Running Boolean Matrix Factorization in Parallel

Jan Outrata **Martin Trnecka**

Department of Computer Science
Palacký University Olomouc, Czech Republic
Email: jan.outrata@upol.cz, martin.trnecka@gmail.com

Abstract

Boolean matrix factorization (also known as Boolean matrix decomposition) is a well established method for analysis and preprocessing of data. There is a number of various algorithms for Boolean matrix factorization, but none of them uses benefits of parallelization. This is mainly due to the fact that the algorithms utilize greedy heuristics that are inherently sequential. In this work, we propose a general parallelization scheme—and an algorithm which uses it—for Boolean matrix factorization. Our approach computes several possible locally most optimal (from heuristic perspective) partial decompositions and constructs several most optimal final decompositions in more processes running simultaneously in parallel. As a result of the computation, either the single most optimal decomposition or several top-k of them can be returned. The approach could be applied to any sequential heuristic Boolean matrix factorization algorithm. Moreover, we present results of various experiments involving this new algorithm on synthetic and real datasets.

Keywords: Boolean matrix decomposition; parallel algorithm

1 Introduction

Boolean Matrix Factorization (BMF) is a problem of decomposing a Boolean matrix into two Boolean matrices such that the (Boolean) matrix product of the two matrices exactly or approximately equals the given matrix. In a variant of the problem called Approximate Factorization Problem (AFP) (Belohlavek et al. 2010), a (non-trivial) solution with the inner matrix product dimension as low as possible, for a given maximal, usually zero, difference (error) of the product from the input matrix, is requested. Another variant, called Discrete Basis Problem (DBP) (Miettinen et al. 2008), demands minimal error for a given maximal inner dimension. We will focus on the AFP in this paper. The least dimension for which an exact decomposition of a Boolean matrix exists is called the *Boolean rank* (or Schein rank) of the matrix. As the problem of finding the Boolean rank for a given Boolean matrix as well as the AFP and the DBP is NP-hard (due to the NP-hardness of the set basis problem (Stockmeyer 1975)), existing BMF algorithms seek for a sub-optimal decomposition with the dimension as close to the Boolean rank as possible, utilizing some heuristic approach.

Well-recognized efficient algorithms are GreConD (Belohlavek et al. 2010) and GreEss (Belohlavek et al. 2015), both designed for the AFP, and Asso (Miettinen et al. 2008) for the DBP. Other competitive algorithms for either AFP or DBP include e.g. PaNDa (Lucchese et al. 2010) or Hyper (Xiang et al. 2011). Being heuristic, the algorithms construct the final decomposition from partial (approximate) decompositions which are only locally optimal among all possible partial decompositions. The choice of optimal partial decomposition is usually hardcoded in the algorithm design and for performance reasons one cannot afford in the algorithm to explore several most optimal decompositions (or even all of them) and then choose among them the most optimal one discarding the others. This is true for sequential algorithms, as all the above mentioned algorithms are sequential, and this is where a parallel computation approach could be used. Moreover, with the development and growing affordability of multicore processors and other hardware allowing parallel computations, interest in parallel computing increases and parallel algorithms are preferred to better utilize the hardware.

Interestingly, to our knowledge, there is no parallel algorithm for *Boolean* matrix factorization today introduced in the literature. The adjective 'Boolean' needs to be emphasized here. There are parallel algorithms for some of the many existing factorization methods designed originally for real-valued matrices, see for instance (Berry et al. 2006) or (Kannan et al. 2016), and those algorithms obviously can be applied also to Boolean matrices (and they are, though). However, as (Tatti et al. 2006) and other authors conclude, a problem with applying to Boolean matrices the methods designed originally for real-valued matrices is the lack of interpretability. And because of interpretability, which is crucial from the knowledge discovery point of view, BMF is considerably more appropriate to use with Boolean matrices than the methods designed originally for real-valued matrices. One of the reasons for the absence of a parallel BMF algorithm, however, may be that the most commonly used greedy heuristic approach, utilized in GreConD, GreEss and also in Asso, is inherently sequential. Other reason could be that factorization

The research was supported by grant No. GA15-17899S of the Czech Science Foundation. M. Trnecka also acknowledges partial support by grant No. PrF_2016_027 of IGA of Palacký University Olomouc.

of Boolean matrices is as a standalone research area relatively young within the data mining research and not so elaborated as the factorization of real-valued matrices.

Our contribution in this paper does not lie in a parallel algorithm for BMF which would compute a decomposition in a parallel manner either. Instead, we show a general parallelization scheme consisting in a viable way to compute in parallel several locally optimal decompositions and then select the most optimal one(s) hoping to find the globally optimal. In essence, as suggested above, the approach consists in following several possible choices of locally most optimal partial decompositions in the heuristic approach and construct several most optimal final decompositions in more processes running simultaneously in parallel. As a result of computation, either the single most optimal decomposition or several top-k of them can be returned—a distinctive feature of our approach. The approach could be applied, with more or less effort, to any sequential heuristic BMF algorithm. We chose as our base algorithm to demonstrate the approach the GreConD algorithm, due to its relative simplicity and high efficiency (for the AFP).

In the rest of the paper, the following Section 2 provides basic notions of Boolean matrix factorization, the main Section 3 contains first a brief description of the base GreConD algorithm and then a presentation of our approach of computing several matrix decompositions in parallel demonstrated on GreConD, including full pseudocodes of both original GreConD and our modification of it, Section 4 then presents results from basic experiments evaluating the approach, and finally Section 5 concludes the paper.

2 Boolean Matrix Factorization

Boolean matrix factorization (BMF), called also Boolean matrix decomposition, comprises various methods for analysis and processing of Boolean data, mostly for factorization or decomposition of the data. The data is in the form of Boolean matrices, i.e. matrices with entries either 1 or 0. We interpret such matrices primarily as object-attribute incidence relations, that is, the entry I_{ij} of a Boolean matrix I corresponding to the row i and the column j indicates that the object i does (value 1) or does not have (value 0) the attribute j. The ith row and jth column vector of I is denoted by $I_{i\_}$ and $I_{\_j}$, respectively. The set of all $n \times m$ Boolean matrices is denoted $\{0,1\}^{n\times m}$.

Generally speaking, the basic problem in BMF is to find for a given Boolean matrix $I \in \{0,1\}^{n\times m}$ Boolean matrices $A \in \{0,1\}^{n\times k}$ and $B \in \{0,1\}^{k\times m}$ for which

$$I \text{ (approximately) equals } A \circ B, \qquad (1)$$

where $\circ$ is the Boolean matrix product, i.e.

$$(A \circ B)_{ij} = \max_{l=1}^{k} \min(A_{il}, B_{lj}).$$

Interpreting the matrices A and B as object-factor and factor-attribute incidence relations, respectively, such a decomposition of I into A and B may be interpreted as a discovery of k factors (k is the inner dimension of the product) exactly or approximately explaining I. In the factor model given by (1) matrices A and B explain I as follows: the object i has the attribute j ($I_{ij} = 1$) if and only if there exists factor l such that l applies to i ($A_{il} = 1$) and j is one of the particular manifestations of l ($B_{lj} = 1$). Thus, a factor in the model is naturally interpreted as an abstract property (or attribute), generally distinct from the m original attributes, which applies to some of the n objects and which is characterized by some of the m original attributes.

This easy interpretation of factors is further supported by the so-called geometric view on factors and on BMF, which is unfortunately not always recognized in the literature. We will use the view in the description of the GreConD algorithm below. Briefly, in the view each factor is identified with a rectangular matrix, or *rectangle* for short, a Boolean matrix whose entries with 1 form, upon a suitable permutation of rows and columns, a rectangular area (full of 1s). A decomposition of a Boolean matrix I using k factors then corresponds to a *coverage* of the entries of I containing 1s by k such rectangles (a Boolean matrix product from (1) is looked at as a max-superposition $\max_{l=1}^{k}(J_l)_{ij}$ of rectangles $J_l = A_{\_l} \circ B_{l\_}$).

In optimization versions of the basic BMF problem, the number k of factors is requested to be as small as possible (see the AFP below) and, as already mentioned in the beginning of the introduction section, the least k for which an exact decomposition $I = A \circ B$ exists is called the *Boolean rank* (or Schein rank) of I. The deviation from an exact decomposition, i.e. the approximate equality in (1), is assessed by means of the well-known L_1-norm $||I|| = \sum_{i,j=1}^{m,n} |I_{ij}|$ and the difference of $A \circ B$ from I is measured by a distance (error) function $E(I, A \circ B)$ defined as

$$E(I, A \circ B) = ||I - A \circ B|| = \sum_{i,j=1}^{m,n} |I_{ij} - (A \circ B)_{ij}|.$$

Using E, quality of decompositions delivered by BMF algorithms is commonly assessed (Belohlavek et al. 2010, 2015, Geerts et al. 2004, Miettinen et al. 2008) by the following function representing the *coverage quality* of the first l factors delivered by the particular algorithm and measuring how well the data is explained by the l factors:

$$c(l) = 1 - E(I, A \circ B)/||I||.$$

We will use this function in Section 4 devoted to experiments where we also state what $c(l)$ should satisfy for a good factorization algorithm.

A (optimization) variant of the basic BMF problem of our concern is the approximate factorization problem (AFP):

Definition 1 (Approximate Factorization Problem, AFP (Belohlavek et al. 2015))**.** *Given* $I \in \{0,1\}^{n\times m}$ *and prescribed error* $\varepsilon \geq 0$*, find* $A \in \{0,1\}^{n\times k}$ *and* $B \in \{0,1\}^{k\times m}$ *with* k *as small as possible such that* $||I - A \circ B|| \leq \varepsilon$.

AFP emphasizes the need to account for (and thus to explain) a prescribed (presumably reasonably large) portion of data, which is specified by ε.

For a more throughout study of the factor model and the geometric view on it described above, of the AFP and of the BMF in general, we refer the reader to (Belohlavek et al. 2010) or (Belohlavek et al. 2015).

3 BMF in Parallel Runs

In this section we present our approach of computing several Boolean matrix decompositions simultaneously in parallel. First, however, we need to recall the GreConD algorithm which we use as a base for our parallelization scheme.

3.1 GreConD Algorithm

The algorithm, proposed in (Belohlavek et al. 2010) and called Algorithm 2 there, solves the AFP by implementing greedy search for factors. I.e., each factor is sought to explain as much of the input Boolean matrix being decomposed as possible. The factors satisfying this property are, however, not selected among all candidate factors (as in the "classical" greedy approach), rather they are incrementally, or "on demand", computed with the aim to fulfill the property. And the computation is again in a greedy manner, see the description below. In the description we will use the geometric view on factors (as (Belohlavek et al. 2010) does too), identifying each factor with a rectangular matrix (rectangle) full of 1 as introduced in Section 2. Finding a decomposition of the input Boolean matrix I then means finding a coverage of 1s in I by such rectangles. GreConD finds factors as *maximal rectangles*. This is not accidental, maximal rectangles make factors better interpretable. Informally, maximal rectangles are rectangles which cannot be enlarged by adding another row or another column so that it remains a rectangle—hence factors, in this sense, apply to a maximal number of objects and are characterized by a maximal number of attributes. This rationale actually stems *Formal concept analysis* (FCA) (Ganter 1999) in which maximal rectangles correspond to so-called formal concepts (basic data units studied in FCA) and which is used as a description platform of the algorithm in (Belohlavek et al. 2010) (now, GreConD means "Greedy Concepts on Demand"). We do not use FCA in this paper, readers interested in (a fruitful) connection between BMF and FCA are referred to (Belohlavek et al. 2010) or (Belohlavek et al. 2015). We will briefly describe the GreConD algorithm now.

The algorithm, in its seek for a factor, starts with the empty set of attributes (characterized as the empty Boolean vector of size m or the empty $1 \times m$ Boolean matrix) which is repeatedly grown by a selected attribute. Together with the selected attribute other attributes may be possibly added to the set due to the construction of each factor as a maximal rectangle. Namely, the set of attributes after each addition is completed, or *closed*, to contain all attributes which are shared by all objects having the attributes. Such a set of attributes is called closed. The closed set of attributes together with the corresponding set of all objects having all the attributes determine a maximal rectangle. The selected attribute is such that the rectangle grown by the attribute covers as many still uncovered 1s in the input Boolean matrix I as possible. Within the aim of computing factors as rectangles which cover as many still uncovered 1s in matrix I as possible, the rectangle is grown repeatedly as long as the number of still uncovered 1s in I covered by the rectangle increases. The final maximal rectangle then represents a computed factor. Note the greedy and "on demand" computation of the factor. Further factors as maximal rectangles are, within the greedy factor search, sought the same way until the prescribed number of 1s in I is covered by the rectangles (the prescribed maximal error E is reached, recall that the algorithm is designed for the AFP). Finally, characteristic vectors of object sets determining found maximal rectangles/factors constitute columns of the object-factor matrix A and characteristic vectors of attribute sets of the rectangles/factors constitute rows of the factor-attribute matrix B. Matrices A and B, which determine the decomposition of the matrix I, form an output of the algorithm.

The above description results in a pseudocode of the algorithm depicted in Algorithm 1. D denotes the set of attributes determining a maximal rectangle (after closing) and is repeatedly grown by a selected attribute h between lines 4 and 16. Under the vector/matrix notation we use in the pseudocode D_j denotes the jth item of $D \in \{0,1\}^{1\times m}$ as a Boolean vector which effectively corresponds to the presence/absence of attribute j in D. The addition of h and the closure of D with h added are performed at line 14. Here, $[h] \in \{0,1\}^{1\times m}$ denotes the Boolean vector with hth item equal to 1 and all other items equal to 0 (i.e. the set with attribute h only) and the closure operator $^{\downarrow\uparrow}$ is a composition of the (so-called formal concept-forming (Ganter 1999)) operators $^{\uparrow}$ and $^{\downarrow}$. The operators are defined for a (column) Boolean vector $C \in \{0,1\}^{n\times 1}$ and a (row) Boolean vector D, respectively, as

$$C^{\uparrow} = \textstyle\bigplus [j] \in \{0,1\}^{1\times m}\,;\ I_{ij} = 1 \text{ for all } i \text{ s.t. } C_i = 1,$$
$$D^{\downarrow} = \textstyle\bigplus [i] \in \{0,1\}^{n\times 1}\,;\ I_{ij} = 1 \text{ for all } j \text{ s.t. } D_j = 1.$$

Note the meaning of the operators (in the set notation): $C^{\uparrow}$ contains all attributes shared by all objects in C and $D^{\downarrow}$ contains all objects having all attributes in D. The selection of attribute h is done between lines 7 and 13. The number of still uncovered 1s in the input Boolean matrix I covered by the rectangle determined by D, on which the selection is based, is computed by a function $cover(D, I, A, B)$ defined as

$$\|(D^{\downarrow} \times D^{\downarrow\uparrow}) \cdot (I - A \circ B)\|.$$

The $\times$ and $\cdot$ (dot) operations in the function definition denote the usual Cartesian and scalar matrix products, respectively. By line 6, D is repeatedly grown as long as the number computed by the function *cover* increases. I is then covered by the final maximal rectangles determined by D after growing which represent factors and the factors are "stored" in matrices A and

Algorithm 1: Original GreConD algorithm

```
Input: A Boolean matrix $I \in \{0,1\}^{n\times m}$ and a
       prescribed error $\varepsilon \ge 0$
Output: Boolean matrices $A \in \{0,1\}^{n\times k}$ and
        $B \in \{0,1\}^{k\times m}$
1  $A \leftarrow$ empty $n \times 0$ Boolean matrix
2  $B \leftarrow$ empty $0 \times m$ Boolean matrix
3  while $\|I - A \circ B\| > \varepsilon$ do
4      $D \leftarrow$ empty $1 \times m$ Boolean matrix
5      $V \leftarrow 0$
6      while there is $j$ such that $D_j = 0$ and
        $cover(D + [j], I, A, B) > V$ do
7          $W \leftarrow 0$
8          forall $j$ do
9              if $cover(D + [j], I, A, B) > W$ then
10                 $h \leftarrow j$
11                 $W \leftarrow cover(D + [j], I, A, B)$
12             end
13         end
14         $D \leftarrow (D + [h])^{\downarrow\uparrow}$
15         $V \leftarrow W$
16     end
17     $A \leftarrow [A\ D^{\downarrow}]$, $B \leftarrow \begin{bmatrix} B \\ D \end{bmatrix}$
18 end
19 return $A$ and $B$
```

B at line 17. Finally, finding the factors until they cover the prescribed number of 1s in I given by ε is wrapped between lines 1 and 18.

We can now proceed to the description of our approach to running the algorithm in parallel.

3.2 Parallel Runs of GreConD

As we saw above the GreConD algorithm implements a greedy search for factors in which each factor is computed to explain as much of the input Boolean matrix being decomposed as possible. While the alone factor computed this way may be optimal within the aim to explain by factors as much of the input matrix as possible, several (or all) factors together, forming a (final) decomposition, may be not. Hence, the partial decomposition formed by the factor and all factors computed previously is only locally optimal. Moreover, there can be more equally optimal factors instead of just one, forming more partial decomposition to choose from and generally leading to different final decompositions. Likely, the computation of a factor is also greedy, by growing a maximal rectangle by attributes selected so that the rectangle covers as many still uncovered 1s in the input matrix as possible. Similarly, while the alone attribute selected in such a way may be optimal within the aim to cover by the (final) maximal rectangle after growing as many still uncovered 1s in the input matrix as possible, more attributes together, determining a factor, may be not. Hence, here the partial factor (formed by the attribute and all attributes added to the corresponding maximal rectangle previously) is also only locally optimal. And finally, there can also be more equally optimal attributes to select, forming more partial factors to choose from and leading to different factors.

In our approach to parallel runs of a BMF algorithm, as aforementioned in the introduction section, we construct, simultaneously in parallel, several (locally) most optimal partial decompositions and select among them several most optimal final decompositions in hope to find the globally optimal one. For the GreConD algorithm it means that in the search for factors in each iteration several factors explaining most of the input Boolean matrix being decomposed are computed instead of just one, and in the factor computation in each iteration several attributes are selected so that the corresponding maximal rectangle covers most still uncovered 1s in the input matrix, instead of just one. Each of the factors computed, together with the factors computed previously, forms a (locally optimal) partial decomposition and each of the attributes (possibly with other attributes due to the closure), together with the attributes selected previously, forms a (locally optimal) partial factor. After each iteration, several most optimal partial decompositions or factors are selected for the next iteration. And while the computation of factors from the selected partial factors remains serial, the construction of the final decompositions from the selected partial ones is done in parallel.

In terms of a pseudocode the idea is included in the algorithm depicted in Algorithms 2 and 3. Algorithm 2 depicts an algorithm which, loosely speaking, represents several instances of the (modified) GreConD algorithm from Algorithm 1 where each instance is represented by the procedure depicted in Algorithm 3. All instances are running simultaneously in parallel—hence the name GreConDP as "GreConD in Parallel runs"—and *jointly* construct several most optimal decompositions of input Boolean matrix I. The number of instances equal to the number

Algorithm 2: GreConDP – GreConD in parallel runs

```
Input: A Boolean matrix $I \in \{0,1\}^{n\times m}$ and a
       prescribed error $\varepsilon \geq 0$
Output: Boolean matrices $A_1 \in \{0,1\}^{n\times k}$ and
        $B_1 \in \{0,1\}^{k\times m}$
Uses: Boolean matrices $A_r \in \{0,1\}^{n\times k}$ and
      $B_r \in \{0,1\}^{k\times m}$, values $U_r$, $r \in \{1,\ldots,P\}$,
      and a number $P \geq 1$ of processes
1  $A_r \leftarrow$ empty $n \times 0$ Boolean matrix
2  $B_r \leftarrow$ empty $0 \times m$ Boolean matrix
3  $U_r \leftarrow 0$ $(r \in \{1,\ldots,P\})$
4  GreConDP-i($I, \varepsilon, A_1, B_1, true$)
5  for $r = 1,\ldots,P$ do
6      with process $r$
7          GreConDP-i($I, \varepsilon, A_r, B_r, false$)
8      end
9  end
10 wait for all processes
11 return $A_1$ and $B_1$
```

of decompositions is given by (presently equal to) the number P of processes in which the instances are run, see lines 5 to 9 of Algorithm 2. The decompositions of matrix I are determined by Boolean matrices A_r and B_r, $r \in \{1,\ldots,P\}$, sorted in the descending order from the most optimal one (by increasing r). The first one only, i.e. the most optimal one, determined by A_1 and B_1, is output.

Let us now focus on Algorithm 3 depicting the core of the modified GreConD algorithm and compare it to the original version of GreConD depicted in Algorithm 1. The procedure GreConDP-i constructs the ith decomposition (ith in the time of calling from Algorithm 2, the order of decompositions may change, see below) determined by matrices A_i and B_i. Several, actually P, sets of attributes determining P maximal rectangles (after closing) representing partial factors are denoted by D_r. By lines 4 and 5, all those sets are repeatedly (and serially) grown by several selected attributes between lines 2 and 33. Just in the first iteration of growing, when all D_rs are empty Boolean matrices, only D_1 is grown, see line 30. The selection of the attributes for a particular D_r is done between lines 6 and 18. Compare it to lines 7 to 13 in Algorithm 1. Instead of just one attribute h, P attributes h_s such that the maximal rectangle determined by D_r with h_s added covers most still uncovered 1s in the input matrix I are selected and the attributes are, as a bonus here, sorted (by increasing s, using the Insertsort sorting algorithm) in the descending order from the greatest number of 1s covered by the rectangle (computed by function *cover*). In addition, we need to check that the rectangles are distinct, at line 10 (adding different attributes to the same maximal rectangle can result in the same maximal rectangle).

Among all the sets grown from a D_r by all P selected attributes h_p, P of the sets only determining maximal rectangles which cover most still uncovered 1s in I, over all D_rs, are stored as E_s, between lines 19 and 29 (extending lines 14 and 15 in Algorithm 1). The sets E_s are sorted (by increasing s) in the descending order from the greatest number of 1s covered by the rectangle and here the sorting helps to choose the P sets. After an iteration of growing of all D_rs each E_s is renamed to D_s for another iteration of growing, line 32. When the repeated growing is finished (when the number computed by the func-

Algorithm 3: Procedure GreConDP-i

```
   Input: A Boolean matrix $I \in \{0,1\}^{n\times m}$, a
          prescribed error $\varepsilon \geq 0$, Boolean matrices
          $A_i \in \{0,1\}^{n\times k}$ and $B_i \in \{0,1\}^{k\times m}$ and a
          Boolean flag $first$
   Uses: Boolean matrices $A_r \in \{0,1\}^{n\times k}$ and
         $B_r \in \{0,1\}^{k\times m}$, values $U_r$, $r \in \{1,\dots,P\}$,
         and a number $P \geq 1$ of processes
 1 while $||I - A_i \circ B_i|| > \varepsilon$ do
 2     $D_r \leftarrow$ empty $1 \times m$ Boolean matrix
 3     $V_r \leftarrow 0$ ($r \in \{1,\dots,P\}$)
 4     while there is $\langle r,j \rangle$ such that $(D_r)_j = 0$ and
         $cover(D_r + [j], I, A_i, B_i) > V_r$ do
 5         forall $r$ from the $\langle r,j \rangle$ do
 6             $W_s \leftarrow 0$ ($s \in \{1,\dots,P\}$)
 7             forall $j$ from the $\langle r,j \rangle$ do
 8                 $s \leftarrow P$
 9                 while $s > 0$ and
                     $cover(D_r + [j], I, A_i, B_i) > W_s$
                     do
10                     if $s > 1$ and
                         $(D_r + [j])^{\downarrow\uparrow} = (D_r + [h_{s-1}])^{\downarrow\uparrow}$
                         then break
11                     if $s < P$ then
12                         $h_{s+1} \leftarrow h_s$, $W_{s+1} \leftarrow W_s$
13                     end
14                     $h_s \leftarrow j$
15                     $W_s \leftarrow cover(D_r + [j], I, A_i, B_i)$
16                     $s \leftarrow s - 1$
17                 end
18             end
19             for $p = 1,\dots,P$ do
20                 $s \leftarrow P$
21                 while $s > 0$ and $W_p > V_s$ do
22                     if $s < P$ then
23                         $E_{s+1} \leftarrow E_s$, $V_{s+1} \leftarrow V_s$
24                     end
25                     $E_s \leftarrow (D_r + [h_p])^{\downarrow\uparrow}$
26                     $V_s \leftarrow W_p$
27                     $s \leftarrow s - 1$
28                 end
29             end
30             if $D_1$ = empty $1 \times m$ Boolean matrix
                 then break
31         end
32         for $r = 1,\dots,P$ do $D_r \leftarrow E_r$
33     end
34     $U_i \leftarrow 0$
35     synchronization barrier
36     for $r = 1,\dots,P$ do
37         $s \leftarrow P$
38         begin critical section
39             while $s > 0$ and $V_r > U_s$ do
40                 if $s < P$ then
41                     $A_{s+1} \leftarrow A_s$, $B_{s+1} \leftarrow B_s$
42                     $U_{s+1} \leftarrow U_s$
43                 end
44                 $A_s \leftarrow [A_i\ D_r^{\downarrow}]$, $B_s \leftarrow \begin{bmatrix} B_i \\ D_r \end{bmatrix}$
45                 $U_s \leftarrow V_r$
46                 $s \leftarrow s - 1$
47             end
48         end
49     end
50     synchronization barrier
51     if $first = true$ then break
52 end
```

tion *cover* at line 4 does not increase) we have P final maximal rectangles (determined by D_r) representing factors which have to be "stored" in matrices A_i and B_i.

This is done between lines 34 and 50, over all the P decompositions (determined by A_is and B_is) constructed in the P processes running the procedure GreConDP-i in parallel. Therefore, due to data consistency reasons when storing the factors, we need to wait until all processes compute the factors and before they start to compute further factors. Hence the synchronization barriers at lines 35 and 50. Among all the factors computed by the processes, P of the factors only which explain most of the input Boolean matrix I are stored in matrices A_i and B_i (line 44) and the matrices are sorted (by increasing s) in the descending order from the greatest number of 1s covered by their factors. Again, due to the data consistency reasons, the storing of factors and sorting need to be done in a critical section, i.e. with inter-process switching disabled between lines 38 and 48. Note also that due to the sorting of matrices A_i and B_i determining the ith decomposition, the order of decompositions may change. But each process constructs still the same decomposition (ith in the time of calling the procedure GreConDP-i in Algorithm 2), until the prescribed number of 1s in I given by ε is covered.

The last comment is on line 51 of Algorithm 3 and line 4 of Algorithm 2. Since calling the procedure GreConDP-i in Algorithm 2 with the empty decomposition as input in P processes does not make sense (because all would compute the same P factors from which the same first one explaining most of the input Boolean matrix would just be stored to P copies the same decomposition), we first call the procedure once to construct the first P non-empty decompositions (in the first iteration) only and after that we can continue the construction of the decompositions in the P processes.

4 Experimental Evaluation

Now we provide an experimental evaluation of the GreConDP algorithm described in the previous section and present a comparison with the base algorithm GreConD. We do not include a comparison with other algorithms and approaches to the general BMF. A comparison of GreConD with other BMF algorithms can be found e.g. in (Belohlavek et al. 2015).

4.1 Datasets

As in the typical experiment scenario, which occurs in various BMF papers, we use both synthetic and real datasets which are described below. Experiments on synthetic datasets enable us to analyze performance of the algorithms on data with the same and known characteristics—we can analyze results in the average case. On the other hand such data are fully artificial while real data are influenced by real factors.

4.1.1 Synthetic Data

We created 1000 of randomly generated datasets. Every dataset X_i has 500 rows (objects) and 250 columns (attributes) and was obtained as a Boolean product $X_i = A_i \circ B_i$ of Boolean matrices A_i and B_i that were both generated randomly. Final densities (the ratio of 1s in the Boolean matrix) of datasets are 0.05, 0.1, 0.15, 0.2 and 0.3 and there is the same number of datasets with each density. The inner dimension of matrices A_i and B_i in the Boolean matrix product was set to 40 for all datasets, i.e. the expected number of factors is 40 (but the Boolean rank can be lower). Data generated in this way are standard for evaluation of BMF algorithms (Belohlavek et al. 2015, Miettinen et al. 2008).

4.1.2 Real Data

We used the datasets Emea (Ene et al. 2008), DBLP (Miettinen et al. 2008), Firewall 1 (Ene et al. 2008), Mushroom (Bache at al. 2013), Paleo[1] and Zoo (Bache at al. 2013), see Table 1. All of them are well known and used in the literature on BMF. The characteristics of the datasets in the table are number of objects × number of attributes (column Size), ratio of 1s in the dataset Boolean matrix (Dens. 1) and the average number of equally locally optimal factors per factor in the GreConD algorithm (column Equal).

Dataset	Size	Dens. 1	Equal
Emea	3046×35	0.095	157.279
DBLP	19×6980	0.130	2.105
Firewall 1	365×709	0.124	31.168
Mushroom	8124×119	0.193	3.148
Paleo	501×139	0.051	5.868
Zoo	101×28	0.305	5.867

Table 1: Real datasets and their characteristics

The last characteristics is an important one. Recalling Section 3.2, factors computed by GreConD (and other heuristic BMF algorithms) are alone locally optimal within the aim to explain by factors as much of the input Boolean matrix being decomposed as possible. For a partial decomposition formed by a factor and all factors computed previously there can, however, be more than one equally optimal factors to add to the decomposition, generally leading to different final decompositions. Classical sequential heuristic BMF algorithms like GreConD then have to select one and this selection is algorithm or implementation dependent. This problem is reduced in our parallelization scheme—instead of selecting one of the equally optimal factors several (or potentially all) of them producing most optimal decompositions are considered. And since the choice of equally optimal factors influences results obtained by the evaluated algorithms, as we will see below, the last column in Table 1 includes the total number of the equally optimal factors computed during the whole computation divided by the final number of factors delivered by the GreConD algorithm. Let us also note that this characteristics is not mentioned in any previous work.

4.2 Quality of Decomposition

We provide results on the most important aspect of evaluation of the performance of BMF algorithms—the quality of decomposition delivered by an algorithm (recall Section 2). As common in the literature on BMF, we evaluate the obtained results from the viewpoints of the two main BMF optimization problems: DBP (Discrete Basis Problem) and AFP (Approximate Factorization Problem), cf. the introduction section and Section 2.

DBP emphasizes the importance of the first few (presumably most important) factors. In this perspective, the quality of factors obtained by a BMF algorithm may be assessed by observing the values of coverage c for small numbers of factors.

AFP emphasizes the need to account for (and thus to explain) a prescribed (presumably reasonably large) portion of data. In this perspective, the quality of factors obtained by a BMF algorithm may be assessed by observing the numbers of factors needed to attain a prescribed coverage c.

4.2.1 Comparsion with GreConD Algorithm

We observe the values of $c(l)$ (see Section2) for $l = 0 \ldots k$, where k is the number of factors delivered by a particular algorithm. Clearly, for $l = 0$ (no factors, A and B are "empty") we have $c(l) = 0$. In accordance with general requirements on BMF, for a good factorization algorithm $c(l)$ should be increasing in l, should have relatively large values for small l (i.e. should be steeply increasing in the beginning), and it is desirable that for $l = k$ we have $I = A \circ B$, i.e. the data is fully explained by all k computed factors (in which case $c(l) = 1$).

The values of $c(l)$ (average over 1000 iterations) for synthetic data are shown in Figures 1, 2 and 3. In case of the GreConDP algorithm we present the values for the best factorization. We study similarities of particular factorizations delivered by GreConDP later in Section 4.3.

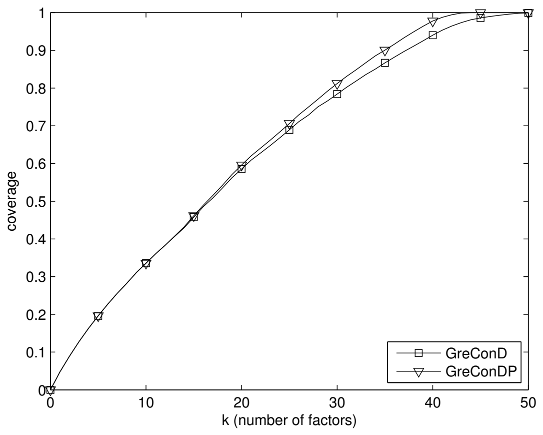

Figure 1: Comparison of GreConDP with GreConD on synthetic data, $P = 4$

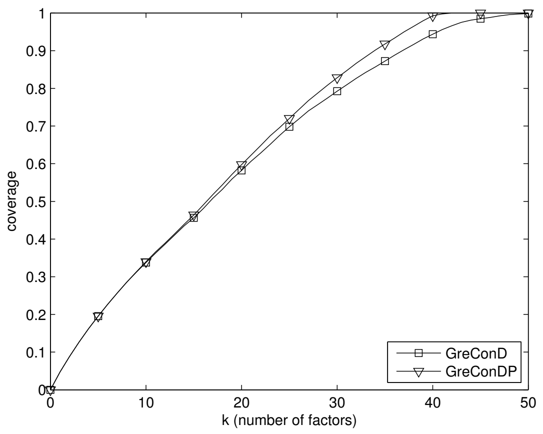

Figure 2: Comparison of GreConDP with GreConD on synthetic data, $P = 8$

[1]NOW public release 030717, available from http://www.helsinki.fi/science/now/

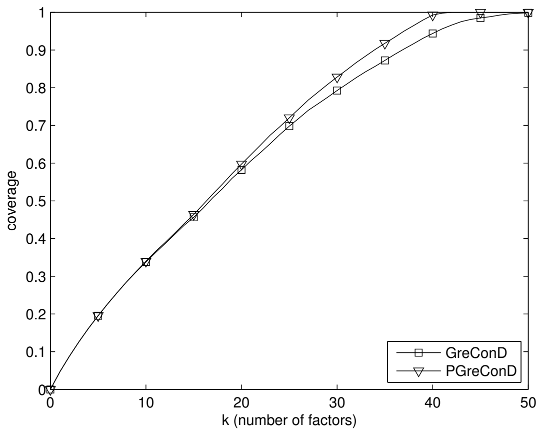

Figure 3: Comparison of GreConDP with GreConD on synthetic data, $P = 16$

We can see that GreConDP considerably outperforms original GreConD algorithm, especially from the AFP point of view and slightly from the DBP point of view. Namely, the coverage values are higher for a given number of factors and the difference grows with the number of factors. Eventually, a full coverage of input data is obtained with less factors – let us note that the number of factors delivered by GreConDP is in most cases equal to the expected number of factors (40, see Section 4.1.1) and, moreover, the factors are the original factors used to generate the data. For small numbers of factors (values of $k < 10$), however, the difference between the two algorithms is slight. On the other hand, for increasing number P of processes this difference slowly increases (for $P = 1$, GreConDP produces the same results as GreConD).

For real datasets we obtain similar results, see Figures 4, 5, 6, 7 and 8. GreConDP outperforms GreConD on Mushroom, Paleo and Zoo datasets. Here, however, full coverage of data is obtained from both algorithms with the same number of factors but GreConDP gives higher coverage values for small values of k. In particular, this is quite notable on the Mushroom dataset, see the Figure 6. On Emea, GreConDP produces better results than GreConD from the DBP point of view but from the AFP point of view it is outperformed by GreConD. This is due to the fact that the average number of equally locally optimal factors per factor (see above) for this dataset is extremely high and the advantage of GreConDP over GreConD in utilizing more (but few compared to the number) of the equally optimal factors rather than just one vanishes. We also observed a similar behavior on Firewall 1 dataset. On the DBLP dataset, GreConDP produces exactly the same decompositions as GreConD so we do not include a graph for it. Let us also note that for this dataset we know the Boolean rank (19) and decompositions produced by both algorithms are optimal.

4.3 Similarity of Factorizations

As we saw in Section 3.2, the GreConDP algorithm produces P most optimal factorizations of the given input Boolean matrix (instead of just one) where P is the number of processes. Hence a natural question arises: How much are those factorizations similar to each other? For $P = 8$, we obtained for all datasets from Table 1 similar results like those shown in Table 2. The numbers in the table refer to the degree

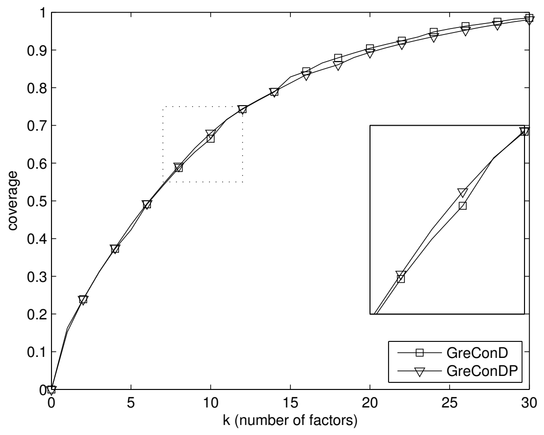

Figure 4: Comparison of GreConDP with GreConD on Emea dataset, $P = 4$

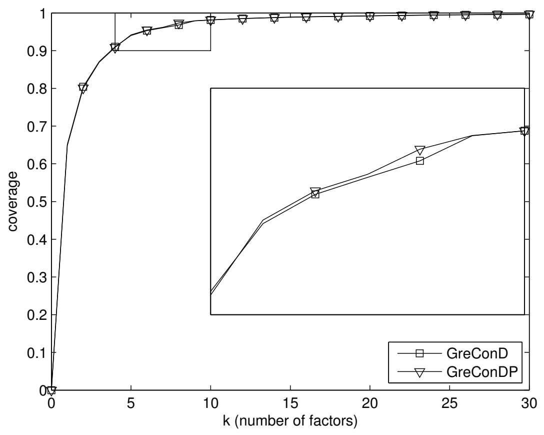

Figure 5: Comparison of GreConDP with GreConD on Firewall 1 dataset, $P = 4$

of similarity of two factorizations defined as follows: factorization F_i in a row of the table and factorization F_j in a column of the table are similar to degree $p \in [0, 1]$ if $p \cdot 100$ percent of factors of F_i is also present (as the same factors) in F_j. Note that in this simple and easily interpretable (in general non-symmetric) similarity measure we do not consider indices of factors in the factorizations (i.e., in particular, if one of the factorizations is a permutation of the other one these are measured as equal).

We can see that the factorizations obtained by GreConDP are rather similar to each other (especially the best ones, denoted by F_i with index i close to 1). This is not surprising since the factor search strategy in the GreConD algorithm, which is included also in GreConDP, has its limits.

Moreover, in Figure 9 we can see the progress of the similarities of factorizations F_1, F_4 and F_7 of the Mushroom dataset for the number of the first factors going from 1 to 30. As we can see, the GreConDP algorithm starts with different factors (the factorizations are not similar at all) and with more factors the factorizations become more similar. That means that the algorithm finds the same factorizations in different ways. Similar results were obtained also for the others factorizations.

	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8
F_1	1.000	0.991	0.991	0.991	0.983	0.983	0.991	0.983
F_2	0.991	1.000	0.991	0.983	0.991	0.983	0.983	0.991
F_3	0.991	0.991	1.000	0.983	0.983	0.991	0.983	0.983
F_4	0.991	0.983	0.983	1.000	0.991	0.991	0.991	0.983
F_5	0.983	0.991	0.983	0.991	1.000	0.991	0.983	0.991
F_6	0.983	0.983	0.991	0.991	0.991	1.000	0.983	0.983
F_7	0.991	0.983	0.983	0.991	0.983	0.983	1.000	0.991
F_8	0.983	0.991	0.983	0.983	0.991	0.983	0.991	1.000

Table 2: Similarity of the first eight factorizations of Mushroom dataset

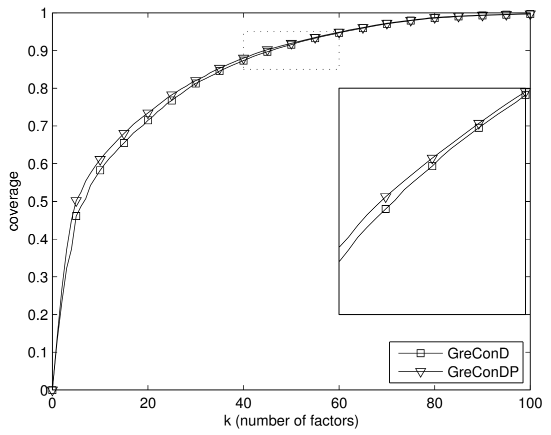

Figure 6: Comparison of GreConDP with GreConD on Mushroom dataset, $P = 4$

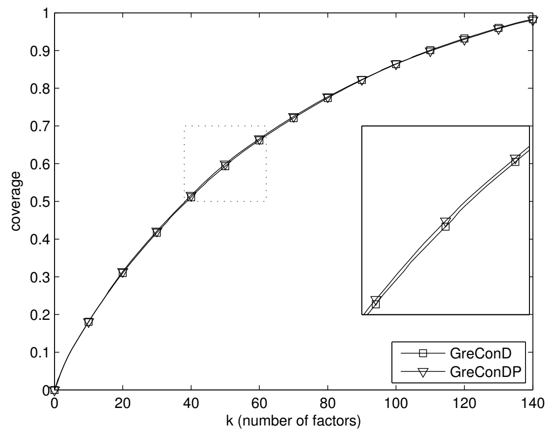

Figure 7: Comparison of GreConDP with GreConD on Paleo dataset, $P = 4$

4.4 Running Time

Usually, (theoretical asymptotic) time complexity of a BMF algorithm is not a primary concern in the BMF community, see e.g. (Belohlavek et al. 2015). Nevertheless, below we provide brief remarks on the observed running time of the GreConDP algorithm compared to the GreConD algorithm.

We implemented both GreConD and GreConDP in MATLAB. Critical parts (computing the operators $^{\uparrow}$ and $^{\downarrow}$, recall Section 3.1) were written in C and compiled to binary MEX files. For parallelization we used the Parallel Computing Toolbox MATLAB package. None of the algorithms was optimized for speed.

Despite that, each of the evaluated datasets (Table 1) was factorized by both algorithms, on an ordinary PC, in order of minutes[2]. The slowdown of GreConDP to GreConD depends on the number of processes run in GreConDP vs. the number of processor units. If we use less processes than we have processor units, GreConDP is only a slightly slower than GreConD, mainly due to the parallelization overhead (synchronization barriers and the critical section). If we use p times more processes than we have processor units, our observation is that GreConDP is approximately $p/2$ slower than GreConD.

5 Conclusions

In the paper we presented a general parallelization scheme for Boolean matrix factorization algorithms and also a new algorithm, called GreConDP, utilizing this scheme. The algorithm is based on one of the well established algorithms for Boolean matrix decomposition, the GreConD algorithm (Belohlavek et al. 2010). We chose GreConD as our base algorithm due to its relative simplicity and high efficiency but the proposed parallelization scheme can be applied to arbitrary sequential heuristic algorithm for Boolean matrix decomposition.

We evaluated properties and results delivered by GreConDP in various experiments involving synthetic (randomly generated) and real datasets. From presented results we can see that the algorithm outperforms in most cases the base algorithm, GreConD, most importantnly from the quality of decomposition standpoint, at almost none or moderate computing time expenses (depending the number of parallel runs/processes vs. the number of available processor units)—which were the objectives of our parallelization scheme. Namely, for the synthetic data, coverage produced by GreConDP is higher than coverage produced by GreConD for the same number of factors and in the end the data is fully explained by considerably less number of factors; for the real datasets, at least for those we used, the data is, however, fully explained by the same number of factors (and the final decompositions are very similar) but GreConDP produces higher coverage than GreConD for small numbers of factors in the beginnings of decomposition computation, provided the numbers of equally locally optimal factors in partial decomposition constructions are not very high (not significantly higher than the number of parallel runs)—only then the utilization of more equally optimal factors is beneficial. Moreover, as expected and intended, GreConDP tends to produce better results than GreConD with the increasing number of parallel runs. What was also expected, and has been observed, is that although producing rather similar

[2]An implementation of GreConD in C optimized for speed factorizes the datasets on an ordinary PC in order of seconds, see e.g. (Belohlavek et al. 2010).

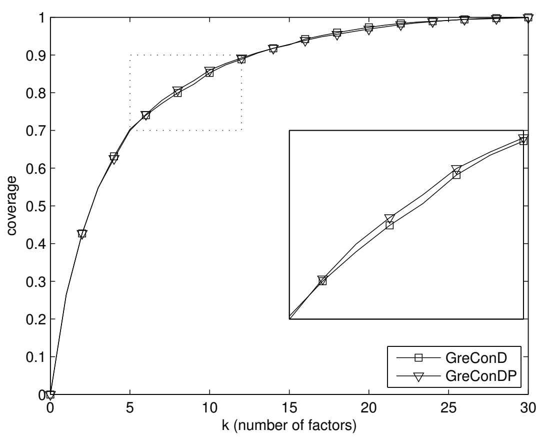

Figure 8: Comparison of GreConDP with GreConD on Zoo dataset, $P = 4$

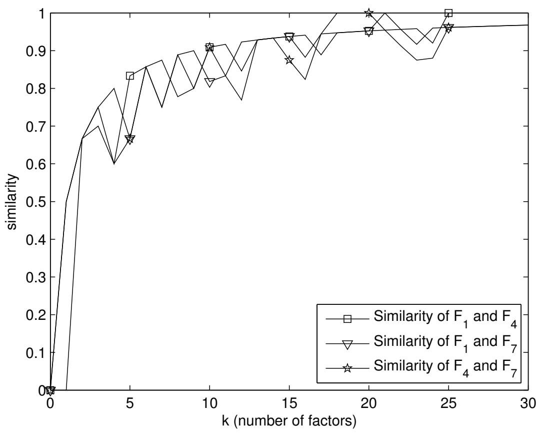

Figure 9: Progress of similarities of factorizations of Mushroom dataset for the first 30 factors

final decompositions (especially the best ones), the algorithm starts with different factors in the decompositions, i.e. the similar final decompositions are constructed in different ways.

The observed results encourage us to the following future research directions. First, apply the proposed general parallelization scheme to other BMF algorithms, especially to GreEss (Belohlavek et al. 2015) and Asso (Miettinen et al. 2008) which both involve a similar heuristic strategy like GreConD but in a different manner. Second, study the properties of the equally locally optimal factors—the problem which every heuristic algorithm faces—in order to find further ways leading to better factorizations.

References

Bache, K., Lichman, M. (2013), `http://archive.ics.uci.edu/ml`, University of California, School of Information and Computer Science, Irvine, CA.

Belohlavek, R. & Vychodil, V. (2010), Discovery of optimal factors in binary data via a novel method of matrix decomposition, *Journal of Computer and System Sciences* **76**(1), 3–20.

Belohlavek, R. & Trnecka, M. (2015), From-Below Approximations in Boolean Matrix Factorization: Geometry and New Algorithm, *Journal of Computer and System Sciences* **81**(8), pp 1678–1697.

Berry, M. W., Mezher, D., Philippe, B. & Sameh, A. (2006), Parallel Algorithms for the Singular Value Decomposition, *in* Kontoghiorghes, E. (ed.): 'Handbook on Parallel Computing and Statistics', Stat. Textb. Monogr., 184, Chapman & Hall/CRC, pp. 117–164.

Ene, A., Horne, W., Milosavljevic, N., Rao. P, Schreiber, R. & Tarjan, E. R. (2008), Fast exact and heuristic methods for role minimization problems, *in* 'ACM SACMAT Proceedings of the 13th ACM Symposium on Access Control Models and Technologies', pp. 1–10.

Ganter, B. & Wille, R. (1999), *Formal Concept Analysis: Mathematical Foundations*, Springer.

Geerts, F., Goethals, B., & Mielikäinen, T. (2004), Tiling databases, *in* 'Proc. Discovery Science', pp. 278–289.

Kannan, R., Ballard, G. & Park, H. (2016), A high-performance parallel algorithm for nonnegative matrix factorization, *in* 'Proc. 21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'16)'.

Lucchese, L., Orlando, S., & Perego, R. (2010), Mining Top-K Patterns from Binary Datasets in presence of Noise, *in* 'SIAM ICDM International Conference on Data Mining', pp. 165–176.

Miettinen, P., Mielikinen, T., Gionis, A., Das, G. & Mannila, H. (2008), The Discrete Basis Problem, *IEEE Transactions on Knowledge and Data Engineering* **20**(10), pp 1348–1362.

Stockmeyer, L. (1975), The set basis problem is NP-complete, Tech. Rep. RC5431, IBM, Yorktown Heights, NY, USA.

Tatti, N., Mielikainen, T., Gionis, A. & Mannila, H. (2006), What is the Dimension of Your Binary Data?, *in* 'IEEE ICDM International Conference on Data Mining', pp. 603–612.

Xiang, Y., Jin, R., Fuhry, D. & Dragan, F. F. (2011), Summarizing transactional databases with overlapped hyperrectangles, *Data Mining and Knowledge Discovery* **23**(2), 215–251.