to our problem but has some repeated calculations. Other algorithms of parallel coordinate descent, such as \cite{6, 14} cannot be directly applied to FMF because of complex feature dependencies. The convergence of existing parallel coordinate descent algorithms for linear regression \cite{2, 19, 20} depends on the spectrum of the covariance matrix, which changes in each round in our learning setting (due to the changes in $U$ and $V$), 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 \cite{11}. Our algorithm makes use of parallel update to minimize an upper bound re-estimated each round to ensure convergence, which
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 \( g_{ij} = \frac{\partial}{\partial Y_{ij}} l(\mathbf{Y}_{ij}, \mathbf{Y}_{ij}) \) be the gradient of each instance over prediction and \( \beta = \sup_{ij} \frac{\partial}{\partial Y_{ij}} l(y, \mathbf{Y}_{ij}) \). 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 \( \mathbf{Y}_{ij} \) 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 \( O(|\mathcal{O}|(|\mathbf{X}| + |\mathbf{Z}| + q)) \), where \( |\mathbf{X}| \) and \( |\mathbf{Z}| \) denote numbers of nonzero entries in feature matrices \( \mathbf{X} \) and \( \mathbf{Z} \), and \( q \) denotes number of query and target instances, and \( \left( \frac{|\mathbf{X}|}{q} + \frac{|\mathbf{Z}|}{p} \right) \) 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 \( O(|\mathcal{O}|) \) 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

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

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.

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 \( G_{ks} \) 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 \( \Delta \mathbf{P}_{ks} \) to represent the change in \( \mathbf{P}_{ks} \). After this change \( \hat{L} \) will be

\[
\hat{L}(\Delta \mathbf{P}_{ks}) = \sum_{i, j \in \mathcal{O}} l(\hat{Y}_{ij}, \hat{Y}_{ij}) + \sum_{s \in S} \sum_{t \neq s} \sum_{i} H_{ki} X_{it} X_{is} \Delta \mathbf{P}_{kt} \Delta \mathbf{P}_{ks} + \frac{1}{2} \sum_{i} H_{ki} X_{is}^2 \Delta \mathbf{P}_{ks}^2
\]

In a specific case in which \( X_{it} X_{is} = 0 \) 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
an adaptive estimation of the conflicts caused by parallel updates, more specifically, by the inequality:

$$\Delta P_{kl} \Delta P_{ks} X_{kl} X_{ls} \leq \frac{1}{2} \|X_{kl} X_{ls}\| \left(\Delta P_{kl}^2 + \Delta P_{ks}^2\right)$$

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

$$\tilde{L}(\Delta P_{k,:}) \leq c + \sum_{s \in S} \left(\sum_i G_{ki} X_{kl} \Delta P_{ks}\right) + \sum_{s \in S} \left(\sum_i H_{ki} X_{is}\right) \left(\sum_{t \in S} |X_{it}|\right) \Delta P_{ks}^2$$

$$\triangleq \tilde{L}(\Delta P_{k,:}).$$

Obviously this new upper bound $\tilde{L}(\Delta P_{k,:})$ can be separated into $|S|$ independent parts and optimized in parallel. Moreover, the sum $\sum_{t \in S} |X_{it}|$ 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 $\tilde{L}(\Delta P_{k,:})$ into $\tilde{L}(\Delta P_{k,:})$ 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 $\Delta P_{k,:}^*$ be the change in $P_{k,:}$ after each parallel update. Since each parallel update optimizes $\tilde{L}(\Delta P_{k,:})$, we have the following inequality

$$\tilde{L}(\Delta P_{k,:}) + \Omega(P_{k,:} + \Delta P_{k,:}^*) \leq \tilde{L}(\Delta P_{k,:}) + \Omega(P_{k,:} + \Delta P_{k,:}) \leq \tilde{L}(0) + \Omega(P_{k,:} + 0)$$

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 $C_{ki} = \sum_{t \in S} |X_{it}|$. With $\eta_{ks} = \sum_{s \in S} H_{ks} X_{is}^2 + \lambda$, it can be shown that the parallel update of $\Delta P_{ks}$ is shrunken by $\eta_{ks}$ compared to sequential update. Intuitively $\eta_{ks}$ depends on the co-occurrence between features $s \in S$. When features in $S$ rarely co-occur, $\eta_{ks}$ will be close to one, which means that we can update “aggressively”. When features in $S$ co-occur frequently, $\eta_{ks}$ 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 ($X_{is} = X_{it}$ 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

$$\tilde{L}(\Delta P_{k,:}) + \Omega(P_{k,:} + \Delta P_{k,:}^*) - \tilde{L}(0) - \Omega(P_{k,:} + 0) \leq - \frac{1}{4} \sum_{s \in S} \eta_{ks} \left(\sum_i G_{ki} X_{is}\right)^2$$

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 $\eta_{ks}$. The above analysis also intuitively justifies that $\eta_{ks}$ 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 $O(\frac{1}{K} d(||X||_0 + ||Z||_0 + |O|))$. 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 $O(\frac{1}{K} d(||X||_0 + ||Z||_0(1 + \sigma) + |O|))$, where $\sigma$ stands for variance of time costs by parallel tasks. Assume that we have $K$ tasks whose time cost are $T_1, T_2, \ldots, T_K$. We can define $\sigma = \text{max}_{i}(T_i)/\text{mean}(T_i)$. 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 OpenMP2. *All matrix operations mentioned in the algorithms take the advantages of data sparsity. All algorithms in experiments share the same codes of elemental 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 $|S| = T$.

2http://www.openmp.org

---

### Table I

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

| Dataset       | $q \times p$ | $|X|_0$ | $|Z|_0$ | $|O|$ | Task               | Metric    | Available Features |
|---------------|--------------|--------|--------|------|--------------------|-----------|-------------------|
| Yahoo! Music  | 1M $\times$ 0.6M | 265M   | 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    | 93M  | Collaborative Filtering | RMSE      | User Feedback [9] |

---
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 P and Q, after several updates, for example, 20 rounds, the values of P and 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


