Interactive Constrained Boolean Matrix Factorization

Nelson Mukuze
Max-Planck-Institut für Informatik
Saarbrücken, Germany
nelson.mukuze@mpi-inf.mpg.de

Pauli Miettinen*
Max-Planck-Institut für Informatik
Saarbrücken, Germany
pauli.miettinen@mpi-inf.mpg.de

ABSTRACT

Boolean matrix factorization (BMF) has become one of the standard methods in data mining with applications to fields such as lifted inference, bioinformatics, and role mining, to name a few. But the standard formalization of BMF assumes all errors are equal, at most giving the user a chance to weigh different types of errors differently. In many cases, however—and here role mining is a good example—making errors at one element of the matrix can be inacceptable, while the value in another element might be rather inconsequential. It is therefore preferable that the user can express her constraints to the mining algorithm. Unfortunately, deciding on the constraints for every element of the matrix easily becomes infeasible. To solve that problem, we propose to query the constraints from the user only when they are needed. In this paper we demonstrate our system for interactive constrained BMF. We will present the problem and the algorithm, and in addition to the demonstration, we will also present a short experimental evaluation showing that our approach can find good factorizations in the presence of constraints.

CSS Concepts

•Human-centered computing → Interactive systems and tools; **•Information systems → Data mining**

Keywords

Boolean matrix factorization; interactive data mining; role mining

1. INTRODUCTION

In *role mining*, we are given a relation between a set of users and a set of rights telling us which user has which right. Such relation is naturally expressed as a binary matrix, and an example of such a matrix is presented in Figure 1. In Figure 1, we have three users, Alice, Bob, and Charles (A, B, and C, respectively) and three rights, print, delete, and execute (respectively p, d, and e), and Alice, for instance, has the right to print and execute, but not delete. The goal of role mining is to find a small collection of *roles*, that is, sets of rights, and corresponding collection of sets of users so that each user of set i has all the rights in role i.

$$\begin{array}{c} \\ p \\ e \\ d \end{array} \begin{array}{c} \begin{array}{ccc} A & B & C \end{array} \\ \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} \end{array}$$

Figure 1: Example role mining dataset with three users, (A)lice, (B)ob, and (C)harles, and three rights, (p)rint, (e)xecute, and (d)elete

There are two common variants of the general role mining problem: either every user must get exactly the rights they held via the roles and the goal is to achieve this with the minimum number of roles, or we are given the maximum allowable number of roles, and the goal is minimize the errors we make (giving users new rights or taking existing ones away). If the data admits a description using a small sets of roles, the former variant is clearly more desirable (and indeed, many real-world data do admit it [2]). But other data could potentially take hundreds of roles to describe exactly (e.g. if the users are smartphone applications and the rights the permissions they ask [3]). In such cases we might be willing to give some users some new rights, and take some existing rights away, in an attempt to find a concise set of roles. Consider the example in Figure 1: to exactly express every user's right, we would need three roles, that is, one role for each user. But if we give Bob the 'execute' right, we can do with just two roles, namely $r_1 = \{p, e\}$ and $r_2 = \{e, d\}$, and Alice would have only role r_1, Charles only role r_2, and Bob both roles r_1 and r_2.

But a short moment of thought will immediately tell us that not every right is of equal importance to every user. Indeed, there might be a very good reason why Bob is not granted the right to execute. Instead, if we want to describe the data with just two roles, we have to take away at least two rights from the users: we can, for instance, remove Bob's right to delete and Alice's right to execute, or we can remove Bob's right to print and Charles' right to execute. Both of these two cases will cause the same amount of error, but again, they might not be equal in other ways. Printing, for example, can be vital for Bob's job and he shouldn't loose

*Corresponding author

KDD 2016 Workshop on Interactive Data Exploration and Analytics (IDEA'16) August 14th, 2016, San Francisco, CA, USA.

that right.

The user can enter these constraints before she starts finding the roles and tell the algorithm which user–right pairs cannot be changed by giving the user the right or taking it away. But often there are too many constraints to consider: if the data has n users and m rights, there are nm user-right combinations, and each of them is a potential constraint. For example, in Section 5, we report results of our experiments using an Android application permission data that has 117 036 applications and 173 permissions. Considering all of the over twenty million application–permission pairs a priori is practically impossible.

Not only impossible, considering each of these pairs a priori is also often useless: often the role mining algorithm can honor the vast majority of the right assignments. In order to allow the user to concentrate on those right assignments whose changing would help the algorithm to find a smaller set of roles, we propose an *interactive* approach, where the algorithm will inform the user whenever it is about to add a new or remove an existing right. The user can then decide whether the change can be done or whether this user–right pair should be left untouched. In the latter case, the algorithm will never try to change that pair again.

To formalize our problem, we cast it as an equivalent Boolean matrix factorization (BMF) problem, and indeed, our algorithm is designed for interactive constrained BMF. As BMF has many other applications areas besides role mining, so has our algorithm. Although we will use role mining as the motivating application throughout this paper, in Section 5, we will show that our algorithm can be used with many other datasets as well.

The main purpose of this paper is to demonstrate our algorithm, `iConFaRe` (Interactive Constrained Factor Reducer), and we will explain the proposed demonstration in Section 6. Let us first, however, formally define BMF in Section 2 and cover the related work in Section 3. After presenting our algorithm in Section 4, we present our experimental evaluation in Section 5 before concluding in Section 7.

2. NOTATION AND DEFINITIONS

As we explained, in order to have a general framework, we will describe our method in the terms of Boolean matrix factorization. We denote a matrix by upper-case boldface letters ($\boldsymbol{A}$), and vectors by lower-case boldface letters ($\boldsymbol{a}$). For a matrix $\boldsymbol{A}$ we denote its ith row by $\boldsymbol{A}_i$ and its jth column by $\boldsymbol{A}^j$.

We use the shorthand $[n]$ to denote the set of integers up to n, $[n] = \{1, 2, \ldots, n\}$.

Let $\boldsymbol{A} \in \{0,1\}^{n\times m}$, $\boldsymbol{B} \in \{0,1\}^{n\times k}$ and $\boldsymbol{C} \in \{0,1\}^{k\times m}$. We denote by $\boldsymbol{B} \circ \boldsymbol{C}$ the n-by-m *Boolean product* of matrices $\boldsymbol{B}$ and $\boldsymbol{C}$. The Boolean matrix product is defined like the normal product, but over the Boolean semiring, that is, $(\boldsymbol{B} \circ \boldsymbol{C})_{ij} = \bigvee_{\ell=1}^{k} \boldsymbol{B}_{i\ell}\boldsymbol{C}_{\ell j}$.

Let $\langle \boldsymbol{B}, \boldsymbol{C}\rangle$ be an (approximate) Boolean decomposition of $\boldsymbol{A}$, $\boldsymbol{A} \approx \boldsymbol{B} \circ \boldsymbol{C}$. We call $\boldsymbol{B}$ and $\boldsymbol{C}$ *factors* of this decomposition, and for any $1 \le l \le k$, we refer to the rank-1 matrix formed by the vector pair $\langle \boldsymbol{B}^l, \boldsymbol{C}_l\rangle$ as a *block*. If $\boldsymbol{X}$ and $\boldsymbol{Y}$ are n-by-m binary matrices, we use $\boldsymbol{X} \oplus \boldsymbol{Y}$ to denote their *element-wise exclusive or*. Finally, we denote by $|\boldsymbol{A}|$ the number of non-zeros in Boolean matrix $\boldsymbol{A}$, that is, $|\boldsymbol{A}| = \sum_{i,j} a_{ij}$.

The standard *Boolean matrix factorization* (BMF) problem is now:

Problem 1. Given $\boldsymbol{A} \in \{0,1\}^{n\times m}$ and $k \in \mathbb{N}$, find $\boldsymbol{B} \in \{0,1\}^{n\times k}$ and $\boldsymbol{C} \in \{0,1\}^{k\times m}$ that minimize

$$|\boldsymbol{A} - \boldsymbol{B} \circ \boldsymbol{C}| \ . \tag{1}$$

As we explained, the standard BMF considers all errors equal, but for example in the role mining application, this is not what is wanted. To address that problem, we can formulate the *constrained BMF* (cBMF) problem, where we are additionally given a set of index pairs denoting the locations of $\boldsymbol{A}$ where the factorization is not allowed to make mistakes:

Problem 2. Given $\boldsymbol{A} \in \{0,1\}^{n\times m}$, $k \in \mathbb{N}$, and a set of constraints $C = \{(i,j) : i \in [n], j \in [m]\}$, find $\boldsymbol{B} \in \{0,1\}^{n\times k}$ and $\boldsymbol{C} \in \{0,1\}^{k\times m}$ that minimize (1) while admitting the constraints, that is,

$$a_{ij} = (\boldsymbol{B} \circ \boldsymbol{C})_{ij} \quad \text{for all } (i,j) \in C. \tag{2}$$

Notice that our definition of cBMF has a significant problem: it is possible that there exists no valid solution. A simple example is if we let $\boldsymbol{A}$ to be the n-by-n identity matrix, set $C = [n] \times [n]$ (i.e. require exact decomposition), and set $k < n$. We can avoid this problem by requiring that the rank k is always high-enough, for example, by requiring that $k \ge \max\{|\{i \in [n] : (i,j) \in C\}|, |\{j \in [m] : (i,j) \in C\}|\}$; with this inequality, we know we can always represent the rows (or columns) with constraints exactly.

Another, and arguably more severe, problem of the above formulation is that it requires the user to pre-specify all constraints. Our approach is to make the algorithm query for the constraints when it needs to, and let the rank k be implicitly set: the algorithm tries to reduce the rank as much as possible without violating any constraints. To formalize the process of obtaining the constraints, consider a function $\mathcal{Q}_{\boldsymbol{A}} \colon [n]\times[m] \to \{0,1\}$. For every element (i,j) of $\boldsymbol{A}$, function $\mathcal{Q}_{\boldsymbol{A}}(i,j)$ returns 0 if the element is not constrained, and 1 if the element is constrained. We assume that our algorithm does not know the definition of $\mathcal{Q}_{\boldsymbol{A}}$, but it does have a way to evaluate it for any element (by asking the user). With these, we can define the problem we study in this paper, the *interactive constrained BMF* (icBMF):

Problem 3. Given $\boldsymbol{A} \in \{0,1\}^{n\times m}$ and a way to evaluate the function $\mathcal{Q}_{\boldsymbol{A}}$, find $\boldsymbol{B} \in \{0,1\}^{n\times k}$ and $\boldsymbol{C} \in \{0,1\}^{k\times m}$ such that k is minimized and the factorization $\boldsymbol{B} \circ \boldsymbol{C}$ does not violate any constraints, that is

$$\sum_{i=1}^{n}\sum_{j=1}^{m}(a_{ij} - (\boldsymbol{B} \circ \boldsymbol{C})_{ij})\mathcal{Q}_{\boldsymbol{A}}(i,j) = 0 \ . \tag{3}$$

Problem 3 does not consider the error at all. Obviously, it should be considered, but in the definition of Problem 3, it is implicit in the constraint query $\mathcal{Q}_{\boldsymbol{A}}$: if the user feels that there is going to be too much error, she can limit it by imposing more constraints. There are definitely other possible approaches, and we point the reader to Section 7 for more discussion on this topic.

3. RELATED WORK

Boolean matrix factorizations have received considerable research interest in data mining. The problem was introduced to the field in [11] together with the `Asso` algorithm. Subsequent papers have proposed new algorithms [1, 8], new

Algorithm 1 `iConFaRe`

Input: matrix $\boldsymbol{A} \in \{0,1\}^{n \times m}$, a way to evaluate function $\mathcal{Q}_{\boldsymbol{A}}$
Output: Factors $\boldsymbol{B} \in \{0,1\}^{n \times k}$ and $\boldsymbol{C} \in \{0,1\}^{k \times m}$

```
1: function iConFaRe(A, Q_A)
2:     ⟨B, C⟩ ← exact Boolean factorization of A
3:     repeat
4:         d ← the block ⟨B^d, C_d⟩ that causes the least error if deleted
5:         (m_1, m_2) ← the pair of blocks ⟨B^{m_1}, C_{m_1}⟩, ⟨B^{m_2}, C_{m_2}⟩ that cause the least error if merged
6:         op ← the delete or merge operation that causes the least error
7:         query Q_A if op violates any constraints
8:         if op does not violate any constraints then
9:             perform op to obtain new B and C
10:            mark all elements with errors unconstrained
11:        else
12:            mark op as inadmissible operation
13:            update the constraints
14:        end if
15:    until there are no admissible operations
16:    return B and C
17: end function
```

optimization goals [12], and new algorithms aiming to optimize these goals [5].

The special cases of exact Boolean matrix factorization (a.k.a. Boolean rank) and dominated Boolean matrix factorization were studied even earlier [4], although under the different name of *tiling*.

The role mining problem and its connections to BMF, were popularized in [15]. This approach was later extended in [6, 16], leading to constraint-aware role mining in [7]. Notice, however, that in [7], the constraints are something the algorithm is supposed to mine and express via *negative permissions*, while in our work, the user has to explicitly state the constraints. On the other hand, [2] provided an algorithm for computing the optimal Boolean matrix factorization (in exponential time) and applied it to many real-world user–right datasets. All of these approaches are based on combinatorial approaches; recently, [3] proposed a probabilistic approach.

4. OUR ALGORITHM

In this section we will present our approach for the icBMF problem. We divide our treatise into two parts, the back-end that is responsible for doing the computation, and the front-end that provides the user interface and in essence implements the evaluation of $\mathcal{Q}_{\boldsymbol{A}}$, although our front-end also allows for the user to steer the computation in other ways, as well.

The source code for our algorithm is freely available at `http://people.mpi-inf.mpg.de/~pmiettin/bmf/interactive/`.

4.1 The Back-End

The general process of our algorithm, `iConFaRe`, is straightforward: it starts with an exact factorization and then reduces the factors, either by removing them, or by merging them, while making sure that it never violates any constraints. When it cannot anymore do any changes, it terminates. The pseudo-code of `iConFaRe` is presented in Algorithm 1.

Finding the exact decomposition.

We start `iConFaRe` by finding an exact Boolean matrix factorization of $\boldsymbol{A}$ in line 2. As this is an NP-hard problem, we use a heuristic. Instead of developing a new algorithm, we use the minimum tiling algorithm of Geerts et al. [4]. That algorithm works essentially by first finding all closed itemsets of the data, and then solving the minimum set cover problem on an instance where each 1 of the data is an element, and each closed itemset is a set. For efficiency we use a slight modification of that idea, and instead of considering all closed itemsets, we allow the user to set a minimum-frequency threshold. To ensure that we can find an exact decomposition, we add all columns as closed itemsets, even if their frequency is below the user-set minimum-frequency threshold.

Instead of using the tiling algorithm, we could of course use any other algorithm returning an exact BMF; for smaller matrices, for instance, the method proposed by Ene et al. [2] can provide the optimal exact decomposition. We could also use non-exact decompositions, but then we would have to ensure that we do not violate any constraints as our algorithm is not guaranteed to fix errors that were committed during the exact factorization process.

Reducing the rank.

The exact decomposition usually has a rank that is too high for the application, and the main part of `iConFaRe` is to reduce that rank while ensuring it does not violate any constraints. It considers two ways of reducing the rank: delete a block, or merge two blocks (see below for how the blocks are selected for these operations). Both of these operations result in a new factorization that has one block less. To choose which of these operations it should perform, `iConFaRe` compares the errors the operations cause and selects the one that causes the least error. While we do not directly aim at minimizing the error, an operation that causes only little error is less likely to violate any constraint than an operation that causes a large error. Therefore, selecting the operation that causes the least error (lines 4–6) is a sensible heuristic.

The error is defined to be the number of elements where the operation would cause an error and that are not known to be constrained. If the operation would cause an error on any constrained element, it cannot be performed, and consequently, `iConFaRe` does not consider it. On the other hand, `iConFaRe` does not penalize operations for causing errors on known-unconstrained elements. The logic behind this is that if an element is known to be unconstrained, we can set it to whatever value we want. Further, `iConFaRe` will only consider elements to be unconstrained if it has already committed errors on those elements, and ignoring those elements avoids double-counting the errors.

Before `iConFaRe` commits the operation, it evaluates $\mathcal{Q}_{\boldsymbol{A}}$ (i.e. queries the user) to guarantee that it is not violating any constraints. This querying needs to be done for all new errors committed by the operation (the user has already allowed the old errors, so those elements cannot be constrained). If the operation does not violate any constraints, `iConFaRe` commits it in line 9. It also marks all elements where the new error is committed as unconstrained.

If the operation violates some constraints, `iConFaRe` marks it as inadmissible (to avoid considering it again) and updates the information regarding the constrained elements (in practise, though, the latter information might not be available; see Section 4.2).

The main loop, and the algorithm, end when `iConFaRe` cannot anymore find any operations that would not violate

some constraints. At that point it simply returns the current factorization.

The delete and merge operations.

The operation of deleting a block is straightforward, and so is finding the block to delete. For every block in the factorization, `iConFaRe` keeps a list of elements that are 1 only in this block. If that element is 1 also in the data, we know that removing this block would also commit an error of not covering the 1 (e.g. removing a right from a user). Deleting a block, obviously, cannot ever add 1s in the factorization. An inverted index, matching every 1 in the data to the blocks in the factorization, can be used to efficiently update the information whether a block is the only one covering a 1.

The merge operation is more complicated, and in fact we consider two different merge operations: *and-merge* and *or-merge*. Consider two blocks

$$\boldsymbol{B}^1 = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \end{pmatrix} \qquad \boldsymbol{C}^1 = \begin{pmatrix} 1 & 1 & 0 & 0 \end{pmatrix}$$

$$\boldsymbol{B}^2 = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \end{pmatrix} \qquad \boldsymbol{C}^2 = \begin{pmatrix} 1 & 0 & 1 & 0 \end{pmatrix}$$

The *and-merge* of the two blocks would result to factors $\boldsymbol{B}^{\text{and}} = (1000)^T$ and $\boldsymbol{C}^{\text{and}} = (1000)^T$, while the *or-merge* would result to factors $\boldsymbol{B}^{\text{or}} = (1110)^T$ and $\boldsymbol{C}^{\text{or}} = (1110)^T$. Notice that the *and-merge* can only remove 1s from the factorization, while the *or-merge* can only add them.

Finding good blocks to merge is harder than with deletions as it is hard to know how much error each operation would cause without first computing all pair-wise merges (or both types). That, naturally, is infeasible. Rather, we try to find two rows of $\boldsymbol{C}$ that have a high Jaccard similarity, that is, the value $J(\boldsymbol{x}, \boldsymbol{y}) = |\boldsymbol{x} \wedge \boldsymbol{y}| \,/\, |\boldsymbol{x} \vee \boldsymbol{y}|$ is high. For such vectors, both types of merges should yield small errors as the vectors are already rather similar and hence we will only consider the pair of vectors with the highest similarity for merging.

Finding pairs of vectors with high Jaccard similarity will still require us to consider all $\binom{k}{2}$ pairs of rows of $\boldsymbol{C}$ for every run of the loop. As `iConFaRe` is supposed to be an interactive algorithm, we might not be able to wait so long. To speed up the processing, we use the minhash signatures [14, Ch. 3]: we use the min-wise hashing (or *minhashing*) to compute a short signature for each row of $\boldsymbol{C}$, and the similarity between these signatures gives us a good approximation of the Jaccard similarity of the rows.

After `iConFaRe` has selected the pair of blocks for merging, it simply checks which of *and-merge* or *or-merge* yields the smallest error and recommends it to the user.

4.2 The Front-End

The back-end of `iConFaRe` is essentially non-interactive, and all of the interaction happens during the evaluation of the function $\mathcal{Q}_{\boldsymbol{A}}$: this evaluation is done by the front-end that queries the user whether the recommended operation is admissible. The front-end will then inform the back-end whether it can commit the operation or not. Computing the exact decomposition is not done in an interactive way, and as it can be time-consuming, we have implemented it as a separate pre-processing step.

The front-end allows for other types of interaction than just verifying the recommended operations. In particular, it lets the user remove and merge blocks herself, even if `iConFaRe` did not recommend those operations. Our goal here is to allow the user to use her domain knowledge and semantic understanding of the data to perform operations that might seem sub-optimal for the back-end, but that the user knows are admissible in the domain. For example, the user might know that certain users have a right to use machinery that is decommissioned. Removing that right, then, will not harm the user's capability to carry on their duties, but it can help `iConFaRe` to find more concise set of roles.

The main user interface.

The main user interface of `iConFaRe`'s front-end consists of a list view showing the current blocks (see Figure 2). This view allows the user to see all of the factors, sort them using different criteria, and manually perform merge and delete operations. It also allows the user to interact with the back-end, asking it to produce the next recommended operation (i.e. run one iteration of the main loop until the next evaluation of the constraints). The next tabs of the main view allow the user to see more information on the rows and columns in the data, and specify entire rows and columns as constraints in a fashion similar to constrained clustering.

A simple but powerful application of the main list view is to quickly delete a number of blocks. In particular, the exact initial decomposition can yield an excessive number of very small blocks (covering only few, or just one, rows and columns). In many applications, they can be deleted with only minimal consideration. In the main view, the user can easily sort the blocks based on their size and quickly delete all small blocks from the list.

In addition to the list view, `iConFaRe` has also a persistent global view of the data (Figure 3). This view shows the effects of operations on the full data. The main information this visualization conveys is the effects of the operations on the current factorization. For example, if we want to delete a block, we can see which 1s in the current factorization would turn into 0s (e.g. which rights would be removed from which users), which 1s would be covered by only one block (removal of which would remove the 1s from the factorization), and which other 1s this block covers (together with at least two other blocks). Further information is shown for merge operations. This allows the user to do long-term planning beyond that of `iConFaRe`: if an operation is going to make many 1s dependable on one other block only, the user might cancel the operation to allow for deleting other factors in the future.

Especially with big datasets, the global view might not be detailed enough and the effects scattered around the data can be hard to interpret. For that purpose, `iConFaRe` also includes a local view (Figure 4) that shows only the elements that the operation is going to affect.

Figure 4 shows a case of *or-merge* where few elements would turn into 1 in the factorization (denoted using blue color). As the overall area of the blue color is rather small, the user might as well decide to merge these two blocks unless she knows that *Eptesicus nilssonii* (a type of a bat) should never appear in the areas corresponding to the rows,

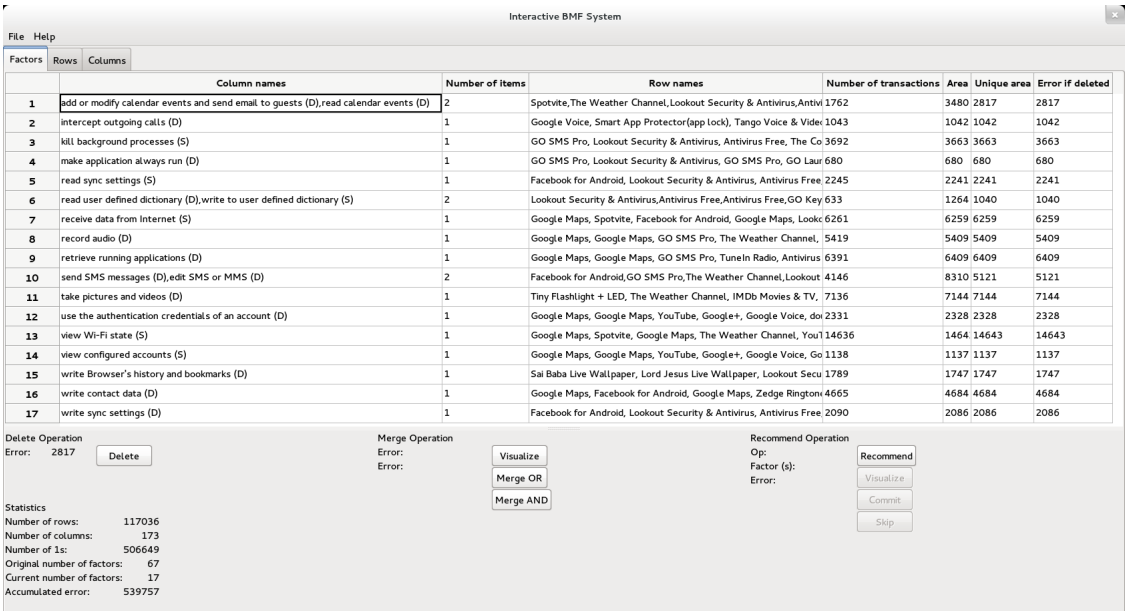

Figure 2: The main user interface of `iConFaRe`. The list shows the current factors with associated statistics, and the space at the bottom-left shows detailed statistics for the selected factor. At the middle, the buttons allow the user to commit a delete or merge operation, with relevant statistics shown next to the button, or to ask the `iConFaRe` to propose the next operation. This view shows factors from the AndroidApps data.

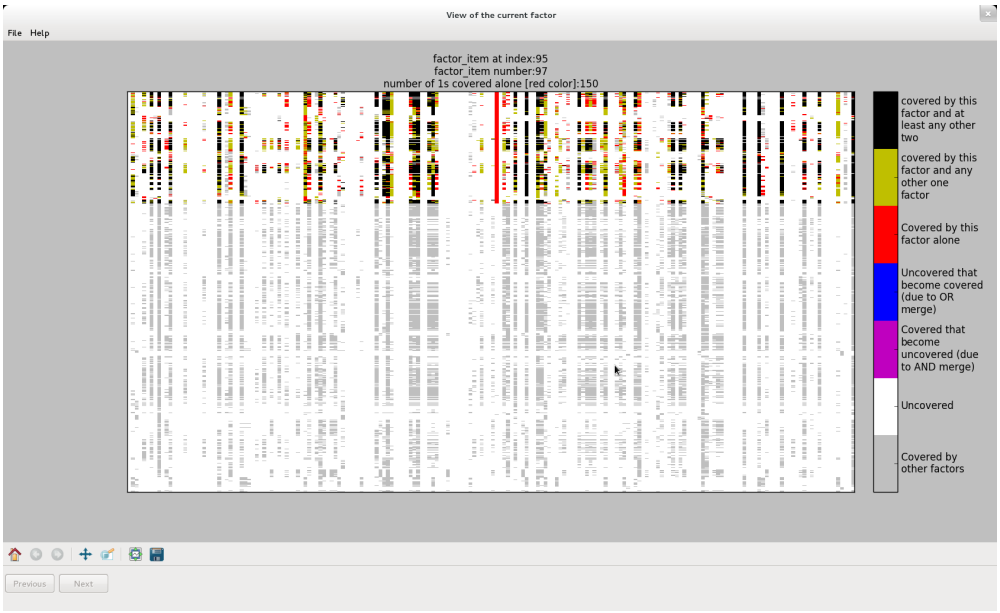

Figure 3: Visualization of the full data. One factor is selected, and the visualization shows how it effects the current factorization. For example, deleting this factor would mean that all the red dots in the data would turn to 0 in the factorization. The dataset shown is the Mammals data.

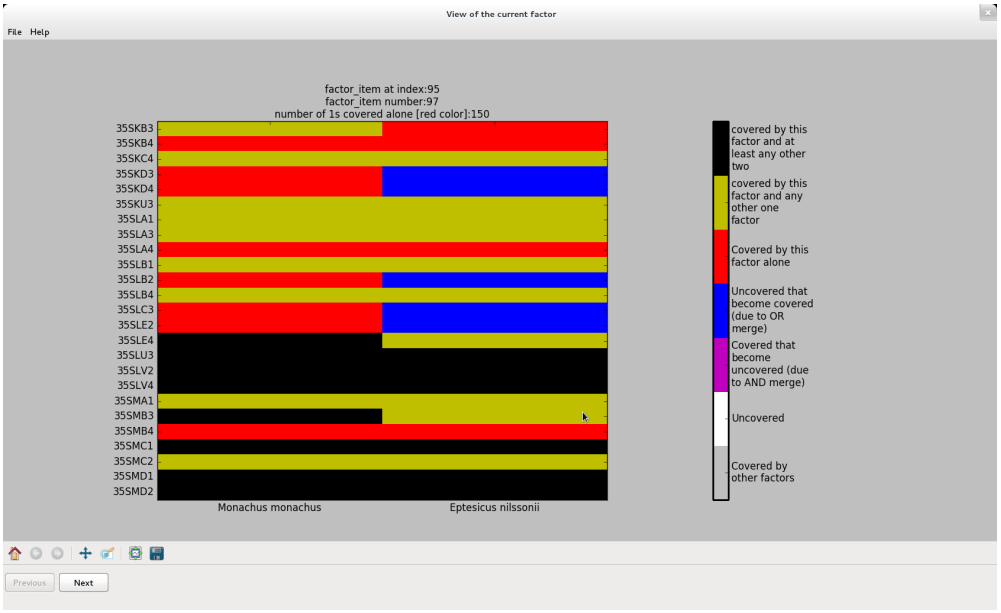

Figure 4: Visualization of the local view. Two factors are to be merged with *or-merge*, and the visualization shows how it effects the current factorization. The colors are the same as in Figure 3. The dataset shown is the Mammals data.

for example, in square 35SKD3 (latitude 39.26, longitude 24.00, close to the East coast of Greece); indeed, *E. nilssonii* is not supposed to live so south, and the user might well reject this block for being unintuitive.

Communicating with the back-end.

When the user clicks the "Recommend" button at the main view, `iConFaRe` computes the next recommended operation that is then shown to the user. If the user commits the operation, `iConFaRe` knows that all errors committed are allowed, and consequently marks all those elements to be unconstrained. If, on the other hand, the user skips the operation, `iConFaRe` does not know what was the reason: not all errors might have been in the constrained elements, or the user might have decided to skip the recommended operation for other reasons. Hence, by default, `iConFaRe` will not mark any elements constrained even if the user does not allow an operation. However, `iConFaRe` does not recommend the same operation again as the user skipped it before, but recommends the user the next operation resulting in the smallest error.

To mark elements as constrained, the user must take specific action. In particular, she has to select the row on which the element is, and left-click the element on the local view to constrain it. This extra step is needed as otherwise `iConFaRe` cannot infer the constraints. Also, in many situations the user might elect to not give the constraints explicitly the first time they are violated. It might be simplest to just skip the recommended operation, and only mark it as a constraint if `iConFaRe` repeatedly recommends violating it.

5. EXPERIMENTAL EVALUATIONS

While the main purpose of this manuscript is to demonstrate the interactive `iConFaRe` system, we have also run some off-line experiments to validate our approach.

5.1 Real-World Datasets

We have tested `iConFaRe` with three real-world datasets. We summarize the characteristics of the datasets in Table 1.

The first is the DBLP dataset. It contains the names of 6 980 authors and which of the 19 conferences they have published to. The dataset was collected from the DBLP database[1], and is pre-processed as in [9]. It is the first dataset on which `iConFaRe` was tested to see its performance as a system for constrained Boolean matrix factorization given user's constraints, because it is sparse and has less features (only 19 columns, corresponding to the 19 conferences).

The second real-world dataset that `iConFaRe` was tested on is the Mammals dataset. It consists of presence–absence data[2] of European mammals with geographical areas of 50-by-50 kilometers [13]. This dataset is denser than the DBLP dataset and has far more features (194 columns).

The third real-world dataset that `iConFaRe` was tested on is the AndroidApps dataset[3] [3]. For each application, the dataset provides the permissions requested by the application, the price, the number of downloads, the average user rating, and a short prosaic description. The original data was pre-processed by removing all applications with ratings and number of downloads less than the average rating and number of downloads. We also removed all applications which request no permission at all during the pre-processing step. Additionally, all columns which are not permissions (the price, the number of downloads, the average user rating, and a short prosaic description) were removed from the data. After the pre-processing step, the dataset contained 117 036 android applications and the presence (1) or absence (0) of 173 permission. This dataset is a good use-case for the role

[1] `http://www.informatik.uni-trier.de/~ley/db/`
[2] Available for research purposes from the Societas Europaea Mammalogica at `http://www.european-mammals.org`
[3] `http://www.mariofrank.net/andrApps/`

Dataset	Rows	Columns	Density (%)
DBLP	6 980	19	13
Mammals	2 618	194	16
AndroidApps	117 036	173	2

Table 1: Real world datasets overview.

Dataset	Rank
DBLP	11
Mammals	129
AndroidApps	67

Table 2: Factorization ranks returned by `iConFaRe` with real-world data.

mining application of `iConFaRe`.

5.2 Algorithms Used

To test `iConFaRe` in a controlled manner, we used it in a non-interactive way. Namely, we sampled random constraints for the data and replaced the evaluation of $\mathcal{Q}_{\boldsymbol{A}}$ with a function that checked whether the proposed option would violate the constraints or not.

To the best of our knowledge, `iConFaRe` is the first algorithm for icBMF (or cBMF). To compare it against other algorithms, we took `Asso` [11], a popular algorithm for standard BMF, and edited it to accept pre-defined constraints. This edit was done essentially by adjusting the evaluation function of `Asso` to make any factorization that would violate the constraints infinitely bad. Notice however, that as `Asso` builds the factorization from bottom up (i.e. it starts with an empty factorization), it cannot guarantee that the final factorization does cover all constrained 1s in the data.

5.3 Results

To test these two algorithms, we computed the cBMF factorizations with both algorithms using the same sets of random constraints. As `Asso` requires the rank as an input, we could not compare the methods based on the rank they returned. Instead, we first ran `iConFaRe` with the given datasets to obtain the error it gave and the rank it returned (`iConFaRe` was reducing the rank until it could not find any admissible operations). The ranks proposed by `iConFaRe` are listed in Table 2. We want to emphasize that these ranks do not denote any kind of "latent rank" of the data (see, e.g. [5,12]) as they depend heavily on the constraints we have randomly set.

After we got the ranks, we ran the modified `Asso` with the same ranks and constraints, and recorded the error. Errors for both `iConFaRe` and `Asso` are reported in Table 3. In DBLP, the modified `Asso` algorithm obtains smaller reconstruction error, but in the other datasets, Mammals and AndroidApps, `iConFaRe` is actually better.

The error especially on DBLP, and arguably also on Mammals, is high with both systems. This is probably due to the constraints that we imposed as the standard version of `Asso` is known to perform relatively well on both of these datasets. Hence, in this experiment the actual error is much less important than the error relative to modified `Asso`. In this respect `iConFaRe` performs very well especially as, unlike `Asso`, it actually guarantees to admit all constraints.

Dataset	`iConFaRe`	`Asso`
DBLP	78.3148	**68.3515**
Mammals	**44.7505**	48.5783
AndroidApps	**1.3058**	3.7815

Table 3: Errors in percentages of 1s in the data for `iConFaRe` and a constrained version of `Asso` on real-world datasets.

In our final experiment we tested the effects of the different constraint sets. For this experiment, we generated five different random sets of constraints and ran both `iConFaRe` and the modified `Asso` on all of them. The average error of `iConFaRe` was 1.31% (with standard deviation of ± 0.01), while for `Asso` the average error was 6.62 (± 3.28), showing that not only is `iConFaRe` significantly better than `Asso`, but also more consistent.

6. THE DEMONSTRATION

`iConFaRe` is inherently interactive, and hence best demonstrated in a way that allows the audience to have hands-on time with it. On the other hand, the initial pre-processing step can be time-consuming, and consequently unsuitable for demonstrations. Hence we have to limit the demonstration to the pre-processed datasets, for which we are planning to use the three datasets used in the experiments.

At the beginning of the demonstration, the audience is explained the goal of constrained BMF and the general ideas behind `iConFaRe`. They can then choose one of the pre-processed datasets based on their interests. For the purpose of the demonstration, we plan to use the AndroidApps dataset, as most audience members are expected to have at least a passing familiarity with smartphone application permissions. Hence, barring special request from the audience, we will use it. The demonstrator will then walk them through the basic functionality of `iConFaRe` by means of example merges and deletions. This step also explains the visualizations and their interpretations. The audience could then come up with constraints as they see fit, and test if `iConFaRe` indeed recommends operations which do not violate any of those constraints.

After the user has become used to the system—and provided that they are interested—we can load a new dataset (or re-load the AndroidApps data) and set the user a task: come up with the least-error decomposition of given rank (to be defined later) and admitting some constraints (also to be defined). The goal of this experiment is two-fold: for one, it should give the user better understanding of `iConFaRe`, but it should also give us important data on how well `iConFaRe` performs in these situations. Furthermore, it is interesting to see whether the humans with their semantic understanding and pattern recognition skills can perform better than the automated `iConFaRe` setup.

7. CONCLUSIONS

In this paper we have presented our system called `iConFaRe` for interactive constrained Boolean matrix factorization. In some sense `iConFaRe` presents a first-order system: after starting with an exact decomposition, it always considers only one step ahead. Hence, it never tries to add any new blocks but only to merge or delete the existing ones—adding a

new block will never be an optimal move alone. On the other hand, a higher-order system would consider multiple steps ahead, and it could consider adding new blocks if they would allow it to remove many existing blocks in the subsequent steps. Such higher-order systems are, however, significantly more complicated, and it is also unclear how to present their operations to the user.

As of now, `iConFaRe` attempts only to minimize the rank. This was partially done to avoid the problematic bi-optimization criterium that tries to balance the rank and the error. This criterium is problematic as it requires us to decide the relative weights between reducing the rank and increasing the error. One way to do that, though, would be to use the Minimum Description Length (MDL) principle. Using the MDL principle for BMF was pioneered in [12], and algorithms optimizing MDL directly have been proposed in recent years [5, 8].

Systems similar to `iConFaRe` could use the MDL principle to choose which operation to perform and when to stop. We argue, however, that in the `iConFaRe` system this would be unlikely to provide much, or any, benefits. For one, being able to infer the rank is a less important problem in `iConFaRe` as the user is able to stop the algorithm when she feels it has obtained small-enough decomposition. The selection of the next operation, on the other hand, would probably not see much changes: in a first-order system like `iConFaRe`, the MDL-optimal way to reduce the rank by 1 is often to do that in a way that minimizes the increase in the error. Here a higher-order system could behave differently.

Another aspect omitted from `iConFaRe`'s formal problem statement is the amount of user-involvement required. In practical terms, an interactive system like `iConFaRe` should aim to minimize the cases the user needs to consider. This is not explicitly stated in the definition of Problem 3, but we have designed `iConFaRe` to follow these guidelines as much as possible. In particular, the goal to minimize the error the operation causes implies that the user needs to consider the least number of elements for constraints.

We believe that `iConFaRe` is capable of providing practical benefits to real-world applications of BMF beyond our poster child application of role mining. Being able to tell the algorithm to avoid non-intuitive factors before they are being created, but without having to pre-specify what "non-intuitive" means, can greatly improve the usability of the results to the end-user. On the other hand, such great powers come with great responsibly, and there is a real risk that when `iConFaRe` (and similar tools) are applied to general data mining, the user inadvertently guides the system to find only results she knew a priori [10]. Designing checks to prevent such outcomes is an interesting direction of future work.

8. REFERENCES

[1] R. Bělohlávek and V. Vychodil. Discovery of optimal factors in binary data via a novel method of matrix decomposition. *J. Comput. Syst. Sci.*, 76(1):3–20, 2010.

[2] A. Ene, W. Horne, N. Milosavljevic, P. Rao, R. Schreiber, and R. E. Tarjan. Fast exact and heuristic methods for role minimization problems. In *SACMAT '08*, pages 1–10. ACM Press, 2008.

[3] M. Frank, B. Dong, A. Porter Felt, and D. Song. Mining Permission Request Patterns from Android and Facebook Applications. In *2012 IEEE 12th International Conference on Data Mining (ICDM)*, pages 870–875. IEEE, 2012.

[4] F. Geerts, B. Goethals, and T. Mielikäinen. Tiling databases. In *DS '04*, pages 278–289, 2004.

[5] S. Karaev, P. Miettinen, and J. Vreeken. Getting to Know the Unknown Unknowns: Destructive-Noise Resistant Boolean Matrix Factorization. In *SDM '15*, 2015.

[6] H. Lu, J. Vaidya, and V. Atluri. Optimal Boolean Matrix Decomposition: Application to Role Engineering. In *ICDE '08*, pages 297–306, 2008.

[7] H. Lu, J. Vaidya, V. Atluri, and Y. Hong. Constraint-Aware Role Mining Via Extended Boolean Matrix Decomposition. *IEEE Trans. Depend. Secure*, 9(5):655–669, 2012.

[8] C. Lucchese, S. Orlando, and R. Perego. A Unifying Framework for Mining Approximate Top-k Binary Patterns. *IEEE Trans. Knowl. Data Eng.*, 26(12):2900–2913, Dec. 2013.

[9] P. Miettinen. *Matrix Decomposition Methods for Data Mining: Computational Complexity and Algorithms*. PhD thesis, Department of Computer Science, University of Helsinki, 2009.

[10] P. Miettinen. Interactive Data Mining Considered Harmful (If Done Wrong). In *IDEA '14*, pages 85–87, July 2014.

[11] P. Miettinen, T. Mielikäinen, A. Gionis, G. Das, and H. Mannila. The Discrete Basis Problem. *IEEE Trans. Knowl. Data Eng.*, 20(10):1348–1362, Oct. 2008.

[12] P. Miettinen and J. Vreeken. MDL4BMF: Minimum Description Length for Boolean Matrix Factorization. *ACM Trans. Knowl. Discov. Data*, 8(4), Oct. 2014.

[13] A. J. Mitchell-Jones, G. Amori, W. Bogdanowicz, B. Krystufek, P. Reijnders, F. Spitzenberger, M. Stubbe, J. Thissen, V. Vohralik, and J. Zima. *The atlas of European mammals*. Poyser, 1999.

[14] A. Rajaraman, J. Leskovec, and J. D. Ullman. *Mining of Massive Datasets*. Cambridge University Press, 1.3 edition, July 2013.

[15] J. Vaidya, V. Atluri, and Q. Guo. The role mining problem: finding a minimal descriptive set of roles. In *SACMAT '07*, pages 175–184, 2007.

[16] J. Vaidya, V. Atluri, Q. Guo, and H. Lu. Edge-RMP: Minimizing administrative assignments for role-based access control. *J. Comp. Secur.*, 17(2):211–235, 2009.