Boolean matrix factorization meets consecutive ones property

Nikolaj Tatti[*] Pauli Miettinen[†]

Abstract

Boolean matrix factorization is a natural and a popular technique for summarizing binary matrices. In this paper, we study a problem of Boolean matrix factorization where we additionally require that the factor matrices have consecutive ones property (OBMF). A major application of this optimization problem comes from graph visualization: standard techniques for visualizing graphs are circular or linear layout, where nodes are ordered in circle or on a line. A common problem with visualizing graphs is clutter due to too many edges. The standard approach to deal with this is to bundle edges together and represent them as ribbon. We also show that we can use OBMF for edge bundling combined with circular or linear layout techniques.

We demonstrate that not only this problem is NP-hard but we cannot have a polynomial-time algorithm that yields a multiplicative approximation guarantee (unless P = NP). On the positive side, we develop a greedy algorithm where at each step we look for the best 1-rank factorization. Since even obtaining 1-rank factorization is NP-hard, we propose an iterative algorithm where we fix one side and and find the other, reverse the roles, and repeat. We show that this step can be done in linear time using pq-trees. We also extend the problem to cyclic ones property and symmetric factorizations. Our experiments show that our algorithms find high-quality factorizations and scale well.

1 Introduction

Matrix factorization is an immensely popular way of summarizing data as well as discovering signal from the data. While being useful, the interpretation and visualization of discovered factor matrices may be difficult. A popular variant for factorizing binary matrices is a k-Boolean matrix factorization, which, essentially, summarizes the binary data as a union of k tiles, that is, submatrices full of 1s. However, visualizing such factorization is difficult as the discovered rows and columns can be any sets, and there is no insightful way of visualizing them all at once.

In this paper we consider k-Boolean matrix factorization such that the resulting matrix has a certain property: we can order the columns and the rows such that the matrix consists of union of k *contiguous* tiles. We do not know the order before-hand, and we discover the order as we also discover the factorization.

[*]University of Helsinki, Helsinki, Finland, nikolaj.tatti@helsinki.fi

[†]University of Eastern Finland, Kuopio, Finland, pauli.miettinen@uef.fi. Part of this work was done while the author was with MPI-INF, Saarbrücken, Germany.

Our motivation for discovering such factorization is primarily due to easy exploration of the factorization: we can draw the factorization as k tiles. While in certain cases, such a constraint may be too restrictive, there are many settings, where this constraint comes naturally. As a specific example, consider visualizing graphs. A classic technique for visualizing a graph is using linear or circular layout, where the nodes are drawn on a line or circle, and they are connected with arcs. The most common problem with visualizing graphs is clutter due to too many edges. To combat the clutter, edges are often grouped, and drawn in ribbons (see Figure 3 for an example). The problem is to discover such ribbons and the node order, while minimizing the error. We show that we can use matrix factorization on the adjacency matrix of a graph to find the order and the groups.

We show that the factorization we seek can be expressed with consecutive ones property (C1P). Namely, we will look for factor matrices $\boldsymbol{X}$ and $\boldsymbol{Y}$ whose columns can be shuffled such that each row has a form of $[0, \ldots, 0, 1, \ldots, 1, 0, \ldots, 0]$. We show that the problem is **NP**-hard, even if $k = 1$, and it is inapproximable for $k > 1$. On the positive side, we propose a greedy algorithm that searches the factors in iterative manner. The search is done by first fixing a vector in $\boldsymbol{X}$ and finding the optimal counterpart in $\boldsymbol{Y}$, then fixing the vector in $\boldsymbol{Y}$ and finding the optimal vector in $\boldsymbol{X}$, and so on, until convergence. We show that we can find the optimal counterpart in linear time using pq-trees.

We also consider 3 extensions of this factorization: the first variant, cyclic decomposition, consists of allowing factors to "wrap around the border." the second variant is specifically designed for symmetric matrices, while the last variant combines the two. Performing cyclic and symmetric decomposition proves to be useful for cyclic layout of graphs.

The rest of the paper is organized as follows: We present preliminary notation and define the matrix factorization and the cyclic version in Section 2. We present the search algorithm in Section 3. The symmetric extensions are given in Section 4. Section 6 is dedicated to related work, and Section 5 is dedicated to experimental evaluation. Finally, we conclude the paper with remarks given in Section 7. The proofs are given in the full version of this paper [17].

Copyright © 2019 by SIAM
Unauthorized reproduction of this article is prohibited.

2 Preliminary notation and problem definitions

We begin by presenting preliminary notation, and then present the two main problem definitions. Extended problems are discussed in Section 4.

2.1 Notation Given an n-by-k binary matrix $\boldsymbol{A}$ and a k-by-m binary matrix $\boldsymbol{B}$, the *Boolean matrix product* $\boldsymbol{A} \circ \boldsymbol{B}$ is defined element-wise as

$$(\boldsymbol{A} \circ \boldsymbol{B})_{ij} = \bigvee_{\ell=1}^{k} a_{i\ell} b_{\ell j} \; . \tag{2.1}$$

The *Boolean matrix sum* of $\boldsymbol{A} \in \{0,1\}^{n \times m}$ and $\boldsymbol{B} \in \{0,1\}^{n \times m}$ is defined elementwise as $(\boldsymbol{A} \vee \boldsymbol{B})_{ij} = a_{ij} \vee b_{ij}$.

To measure the distance between two binary matrices, we use the *squared Frobenius norm* of their (normal) difference, $\|\boldsymbol{A} - \boldsymbol{B}\|_F^2$. Notice that as $\boldsymbol{A}$ and $\boldsymbol{B}$ are both binary, this is the same as calculating the number of disagreements between $\boldsymbol{A}$ and $\boldsymbol{B}$: $\|\boldsymbol{A} - \boldsymbol{B}\|_F^2 = |\{(i,j) : a_{ij} \neq b_{ij}\}|$.

We say that a binary matrix $\boldsymbol{X}$ has a *consecutive ones property* (C1P) if its columns can be permuted such that each row has a form of $[0, \ldots, 0, 1, \ldots, 1, 0, \ldots, 0]$, that is, 1s form a contiguous interval. For the sake of presentation, we will also refer these matrices as *unimodal.*

We say that a binary matrix $\boldsymbol{X}$ is *cyclic* if its columns can be permuted such that each row has a form of $[0, \ldots, 0, 1, \ldots, 1, 0, \ldots, 0]$ or $[1, \ldots, 1, 0, \ldots, 0, 1, \ldots, 1]$.

2.2 Problem definitions Next we will give our two main optimization problems.

Problem 1. (Ordered BMF, obmf) *Given a binary matrix $\boldsymbol{D}$ and an integer $k \in \mathbb{N}$, find two unimodal binary matrices $\boldsymbol{X}$ and $\boldsymbol{Y}$ that minimize the number of disagreements*

$$\left\| \boldsymbol{D} - (\boldsymbol{X}^T \circ \boldsymbol{Y}) \right\|_F^2 \; . \tag{2.2}$$

Problem 2. (Cyclic Ordered BMF, cobmf) *Given a binary matrix $\boldsymbol{D}$ and an integer $k \in \mathbb{N}$, find two cyclic binary matrices $\boldsymbol{X}$ and $\boldsymbol{Y}$ that minimize the number of disagreements*

$$\left\| \boldsymbol{D} - (\boldsymbol{X}^T \circ \boldsymbol{Y}) \right\|_F^2 \; . \tag{2.3}$$

The matrix $\boldsymbol{Z} = \boldsymbol{X}^T \circ \boldsymbol{Y}$ given in Eq. 2.2 has another natural alternative characterization: the columns and the rows of $\boldsymbol{Z}$ can be permuted such that the resulting matrix is a union of k contiguous tiles of 1s. Similarly, the matrix $\boldsymbol{Z} = \boldsymbol{X}^T \circ \boldsymbol{Y}$ given in Eq 2.3 can be permuted such that the resulting matrix is a union of k contiguous tiles, but we also allow the tiles to wrap around the border.

Unsurprisingly, the problems are computationally infeasible. First, we demonstrate that obmf is difficult even if $k = 1$.

Theorem 2.1. *The obmf problem is* NP*-hard, even if $k = 1$.*

Our next result shows that not only obmf is difficult, but it is also impossible to approximate. To show this, it is enough to demonstrate that testing for zero-error solution is expensive.

Theorem 2.2. *Deciding whether obmf has a zero-error solution is* NP*-complete.*

The proofs of these and other statements are given in the full version of this paper [17].

3 Iterative greedy algorithm

3.1 Greedy algorithm As we saw in the previous section, not only the problem is NP-hard, we cannot construct any polynomial-time algorithm with a multiplicative guarantee. Hence, we need to resort to heuristics. The most natural heuristic is a greedy heuristic, where given a $(k-1)$-sized factorization we look for a k-sized factorization by adding one row and one column to $\boldsymbol{X}$ and $\boldsymbol{Y}$. Note that these rows need to be selected carefully such that $\boldsymbol{X}$ and $\boldsymbol{Y}$ remain unimodal, and we also need to maintain the permutation(s).

Unfortunately, Theorem 2.1 states that we cannot even find the best solution for $k = 1$ in polynomial-time. Fortunately, we can solve quickly a subproblem, where we have fixed one side.

Problem 3. (Ordered BMF step, obmfstep) *Given a binary matrix $\boldsymbol{D}$ of size n-by-m and two unimodal matrices, $\boldsymbol{X}'$ of size k-by-n and $\boldsymbol{Y}'$ of size $(k-1)$-by-m, find the decomposition $\boldsymbol{X}^T \circ \boldsymbol{Y}$ solving* obmf *such that $\boldsymbol{X} = \boldsymbol{X}'$ and $\boldsymbol{Y}$ is obtained by adding one new row to $\boldsymbol{Y}'$.*

We can use obmfstep as follows. Assume that we have already found $(k-1)$-by-m matrices $\boldsymbol{X}$ and $\boldsymbol{Y}$. We first extend $\boldsymbol{X}$ with a new row using a given seed, and find the optimal new row for $\boldsymbol{Y}$ (strategy for such selection is given later using obmfstep. We fix the discovered row, and use obmfstep to find the corresponding row for $\boldsymbol{X}$. Since we solve each step optimally, the error will never increase. We stop when the error stops decreasing. Note that we will need to provide a seed for the initial row in $\boldsymbol{X}$. Here, we test several possible seeds S, and select the best. We

Copyright © 2019 by SIAM
Unauthorized reproduction of this article is prohibited.

Algorithm 1: Greedy iterative algorithm for estimating OBMF. The algorithm takes as input the dataset, the desired dimension k, and the seed set S used for selecting the first candidate for a column.

```
1  $\boldsymbol{X} \leftarrow$ matrix of size 0-by-$n$;
2  $\boldsymbol{Y} \leftarrow$ matrix of size 0-by-$m$;
3  foreach $i = 1 \ldots, k$ do
4      foreach $s \in S$ do
5          $c \leftarrow s$;
6          while error decreases do
7              $r \leftarrow$ best row for fixed columns $[\boldsymbol{Y}; c]$;
8              $c \leftarrow$ best column for fixed rows $[\boldsymbol{X}; r]$;
9          $\boldsymbol{X} \leftarrow [\boldsymbol{X}; r]$;
10         $\boldsymbol{Y} \leftarrow [\boldsymbol{Y}; c]$;
```

experiment with several options in experiments, but the default is that S is equal to all singleton columns. The pseudo-code for the algorithm is given in Algorithm 1.

The remainder of this section is about solving OBMFSTEP in linear time. Almost the same approach will also work for the cyclic version, COBMFSTEP; we will point the minute difference.

3.2 Expressing permutations with pq-trees The complicated aspect of OBMFSTEP is that we need to make sure that the new matrix is unimodal. Luckily, we can use pq-trees, a classic structure that allows us to express every permutation for which a set of binary vertices remain unimodal. In this section we will give a brief review of pq-trees and the two main properties that are relevant to us.

Assume that we are given a universe U; in our case this will be either rows or columns of the input matrix. A pq-tree is a tree with each leaf corresponding to $u \in U$. There are two types of non-leaf nodes, these types will dictate what permutations we can perform on the children. We can permute children of p-node in any order whereas the order of the children of q-node is fixed but we can flip the direction. The leaves of the permuted tree will then indicate an order. We will denote such orders by *order*(T), where T is the pq-tree.

Two seminal results are important to us. The first result states that there is a pq-tree T such that *order*(T) are exactly the orders under which a set of binary vertices remain unimodal.

THEOREM 3.1. (BOOTH AND LUEKER [3]) *Given a universe U and k sets $S_i \subseteq U$, there is a pq-tree T such that order(T) are exactly the permutations of U under which each S_i is contiguous.*

The second result states that we can efficiently update the pq-tree.

THEOREM 3.2. (BOOTH AND LUEKER [3]) *Assume that we have a pq-tree T over a universe U and a set $S \subseteq T$. Let P be the set of all permutations of U where S is contiguous. If order$(T) \cap P \neq \emptyset$, then there is an $\mathcal{O}(|U|)$-time algorithm that constructs a tree T' such that order$(T') =$ order$(T) \cap P$. If order$(T) \cap P = \emptyset$, then the same algorithm detects a failure.*

The detailed description of the algorithm for updating the pq-tree can be found in [3].

3.3 Finding the optimal row In this section we describe the algorithm that solves OBMFSTEP. Assume that we have a pq-tree T representing the permutations of columns in $\boldsymbol{D}$ allowed by the previously discovered rows in $\boldsymbol{Y}'$. When dealing with pq-trees it is notationally easier to deal with sets rather than with vectors. Naturally every binary vector $\boldsymbol{y}$ can be represented as a set $S = \{i : y_i = 1\}$.

Let us define U to be the column indices of $\boldsymbol{D}$; these are exactly the leaves of T. We say that a set $S \subseteq U$ is *compatible* with a pq-tree T, if there is an order in *order*(T) where S is contiguous. Obviously, compatible sets S correspond exactly to suitable new rows in $\boldsymbol{Y}$.

We can express OBMFSTEP as an instance of the following problem.

PROBLEM 4. (OPTSET) *Given a universe U, weights $w(u)$ for each $u \in U$, and a pq-tree T over the universe U, find a set S that is compatible with T and maximizes the total weight $\sum_{u \in S} w(u)$.*

Recall that $u \in U$ corresponds to a column index of $\boldsymbol{D}$. Define $w(u)$ to be the gain in the error-function if we were to use u in our new row for $\boldsymbol{Y}$. More formally, let $\boldsymbol{x}$ be the fixed counterpart in $\boldsymbol{X}$ for the new row in $\boldsymbol{Y}$. Let p be the number of ones in $\boldsymbol{D}$ at rows $\boldsymbol{x}$ and column u that are not yet covered by the previous factors. Let n be the number of zeros in $\boldsymbol{D}$ at rows $\boldsymbol{x}$ and column u that are not yet covered by the previous factors. We define $w(u) = p - n$. Solving OPTSET with these weights solves OBMFSTEP.

In order to solve COBMFSTEP, we solve OPTSET using $w(u) = p-n$, as above, yielding a set, say S_1. In addition, we also solve OPTSET using $w(u) = n - p$, yielding a set, say S_2. Then, we use either S_1 or $U \setminus S_2$, whichever yields a better gain.

In order to solve OPTSET, we need an additional definition: Let S be a compatible set of a pq-tree T. If there is a permutation in *order*(T) with the first or the last element in S, we call S a *border-compatible* set.

Copyright © 2019 by SIAM
Unauthorized reproduction of this article is prohibited.

Let T be a pq-tree. To solve OPTSET we will compute 3 counters for a node v in T, namely, $\mathit{inner}(v)$, $\mathit{border}(v)$, and $\mathit{total}(v)$. The counter $\mathit{total}(v)$ corresponds to the total weight of leaves under v, while the counter $\mathit{inner}(v)$ corresponds to the best S that is compatible with the subtree starting at v. Finally, $\mathit{border}(v)$ corresponds to the best S that is border-compatible with the subtree starting at v.

We should stress that, strictly by definition, $\mathit{inner}(v)$ can represent an empty set, whereas $\mathit{total}(v)$ and $\mathit{border}(v)$ should be never empty, even if they produce a negative value. Thus, $\mathit{inner}(v) \geq 0$ but $\mathit{border}(v)$ and $\mathit{total}(v)$ can have negative values. Moreover, it is possible that $\mathit{border}(v)$ represents every leaf of v, in which case, $\mathit{border}(v) = \mathit{total}(v)$.

Naturally, we want to compute $\mathit{inner}(r)$, where r is the root of T. To obtain this value we compute each value iteratively, children first. We also maintain the lists of the children that were responsible for producing the optimal value. These lists are clear from the proofs of the following lemmata. This allows us to extract the optimal S.

First, note that computing $\mathit{total}(v)$ is trivial since $\mathit{total}(v) = \sum_{c \in ch(v)} \mathit{total}(c)$. If v is a leaf-node, then $\mathit{border}(v) = \mathit{total}(v)$ and $\mathit{inner}(v) = \max(0, \mathit{total}(v))$.

The next two lemmata establish how to compute the counters for q-nodes.

LEMMA 3.1. *Let v be a q-node and let $c_1, \ldots, c_\ell$ be its children. Then*

$$\begin{aligned} \mathit{border}(v) &= \max(x, y), \quad \textit{where} \\ x &= \max_i \mathit{border}(c_i) + \sum_{j=1}^{i-1} \mathit{total}(c_j)\,, \\ y &= \max_i \mathit{border}(c_i) + \sum_{j=i+1}^{\ell} \mathit{total}(c_j) \quad . \end{aligned}$$

LEMMA 3.2. *Let v be a q-node and let $c_1, \ldots, c_\ell$ be its children. Then*

$$\begin{aligned} \mathit{inner}(v) &= \max(x, y), \quad \textit{where} \\ x &= \max_i \mathit{inner}(c_i)\,, \\ y &= \max_{i<j} \mathit{border}(c_i) + \mathit{border}(c_j) + \sum_{\ell=i+1}^{j-1} \mathit{total}(c_\ell) \; . \end{aligned}$$

Our next step is to compute the counters for p-nodes. For that we need to define the following helper function: given a node v we define $g(v) = \mathit{border}(v) - \max(\mathit{total}(v)\,, 0)$. We will use $g(v)$ in the next two lemmata describing on how to compute the counters for p-node.

LEMMA 3.3. *Let v be a p-node and let $c_1, \ldots, c_\ell$ be its children. Define $b = \max g(c_i)$. Then*

$$\mathit{border}(v) = b + \sum_i \max(\mathit{total}(c_i)\,, 0) \quad .$$

Note that since we require the set responsible for $\mathit{border}(v)$ be non-empty, it is possible that $\mathit{border}(v) < 0$. This can happen only if $b < 0$ and every child w of v has $\mathit{total}(w) < 0$.

LEMMA 3.4. *Let v be a p-node and let $c_1, \ldots, c_\ell$ be its children. Define b_1 and b_2 be the top-2 values of $g(c_i)$. Then*

$$\begin{aligned} &\mathit{inner}(v) = \max(x, y), \quad \textit{where} \\ &\quad x = \max_i \mathit{inner}(c_i)\,, \\ &\quad y = \max(b_1, 0) + \max(b_2, 0) + \sum_i \max(\mathit{total}(c_i)\,, 0) \; . \end{aligned}$$

Note that using these lemmas every counter can be trivially solved in linear time, except for $\mathit{inner}(v)$, where v is q-node. To compute $\mathit{inner}(v)$ in linear time, it is enough if we can solve

$$\mathit{border}(c_j) + \max_{i<j} \mathit{border}(c_i) + \sum_{\ell=i+1}^{j-1} \mathit{total}(c_\ell)$$

in *constant* time for a *fixed* j. Luckily, we can rewrite this function as

$$\mathit{border}(c_j) + \left(\sum_{\ell=1}^{j-1} \mathit{total}(c_\ell) \right) + \max_{i<j} t(i, j),$$

where

$$t(i, j) = \max_{i<j} \mathit{border}(c_i) - \sum_{\ell=1}^{i} \mathit{total}(c_\ell) \quad .$$

Let $i(j)$ to be the optimal i for a fixed j. Since

$$\max_{i<j} t(i, j) = \max \left(t(j-1, j) \max_{i<j-1} t(i, j) \right),$$

we have either $i(j) = i(j-1)$ or $i(j) = j - 1$. If we were to test each j consecutively, then this allows us to compute $i(j)$ in constant time: we simply compare the solution $i = j - 1$ to the best previous solution $i(j-1)$.

In summary, each counter of v can be computed in $\mathcal{O}(|ch(v)|)$. Thus we need $\mathcal{O}(\ell)$, where ℓ is the number of nodes in T. Since $\ell \in \mathcal{O}(|U|)$, we can compute the counters in $\mathcal{O}(|U|)$ time, where $|U|$ is the number of columns in $\boldsymbol{D}$.

When computing the counters we also store which children were responsible for this value. Once we have

Copyright © 2019 by SIAM
Unauthorized reproduction of this article is prohibited.

computed *inner*(r), where r is the root of the tree, we can backtrack to obtain the optimal S. This can be also done in linear time.

Computing the weights w in OPTSET can be done in $\mathcal{O}(p)$ time, where p is the number of 1s in the dataset $\boldsymbol{D}$ of size n-by-m. Consequently, OBMFSTEP can be done in $\mathcal{O}(p + n + m)$ time.

4 Symmetric decomposition

We now propose an extension for symmetric matrices.

4.1 Definition If $\boldsymbol{D}$ is symmetric (e.g. an adjacency matrix of an undirected graph), we have the following problem:

PROBLEM 5. (SYMMETRIC OBMF, OBMF$_{\text{sym}}$) *Given a binary matrix $\boldsymbol{D}$ and an integer $k \in \mathbb{N}$, find two binary matrices $\boldsymbol{X}$ and $\boldsymbol{Y}$ such that $[\boldsymbol{X}; \boldsymbol{Y}]$ is unimodal, that minimize the number of disagreements*

$$\left\| \boldsymbol{D} - \left((\boldsymbol{X}^T \circ \boldsymbol{Y}) \vee (\boldsymbol{Y}^T \circ \boldsymbol{X}) \right) \right\|_F^2 . \tag{4.4}$$

We define similarly COBMF$_{\text{sym}}$, a cyclic and symmetric variant of OBMF.

The unimodality condition in OBMF$_{\text{sym}}$ states that we should be able to permute $\boldsymbol{X}$ and $\boldsymbol{Y}$ with the *same* permutation so that the rows are in form of $[0, \ldots, 0, 1, \ldots, 1, 0, \ldots, 0]$.

Notice that we do not use the more common symmetric decomposition $\boldsymbol{D} \approx \boldsymbol{X}^T \circ \boldsymbol{X}$ as this would lead to necessarily having the blocks around the diagonal.

4.2 Algorithm The discovery algorithm for symmetric OBMF is similar. Like with the regular OBMF, we use a greedy algorithm as an iterative step for discovering new rows.

The first difference is that we maintain only one pq-tree, corresponding to the rows in both $\boldsymbol{X}$ and $\boldsymbol{Y}$.

The second difference is that – as $\boldsymbol{X}^T \circ \boldsymbol{Y}$ and $\boldsymbol{Y}^T \circ \boldsymbol{X}$ can have overlapping 1s – maximizing OPTSET does not necessarily produce the optimal row. Instead, we can show that solving OPTSET, with the weights as described in the previous section, minimizes $\left\| \boldsymbol{D} - \boldsymbol{X}^T \circ \boldsymbol{Y} \right\|_F^2 + \left\| \boldsymbol{D} - \boldsymbol{Y}^T \circ \boldsymbol{X} \right\|_F^2$. It follows easily that minimizing this function yields a 2-approximation for finding optimal counterpart row.

5 Experimental evaluation

In this section we study how well the algorithms from Sections 3 and 4.2 work with synthetic and real-world data. We denote the algorithms with the same names as the problems they are solving, and differentiate the algorithms from the problems via the font. That is, `obmf` is the algorithm for OBMF, and so on. The algorithms are implemented in C++, and we make the source code and synthetic experiments freely available.[1]

5.1 Resilience to Noise We start by evaluating the algorithms' resilience to noise. To that end, we synthesized random matrices of size 95×95 with block structure (6 blocks of size 20×20 along the diagonal, with 5 overlapping rows and columns) and corrupted those matrices with flipping a varying amounts of entries. The amount of flipped entries varied from 0 % to 50 % (of total elements) and we compared the quality of the results to both the noise-free matrix and noisy matrix. The results are shown in Figure 1.

With lower leves of noise (35 % for `obmf` and `cobmf` and 25 % for the symmetric variants), the reconstruction of the original data is more accurate. With higher levels of noise, the noise has destroyed so much of the structure that the algorithms start fitting to the noise only, with a clear reduction of the quality versus the original data.

It is also worth noticing that `obmf` obtains exact decompositions when the data has no noise; the other methods introduce a slight error even in these cases emphasizing their more complex setting.

5.2 Scalability In this section we test how well `obmf` scales to larger data sets and how well it benefits from multiple cores. These experiments were executed on a server with 40 cores of Intel Xeon E7-4870 processors running at 2.4 GHz. The algorithm was compiled using GCC 8.1.0 and the parallel code uses the OpenMP library.

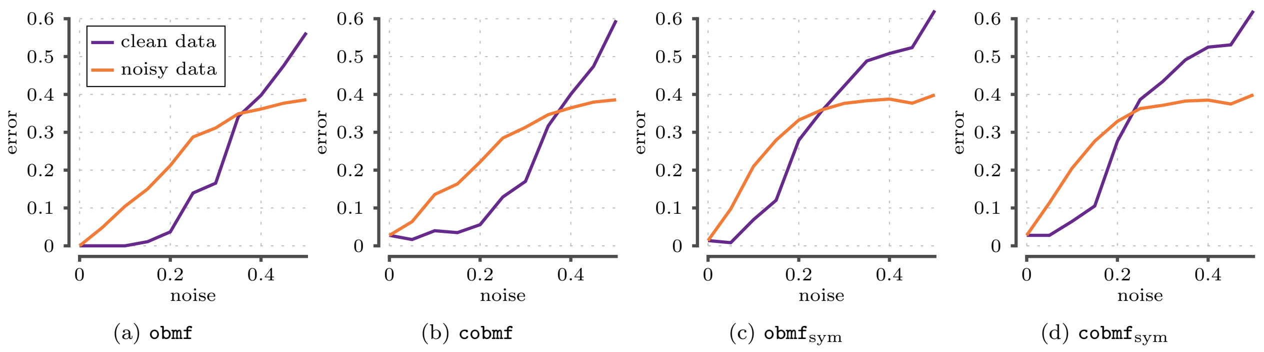
To test the scalability, we generated n-by-n square matrices with $n = 2^i$ for $i = 9, \ldots, 13$. All matrices have a density of approximately 24 %. The results are presented in Figure 2a.

The algorithm shows very good scalability over the full range, although it does get slower when the data size increases from 2^{12} to 2^{13}. It should be noted, though, that as the density is constant, the number of non-zeros in the matrices increases as the square of the matrix size. Hence, `obmf` exhibits linear growth with respect to the number of non-zero elements.

Algorithm 1 is almost embarrassingly parallel over the different seeds vectors. Hence, we parallellized the test of different seeds, and tested how the algorithm behaves with increased number of cores. The results are in Figure 2b, where we can see that the speed-up is essentially linear up to 4 cores, slightly slower until 16 cores, and only marginal gains are available when increasing the number of cores to 32, indicating that at

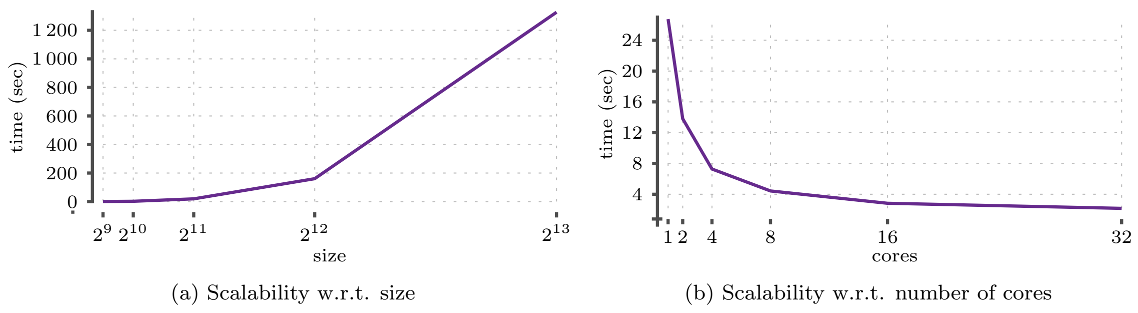
[1] `https://cs.uef.fi/~pauli/bmf/ordered_bmf/`

Copyright © 2019 by SIAM
Unauthorized reproduction of this article is prohibited.



(a) `obmf` (b) `cobmf` (c) $\texttt{obmf}_{\text{sym}}$ (d) $\texttt{cobmf}_{\text{sym}}$

Figure 1: Error as a function of noise. Here the error is the proportion of disagreements between the reconstructed matrix and either the noise-free or the noisy matrix. The decomposition was done using the noisy matrix.



(a) Scalability w.r.t. size (b) Scalability w.r.t. number of cores

Figure 2: Scalability with respect to the size and number of cores.

the algorithm has become memory bus constrained.

Overall, the experiments show that the algorithm scales very well, and is able to benefit from modern multi-core computers. We study further speed-up options later in Section 5.3.2.

5.3 Experiments with Real-World Data We now turn to real-world data sets. We used six different real-world data sets, selected to offer a wide variety of different types of data. The data sets we used are as follows. *Les Misérables* is a standard benchmark data[2] of the characters of Victor Hugo's novel *Les Misérables*. *Paleo* is a palaeontological data[3] in the form of a locations-by-genera matrix, giving information where different fossiles have been found. *Newsgroups* is a subset of the famous 20Newsgroups data[4] consisting four newsgroups and 100 terms. *Terms* the terms-by-terms co-occurrence matrix based on *Newsgroups*. *Locations* is locations-by-locations matrix indicating mammal species co-location in the northern hemisphere: the data has a 1 in element (i, j) if locations i and j have at least five mammals in common. The data is based on the IUC Red List data.[5] The final data set, *Mammals*, contains a species-by-species co-inhabitation matrix.[6] The data set properties are summarized in Table 1.

Table 1: Properties of real-world data sets. Rank indicates the rank used in the decomposition.

data	rows	cols	% of 1s	sym.	rank
Les Misérables	77	77	8.57	Yes	10
Paleo	124	139	11.48	No	10
Newsgroups	100	348	6.30	No	10
Terms	100	100	48.54	Yes	10
Locations	3203	3203	8.42	Yes	50
Mammals	194	194	58.04	Yes	10

[2]`http://moreno.ss.uci.edu/data.html`

[3]NOW 030717, `http://www.helsinki.fi/science/now/`

[4]`http://qwone.com/~jason/20Newsgroups/`

[5]`http://www.iucnredlist.org/technical-documents/spatial-data`

[6]Available for research purposes from the Societas Europaea Mammalogica at `http://www.european-mammals.org`

Copyright © 2019 by SIAM
Unauthorized reproduction of this article is prohibited.

To the best of our knowledge, this is the first work to address the ordered Boolean matrix factorization problem. To understand what kind of an effect the ordering constraint has to the reconstruction error, we compare our results with those of `asso` [14]. The `asso` algorithm is a well-known method for computing the standard Boolean matrix factorization. We used an implementation available from the author[7] and set the rank for `asso` the same as for our algorithms, and used threshold values $\tau = \{0.2, 0.4, 0.6, 0.8\}$.

For symmetric data sets, we also computed the symmetric Boolean factorization. This was done by first computing the standard $\boldsymbol{X}^T \circ \boldsymbol{Y}$ factorization, and then testing whether $\boldsymbol{X}^T \circ \boldsymbol{X}$ or $\boldsymbol{Y}^T \circ \boldsymbol{Y}$ gives smaller reconstruction error and using that one. This version of `asso` is denoted $\texttt{asso}_{\text{sym}}$.

5.3.1 Reconstruction errors We first compute the reconstruction errors for the various data sets. To facilitate the comparisons, we report the *relative* reconstruction error

$$\frac{\|\boldsymbol{D} - \boldsymbol{X}^T \circ \boldsymbol{Y}\|_F^2}{\|\boldsymbol{D}\|_F^2} .$$

The results of all datasets are given in Table 2.

In case of asymmetric decompositions, `asso` is – as expected, as its factor matrices are not restricted to unimodal or cyclic – almost always slightly better than either `obmf` or `cobmf`. This difference is, however, very small in many data sets (only 8 % in *Les Misérables* and 0.50 % in *Paleo*). A remarkable exception is the *Mammals* data, where `asso` is in fact worse than either `obmf` or `cobmf`. As the data set is the densest of the ones we tested, it is possible that `asso` was unable to obtain good candidates from it with the rounding thresholds we tried.

There is almost no difference between `obmf` and `cobmf` in the terms of reconstruction error in these data sets. Usually, `obmf` is on par or slightly better than `cobmf`, except again in *Mammals*, where `cobmf` is slightly better. The asymmetric data sets, *Paleo* and *Newsgroups*, cause the highest reconstruction errors at over 70 %. It should be noted, though, that also `asso` has similarly high errors with these data sets, indicating that they might not have strong Boolean low-rank structure.

In symmetric decompositions, the relationship between the ordered BMF algorithms and `asso` is reversed, with $\texttt{asso}_{\text{sym}}$ being often the worse method (with the exception of *Terms*). This is not very surprising, given that `asso` is not designed for symmetric decompositions. The errors are slightly worse than with the asymmetric algorithms, highlighting the complexity of finding the symmetric decompositions.

[7]https://cs.uef.fi/~pauli/basso/basso-0.5.tar.gz

5.3.2 Changing the seeds In the above experiments, we used the columns as the seeds S for the algorithm (cf. Algorithm 1). This slows the algorithm down, as it has to attempt all of the potential seeds. In this section we study if we can improve the running time without hurting the reconstruction error by sampling only some of the columns for the seed set S.

In particular, we sampled 10 % of the columns uniformly at random to create the seed set. As the algorithm scales linearly with the number of seeds, this provides an order of magnitude speed-up. To test the quality, we repeated the sampling ten times and report the average relative reconstruction errors and standard deviations in Table 3.

The first thing to notice in Table 3 are the low standard deviations; less than 3 % in almost all data sets. The reconstruction errors are also only slightly higher than those in Table 2; for instance, `obmf` with *Paleo* has only 6 % higher error on average when using random sampling. In most cases the speed-up obtained by the sampling is significant compared to the loss in accuracy.

5.4 Visualizing the Graphs One of the motivations for the ordered BMF is that it allows the convenient visualization of the graphs using edge bundles (or ribbons) between nodes that are placed in a circle. In this section we explore some of these visualizations and explain what we can learn from the respective data sets using them. In the following plots, the edge bundles and the ordering are obtained form the factorization. Further visualizations can be found in the full version of this paper [17].

The *Les Misérables* data: The visualization of the *Les Misérables* data is presented in Figure 3. Most edge bundles form a circular segment indicating that all of the nodes under the segment are connected to each other (the characters appear in the same parts of the book). Some of the bundles are contained in other bundles, indicating important subset of characters. Multiple bundles intersect on a node at south-east of the circle called *Valjean* – the protagonist of the book.

The *Mammals* data: The second data set is the *Mammals* data, in Figure 4. For a clearer visualization, we only consider 134 species that do not appear too frequently in the data, as such species are neighbours of every other species in graph. The edge bundles in Figure 4 are essentially rotating around the middle. This probably corresponds to the change of fauna when moving from north to south. The change is gradual, hence two consecutive edge bundles have a significant

Copyright © 2019 by SIAM
Unauthorized reproduction of this article is prohibited.

Table 2: Relative errors with asymmetric (left) and symmetric (right) algorithms on real-world data.

	Les Mis	*Paleo*	*News*	*Terms*	*Locations*	*Mammals*
`obmf`	0.36	0.71	0.74	0.32	0.40	0.26
`cobmf`	0.36	0.72	0.74	0.32	0.40	0.26
`asso`	0.33	0.71	0.72	0.29	0.34	0.26

	Les Mis	*Terms*	*Locations*	*Mammals*
$\texttt{obmf}_{\text{sym}}$	0.40	0.35	0.48	0.27
$\texttt{cobmf}_{\text{sym}}$	0.41	0.36	0.45	0.27
$\texttt{asso}_{\text{sym}}$	0.66	0.33	1.02	0.33

Table 3: Average relative errors and standard deviation with random columns as seeds for asymmetric algorithms on real-world data. Ten random samples.

	Les Misérables	*Paleo*	*Newsgroups*	*Terms*	*Locations*	*Mammals*
`obmf`	0.51 ± 0.08	0.76 ± 0.02	0.83 ± 0.02	0.32 ± 0.00	0.53 ± 0.00	0.26 ± 0.00
`cobmf`	0.46 ± 0.05	0.76 ± 0.02	0.83 ± 0.02	0.32 ± 0.01	0.51 ± 0.00	0.26 ± 0.00

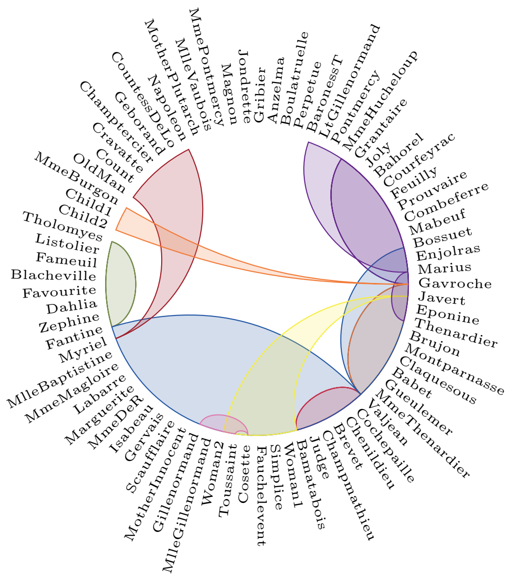


Figure 3: Visualization of the *Les Misérables* data with the ribbons and ordering from `cobmf`.

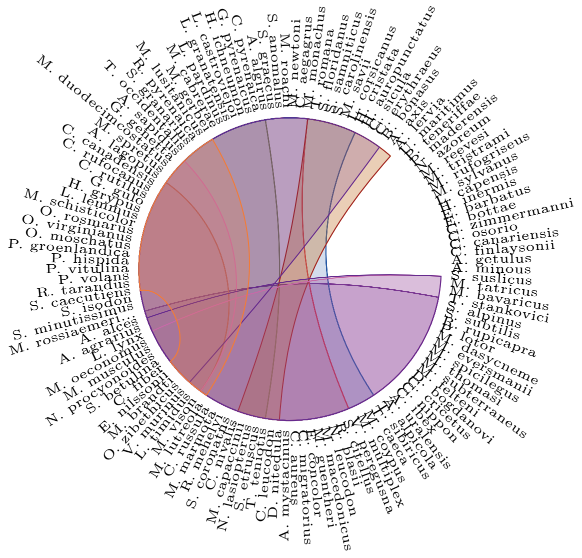


Figure 4: Visualization of the *Mammals* data with the ribbons and ordering from `obmf`.

overlap, but over longer distance, the change in the fauna becomes more obvious and the edge bundles are more disjoint. This gives a good intuition about the structure of the data.

6 Related Work

Boolean matrix factorization (BMF) has received increasing interest in the data analysis community [2, 8–16], proving to be a versatile tool for analyzing Boolean matrices. Many different algorithms have been proposed, including algorithms based on candidate creation and selection [11, 14], proximal alternations [9], and message passing [15], to name but a few. It has also found applications in diverse fields, such as bioinformatics [5], information extraction [4], and lifted inference [18]. To the best of our knowledge, however, the ordering constraint is not studied in earlier work related to Boolean matrix factorization.

Tiling databases [6] can be seen as a restricted version of BMF, where the factorization cannot express any 0s as 1. *Geometric tiling* [7] is a variation thereof, where the tiles have to be consecutive. The main difference to our work is a different optimization function, [7] uses log-likelihood, and that it assumes that the order is already given, for example, by spectral ordering, whereas we discover the order on the fly.

A binary matrix has the *consecutive ones property* (C1P) if its columns can be permuted so that all rows have all 1s consecutively. The pq-trees can be used

Copyright © 2019 by SIAM
Unauthorized reproduction of this article is prohibited.

to check for the C1P [3] and Atkins et al. [1] propose spectral ordering algorithm. The spectral ordering approach is used in [7] to permute the data for finding the geometric tiles.

7 Conclusions

Ordered Boolean matrix factorization (OBMF) and its variations (COBMF, OBMF_{sym}) are restricted versions of Boolean matrix factorization, requiring the factors to have the consecutive ones property (or be cyclic, in case of COBMF). This restriction facilitates the interpretation of the factorization, in particular in the case of the edge bundle visualizations of graphs, as we saw in Section 5.4. On the other hand, the restriction yields higher reconstruction errors, though our experiments show that the difference to state-of-the-art Boolean matrix factorization algorithm is usually very small.

In this paper we laid the theoretical foundations of the OBMF problem and its variations, and proposed algorithms based on the pq-trees. An important part of the proposed algorithm is the choice of the seed vectors. In this paper, we mostly used all columns of the data as the seed, though the experiments in Section 5.3.2 show that sampling the columns could work equally well. An interesting question for the future is whether other methods for selecting the seeds would yield better reconstruction errors.

In the problem setting of this paper, the user provides the rank of the decomposition and the goal is to minimize the reconstruction error over the rank-k OBMF decompositions. A common variant in the Boolean matrix factorization world is to make the rank a free variable and replace the target function with measure that penalizes for higher ranks (see, e.g. [9, 11, 13]). The *Minimum Description Length* principle is a common approach. The ordered nature of our factor matrices could help with finding more efficient MDL decompositions, as the factor matrices are easier to compress using run-length encoding or similar approaches.

References

[1] J. E. Atkins, E. G. Boman, and B. Hendrickson. A Spectral Algorithm for Seriation and the Consecutive Ones Problem. *SIAM J. Comput.*, 28(1): 297–310, 1998.

[2] 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.

[3] K. S. Booth and G. S. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using pq-tree algorithms. *J. Comput. Syst. Sci.*, 13(3):335–379, 1976.

[4] E. Cergani and P. Miettinen. Discovering relations using matrix factorization methods. In *CIKM '13*, pages 1549–1552, 2013.

[5] G. Corrado, T. Tebaldi, G. Bertamini, F. Costa, A. Quattrone, G. Viero, and A. Passerini. PTRcombiner: mining combinatorial regulation of gene expression from post-transcriptional interaction maps. *BMC Genomics*, 15(1), Apr. 2014.

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

[7] A. Gionis, H. Mannila, and J. K. Seppänen. Geometric and Combinatorial Tiles in 0–1 Data. In *PKDD '04*, pages 173–184, 2004.

[8] S. Hess and K. Morik. C-SALT: Mining Class-Specific ALTerations in Boolean Matrix Factorization. In *ECMLPKDD '17*, pages 547–563, 2017.

[9] S. Hess, K. Morik, and N. Piatkowski. The PRIMPING routine—Tiling through proximal alternating linearized minimization. *Data Min. Knowl. Discov.*, 31(4):1090–1131, May 2017.

[10] S. Karaev, P. Miettinen, and J. Vreeken. Getting to Know the Unknown Unknowns: Destructive-Noise Resistant Boolean Matrix Factorization. In *SDM '15*, pages 325–333, 2015.

[11] 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.

[12] S. Maurus and C. Plant. Ternary Matrix Factorization: problem definitions and algorithms. *Knowl. Inf. Syst.*, 46(1):1–31, Jan. 2016.

[13] P. Miettinen and J. Vreeken. MDL4BMF: Minimum Description Length for Boolean Matrix Factorization. *ACM Trans. Knowl. Discov. Data*, 8(4):–31, Oct. 2014.

[14] 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.

[15] S. Ravanbakhsh, B. Póczos, and R. Greiner. Boolean Matrix Factorization and Noisy Completion via Message Passing. In *ICML '16*, 2016.

[16] T. Rukat, C. C. Holmes, M. K. Titsias, and C. Yau. Bayesian Boolean Matrix Factorisation. In *ICML '17*, pages 2969–2978, July 2017.

[17] N. Tatti and P. Miettinen. Boolean matrix factorization meets consecutive ones property, 2019. arXiv:1901.05797 [cs.DS].

[18] G. van den Broeck and A. Darwiche. On the Complexity and Approximation of Binary Evidence in Lifted Inference. In *NIPS '13*, pages 2868–2876, 2013.

Copyright © 2019 by SIAM
Unauthorized reproduction of this article is prohibited.