Discrete Collaborative Filtering

Hanwang Zhang[1] Fumin Shen[2] Wei Liu[3] Xiangnan He[1] Huanbo Luan[4] Tat-Seng Chua[1]

[1]School of Computing, National University of Singapore

[2]School of Computer Science and Engineering, University of Electronic Science and Technology of China

[3]Didi Research

[4]Department of Computer Science & Technology, Tsinghua University

{hanwangzhang,fumin.shen,xiangnanhe,luanhuanbo}@gmail.com; wliu@ee.columbia.edu; dcscts@nus.edu.sg

ABSTRACT

We address the efficiency problem of Collaborative Filtering (CF) by hashing users and items as latent vectors in the form of binary codes, so that user-item affinity can be efficiently calculated in a Hamming space. However, existing hashing methods for CF employ binary code learning procedures that most suffer from the challenging discrete constraints. Hence, those methods generally adopt a two-stage learning scheme composed of relaxed optimization via discarding the discrete constraints, followed by binary quantization. We argue that such a scheme will result in a large quantization loss, which especially compromises the performance of large-scale CF that resorts to longer binary codes. In this paper, we propose a principled CF hashing framework called Discrete Collaborative Filtering (DCF), which directly tackles the challenging discrete optimization that should have been treated adequately in hashing. The formulation of DCF has two advantages: 1) the Hamming similarity induced loss that preserves the intrinsic user-item similarity, and 2) the balanced and uncorrelated code constraints that yield compact yet informative binary codes. We devise a computationally efficient algorithm with a rigorous convergence proof of DCF. Through extensive experiments on several real-world benchmarks, we show that DCF consistently outperforms state-of-the-art CF hashing techniques, *e.g.*, though using only 8 bits, DCF is even significantly better than other methods using 128 bits.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval - *information filtering;*

Keywords

Recommendation; Discrete Hashing; Collaborative Filtering

1. INTRODUCTION

Over the past decades, we have witnessed continued efforts in increasing the accuracy and efficiency of Recommender Systems, which have been widely known as one of the key technologies for the thrift of Web services like Facebook, Amazon and Flickr. However, their ever-growing scales make today's recommendations even more challenging. Taking a typical Flickr user as an example, a practical recommender system should quickly prompt to recommend photos in a billion-scale collection by exploring extremely sparse user history; and, there are millions of such users[1]!

Collaborative Filtering (CF), more specifically, latent factor based CF (*e.g.*, matrix factorization), has been demonstrated to achieve a successful balance between accuracy and efficiency in real-world recommender systems [4, 16, 1]. Such CF methods factorize an $m \times n$ user-item rating matrix of m users and n items into an r-d low-dimensional latent vector (*a.k.a.* feature) space. Then the predictions for user-item ratings can be efficiently estimated by inner products between the corresponding user and item vectors. In this way, recommendation by CF naturally falls into a similarity search problem—top-K item recommendation for a user can be cast into finding the top-K similar items queried by the user [30, 3, 12]. When m or n is large, storing user (or item) vectors of the size $\mathcal{O}(mr)$ (or $\mathcal{O}(nr)$) and similarity search of the complexity $\mathcal{O}(n)$ will be a critical efficiency bottleneck, which has not been well addressed in recent progress on recommender efficiency [23].

Fortunately, hashing has been widely shown as a promising approach to tackle fast similarity search [29]. First, by encoding real-valued data vectors into compact binary codes, hashing makes efficient in-memory storage of massive data feasible. Second, as similarity calculation by inner product in a vector space is replaced by bit operations in a proper Hamming space, the time complexity of linear scan is significantly reduced and even constant time search is made possible by exploiting lookup tables [28, 31]. Recently, several works have brought the advance of hashing into collaborative filtering for better recommendation efficiency [18, 33, 32]. However, those works essentially divide the hashing into two independent stages: real-valued optimization and binary quantization. More specifically, due to the discrete constraints imposed on the corresponding binary code learning procedure which is generally NP-hard [11], they resort to simply solving relaxed optimization problems by discarding the discrete constraints and then rounding off [32] or

SIGIR '16, July 17-21, 2016, Pisa, Italy

DOI: http://dx.doi.org/10.1145/2911451.2911502

[1]https://www.flickr.com/photos/franckmichel/6855169886

rotating [18, 33] the obtained continuous solutions to target binary codes. We argue that these "two-stage" approaches oversimplify original discrete optimization, resulting in a large quantization loss. Here, we refer to "quantization loss" as the accumulated deviations of the binary bits originating from thresholding real values to integers; unsurprisingly, large deviations will violate the original data geometry in the continuous vector space (*e.g.*, intrinsic user-item relations). As we can foresee, in real-world large-scale applications which require longer codes for accuracy, such accumulated errors will inevitably deteriorate the recommendation performance.

In this paper, we propose a principled approach for efficient collaborative filtering, dubbed Discrete Collaborative Filtering (DCF), which has not been addressed yet. As illustrated in Figure 1, instead of choosing an erroneous two-stage approach, we directly tackle the challenging discrete optimization that should have been treated adequately in hashing. Our formulation is directly based on the loss function of traditional CF, where the user/item features are replaced by binary codes and the user-item inner product is replaced by Hamming similarity (cf. Eq. (2)). By doing so, the proposed DCF explicitly optimizes the binary codes that fit the intrinsic user-item similarities and hence the quantization error is expected to be smaller than those of two-stage approaches. Besides, we explicitly impose the balanced and uncorrelated bits on the codes. Though these two constraints make DCF even more challenging, they are crucial for achieving compact yet informative codes [31]. To tackle the discrete optimization of DCF in a computationally tractable manner, we develop an alternating optimization algorithm which consists of iteratively solving mixed-integer programming subproblems. In particular, we provide efficient solutions that require fast bit-wise updates and eigen–decompositions for small matrices, and therefore they can easily scale up to large-scale recommendations. We evaluate the proposed DCF on three real-world datasets in various million-scale applications including movies, books and business recommendations.

Our contributions are summarized as follows:

- We propose an efficient CF approach called Discrete Collaborative Filtering (DCF). To the best of our knowledge, DCF is the first principled framework that directly learns user-item binary codes via discrete optimization.
- We develop an efficient algorithm for solving DCF. The convergence of our algorithm is rigorously guaranteed.
- Through extensive experiments performed on three real-world datasets, we show that DCF consistently surpasses several state-of-the-art CF hashing techniques.

2. RELATED WORK

We first review efficient Collaborative Filtering (CF) algorithms using latent factor models, and then discuss recent advance in discrete hashing techniques. For comprehensive reviews of CF and hashing, please refer to [5] and [29], respectively.

2.1 Efficient Collaborative Filtering

One line of research towards efficient CF includes designing scalable online methods such as preference ranking [10],

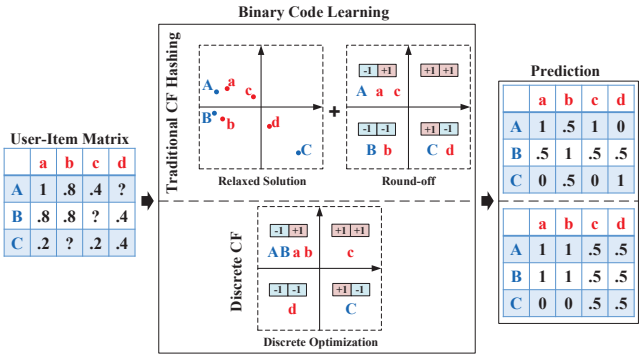

Figure 1: Illustration of the key difference between existing hashing methods for CF (top) and our proposed DCF (bottom). DCF directly learns binary codes that preserve the user-item similarity discovered from the user-item matrix (with missing entries denoted as "?"); while traditional methods first relax the discrete problem and then round-off the real-valued results to binary codes. As can be seen, such a rounding-off quantization step causes errors when two points are quite close but assigned to different bits (*e.g.*, A and B); while two points are quite far away but assigned to the same bits (*e.g.*, C and d). However, DCF can preserve the intrinsic user-item geometry, so it predicts user-item ratings (normalized Hamming similarity, cf. Eq. (2)) with a lower error, *e.g.*, the squared loss of DCF is only half of those of traditional methods.

matrix factorization [20], and regression [2]. In particular, our out-of-sample hashing scheme for new users/items (cf. Section. 4.2) follows a similar spirit of [25, 8], which projects new samples onto a learned factor space. However, these works neglect that CF is essentially a similarity search problem, where even linear time complexity is prohibitive for large-scale recommendation tasks. Therefore, another line of research focuses on encoding users and items into binary codes, *e.g.*, hashing for the purpose of significant efficiency. As a pioneering work, Das *et al.* [6] used Locality-Sensitive Hashing (LSH) [7] to generate hash codes of Google news users based on an item-sharing similarity. Karatzoglou *et al.* [14] learned user-item features with traditional CF and then randomly projected the features to acquire hash codes. Similarly, Zhou and Zha [33] rotated their learned features by running ITQ [9] to generate hash codes. Liu *et al.* [18] imposed the uncorrelated bit constraints on the traditional CF objective for learning user-item features and then rounded them to produce hash codes. Zhang *et al.* [32] argued that inner product is not a proper similarity, which is nevertheless the fundamental assumption about hashing, so subsequent hashing may harm the accuracy of preference predictions. To this end, they proposed to regularize the user/item features to compute their cosine similarities, and then quantized them by respectively thresholding their magnitudes and phases.

One can easily sum up that the aforementioned hashing techniques for CF are essentially "two-stage" approaches, where hash codes for users and items are obtained through two independent steps: relaxed user-item feature learning and binary quantization (cf. Figure 1). As we will review next, such a two-stage relaxation is well-known to suffer from

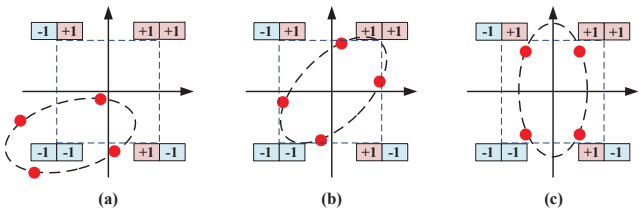

Figure 2: Illustration of the effectiveness of the balance and decorrelation constraints in DCF. The ellipse represents an intrinsic user-item relation manifold and the red dots denote four distinct real-valued user/item vectors that should be separated in Hamming space. (a) Three points are encoded as $(-1,-1)$ and the other one as $(-1,+1)$, which are not discriminative. However, after (b) balancing and (c) decorrelation, the codes are optimized to preserve the user-item relations.

a large quantization loss, which is the main challenge we tackle in this paper.

2.2 Discrete Hashing

In order to reduce quantization errors caused by the over-simplifying rounding-off step, discrete hashing—direct binary code learning by discrete optimization—is becoming popular recently. The most widely used discrete hashing technique is perhaps Iterative Quantization (ITQ) [9], which minimizes the quantization loss by alternatively learning the binary codes and hash functions. However, it has two drawbacks. First, ITQ requires the hash functions to use orthogonal projections, which is not a generic assumption; second, ITQ can also be considered as a two-stage approach since it first learns relaxed solutions and then treats quantization as an independent post-processing, which does not necessarily capture the intrinsic data geometry. Thus, ITQ is suboptimal. Latest improvements on joint optimizations of quantization losses and intrinsic objective functions can be found in Discrete Graph Hashing [17] and Supervised Discrete Hashing [26], which demonstrate significant performance gain over the above two-stage hashing methods. Our work is also an advocate of such a joint discrete optimization but focused on CF which is fundamentally different from the above objectives. To the best of our knowledge, DCF is a research gap that we fill in this paper; and due to the generic matrix factorization formulations in CF, we believe that DCF will have a high potential in plenty of machine learning and information retrieval tasks other than recommendations [18].

3. PROBLEM FORMULATION

3.1 Preliminaries

We use bold uppercase and lowercase letters as matrices and vectors, respectively; In particular, we use $\mathbf{a}_i$ as the i-th row vector of matrix $\mathbf{A}$, A_{ij} as the entry at the i-th row and j-th column of $\mathbf{A}$; alternatively, we rewrite A_{ij} as a_{ij} to highlight the j-th entry of vector $\mathbf{a}_i$. We denote $\|\cdot\|_F$ as the Frobenius norm of a matrix and $\mathrm{tr}(\cdot)$ as the matrix trace. We denote $\mathrm{sgn}(\cdot):\mathbb{R}\to\{\pm 1\}$ as the round-off function.

We focus on discussing matrix factorization CF models, which has been successfully applied in many recommender systems [16]. CF generally maps both users and items to a joint low-dimensional latent space where the user-item similarity (or preference) is estimated by vector inner product. Formally, suppose $\mathbf{u}_i \in \mathbb{R}^r$ is the i-th user vector and $\mathbf{v}_j \in \mathbb{R}^r$ is the j-th item vector, the rating of user i for item j is approximated by $\mathbf{u}_i^T\mathbf{v}_j$. Thus, the goal is to learn user vectors $\mathbf{U}=[\mathbf{u}_1,...,\mathbf{u}_m]\in\mathbb{R}^{r\times m}$ and item vectors $\mathbf{V}=[\mathbf{v}_1,...,\mathbf{v}_n]\in\mathbb{R}^{r\times n}$, where $r \ll \min(m,n)$ is the feature dimension, and $\mathbf{U}^T\mathbf{V}$ is expected to reconstruct the observed ratings as well as predict the unobserved ones. The objective is to minimize the following regularized squared loss on observed ratings:

$$\underset{\mathbf{U},\mathbf{V}}{\operatorname{argmin}} \sum_{i,j\in\mathcal{V}} (S_{ij}-\mathbf{u}_i^T\mathbf{v}_j)^2 + \alpha\|\mathbf{U}\|_F^2 + \beta\|\mathbf{V}\|_F^2, \tag{1}$$

where S_{ij} is the observed rating, whose index set is $\mathcal{V}$. Since the number of observed ratings is sparse, we should properly regularize $\mathbf{U}$ and $\mathbf{V}$ by $\alpha,\beta>0$ in order to prevent from overfitting. After we obtain the optimized user and item vectors, recommendation is then reduced to a similarity search problem. For example, given a "query" user $\mathbf{u}_i$, we recommend items by ranking the predicted ratings $\mathbf{V}^T\mathbf{u}_i\in\mathbb{R}^n$; when n is large, such similarity search scheme is apparently an efficiency bottleneck for practical recommender systems [33, 32].

To this end, we are interested in hashing users and items into binary codes for efficient recommendation since the user-item similarity search can be efficiently conducted in Hamming space. Denote $\mathbf{B}=[\mathbf{b}_1,...,\mathbf{b}_m]\in\{\pm1\}^{r\times m}$ and $\mathbf{D}=[\mathbf{d}_1,...,\mathbf{d}_n]\in\{\pm1\}^{r\times n}$ respectively as r-length user and item binary codes, the Hamming similarity between $\mathbf{b}_i$ and $\mathbf{d}_j$ is defined as [33]:

$$\begin{aligned}\mathrm{sim}(i,j) &= \frac{1}{r}\sum_{k=1}^{r}\mathbb{I}\left(b_{ik}=d_{jk}\right)\\ &= \frac{1}{2r}\left(\sum_{k=1}^{r}\mathbb{I}(b_{ik}=d_{jk}) + r - \sum_{k=1}^{r}\mathbb{I}(b_{ik}\neq d_{jk})\right)\\ &= \frac{1}{2r}\left(r+\sum_{k=1}^{r}b_{ik}d_{jk}\right) = \frac{1}{2}+\frac{1}{2r}\mathbf{b}_i^T\mathbf{d}_j\end{aligned} \tag{2}$$

where $\mathbb{I}(\cdot)$ denotes the indicator function that returns 1 if the statement is true and 0 otherwise. We can easily verify that $sim(i,j)=0$ if all the bits of $\mathbf{b}_i$ and $\mathbf{d}_j$ are different and $sim(i,j)=1$ if $\mathbf{b}_i=\mathbf{d}_j$.

Similar to the problem of conventional CF in Eq. (1), the above similarity score should reconstruct the observed user-item ratings. Therefore, the problem of the proposed Discrete Collaborative Filtering (DCF) is formulated as:

$$\begin{aligned}&\underset{\mathbf{B},\mathbf{D}}{\operatorname{argmin}} \sum_{i,j\in\mathcal{V}}\left(S_{ij}-\mathbf{b}_i^T\mathbf{d}_j\right)^2,\\ &s.t.\ \mathbf{B}\in\{\pm1\}^{r\times m},\mathbf{D}\in\{\pm1\}^{r\times n}\\ &\underbrace{\mathbf{B}\mathbf{1}=0,\mathbf{D}\mathbf{1}=0}_{\text{Balanced Partition}},\ \underbrace{\mathbf{B}\mathbf{B}^T=m\mathbf{I},\mathbf{D}\mathbf{D}^T=n\mathbf{I}}_{\text{Decorrelation}}.\end{aligned} \tag{3}$$

where we slightly abuse the notation S_{ij} as a scaled score in $[-r,r]$ as $\mathbf{b}_i^T\mathbf{d}_j\in\{-r,-r+2,...,r-2,r\}$[2]. Due to the binary constraints in DCF, the regularization $\|\mathbf{B}\|_F^2+\|\mathbf{D}\|_F^2$ as in Eq. (1) is constant and hence is canceled; however, DCF imposes two additional constraints on the binary codes in order to maximize the information encoded in short code

[2]Suppose the original $S_{ij}\in[0,1]$, then we scale $S_{ij}\leftarrow 2rS_{ij}-r$.

length [31]. First, we require that each bit to split the dataset as balanced as possible. This will maximize the information entropy of the bit. Second, each bit should be as independent as possible, *i.e.*, the bits are uncorrelated and the variance is maximized. This removes the redundancy among the bits. Figure 2 illustrates how binary code learning benefits from the two constraints. Note that other latent models with various objective functions such as ranking [24] and regression [2] can be applied in this work with simple algebraic operations.

3.2 Learning Model

It is worth noting that the proposed DCF in Eq. (3) has two key advantages over related work [33, 32]. First, we strictly enforce the binary constraint while theirs relax discrete binary codes to continuous real values. Thus, DCF is expected to minimize the quantization loss during learning. Second, we require the binary codes to be balanced and uncorrelated. Therefore, DCF hashes users and items in a more informative and compact way. However, solving DCF in Eq. (3) is a challenging task since it is generally NP-hard that involves $\mathcal{O}(2^{(m+n)r})$ combinatorial search for the binary codes [11]. Next, we introduce a learning model that can solve DCF in a computationally tractable manner. We propose to solve DCF in Eq. (3) by softening the balance and decorrelation constraints, since strictly imposing them may cause the original DCF infeasible. Let us define two sets: $\mathcal{B} = \{\mathbf{X} \in \mathbb{R}^{r\times m} | \mathbf{X1} = 0, \mathbf{X}\mathbf{X}^T = m\mathbf{I}\}$, $\mathcal{D} = \{\mathbf{Y} \in \mathbb{R}^{r\times n} | \mathbf{Y1} = 0, \mathbf{Y}\mathbf{Y}^T = n\mathbf{I}\}$ and distances $d(\mathbf{B}, \mathcal{B}) = \min_{\mathbf{X}\in\mathcal{B}} \|\mathbf{B}-\mathbf{X}\|_F$, $d(\mathbf{D}, \mathcal{D}) = \min_{\mathbf{Y}\in\mathcal{D}} \|\mathbf{D}-\mathbf{Y}\|_F$. Therefore, we can soften the original DCF in Eq. (3) as:

$$\begin{aligned} &\underset{\mathbf{B},\mathbf{D}}{\operatorname{argmin}} \sum_{i,j\in\mathcal{V}} \left(S_{ij} - \mathbf{b}_i^T\mathbf{d}_j\right)^2 + \alpha d^2(\mathbf{B},\mathcal{B}) + \beta d^2(\mathbf{D},\mathcal{D}) \\ &s.t.,\ \mathbf{B} \in \{\pm 1\}^{r\times m}, \mathbf{D} \in \{\pm 1\}^{r\times n} \end{aligned} \tag{4}$$

where $\alpha > 0$ and $\beta > 0$ are tuning parameters. The above Eq (4) allows a certain discrepancy between the binary codes (*e.g.*, $\mathbf{B}$) and delegate continuous values (*e.g.*, $\mathbf{X}$), to make the constraints computationally tractable. Note that if the constraints in Eq. (3) is feasible, we can enforce the distances $d(\mathbf{B},\mathcal{B}) = d(\mathbf{D},\mathcal{D}) = 0$ in Eq. (4) by imposing very large tuning parameters.

By noting the decorrelation constraints imposed on $\mathbf{B}$, $\mathbf{D}$, $\mathbf{X}$ and $\mathbf{Y}$, Eq. (4) is equivalent to:

$$\begin{aligned} &\underset{\mathbf{B},\mathbf{D},\mathbf{X},\mathbf{Y}}{\operatorname{argmin}} \sum_{i,j\in\mathcal{V}} \left(S_{ij} - \mathbf{b}_i^T\mathbf{d}_j\right)^2 - 2\alpha tr(\mathbf{B}^T\mathbf{X}) - 2\beta tr(\mathbf{D}^T\mathbf{Y}) \\ &s.t., \mathbf{X1} = 0, \mathbf{X}\mathbf{X}^T = m\mathbf{I}, \mathbf{Y1} = 0, \mathbf{Y}\mathbf{Y}^T = n\mathbf{I}, \\ &\quad \mathbf{B} \in \{\pm 1\}^{r\times m}, \mathbf{D} \in \{\pm 1\}^{r\times n}, \end{aligned} \tag{5}$$

which is the proposed learning model for DCF. It is worth noting that we do not discard the binary constraint $\mathbf{B} \in \{\pm 1\}^{r\times m}, \mathbf{D} \in \{\pm 1\}^{r\times n}$ and directly optimize discrete $\mathbf{B}$ and $\mathbf{D}$. Through joint optimization for the binary codes and the delegate real variables, we can obtain nearly balanced and uncorrelated hashing codes for users and items. Next, we will introduce an efficient solution for the mixed-integer optimization problem in Eq. (5).

4. SOLUTION

We alternatively solving four subproblems for DCF model in Eq. (5): $\mathbf{B}$, $\mathbf{D}$, $\mathbf{X}$ and $\mathbf{Y}$. In particular, we show that 1) $\mathbf{B}$ and $\mathbf{D}$ can be efficiently updated by parallel discrete optimization; and 2) $\mathbf{X}$ and $\mathbf{Y}$ can be efficiently updated by small-scale Singular Value Decomposition (SVD).

4.1 Alternating Optimization

It is worth highlighting that the following $\mathbf{B}/\mathbf{D}$-subproblem seeks binary latent features that preserves the intrinsic user-item relations due to the observed loss in Eq. (5); while $\mathbf{X}/\mathbf{Y}$-subproblem attempts to regularize the learned binary codes should be as balanced and uncorrelated as possible.

B-subproblem. In this subproblem, we update $\mathbf{B}$ with fixed $\mathbf{D}$, $\mathbf{X}$ and $\mathbf{Y}$. Since the objective function in Eq. (5) is based on summing over independent users, we can update $\mathbf{B}$ by updating $\mathbf{b}_i$ in parallel according to

$$\underset{\mathbf{b}_i\in\{\pm 1\}^r}{\operatorname{argmin}} \mathbf{b}_i^T\Big(\sum_{j\in\mathcal{V}_i} \mathbf{d}_j\mathbf{d}_j^T\Big)\mathbf{b}_i - 2\Big(\sum_{j\in\mathcal{V}_i} S_{ij}\mathbf{d}_j^T\Big)\mathbf{b}_i - 2\alpha\mathbf{x}_i^T\mathbf{b}_i, \tag{6}$$

where $\mathcal{V}_i$ is the observed rating set for user i.

Due to the binary constraints, the above minimization is generally NP-hard, we propose to use Discrete Coordinate Descent (DCD) to update binary codes $\mathbf{b}_i$ *bit by bit* [26]. Denote b_{ik} as the k-th bit of $\mathbf{b}_i$ and $\mathbf{b}_{i\bar{k}}$ as the rest codes excluding b_{ik}, DCD will update b_{ik} while fixing $\mathbf{b}_{i\bar{k}}$. Thus, the DCD update rule for user binary codes $\mathbf{b}_i$ can be derived as:

$$b_{ik} \leftarrow \operatorname{sgn}\Big(K(\hat{b}_{ik}, b_{ik})\Big), \tag{7}$$

where $\hat{b}_{ik} = \sum_{j\in\mathcal{V}_i}\Big(S_{ij} - \mathbf{d}_{j\bar{k}}^T\mathbf{b}_{i\bar{k}}\Big)d_{jk} + \alpha x_{ik}$.[3], $\mathbf{d}_{j\bar{k}}^T$ is the rest set of item codes excluding d_{jk}, and $K(x,y)$ is a function that $K(x,y) = x$ if $x \neq 0$ and $K(x,y) = y$ otherwise, *i.e.*, when $\hat{b}_{ik} = 0$, we do not update b_{ik}. In this way, $\mathbf{b}_i$ is iteratively updated bit by bit in several passes until convergence (*e.g.*, no more flips of bits). Detailed derivation of Eq. (7) is given in Appendix.

D-subproblem. In this subproblem, we update $\mathbf{D}$ with fixed $\mathbf{B}$, $\mathbf{X}$ and $\mathbf{Y}$. Similar to the B-subproblem, we can update $\mathbf{D}$ by updating $\mathbf{d}_i$ in parallel according to

$$\underset{\mathbf{d}_j\in\{\pm 1\}^r}{\operatorname{argmin}} \mathbf{d}_j^T\Big(\sum_{i\in\mathcal{V}_j} \mathbf{b}_i\mathbf{b}_i^T\Big)\mathbf{d}_j - 2\Big(\sum_{i\in\mathcal{V}_j} S_{ij}\mathbf{b}_i^T\Big)\mathbf{d}_j - 2\beta\mathbf{y}_j^T\mathbf{d}_j. \tag{8}$$

where $\mathcal{V}_j$ is the observed rating set for item j. Denote d_{jk} as the k-th bit of $\mathbf{d}_j$ and $\mathbf{d}_{j\bar{k}}$ as the rest codes excluding d_{jk}, the DCD update for d_{jk} is given as:

$$d_{jk} \leftarrow \operatorname{sgn}\Big(K(\hat{d}_{jk}, d_{jk})\Big), \tag{9}$$

where $\hat{d}_{jk} = \sum_{i\in\mathcal{V}_j}\left(S_{ij} - \mathbf{b}_{i\bar{k}}^T\mathbf{d}_{j\bar{k}}\right)b_{ik} + \beta y_{jk}$.

X-subproblem. When $\mathbf{B}$, $\mathbf{D}$ and $\mathbf{Y}$ are fixed in Eq. (6), the $\mathbf{X}$-subproblem is:

$$\underset{\mathbf{X}}{\operatorname{argmax}}\ tr(\mathbf{B}^T\mathbf{X}), s.t.\ \mathbf{X1} = 0, \mathbf{X}\mathbf{X}^T = m\mathbf{I} \tag{10}$$

It can be solved with the aid of SVD. Denote $\overline{\mathbf{B}}$ is a row-wise zero-mean matrix, where $\overline{B}_{ij} = B_{ij} - \frac{1}{m}\sum_j B_{ij}$. By SVD,

[3] If linear algebra boosting library is available (*e.g.*, Matlab), the first term of $\hat{b}_{ik}$ can be rewritten in matrix form: $(\mathbf{D}_i\mathbf{s}_i)_k - \left(\mathbf{D}_i\mathbf{D}_i^T\right)_k^T\widehat{\mathbf{b}}_i$, where $\mathbf{D}_i$ is the subset of $\mathbf{D}$ selected by rows $j \in \mathcal{V}_i$, $\mathbf{s}_i$ is as $s_{ij} = S_{ij}$, and $\widehat{\mathbf{b}}_i$ is as $\hat{b}_{ik} = 0$. Similar form can be obtained for $\hat{d}_{jk}$ in Eq. (9).

we have $\overline{\mathbf{B}} = \mathbf{P}_b \mathbf{\Sigma}_b \mathbf{Q}_b^T$, where $\mathbf{P}_b \in \mathbb{R}^{r \times r'}$ and $\mathbf{Q}_b \in \mathbb{R}^{m \times r'}$ are left and right singular vectors corresponding to the r' ($\leq r$) positive singular values in the diagonal matrix $\mathbf{\Sigma}_b$. In practice, we first apply eigendecomposition for the small $r \times r$ matrix $\overline{\mathbf{B}}\ \overline{\mathbf{B}}^T = [\mathbf{P}_b\ \widehat{\mathbf{P}}_b] \begin{bmatrix} \mathbf{\Sigma}_b^2 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} [\mathbf{P}_b\ \widehat{\mathbf{P}}_b]^T$, where $\widehat{\mathbf{P}}_b$ are the eigenvectors of the zero eigenvalues. Therefore, by the definition of SVD, we have $\mathbf{Q}_b = \overline{\mathbf{B}}^T \mathbf{P}_b \mathbf{\Sigma}_b^{-1}$. In order to satisfy the constraint $\mathbf{X1} = \mathbf{0}$, we further obtain additional $\widehat{\mathbf{Q}}_b \in \mathbb{R}^{m \times (r-r')}$ by Gram-Schmidt orthogonalization based on $[\mathbf{Q}_b\ \mathbf{1}]$, thus, we have $\widehat{\mathbf{Q}}_b^T \mathbf{1} = \mathbf{0}$. The fact $\mathbf{Q}_b^T \mathbf{1} = \mathbf{0}$ is detailed in Appendix.

Now we are ready to obtain a closed-form update rule for the $\mathbf{X}$-subproblem in Eq. (10):

$$\mathbf{X} \leftarrow \sqrt{m}[\mathbf{P}_b\ \widehat{\mathbf{P}}_b][\mathbf{Q}_b\ \widehat{\mathbf{Q}}_b]^T. \tag{11}$$

$\mathbf{Y}$-subproblem. When $\mathbf{B}$, $\mathbf{D}$ and $\mathbf{X}$ are fixed in Eq. (5), the $\mathbf{Y}$-subproblem is:

$$\underset{\mathbf{Y}}{\text{argmax}}\ tr(\mathbf{D}^T\mathbf{Y}),\ s.t.\ \mathbf{Y1} = \mathbf{0}, \mathbf{Y}\mathbf{Y}^T = n\mathbf{I} \tag{12}$$

According to the above analysis, we can derive a closed form update rule for $\mathbf{Y}$ as:

$$\mathbf{Y} \leftarrow \sqrt{n}[\mathbf{P}_d\ \widehat{\mathbf{P}}_d][\mathbf{Q}_d\ \widehat{\mathbf{Q}}_d]^T, \tag{13}$$

where $\mathbf{P}_d$ and $\mathbf{Q}_d$ are the left and right singular vectors of the row-centered matrix $\overline{\mathbf{D}}$, $\widehat{\mathbf{P}}_d$ are the left singular vectors corresponding to zero singular values, and $\widehat{\mathbf{Q}}_d$ are the vectors obtained by Gram-Schimidt process based on $[\mathbf{Q}_d\ \mathbf{1}]$.

4.2 Out-of-Sample Extension

When new users, items and the corresponding ratings come in, it is impractical to retrain DCF for obtaining hashing codes of these out-of-sample data. Instead, an economical way is to learn ad-hoc codes for new data online and then update for the whole data offline when possible [25, 8].

Without loss of generality, we only discuss the case when a new user comes in. Denote $\{s_j | j \in \mathcal{N}\}$ as the set of observed ratings for existing items by the new user, whose binary codes are $\mathbf{b}$. For a single user, it is too expensive and unnecessary to impose the global balance and decorrelation constraints as batch DCF in Eq. (5). Therefore, we only focus on minimizing the rating prediction loss:

$$\underset{\mathbf{b} \in \{\pm 1\}^r}{\text{argmin}} \sum_{j \in \mathcal{N}} (s_j - \mathbf{b}^T \mathbf{d}_j)^2 \tag{14}$$

It is easy to see that Eq. (14) is a special case of $\mathbf{B}$-subproblem in Eq. (6). Therefore, we can quickly develop the DCD update rule for the k-th bit b_k of $\mathbf{b}$ as:

$$b_k \leftarrow \text{sgn}\left(K(\hat{b}_k, b_k)\right), \tag{15}$$

where $\hat{b}_k = \sum_{j \in \mathcal{N}} \left(s_j - \mathbf{d}_{j\bar{k}}^T \mathbf{b}_{\bar{k}}\right) d_{jk}$.

Similarly, for a new item, whose ratings are $\{s_i | i \in \mathcal{N}\}$ made by existing users, the update rule for the k-th bit d_k of the new item codes $\mathbf{d}$ is:

$$d_k \leftarrow \text{sgn}\left(K(\hat{d}_k, d_k)\right), \tag{16}$$

where $\hat{d}_k = \sum_{i \in \mathcal{N}} \left(s_i - \mathbf{b}_{i\bar{k}}^T \mathbf{d}_{\bar{k}}\right) b_{ik}$.

Algorithm 1: Discrete Collaborative Filtering

Input : $\{S_{ij} | i, j \in \mathcal{V}\}$: observed user-item ratings,
r: code length,
α and β: trade-off parameter
Output: $\mathbf{B} \in \{\pm 1\}^{r \times m}$: user codes,
$\mathbf{D} \in \{\pm 1\}^{r \times n}$: item codes

1 **Initialization**: $\mathbf{B}$, $\mathbf{D}$, $\mathbf{X} \in \mathbb{R}^{r \times m}$ and $\mathbf{Y} \in \mathbb{R}^{r \times n}$ by Eq. (18)
2 **repeat**
  // B-subproblem, parallel outer for loop
3 **for** *i=1* **to** m **do**
4 **repeat**
5 **for** *k=1* **to** r **do**
6 $\hat{b}_{ik} \leftarrow \sum_{j \in \mathcal{V}_i} \left(S_{ij} - \mathbf{d}_{j\bar{k}}^T \mathbf{b}_{i\bar{k}}\right) d_{jk} + \alpha x_{ik}$;
7 $b_{ik} \leftarrow \text{sgn}\left(K(\hat{b}_{ik}, b_{ik})\right)$;
8 **end**
9 **until** *converge*;
10 **end**
  // D-subproblem, parallel outer for loop
11 **for** *j=1* **to** n **do**
12 **repeat**
13 **for** *k=1* **to** r **do**
14 $\hat{d}_{jk} \leftarrow \sum_{i \in \mathcal{V}_j} \left(S_{ij} - \mathbf{b}_{i\bar{k}}^T \mathbf{d}_{j\bar{k}}\right) b_{ik} + \beta y_{jk}$;
15 $d_{jk} \leftarrow \text{sgn}\left(K(\hat{d}_{jk}, d_{jk})\right)$;
16 **end**
17 **until** *converge*;
18 **end**
  // X-subproblem
19 $\left([\mathbf{P}_b \widehat{\mathbf{P}}_b], \mathbf{Q}_b\right) \leftarrow \text{SVD}(\overline{\mathbf{B}})$, $\widehat{\mathbf{Q}}_b \leftarrow \text{GramSchmidt}\left([\mathbf{Q}_b 1]\right)$;
20 $\mathbf{X} \leftarrow \sqrt{m}[\mathbf{P}_b\ \widehat{\mathbf{P}}_b][\mathbf{Q}_b\ \widehat{\mathbf{Q}}_b]^T$;
  // Y-subproblem
21 $\left([\mathbf{P}_d \widehat{\mathbf{P}}_d], \mathbf{Q}_d\right) \leftarrow \text{SVD}(\overline{\mathbf{D}})$, $\widehat{\mathbf{Q}}_d \leftarrow \text{GramSchmidt}\left([\mathbf{Q}_d 1]\right)$;
22 $\mathbf{Y} \leftarrow \sqrt{n}[\mathbf{P}_d\ \widehat{\mathbf{P}}_d][\mathbf{Q}_d\ \widehat{\mathbf{Q}}_d]^T$;
23 **until** *converge*;
24 **return** $\mathbf{B}$, $\mathbf{D}$

5. ALGORITHMIC ANALYSIS

We summarize the solution for DCF in Algorithm 1. We will discuss the convergence, complexity and initialization issues in this section.

5.1 Convergence

The convergence of the proposed DCF algorithm is guaranteed by the following theorem.

Theorem 1 (Convergence of Algorithm 1). *The sequence $\{\mathbf{B}^{(t)}, \mathbf{D}^{(t)}, \mathbf{X}^{(t)}, \mathbf{Y}^{(t)}\}$ generated by Algorithm 1 monotonically decreases the objective function L of Eq. (5); the objective function sequence $\{L\left(\mathbf{B}^{(t)}, \mathbf{D}^{(t)}, \mathbf{X}^{(t)}, \mathbf{Y}^{(t)}\right)\}$ converges; the sequence $\{\mathbf{B}^{(t)}, \mathbf{D}^{(t)}, \mathbf{X}^{(t)}, \mathbf{Y}^{(t)}\}$ converges.*

In nutshell, we need to prove two key facts. First, we show that the updating steps in Step 7, 15, 20 and 22 monotonically decreases the objective function in Eq. (5), which is proved to be bounded below. Then, we use the fact that $L(\mathbf{B}, \mathbf{D})$ has finite values to show that $\{\mathbf{B}^{(t)}, \mathbf{D}^{(t)}\}$ converges. See Appendix for detailed proof.

5.2 Initialization

Since DCF deals with mixed-integer non-convex optimization, initialization is crucial for better convergence and local optimum. Here, we suggest an efficient initialization

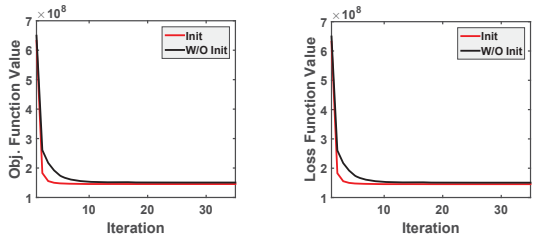

Figure 3: Convergence curve of the overall objective function value (left) and the squared loss value (right) using DCF with/without initializations on Yelp ($\alpha = \beta = 0.001$). We can see that the proposed initialization strategy helps to achieve faster convergence and lower objective/loss values.

heuristic, which essentially relaxed the binary constraints in Eq. (5) as:

$$\begin{aligned}\underset{\mathbf{U},\mathbf{V},\mathbf{X},\mathbf{Y}}{\operatorname{argmin}} \sum_{i,j\in\mathcal{V}} \left(S_{ij} - \mathbf{u}_i^T\mathbf{v}_j\right)^2 + \alpha\|\mathbf{U}\|_F^2 + \beta\|\mathbf{V}\|_F^2 \\ - 2\alpha tr(\mathbf{U}^T\mathbf{X}) - 2\beta tr(\mathbf{V}^T\mathbf{Y}) \\ s.t., \mathbf{X1} = 0, \mathbf{X}\mathbf{X}^T = m\mathbf{I}, \mathbf{Y1} = 0, \mathbf{Y}\mathbf{Y}^T = n\mathbf{I}.\end{aligned} \tag{17}$$

In fact, we can consider the above initialization as traditional CF in Eq. (1) with balance and decorrelation constraints imposed on real-valued $\mathbf{U}$ and $\mathbf{V}$. A possible explanation for why Eq. (17) may offer better initialization is illustrated in Figure 2, where the two constraints can restrict real-valued results within a subspace with small quantization error.

To solve Eq. (17), we can further initialize real-valued $\mathbf{U}$ and $\mathbf{V}$ randomly and find feasible initializations for $\mathbf{X}$ and $\mathbf{Y}$ by solving $\mathbf{X}/\mathbf{Y}$-subproblems in Section 4. Then, the optimization can be done alternatively by solving $\mathbf{U}$ and $\mathbf{V}$ by traditional CF in Eq. (1), and solving $\mathbf{X}$ and $\mathbf{Y}$ by $\mathbf{X}/\mathbf{Y}$-subproblem. Suppose the solutions are $(\mathbf{U}^*, \mathbf{V}^*, \mathbf{X}^*, \mathbf{Y}^*)$, we can initialize Algorithm 1 as:

$$\mathbf{B} \leftarrow \operatorname{sgn}(\mathbf{U}^*), \mathbf{D} \leftarrow \operatorname{sgn}(\mathbf{V}^*), \mathbf{X} \leftarrow \mathbf{X}^*, \mathbf{Y} \leftarrow \mathbf{Y}^*. \tag{18}$$

It is easy to see that the initializations above are feasible to Eq. (5). The effectiveness of the proposed initialization is illustrated in Figure 3.

5.3 Complexity

For space complexity, Algorithm 1 requires $\mathcal{O}(|\mathcal{V}|)$ for storing $\{S_{ij}\}$ and $\mathcal{O}(r \cdot \max(m, n))$ for $\mathbf{B}$, $\mathbf{D}$, $\mathbf{X}$ and $\mathbf{Y}$. As r is usually less than 256 bits, we can easily store the above variables at large-scale in memory.

We first analyze the time complexity for each of the subproblems. For $\mathbf{B}$-subproblem, it takes $\mathcal{O}(r^2|\mathcal{V}_i|T_s)$ for completing the inner loop of updating $\mathbf{b}_i$, where T_s is the number of iterations needed for convergence (Step 4 to 8). Further, suppose we have p computing threads, then the overall complexity for $\mathbf{B}$-subproblem is $\mathcal{O}(r^2T_s|\mathcal{V}|/p)$. Similarly, the overall complexity for $\mathbf{D}$-subproblem is also $\mathcal{O}(r^2T_s|\mathcal{V}|/p)$. In practice, T_s is usually 2∼5. For $\mathbf{X}$-subproblem, it requires $\mathcal{O}(r^2m)$ to perform the SVD and Gram-Schimdt orthogonalization in Step 19, and $\mathcal{O}(r^2m)$ matrix multiplication in Step 20. Similarly, it takes $\mathcal{O}(r^2n)$ for solving $\mathbf{Y}$-subproblem. Suppose the entire algorithm requires T iterations for convergence, the overall time complexity for Algorithm 1 is $\mathcal{O}\left(Tr^2\left(T_s|\mathcal{V}|\frac{1}{p} + m + n\right)\right)$, where we observe that T is usually 10∼20. In summary, training DCF is efficient since it scales linearly with the size of the data, *e.g.*, $|\mathcal{V}|$ and $m + n$.

6. EXPERIMENTS

As the proposed DCF fundamentally tackles the problem of quantization loss caused by traditional hashing methods of CF, the goal of our experiments is to answer the following three research questions:

RQ1 How does DCF perform as compared to other state-of-the-art hashing methods?

RQ2 Does DCF generalize well to new users? If yes, how much training overhead is needed?

RQ3 How do the discrete, balanced and uncorrelated constraints contribute to the overall effectiveness of DCF?

Table 1: Statistics of datasets in evaluation.

Dataset	Rating#	User#	Item#	Density
Yelp	696,865	25,677	25,815	0.11%
Amazon	5,057,936	146,469	189,474	0.02%
Netflix	100,480,507	480,189	17,770	1.18%

6.1 Datasets

We used three publicly available datasets from various real-world online websites:

Yelp: This is the latest Yelp Challenge dataset[4]. It originally includes 366,715 users, 60,785 items (*e.g.*, restaurants and shopping malls), and 1,569,264 ratings.

Amazon: This is a collection of user ratings on Amazon products of Book category [21], which originally contains 2,588,991 users, 929,264 items and 12,886,488 ratings.

Netflix: This is the classic movie rating dataset used in the Netflix challenge[5]. We use the full dataset that contains 480,189 users, 17,770 items and 100,480,507 ratings.

Due to the severe sparsity of Yelp and Amazon original datasets, we follow the convention in evaluating CF algorithms [24] by removing users and items that have less than 10 ratings[6]. When a user rates an item multiple times, we merge them into one rating by averaging the duplicate rating scores. Table 1 summarizes the filtered, experimental datasets. For each user, we randomly sampled 50% ratings as training and the rest 50% as testing. We repeated for 5 random splits and reported the averaged results.

6.2 Setups

6.2.1 Evaluation Metrics

As practical recommender systems usually generate a ranked list of items for user, we diverge from the error-based measure (*e.g.,* RMSE [15]), which is a suboptimal to recommendation task [3]. Instead, we treat it as a ranking problem, evaluating the ranking performance on test ratings with NDCG (Normalized Discounted Cumulative Gain), which is a widely used measure for evaluating recommendation algorithms [30, 3], owing to its comprehensive consideration of both ranking precisions and the positions of ratings.

[4] http://www.yelp.com/dataset_challenge
[5] http://www.netflixprize.com
[6] Note the similar filtering was conducted to Netflix dataset [15], so we use the Netflix dataset as-it-is.

6.2.2 Search Protocols

We adopted two search protocols which are widely used in search with binary codes. We used code length within $\{8, 16, 32, 64, 128, 256\}$.

Hamming Ranking: Items are ranked according to their Hamming distance (or similarity) from the query user. Although the search complexity of Hamming ranking is still linear, it is very fast in practice since the Hamming distance (or similarity) calculation can be done by fast bit operations and the sorting is constant time due to integer distance.

Hashtable Lookup: A lookup table is constructed using the item codes and all the items in the buckets that fall within a small Hamming radius (*e.g.*, 2) of the query user are returned. Therefore, search is performed in constant time. However, a single table would be insufficient when the code length is larger than 32 since it would require over $\mathcal{O}(2^{32})$ space to store the table in memory. We adopted Multi-Index Hashing (MIH) table [22], which builds one table for each code subsegment. Items are aggregated by all the tables and then conducted Hamming ranking for the items. By doing this, the search time is significantly reduced to sublinear—linear scan a short returned list. We empirically set the substring length as $\{1, 1, 2, 2, 4, 4\}$ for bit size $\{8, 16, 32, 64, 128, 256\}$ as suggested in [32].

It is worth mentioning that the above two search protocols focus on different characteristics of hashing codes. Hamming ranking provides a better measurement of the learned Hamming space, *i.e.*, the accuracy upper bound that the codes can achieve since it linearly scans the whole data. Hashtable lookup, on the other hand, emphasizes the practical speed of large-scale search. However, a common issue in this protocol is that it may not return sufficient items for recommendation, as a query lookup may miss due to the sparse Hamming space. In our experiments, if a query user returns no items, we treated as a failed query with an NDCG of zero.

6.2.3 Compared Methods

We benchmark the performance using the traditional real-valued CF method, comparing with several state-of-the-art hashing-based CF methods:

MF: This is the classic Matrix Factorization based CF algorithm [16], which learns user and item latent vectors in Euclidean space. We used MF as a baseline to show the performance gap between real values and binary codes. We adopt the ALS algorithm suggested by [34]. Note that it is beyond the scope of this paper to further investigate other popular variants of MF [15, 19].

BCCF: This is a two-stage Binary Code learning method for Collaborative Filtering [33]. At the relaxation stage, it imposes a balanced code regularization instead of the ℓ_2-norm regularization of MF; at quantization stage, it applies orthogonal rotation to user and item features, which is essentially an ITQ method [9].

PPH: This is a two-stage Preference Preserving Hashing [32]. At the relaxation stage, different from MF, it encourages the latent feature norm to reach the maximum ratings and hence smaller discrepancy between inner product (preference preserving) and cosine similarity (hashing friendly) is expected. At quantization stage, PPH quantizes each feature vector into $(r-2)$-bit phase codes and 2-bit magnitude codes. Therefore, in order to keep the code length consistent to other methods, we only learned $(r-2)$-dim latent features at relaxation stage.

CH: Collaborative Hashing [18] is also a two-stage approach, where the relaxation stage is based on full-matrix factorization, *i.e.*, unobserved ratings are considered as zeros. Since the original CH is formulated for visual features, we implemented CH for collaborative filtering as: $\arg\min_{\mathbf{U},\mathbf{V}} \|\mathbf{S} - \mathbf{U}^T\mathbf{V}\|_F^2$, *s.t.*, $\mathbf{U}\mathbf{U}^T = m\mathbf{I}, \mathbf{V}\mathbf{V}^T = n\mathbf{I}$, where $\mathbf{S}$ is the scaled rating matrix where $S_{ij} = 0$ if (i, j) is not observed, *i.e.*, $i, j \notin \mathcal{V}$. By several algebraic transformations, the above problem can be cast into alternating solving SVDs for $\mathbf{US}$ and $\mathbf{VS}^T$, which is analogous to our $\mathbf{X}/\mathbf{Y}$-subproblem. Then, the binary quantization is simply $\text{sgn}(\mathbf{U})$ and $\text{sgn}(\mathbf{V})$.

We also compared the following two settings of DCF to investigate its effectiveness:

MFB: This is the MF results with round-off binary quantization. We used this as a baseline to show how quantization loss degrades performance.

DCFinit: This is the Initialization problem of DCF in Eq. (17), whose binary codes are given in Eq. (18). By comparing with MFB, we can investigate whether the balance and decorrelation constraints are useful for binary quantization. Moreover, DCFinit can be considered as a relaxed two-stage version of DCF. Compared with DCF, we can see whether discrete optimization is effective.

All the hyper-parameters (*e.g.*, α and β) of DCF and the above methods were tuned within $\{1e^{-4}, 1e^{-3}, ..., 1e^{2}\}$ by 5-fold cross validation on the training split. We used MATLAB with mex C for algorithm implementations and we run them on a cluster of 9 machines, each of which has 6-core 2.4GHz CPU and 48GB RAM.

6.3 Result Analysis

6.3.1 Comparison with State-of-The-Arts (RQ1)

Figure 6 shows the performances (NDCG@K) of DCF and the three state-of-the-art hashing methods in terms of Hamming ranking and table lookup. We can have the following key observations:

1). The proposed DCF considerably outperforms all the state-of-the-art methods. For example, as shown in Table 2, in most cases, DCF can even achieve significantly better performance by using only 8 bits as compared to the most competitive CH using 128 bits. This suggests that DCF can reduce a huge amount of memory and time cost as compared to the state-of-the-art methods. One possible reason why CH outperforms BCCF and PPH is that it incorporates the uncorrelated codes constraints during joint optimization, which is beneficial for the subsequent quantization. DCF, on the other hand, minimizing the quantization loss directly by joint discrete optimization with balanced and uncorrelated code constraints, resulting in better performance than CH.

2). When using lookup tables, the performances of all the methods are not as stable as those by using ranking, specifically, at larger K positions on smaller datasets like Yelp and Amazon. This is due to that zero item is returned when lookup misses—Hamming ranking treats the entire items as candidates, while lookup table only indexes a small fraction of items as candidate. This is why NDCG at smaller K positions does not significantly decrease as compared to Hamming ranking. Therefore, as we can observe that DCF considerably outperforms other methods, we can infer that the codes generated by DCF have less lookup misses.

3). As the bit size increases, the performance gap between DCF and other methods becomes larger, especially when

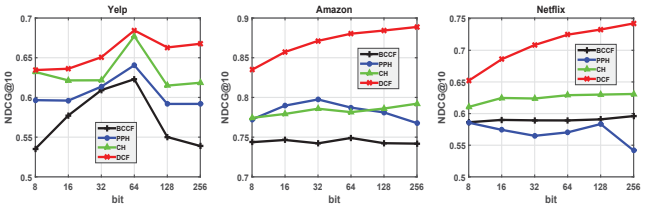

Figure 4: Recommendation performance (NDCG@10) on 50% held-out "new" users (RQ 2).

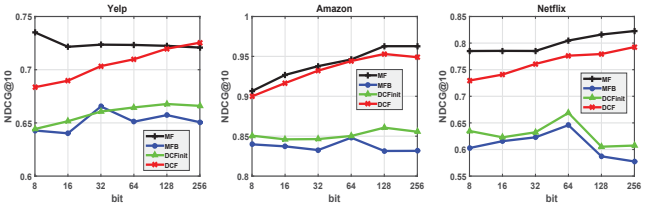

Figure 5: Recommendation performance (NDCG@10) of CF and DCF variants (RQ 3).

using lookup. This demonstrates that the two-stage approaches (BCCF, PPH and CH) increasingly suffer from quantization loss as code length increases.

6.3.2 Generalization to New Users (RQ2)

In this study, we performed the strong generalization test. Following [27], we trained all models on 50% of randomly sampled users with full history; the remaining 50% of users are the new users for testing. In the testing phase, we fed 50% ratings of the test users into the model for an out-of-sample update (cf. Section 4.2), obtaining the hashing codes for test users. The performance was then evaluated against the remaining 50% ratings for the test users. In this way, we simulated the online learning scenario, where a full retraining is prohibitive and the model needs to be instantly refreshed to better serve new users.

Figure 4 shows the performance of the generalization test. We can see that DCF consistently outperforms other methods. The key reason is that the out-of-sample hashing by DCF preserves the original binary separations of the Hamming space by directly performing discrete optimization. On the other hand, since BCCF, PPH and CH are all two-stage approaches, the latent vectors of new users are first approximated with fixed item vectors. As a result, the subsequent quantization loss will be more severe. We can also see that the performances drop of DCF caused by less training data is acceptable, *e.g.*, only 7% NDCG@10 drop as compared to the results obtained by original data.

Table 2: Performance (NDCG@10) of various methods (#bit) using two search protocols on three datasets. * denotes the statistical significance for $p < 0.05$.

Protocol	Hamming Ranking			Table Lookup		
Dataset	Yelp	Amazon	Netflix	Yelp	Amazon	Netflix
BCCF(128)	0.623	0.827	0.633	0.268	0.300	0.262
PPH(128)	0.643	0.841	0.587	0.626	0.799	0.566
CH(128)	0.655	**0.922***	0.693	0.647	0.815	0.680
DCF(8)	**0.684***	0.890	**0.730***	**0.674***	**0.825***	**0.710***

To show the training overhead, we show the time cost for hashing new users of the largest dataset Netflix in Table 3. First, we see CH requires the least hashing time, followed by our DCF method. The efficiency of CH is owing to its simple design — which adopts the conventional SVD and only needs to apply a matrix multiplication operation for hashing new users. Although the proposed DCF is about tens of times slower than CH, we believe this overhead is acceptable since DCF considerably generalizes better than CH. Comparing with BCCF and PPH, our DCF is about 20 times faster. The reason is that both BCCF and PPH need to iteratively solve least square problems, which involve the expensive matrix inverse. In addition, BCCF requires two matrix multiplications after least square, *i.e.*, PCA projection and ITQ rotation, and hence needs more time. In contrast, our DCF only requires Tr matrix multiplications in total, where T is 2∼5 and r is the code length.

Table 3: Time cost (s) of various methods hashing 240,094 new users of Netflix using a single 6-core CPU.

Method/#Bits	8	16	32	64	128	256
BCCF	51.3	64.5	78.0	$1.21e^2$	$4.81e^2$	$2.69e^3$
PPH	51.2	64.3	77.5	$1.21e^2$	$4.81e^2$	$2.69e^3$
CH	$2.12e^{-1}$	$4.11e^{-1}$	$8.37e^{-1}$	1.66	3.34	6.71
DCF	2.25	3.16	5.18	9.52	29.6	93.4

6.3.3 Impact of Discrete, Balanced and Uncorrelated Constraints (RQ3)

As shown in Figure 5, it is not surprising that real-valued user/item features—MF outperforms other user/item binary codes. However, if we simply quantize MF features to MFB binary codes, significant performance drop can be observed, especially as the bit size increases. As analyzed above, this is due to the quantization loss caused by the two-stage approaches. Interestingly, by adding the balanced and uncorrelated constraints, DCFinit generally outperforms MFB. An intuitive explanation is illustrated in Figure 2 that the two constraints can shift and rotate the real-valued features to a new user-item manifold that is beneficial for the binary split of the original vector space into Hamming space, and thus less quantization loss is expected. Moreover, we can see that DCF significantly outperforms DCFinit. This demonstrates the superiority of the proposed joint discrete optimization over the two-stage approaches.

7. CONCLUSIONS

In this paper, a novel hashing approach dubbed Discrete Collaborative Filtering (DCF) was proposed to enable efficient collaborative filtering. In sharp contrast to existing collaborative filtering hashing methods which are generally in a two-stage fashion, DCF directly learns binary codes for users and items according to the Hamming similarity induced rating loss. Through extensive experiments carried out on three benchmarks, we demonstrated that the main disadvantage of those two-stage hashing methods is the severe quantization loss caused by the inconsistency between input real-valued features and subsequent binary quantization. Beyond conventional two-stage methods that are not entirely optimized for hashing, our proposed DCF completes discrete optimization inherent to hashing and therefore achieves considerable performance gain over state-of-the-art collaborative filtering hashing techniques.

To the best of our knowledge, DCF is the first principled learning-to-hash framework for accelerating CF, so we believe that it has a great potential to advance real-world recommendation systems since DCF can effectively compress

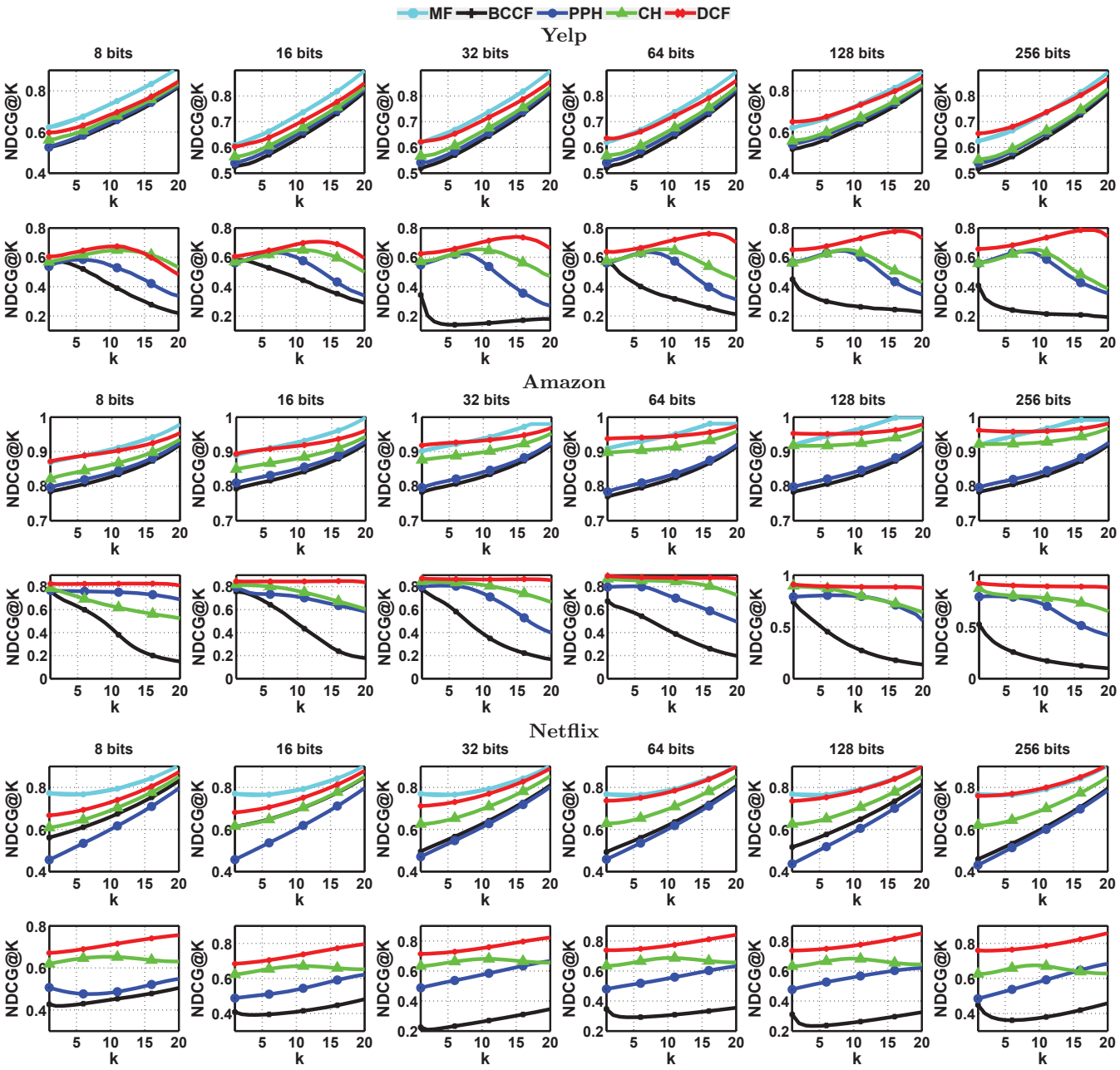

Figure 6: Item recommendation performance (NDCG@K) of all the CF hashing methods of various code lengths. For each dataset, the top row is the performance by Hamming ranking and the bottom row is by table lookup (RQ 1). As MF is continuous, we used cosine ranking and table lookup is not applied.

gigantic users/items to compact binary codes. As moving forward, we are going to apply DCF to various CF applications [27, 13] as well as more generic feature-based factorization approaches [23].

Acknowledgements

NExT research is supported by the National Research Foundation, Prime Minister's Office, Singapore under its IRC@SG Funding Initiative.

8. REFERENCES

[1] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *TKDE*, 2005.

[2] D. Agarwal, B.-C. Chen, and P. Elango. Fast online learning through offline initialization for time-sensitive recommendation. In *KDD*, 2010.

[3] S. Balakrishnan and S. Chopra. Collaborative ranking. In *WSDM*, 2012.

[4] J. Bennett and S. Lanning. The netflix prize. In *KDD*, 2007.

[5] J. Bobadilla, F. Ortega, A. Hernando, and A. Gutiérrez. Recommender systems survey. *Knowledge-Based Systems*, 2013.

[6] A. S. Das, M. Datar, A. Garg, and S. Rajaram. Google news personalization: scalable online collaborative filtering. In *WWW*, 2007.

[7] A. Gionis, P. Indyk, R. Motwani, et al. Similarity search in high dimensions via hashing. In *VLDB*, 1999.

[8] K. Goldberg, T. Roeder, D. Gupta, and C. Perkins. Eigentaste: A constant time collaborative filtering algorithm. *Information Retrieval*, 2001.
[9] Y. Gong, S. Lazebnik, A. Gordo, and F. Perronnin. Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval. *TPAMI*, 2013.
[10] E. F. Harrington. Online ranking/collaborative filtering using the perceptron algorithm. In *ICML*, 2003.
[11] J. Håstad. Some optimal inapproximability results. *JACM*, 2001.
[12] X. He, T. Chen, M.-Y. Kan, and X. Chen. Trirank: Review-aware explainable recommendation by modeling aspects. In *CIKM*, 2015.
[13] X. H. He, H. Zhang, M.-Y. Kan, and T.-S. Chua. Fast matrix factorization for online recommendation with implicit feedback. In *SIGIR*, 2016.
[14] A. Karatzoglou, M. Weimer, and A. J. Smola. Collaborative filtering on a budget. In *AISTAT*, 2010.
[15] Y. Koren. Collaborative filtering with temporal dynamics. In *KDD*, 2009.
[16] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 2009.
[17] W. Liu, C. Mu, S. Kumar, and S.-F. Chang. Discrete graph hashing. In *NIPS*, 2014.
[18] X. Liu, J. He, C. Deng, and B. Lang. Collaborative hashing. In *CVPR*, 2014.
[19] H. Ma, D. Zhou, C. Liu, M. R. Lyu, and I. King. Recommender systems with social regularization. In *WSDM*, 2011.
[20] J. Mairal, F. Bach, J. Ponce, and G. Sapiro. Online learning for matrix factorization and sparse coding. *JMLR*, 2010.
[21] J. McAuley and J. Leskovec. Hidden factors and hidden topics: Understanding rating dimensions with review text. In *RecSys*, 2013.
[22] M. Norouzi, A. Punjani, and D. J. Fleet. Fast search in hamming space with multi-index hashing. In *CVPR*, 2012.
[23] S. Rendle. Scaling factorization machines to relational data. *VLDB*, 2013.
[24] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. In *UAI*, 2009.
[25] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. In *WWW*, 2001.
[26] F. Shen, C. Shen, W. Liu, and H. T. Shen. Supervised discrete hashing. In *CVPR*, 2015.
[27] M. Volkovs and G. W. Yu. Effective latent models for binary feedback in recommender systems. In *SIGIR*, 2015.
[28] J. Wang, S. Kumar, and S.-F. Chang. Semi-supervised hashing for large-scale search. *TPAMI*, 2012.
[29] J. Wang, W. Liu, S. Kumar, and S.-F. Chang. Learning to hash for indexing big data: A survey. *Proceedings of the IEEE*, 2016.
[30] M. Weimer, A. Karatzoglou, Q. V. Le, and A. Smola. Maximum margin matrix factorization for collaborative ranking. *NIPS*, 2007.
[31] Y. Weiss, A. Torralba, and R. Fergus. Spectral hashing. In *NIPS*, 2009.
[32] Z. Zhang, Q. Wang, L. Ruan, and L. Si. Preference preserving hashing for efficient recommendation. In *SIGIR*, 2014.
[33] K. Zhou and H. Zha. Learning binary codes for collaborative filtering. In *KDD*, 2012.
[34] Y. Zhou, D. Wilkinson, R. Schreiber, and R. Pan. Large-scale parallel collaborative filtering for the netflix prize. In *AAIM*, 2008.

APPENDIX

Derivation of Eq. (7). Without loss of generality, suppose $\mathbf{b}_i = [\mathbf{b}_{i\bar{k}}^T \; b_{ik}]^T$ and $\mathbf{d}_j = [\mathbf{d}_{j\bar{k}}^T \; d_{jk}]^T$, the quadratic term in Eq. (6) w.r.t b_{ik} can be rewritten as:

$$\mathbf{b}_i^T(\sum_{j\in\mathcal{V}_i}\mathbf{d}_j\mathbf{d}_j^T)\mathbf{b}_i = \sum_{j\in\mathcal{V}_i}(\mathbf{d}_j^T\mathbf{b}_i)^2 = 2b_{ik}\sum_{j\in\mathcal{V}_i}\left(\mathbf{d}_{j\bar{k}}^T\mathbf{b}_{i\bar{k}}d_{jk}\right) + \underbrace{\sum_{j\in\mathcal{V}_i}\left((\mathbf{d}_{j\bar{k}}^T\mathbf{b}_{i\bar{k}})^2 + (d_{jk}b_{ik})^2\right)}_{\text{constant}}, \quad (19)$$

and the rest two linear terms w.r.t. b_{ik} can be rewritten as:

$$2(\sum_{j\in\mathcal{V}_i}S_{ij}\mathbf{d}_j^T)\mathbf{b}_i - 2\alpha\mathbf{x}_i^T\mathbf{b}_i = -2b_{ik}\sum_{j\in\mathcal{V}_i}S_{ij}d_{jk} - 2\alpha x_{ik}b_{ik} - \underbrace{2\sum_{j\in\mathcal{V}_i}S_{ij}\mathbf{d}_{j\bar{k}}^T\mathbf{b}_{i\bar{k}} - 2\alpha\mathbf{x}_{i\bar{k}}^T\mathbf{b}_{i\bar{k}}}_{\text{constant}}. \quad (20)$$

Therefore, we can derive a set of bit-wise minimizations:

$$\underset{\mathbf{b}_{ik}\in\{\pm 1\}}{\text{argmin}} \; -\hat{b}_{ik}b_{ik}, \quad (21)$$

where $\hat{b}_{ik} = \sum_{j\in\mathcal{V}_i}\left(S_{ij} - \mathbf{d}_{j\bar{k}}^T\mathbf{b}_{i\bar{k}}\right)d_{jk} + \alpha x_{ik}$. It is easy to see that the optimized b_{ik} should be the sign of $\hat{b}_{ik}$. Therefore, the update rule for b_{ik} can be given in Eq. (7). Tthe update rule for d_{jk} in Eq. (9) can be derived in a similar way.

Proof of Theorem 1. We prove the three parts of Theorem 1 one by one.

Part 1. It is easy to see that $\{\mathbf{B}^{(t)}, \mathbf{D}^{(t)}\}$ generated by the DCD Step 7 and Step 15 monotonically decreases the objective functions of **B**-subproblem and **D**-subproblem in Eq. (6) and Eq. (8), respectively. Without loss of generality, we show that $\{\mathbf{X}^{(t)}\}$ generated by Step 20 monotonically decreases the objective function of **X**-subproblem in Eq. (10). In particular, denote $\mathbf{X}'$ as the updated $\mathbf{X}$ at each step, we show that $\mathbf{X}'$ maximizes the **X**-subproblem.

We first show that $\mathbf{X}'$ is feasible, *i.e.*, $\mathbf{X}' \in \mathcal{B}$. Note that $\overline{\mathbf{B}} = \mathbf{B}\mathbf{J}$, where $J = \mathbf{I} - \frac{1}{m}\mathbf{1}\mathbf{1}^T$. Since $\overline{\mathbf{B}}^T$ and $\mathbf{Q}_b$ has the same row space, we have $\mathbf{Q}_b^T\mathbf{1} = 0$ due to $\overline{\mathbf{B}}\mathbf{1} = 0$. As we construct $\widehat{Q}_b$ such that $\mathbf{1}^T\widehat{\mathbf{Q}}_b = \mathbf{0}$, we have $\mathbf{1}^T[\mathbf{Q}_b\widehat{\mathbf{Q}}_b] = \mathbf{1}$, which implies $\mathbf{X}'\mathbf{1} = \mathbf{0}$. Moreover, it is easy to show that $\mathbf{X}\mathbf{X}^T = m[\mathbf{P}_b\widehat{\mathbf{P}}_b][\mathbf{Q}_b\widehat{\mathbf{Q}}_b]^T[\mathbf{Q}_b\widehat{\mathbf{Q}}_b][\mathbf{P}_b\widehat{\mathbf{P}}_b]^T = m\mathbf{I}$. Therefore, $\mathbf{X}'$ is feasible. As SVD $\mathbf{B}\mathbf{J} = [\mathbf{P}_b\ \widehat{\mathbf{P}}_b]\begin{bmatrix}\mathbf{\Sigma}_b & \mathbf{0}\\ \mathbf{0} & \mathbf{0}\end{bmatrix}[\mathbf{Q}_b\ \widehat{\mathbf{Q}}_b]^T$, we have $tr(\mathbf{X}'\overline{\mathbf{B}}^T) = \sqrt{m}\sum_{k=1}^r\sigma_k$, where σ_k is diagonal eigenvalues of $\mathbf{\Sigma}_b$. $\forall\mathbf{X}\in\mathcal{B}$, by using von Neumann's trace inequality, we have $tr(\mathbf{X}\overline{\mathbf{B}}^T) \le \sqrt{m}\sum_{k=1}^r\sigma_k$. Due to $\mathbf{X}\mathbf{1} = \mathbf{0}$, we have $\mathbf{X}\mathbf{J} = \mathbf{X}$. So, $\mathbf{X}'$ is a maximizer. Thus far, we have $tr(\mathbf{X}\mathbf{B}^T) = tr(\mathbf{X}\overline{\mathbf{B}}^T) \le tr(\mathbf{X}'\overline{\mathbf{B}}^T) = tr(\mathbf{X}'\mathbf{B}^T)$.

Part 2. We show that the loss function $L(\mathbf{B},\mathbf{D},\mathbf{X},\mathbf{Y})$ in Eq. (5) is lower bounded. By Cauchy-Schwartz inequality $tr(B^T\mathbf{X}) \le \|\mathbf{B}\|_F\|\mathbf{X}\|_F$, we have

$$\begin{aligned}L(\mathbf{B},\mathbf{D},\mathbf{X},\mathbf{Y}) &\ge 0 - 2\alpha\|\mathbf{B}\|_F\|\mathbf{X}\|_F - 2\beta\|\mathbf{D}\|_F\|\mathbf{Y}\|_F\\ &\ge -2\alpha\sqrt{mr}\sqrt{mr} - 2\beta\sqrt{nr}\sqrt{nr} = -2\alpha mr - 2\beta nr.\end{aligned} \quad (22)$$

Thus, together with the monotonic decrease proved in Part 1, we can conclude that $\{L(\mathbf{B}^{(t)},\mathbf{D}^{(t)},\mathbf{X}^{(t)},\mathbf{Y}^{(t)})\}$ converges.

Part 3. It is easy to see that $L(\mathbf{X},\mathbf{Y},\mathbf{B},\mathbf{D})$ is Lipschitz continuous w.r.t. $\mathbf{X}$ and $\mathbf{Y}$, thus, $\{\mathbf{X}^{(t)},\mathbf{Y}^{(t)}\}$ converges. Therefore, it is sufficient to prove $\{\mathbf{B}^{(t)},\mathbf{D}^{(t)}\}$ converges.

Without loss of generality, we drop the subscript i and only show that $\exists T$ such that $\forall k$, $t > T$, $b_k^{(t+1)} = b_k^{(t)}$. Suppose there is a k' such that $b_{k'}^{(t+1)} \ne b_{k'}^{(t)}$, due to the update rule in Eq. (7), $\hat{b}_{k'} \ne 0$, which implies a strict decrease for the objective function $L(\mathbf{b})$. Since $\mathbf{b}$ is binary valued, $L(\mathbf{b})$ has finite values. If such k' exists, it will lead to infinite values, which contradicts the fact. Therefore, we conclude that $\{\mathbf{B}^{(t)}\}$ converges. Moreover, since $L(\mathbf{B},\mathbf{D})$ has finite values, there exists $\mathbf{B}^{(t+1)}$ such that $\mathbf{D}^{(t+1)} = \mathbf{D}^{(t)}$. Therefore, $\{\mathbf{B}^{(t)},\mathbf{D}^{(t)}\}$ converges and so does $\{\mathbf{B}^{(t)},\mathbf{D}^{(t)},\mathbf{X}^{(t)},\mathbf{Y}^{(t)}\}$. □