J. Parallel Distrib. Comput. 123 (2019) 180–191

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

Parallel exploration of partial solutions in Boolean matrix factorization

Jan Outrata *, Martin Trnecka

Department of Computer Science, Palacký University Olomouc, 17. listopadu 12, CZ–771 46 Olomouc, Czech Republic

HIGHLIGHTS

- General parallelization scheme for Boolean matrix factorization is proposed.
- Several top-k matrix decompositions are obtained in parallel.
- The scheme can be applied to any sequential heuristic BMF algorithm.
- Application on two well known algorithms, GreConD and Asso, is presented.

ARTICLE INFO

Article history:
Received 25 September 2017
Received in revised form 27 April 2018
Accepted 24 September 2018
Available online 4 October 2018

Keywords:
Boolean matrix factorization
Parallel algorithm
Data preprocessing

ABSTRACT

Boolean matrix factorization (BMF) is a well established method for preprocessing and analysis of data. There is a number of algorithms for BMF, but none of them uses benefits of parallelization. This is mainly due to the fact that many of the algorithms utilize greedy heuristics that are inherently sequential. In this work, we propose a general parallelization scheme for BMF in which several locally optimal partial matrix decompositions are constructed simultaneously in parallel, instead of just one in a sequential algorithm. As a result of the computation, either the single best final decomposition or several top-k of them may be returned. The scheme can be applied to any sequential heuristic BMF algorithm and we show the application on two representative algorithms, namely GreConD and Asso. Improvements in decompositions are presented via results from experiments with the new algorithms on synthetic and real datasets.

1. Introduction

Boolean Matrix Factorization (BMF, also known as Boolean matrix decomposition) 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. Two optimization variants of the basic problem are dealt with in the literature: the Approximate Factorization Problem (AFP) [4] and the Discrete Basis Problem (DBP) [11]. In AFP, a (non-trivial) solution with the inner matrix product dimension as low as possible is requested for a given maximal, usually zero, approximation of the solution (i.e. a difference or error of the matrix product from the input matrix). DBP, on the other hand, demands decomposition with a minimal approximation (error) for a given maximal inner dimension. The optimal solution to AFP with zero approximation (i.e. exact decomposition) is a decomposition with the least dimension for which an exact decomposition of the input Boolean matrix exists. This least dimension is called the *Boolean rank* (or Schein rank) of the matrix. However, the problem of finding the Boolean rank for a given Boolean matrix, as well as both the AFP and the DBP, is NP-hard (due to the NP-hardness of the associated Set Basis Problem [13]). Thus, existing BMF algorithms seek for a sub-optimal decomposition with the inner matrix product dimension as close to the Boolean rank as possible (for AFP) or with approximation error as low as possible (for DBP). Usually some heuristic approach is utilized in the search.

Well-recognized efficient algorithms are GreConD [4] and GreEss [3], both originally designed for the AFP, and Asso [11] for the DBP. Other competitive algorithms for either AFP or DBP include e.g. PaNDa [10] or Hyper [15]. Being heuristic, the algorithms iteratively construct the final decomposition of input matrix from some partial, or intermediate, (approximate) decompositions. Typically a sequence of intermediate decompositions is constructed where each decomposition is constructed based on the previous one (or the empty one in case of the first decomposition) with the aim to be a lower approximation of the input matrix. In the most commonly used greedy approach, utilized in GreConD, GreEss and also in Asso, each intermediate decomposition extends the previous one in the sequence by increasing the inner matrix product dimension and the best extension is sought usually in terms

* Corresponding author.
E-mail addresses: jan.outrata@upol.cz (J. Outrata), martin.trnecka@gmail.com (M. Trnecka).

https://doi.org/10.1016/j.jpdc.2018.09.014
0743-7315/

of the input matrix approximation. However, such a sequence of decompositions, although each of them is constructed as an optimal solution from all possible solutions based on the previous one(s), may not end with a globally optimal decomposition (if that is not the only decomposition). As illustrated by the following example.

Example 1. Consider the two below depicted sequences of intermediate decompositions. The input Boolean matrix for both of them is shown before the first decomposition (with numbered rows/objects and columns/attributes). Each decomposition in both sequences is the least approximation of the input matrix which extends the previous decomposition in the sequence (or the empty decomposition) by increasing the inner matrix product dimension by one. However, the dimension of the last decomposition of the bottom sequence is smaller than that of the top sequence. Moreover, the dimension is the smallest, i.e. equal to the Boolean rank of the input matrix, which means that the decomposition is globally optimal.

$$\begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \end{array}\!\!\begin{array}{c} \begin{array}{ccccc} 0 & 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} 1&1&0&1&0 \\ 1&1&0&1&1 \\ 1&1&1&0&0 \\ 1&0&0&0&1 \end{pmatrix} \end{array} \approx \begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \end{pmatrix} \circ \begin{pmatrix} 1&1&0&0&0 \end{pmatrix} \approx \begin{pmatrix} 1&0 \\ 1&1 \\ 1&0 \\ 0&1 \end{pmatrix} \circ \begin{pmatrix} 1&1&0&0&0 \\ 1&0&0&0&1 \end{pmatrix} \approx$$

$$\begin{pmatrix} 1&0&1 \\ 1&1&1 \\ 1&0&0 \\ 0&1&0 \end{pmatrix} \circ \begin{pmatrix} 1&1&0&0&0 \\ 1&0&0&0&1 \\ 1&1&0&1&0 \end{pmatrix} \approx \begin{pmatrix} 1&0&1&0 \\ 1&1&1&0 \\ 1&0&0&1 \\ 0&1&0&0 \end{pmatrix} \circ \begin{pmatrix} 1&1&0&0&0 \\ 1&0&0&0&1 \\ 1&1&0&1&0 \\ 1&1&1&0&0 \end{pmatrix}$$

$$\begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \end{array}\!\!\begin{array}{c} \begin{array}{ccccc} 0 & 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} 1&1&0&1&0 \\ 1&1&0&1&1 \\ 1&1&1&0&0 \\ 1&0&0&0&1 \end{pmatrix} \end{array} \approx \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \end{pmatrix} \circ \begin{pmatrix} 1&1&0&1&0 \end{pmatrix} \approx \begin{pmatrix} 1&0 \\ 1&0 \\ 0&1 \\ 0&0 \end{pmatrix} \circ \begin{pmatrix} 1&1&0&1&0 \\ 1&1&1&0&0 \end{pmatrix} \approx$$

$$\begin{pmatrix} 1&0&0 \\ 1&0&1 \\ 0&1&0 \\ 0&0&1 \end{pmatrix} \circ \begin{pmatrix} 1&1&0&1&0 \\ 1&1&1&0&0 \\ 1&0&0&0&1 \end{pmatrix}$$

To increase the chance of reaching the globally optimal decomposition, more (ideally all) sequences of decompositions, sometimes even not locally optimal, need to be explored. In sequential algorithms like the above mentioned one cannot afford that. Partly due to resulting algorithm time complexity and inferential performance reasons but mainly because of the single choice of locally optimal intermediate decomposition which is usually hard-coded in the algorithm design or its implementation. This is where a parallel computation approach can be used, i.e. more sequences of decompositions can be explored in more concurrent processes. Furthermore, with the development and growing affordability of multi-core processors and other hardware allowing parallel computations, interest in parallel computing increases and parallel algorithms which better utilize hardware are preferred.

Interestingly, to our knowledge, there is no parallel algorithm for *Boolean* matrix factorization introduced in the literature to date, with one recent exception, see below. Of course one can consider a trivial "parallel BMF algorithm" by creating several variants of a sequential algorithm differing in the (hard-coded) choice of locally optimal intermediate decomposition, running them in parallel and selecting the best of obtained final decompositions for output. Such attempts are, however, not considered a parallel algorithm. They are just several altered copies of a sequential algorithm run in parallel. Second, note that the adjective 'Boolean' in Boolean matrix factorization needs to be emphasized. There exist parallel algorithms for some of the many existing factorization methods designed originally for real-valued matrices, see for instance [5] or [9]. Those algorithms obviously can be applied also to Boolean matrices. However, as [14] and other authors conclude, a problem with applying to Boolean matrices 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 (true) parallel BMF algorithms may be that the most commonly used greedy heuristic approach discussed above is inherently sequential. In [16], the only paper introducing a parallel approach to BMF to our knowledge, the authors come round this problem by dividing a solution to BMF into two phases. The first one, called exploration phase, consists in (random) splitting the input Boolean matrix into a large number of quite small row sub-matrices which are in concurrently, but independently, run processes optimally decomposed into matrix products with inner dimension equal to Boolean rank. Parallelization there mitigates the computation time severity (due to the NP-hardness of finding Boolean rank). Then, in the follow-up phase, called aggregation phase, the optimal decompositions are merged and used in serial greedy search for a final decomposition of the whole input matrix with the inner matrix product dimension as low as possible. Note that the problem being solved in [16], the Set Basis Problem, is closely related to the DBP as introduced above. Possibly, other reason for not having parallel algorithms directly for BMF could be that factorization of Boolean matrices as a standalone research area is relatively young within the data mining research and not so elaborated as factorization of real-valued matrices.

Our contribution lies neither in a parallel algorithm for BMF which would compute an input matrix decomposition in concurrent processes, neither in splitting the problem into smaller subproblems and searching for a solution (decomposition) with the help of solutions to those sub-problems obtained concurrently (as in [16]). Instead, as suggested above, we show a general parallelization scheme for (sequential) BMF algorithms consisting in a viable way of exploring in several concurrent processes several different sequences of intermediate decompositions in a hope to find a globally optimal decomposition. In essence, our approach consists in following, in more processes running concurrently, several possible choices of the locally optimal intermediate decompositions building several sequences of the decompositions. As a result of the computation, either the single optimal final decomposition or several top-k of them can be returned, which is a distinctive feature of our approach. The scheme, as such, can be applied, with more or less effort, to any sequential greedy heuristic BMF algorithm and, properly adjusted, also to non-greedy ones. To demonstrate the scheme and the approach, we used as the base sequential algorithm the GreConD algorithm, due to its relative simplicity and high efficiency. Moreover, to show that the scheme can be applied to other (sequential heuristic) BMF algorithms, we also apply it to the Asso algorithm.

In the remainder of the paper, Section 2 recaps basic notions of Boolean matrix factorization. Section 3, the main section in our paper, contains first a brief description of the base GreConD algorithm and then a presentation of our approach of in parallel computing several top matrix decompositions. The approach is demonstrated on GreConD including full pseudocodes of both original GreConD and our modification of it. Section 4 then presents results from experimental evaluation of the approach, Section 5 includes a short illustration of applying our approach to the Asso algorithm and finally Section 6 concludes the paper.

$$I = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{pmatrix} \circ \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 \end{pmatrix} = A \circ B$$

Fig. 1. Factor as rectangle full of 1s.

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 are 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

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

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 (by matrix A) and which is characterized by some of the m original attributes (by matrix B).

This easy interpretation of factors is further supported by the so-called geometric view on factors and on BMF as whole, 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, which is a Boolean matrix whose entries with 1 form, upon a suitable permutation of rows and columns, a rectangular area (full of 1s). See Fig. 1 for illustration. 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 (i.e., a Boolean matrix product from (1) is looked at as the max-superposition $\max_{l=1}^{k}(J_l)_{ij}$ of rectangles $J_l = A_{\_l} \circ B_{l\_}$).

In optimization variants of the basic BMF problem, either the number k of factors (in AFP) or the approximation error of I (in DBP) is requested to be as small as possible. 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 [3,4,8,11] by the following function $c(l)$ representing the *coverage quality* of the first l factors delivered by the particular algorithm:

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

The function measures how well the data are explained by the l factors. We will use this function in Section 4 devoted to experiments where we also state what $c(l)$ should satisfy for a good BMF algorithm.

The two optimization BMF problems we are concerned are defined as follows:

Definition 1 (*Approximate Factorization Problem, AFP [3], Implicitly Already in [4]*)**.** 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 $\|I - A \circ B\| \leq \varepsilon$.

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

Definition 2 (*Discrete Basis Problem, DBP [11]*)**.** Given $I \in \{0, 1\}^{n\times m}$ and a positive integer k, find $A \in \{0, 1\}^{n\times k}$ and $B \in \{0, 1\}^{k\times m}$ that minimize $\|I - A \circ B\|$.

Thus DBP, on the other hand, emphasizes the importance of the first few (presumably most important) k factors.

For a more thorough study of the factor model and the geometric view on it described above (for the AFP and the BMF in general) we refer the reader to [4] or [3].

3. BMF in parallel

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 sequential base for the demonstration of our parallelization scheme.

3.1. GreConD Algorithm

The algorithm, proposed in [4] and called Algorithm 2 there (the name GreConD comes from [3]), implements a greedy search for factors. Each factor is sought to explain as much of the yet unexplained portion 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. The computation proceeds again in a greedy manner, see the description below. In the description of the algorithm we will use the geometric view on factors (as [4] does too), identifying each factor with a rectangular matrix (rectangle) full of 1s 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 from *formal concept analysis* (FCA) [7] in which maximal rectangles correspond to so-called formal concepts (basic data units utilized in FCA) and which is used as a description platform of the algorithm in [4]—now, GreConD means "Greedy Concepts on Demand". We do not use FCA in this paper and refer the readers interested in (a

fruitful) connection between BMF and FCA to [4] or [3]. We will now briefly describe the GreConD algorithm.

The algorithm, in its seek for a factor, starts with the empty set D 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 h. Together with h other attributes may be possibly added to D due to the construction of each factor as a maximal rectangle. Namely, D is after each addition of h completed, or *closed*, to contain all the attributes which are shared by all objects having the attributes in D—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 determines a maximal rectangle. The selected attribute h is such that the rectangle grown by h covers as many still uncovered 1s in the input Boolean matrix I as possible. Then, with respect to the aim of computing factors as rectangles which cover as many still uncovered 1s in matrix I as possible, the rectangle is repeatedly grown by further selected attributes as long as the number of the still uncovered 1s in I covered by the grown 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 (i.e., the prescribed maximal error E is reached) or the prescribed number of factors is obtained—the algorithm was originally designed for the AFP but can also be easily used for the DBP. Finally, the characteristic vectors of object sets determining found maximal rectangles/factors constitute columns of the object–factor matrix A and the characteristic vectors of attribute sets of the rectangles/factors constitute rows of the factor–attribute matrix B. The matrices A and B, which determine the decomposition of the matrix I, form an output of the algorithm.

Example 2. Using numbers 0, 1, 2, 3 for rows/objects and 0, 1, 2, 3, 4 for columns/attributes of the input Boolean matrix I in Example 1, the algorithm has an option in its seek for the first factor to grow the empty set of attributes by either of the two attributes 1 or 3. The set $\{1\}$, closed to the set $\{0, 1\}$, together with the corresponding set $\{0, 1, 2\}$ of objects, determines maximal rectangle which covers six still uncovered 1s in I (highlighted in Fig. 1). Same number for the set $\{3\}$, closed to $\{0, 1, 3\}$, together with $\{0, 1\}$. Those maximal rectangles cover most still uncovered 1s in I among all maximal rectangles determined by all single attributes (and closed). And none of them can be further grown by another selected attribute so that the number of still uncovered 1s in I covered by the grown rectangle increases (actually, the first one can be grown to the second one, by adding attribute 3 and closing, but the number remains the same). So both rectangles can represent the first factor, as seen in Example 1 in the top and bottom sequence of decompositions, and the choice is done by the algorithm implementation. Let us continue with the first one (from the top sequence of decompositions depicted in Example 1), determined by the set $\{0, 1\}$ of attributes. Hence the first column of the constructed object–factor matrix A is $(1\ 1\ 1\ 0)^T$ (transposed characteristic vector of $\{0, 1, 2\}$) and the first row of the factor–attribute matrix B is $(1\ 1\ 0\ 0\ 0)$.

In construction of the second factor, attribute 4 is selected and closed to the set $\{0, 4\}$ of attributes. Since together with the corresponding set $\{1, 3\}$ of objects the determined maximal rectangle covers three still uncovered 1s in I (one 1 is already covered by the first factor). This is the most among all other rectangles determined by all remaining attributes. The rectangle cannot be further grown so it represents the second factor. Hence the second column of A is $(0\ 1\ 0\ 1)^T$ and the second row of B is $(1\ 0\ 0\ 0\ 1)$. For the third factor, the selected attribute is 3, as a set closed to $\{0, 1, 3\}$, set of objects is $\{0, 1\}$ and the maximal rectangle which again cannot be grown covers two (still uncovered) 1s. Finally, for the fourth factor the selected attribute is 2, as a set closed to $\{0, 1, 2\}$, set of objects is $\{2\}$ and the maximal rectangle covers the single last still uncovered 1 in I.

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 in 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 [7]) 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

$$\begin{aligned} C^{\uparrow} &= \text{sum of } [j] \in \{0, 1\}^{1\times m} \text{ for } j \text{ s. t. } I_{ij} = 1 \text{ for all } i \\ &\quad \text{for which } C_i = 1, \\ D^{\downarrow} &= \text{sum of } [i] \in \{0, 1\}^{n\times 1} \text{ for } i \text{ s. t. } I_{ij} = 1 \text{ for all } j \\ &\quad \text{for which } D_j = 1. \end{aligned}$$

According to our vector/matrix notation, the operation "sum of $[j]$" for several j produces a Boolean vector with jth items equal to 1 and all other items equal to 0 (i.e. the set with attributes j). Note also 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 in 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 and the rectangles represent factors which are "stored" in matrices A and 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 (the algorithm represented by the pseudocode solves AFP).

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

3.2. Running GreConD in parallel

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 a single factor computed this way may be locally optimal with respect to the aim to explain by an additional factor as much of the input matrix as possible, a series of factors computed this way, forming a final decomposition, may not be globally optimal. Hence, each but the first intermediate decomposition formed by the factor and all factors computed previously is only locally optimal. Moreover, there can be more different equally optimal factors instead of just one to add to the previous decomposition, forming more different extended intermediate decompositions to choose from and generally leading to different final decompositions. We have already encountered this in Example 1 where we had two different first equally optimal factors forming two different first

Algorithm 1: Original GreConD algorithm

Input: A Boolean matrix $I \in \{0,1\}^{n\times m}$ and a prescribed error $\varepsilon \geq 0$
Output: Boolean matrices $A \in \{0,1\}^{n\times k}$ and $B \in \{0,1\}^{k\times m}$

$A \leftarrow$ empty $n \times 0$ Boolean matrix
$B \leftarrow$ empty $0 \times m$ Boolean matrix
while $||I - A \circ B|| > \varepsilon$ **do**
  $D \leftarrow$ empty $1 \times m$ Boolean matrix
  $V \leftarrow 0$
  while there is j such that $D_j = 0$ and $cover(D + [j], I, A, B) > V$ **do**
    $W \leftarrow 0$
    forall j **do**
      if $cover(D + [j], I, A, B) > W$ **then**
        $h \leftarrow j$
        $W \leftarrow cover(D + [j], I, A, B)$
      end
    end
    $D \leftarrow (D + [h])^{\downarrow\uparrow}$
    $V \leftarrow W$
  end
  $A \leftarrow [A\, D^{\downarrow}], B \leftarrow \begin{bmatrix} B \\ D \end{bmatrix}$
end
return A and B

intermediate decompositions. Likely, the computation of a factor is also greedy, by growing the corresponding maximal rectangle by attributes selected so that the rectangle covers as many still uncovered 1s in the input matrix as possible. Similarly, while a single attribute selected in such way may be locally optimal w.r.t. the aim to cover by the maximal rectangle after growing as many still uncovered 1s in the input matrix as possible, several (and all) consequently selected attributes together, determining a final factor, may not be globally optimal. Hence, here the partial, or intermediate, factor (determined by the attribute and all attributes added to the corresponding maximal rectangle previously, with the exception of the first intermediate factor) is also only locally optimal. Finally, there can also be more equally optimal attributes to select, determining more, in general different, intermediate factors to choose from and leading to different final factors. This is also illustrated in Example 1: the two first factors are (solely) determined by equally optimal attributes, the second one and the fourth one (the maximal rectangles corresponding to the factors are actually determined by more attributes due to the closure operation).

In our approach we compute several different Boolean matrix decompositions simultaneously in parallel. This is done through parallel runs of several instances of a BMF algorithm, i.e. running a BMF algorithm in several concurrent processes, in such a way that (in general) different decompositions are computed by the instances. As mentioned above in the introduction section, several different sequences of locally optimal intermediate decompositions ending with several in general different locally optimal final decompositions are constructed by the instances, in a hope to find the globally optimal decomposition. For concurrently running instances of the GreConD algorithm it means that in the search for factors, in each iteration, several different locally optimal factors explaining most of the input Boolean matrix being decomposed are computed, instead of just one. Likely, in the factor computation, in each iteration (of maximal rectangle growing), several locally optimal 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, can form a (locally optimal) intermediate decomposition and each of the attributes (possibly with other attributes due to the closure), together with the attributes selected previously, can form a (locally optimal) intermediate factor. After each iteration, among all those possible intermediate decompositions and factors, only several top-k ones are selected for the next iteration. While the computation of the final factors from the selected intermediate factors is left serial in our below proposed parallel runs of GreConD, the construction of the final decompositions from the selected intermediate ones is done in parallel across all the instances of the algorithm.

Note that this approach is quite different from the approach of trivial "parallel GreConD" mentioned in the introductory section. In that approach each of the several altered copies of GreConD run in parallel constructs *independently* its own sequence of (locally optimal) intermediate decompositions ending with an optimal final decomposition and the best of the decompositions is selected for output afterwords. In our approach, on the other hand, the sequences of intermediate decompositions are by several concurrently run instances of GreConD constructed *jointly*, selecting top-k best factors (and top-k best attributes for factors) in each iteration of the algorithm—in a hope to find the globally optimal final decomposition.

Example 3. Considering Example 2 and each word 'several' in the above description to be 'two', the empty set of attributes is grown by both selected attributes 1 and 3 in the seek for the first factor. Both two different first factors are computed, determined by the sets $\{0, 1\}$ and $\{0, 1, 3\}$ of attributes. Hence we construct two different first intermediate decompositions, as illustrated in Example 1. Let us denote the decomposition with the first factor determined by set $\{0, 1\}$ (in the top sequence of decompositions in the illustration) by A and the decomposition with the first factor determined by set $\{0, 1, 3\}$ (in the bottom sequence) by B. Note that the computation of the two factors is done in a single instance of the modified GreConD since the other instance would do it exactly the same. The computation of further factors, however, continues already in two concurrently running instances, each one building upon its own first intermediate decomposition. One, building upon decomposition A, computes factors determined by sets $\{0, 4\}$ and $\{0, 1, 3\}$ of attributes with corresponding maximal rectangles covering three and two still uncovered 1s in I, respectively. We have seen this in Example 2. The other instance, building upon decomposition B, computes factors determined by sets $\{0, 4\}$ and $\{0, 1, 2\}$ of attributes (closed from selected attributes 4 and 2, respectively, without further growing) with corresponding maximal rectangles both covering three still uncovered 1s. Hence, for two second intermediate decompositions, we get tree equally optimal factors which means that only two of the factors will be selected for the decompositions. The choice is particular algorithm design and/or implementation dependent.

If the two factors built upon decomposition B are selected, the factors are computed again. Each one by each instance of the algorithm as a single factor, ending with two equal final decompositions which are both equal to the final decomposition depicted in the bottom part in Example 1 (one of the two decompositions has the last two factors swapped). If, on the other hand, one of the two selected factors is the factor built upon decomposition A (determined by set $\{0, 4\}$) and the other one the factor built upon decomposition B (determined by set $\{0, 1, 2\}$), the same factors (determined by $\{0, 4\}$, $\{0, 1, 3\}$ and $\{0, 1, 2\}$) are computed by the instances again and the factors determined by $\{0, 4\}$ and $\{0, 1, 3\}$ (with corresponding maximal rectangles covering three and two still uncovered 1s) are selected for two third intermediate decompositions. One of those decompositions is final (started as B and illustrated in the bottom sequence in Example 1). The other one (started as A, top sequence) is finalized by once more computed factor determined by set $\{0, 1, 2\}$ (and with corresponding maximal rectangle covering the single last still uncovered 1 in I) after the factor determined by $\{0, 1, 3\}$.

Summing up, in both cases of the choice of two factors for the second intermediate decomposition (built upon the decompositions A and B) a final decomposition consisting of three factors is

constructed. At the same time, it is a globally optimal decomposition since the Boolean rank of the input data equals three.

The idea described above is put into pseudocodes depicted in Algorithms 2 and 3. Algorithm 2 depicts an algorithm which, loosely speaking, represents several instances of (modified) GreConD algorithm (Algorithm 1) where each instance is represented by the procedure depicted in Algorithm 3. All instances run simultaneously in parallel – hence the name GreConDP ("P" for "Parallel") – and *jointly* construct several top-k final decompositions of input Boolean matrix I. The number of instances, equal to the number of final decompositions, is given by the number P of concurrent 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 by sub-optimality. Only the first one, the best of them, which is represented by A_1 and B_1, is output.

Algorithm 2: GreConDP – GreCond in parallel

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 ← empty n × 0 Boolean matrix
2  B_r ← empty 0 × m Boolean matrix
3  U_r ← 0 (r ∈ {1, ..., P})
4  GreConDP-I(I, ε, A_1, B_1, true)
5  for r = 1, ..., P do
6  |  with process r
7  |  |  GreConDP-I(I, ε, A_r, B_r, false)
8  |  end
9  end
10 wait for all processes
11 return A_1 and B_1
```

Let us now focus on Algorithm 3 depicting the core of the whole GreConDP algorithm and compare it to the original version of GreConD depicted in Algorithm 1. The procedure GreConDP-I constructs the ith final 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 factors are denoted by D_r. By lines 3 and 5, all those sets are repeatedly (and serially) grown by several selected attributes, in lines 2 and 27. Only in the first iteration of growing, when all D_rs are empty Boolean matrices, only the first D_r is grown, see line 24. The selection of attributes for a particular D_r is done in lines 6 and 15. Compare it to lines 7 to 13 in Algorithm 1. Instead of a single 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. Moreover, the attributes are, as a bonus here, sorted (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, after closing, 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, over all D_rs, determining maximal rectangles which cover most still uncovered 1s in I, are kept as E_s. This is done in lines 16 and 23, extending lines 14 and 15 in Algorithm 1. The sets E_s are also sorted 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 26. Once the repeated growing of maximal rectangles finishes (when the number computed by function *cover* at line 3 does not increase) we have P final maximal rectangles (determined by D_r) representing final factors which have to be "stored" in matrices A_i and B_i.

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, $r \in \{1, \ldots, P\}$, and a number $P \geq 1$ of processes

```
1  while ||I − A_i ∘ B_i|| > ε do
2  |  D_r ← empty 1 × m Boolean matrix, V_r ← 0 (r ∈ {1, ..., P})
3  |  while there is ⟨r, j⟩ such that (D_r)_j = 0 and cover(D_r + [j], I, A_i, B_i) > V_r do
4  |  |  E_s ← ∅ (s ∈ {1, ..., P})
5  |  |  forall r from the ⟨r, j⟩ do
6  |  |  |  W_s ← 0 (s ∈ {1, ..., P})
7  |  |  |  forall j from the ⟨r, j⟩ do
8  |  |  |  |  s ← P
9  |  |  |  |  while s > 0 and cover(D_r + [j], I, A_i, B_i) > W_s do
10 |  |  |  |  |  if s > 1 and W_{s−1} > 0 and
   |  |  |  |  |  (D_r + [j])^{↓↑} = (D_r + [h_{s−1}])^{↓↑} then break
11 |  |  |  |  |  if s < P then h_{s+1} ← h_s, W_{s+1} ← W_s
12 |  |  |  |  |  h_s ← j, W_s ← cover(D_r + [j], I, A_i, B_i)
13 |  |  |  |  |  s ← s − 1
14 |  |  |  |  end
15 |  |  |  end
16 |  |  |  for p = 1, ..., P do
17 |  |  |  |  s ← P
18 |  |  |  |  while s > 0 and W_p > V_s do
19 |  |  |  |  |  if s < P then E_{s+1} ← E_s, V_{s+1} ← V_s
20 |  |  |  |  |  E_s ← (D_r + [h_p])^{↓↑}, V_s ← W_p
21 |  |  |  |  |  s ← s − 1
22 |  |  |  |  end
23 |  |  |  end
24 |  |  |  if D_r = empty 1 × m Boolean matrix then break
25 |  |  end
26 |  |  for r = 1, ..., P do D_r ← E_r
27 |  end
28 |  U_i ← 0
29 |  synchronization barrier
30 |  for r = 1, ..., P do
31 |  |  s ← P
32 |  |  begin critical section
33 |  |  |  while s > 0 and V_r > U_s do
34 |  |  |  |  if s < P then A_{s+1} ← A_s, B_{s+1} ← B_s, U_{s+1} ← U_s
35 |  |  |  |  A_s ← [A_i D_r^{↓}], B_s ← [B_i; D_r], U_s ← V_r
36 |  |  |  |  s ← s − 1
37 |  |  |  end
38 |  |  end
39 |  end
40 |  synchronization barrier
41 |  if first = true then break
42 end
```

This is done in lines 28 and 40, over all P decompositions (determined by A_is and B_is) constructed in the P processes running concurrently the procedure GreConDP-I. Therefore, due to data consistency reasons in 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 29 and 40. Among all 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 35). And the matrices are sorted 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 in lines 32 and 38. 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 final decomposition (ith in the time of calling procedure GreConDP-I in Algorithm 2), until the prescribed number of 1s in I given by ε is covered (solving AFP).

Our last comment concerns line 41 of Algorithm 3 and line 4 of Algorithm 2. Calling 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 of the same decomposition). From this reason we first call the procedure once to construct the first P non-empty decompositions only (in the first iteration) and after that we can continue the construction of the final decompositions in P processes.

Correctness of the GreConDP algorithm (procedures GreConDP and GreConDP-1) follows from the correctness of the GreConD algorithm, see [4], and the above detailed description of GreConDP.

4. Experimental evaluation

We now provide selected results from an extensive experimental evaluation of the GreConDP algorithm described in the previous section and present a comparison with the base GreConD algorithm and the trivial "parallel GreConD" algorithm. We do not include a comparison with other algorithms and approaches – some of them were mentioned in Section 1 – to general BMF. A comparison of GreConD with other BMF algorithms can be found e.g. in [3]. Let us note that GreConD – according to [3,4] and other sources – tends to produce very good results, often the best, and that was also among the reasons for us to use it as a base algorithm for the demonstration and evaluation of our parallelization scheme.

4.1. Datasets

As in the typical experimental 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 used 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 product was set to 40 for all the datasets, i.e. the expected number of factors is 40 (but the Boolean rank of a dataset can be lower). Data generated in this way are standard for evaluation of BMF algorithms (see e.g. [3,11]).

4.1.2. Real data

We used datasets Emea [6], DBLP [11], Domino [6], Firewall 1 [6], Chess [1], Mushroom [1], Paleo[1] and Zoo [1], see Table 1 for their characteristics. All of the datasets are well known and used in the literature on BMF. The characteristics 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).

The last characteristics is an important one. Recalling Section 3.2, factors computed by GreConD (and other heuristic BMF algorithms) are alone locally optimal w.r.t. the aim to explain by factors as much of the input Boolean matrix being decomposed as possible. For an intermediate 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. However, the choice among the equally optimal factors, performed by classical sequential heuristic BMF algorithms like GreConD, influences results obtained by the algorithms. Therefore, to point out the influence, 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.

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

Table 1
Real datasets and their characteristics.

Dataset	Size	Dens. 1	Equal
Emea	3046 × 35	0.095	157.279
DBLP	19 × 6980	0.130	2.105
Domino	79 × 231	0.040	8.048
Firewall 1	365 × 709	0.124	31.168
Chess	3196 × 76	0.487	10.075
Mushroom	8124 × 119	0.193	3.148
Paleo	501 × 139	0.051	5.868
Zoo	101 × 28	0.305	5.867

4.2. Quality of decomposition

We provide results on the most important aspect of evaluation of performance of BMF algorithms—quality of decomposition delivered by an algorithm (recall Section 2). As common in the literature on BMF, we evaluate the obtained results from 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. Comparison with GreConD and trivial "parallel GreConD"

We observe values of $c(l)$ (see Section 2) 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 are fully explained by all k computed factors (in which case $c(l) = 1$).

The values of $c(l)$ (averages over 1000 iterations) for the synthetic data computed with different values of P (see Algorithms 2 and 3) are shown in Figs. 2a–2d. In case of GreConDP algorithm we present only values for the best decomposition (in basic GreConDP produces more than one decomposition). "Parallel GreConD" selects the best decomposition from decompositions produced (independently) by several altered copies of GreConD run in parallel (recall the trivial "parallel GreConD"). We study similarities of the several decompositions later in Section 4.3.

First, see that the trivial "parallel GreConD" performs much the same as the original GreConD algorithm, meaning that the approach almost does not lead to improvement in decomposition (actually the improvement increases with the number P of processes but it is very subtle). Contrary to that and more importantly, we can see that GreConDP considerably outperforms the original GreConD algorithm, especially from the AFP point of view and slightly also from the DBP point of view. Namely, the coverage

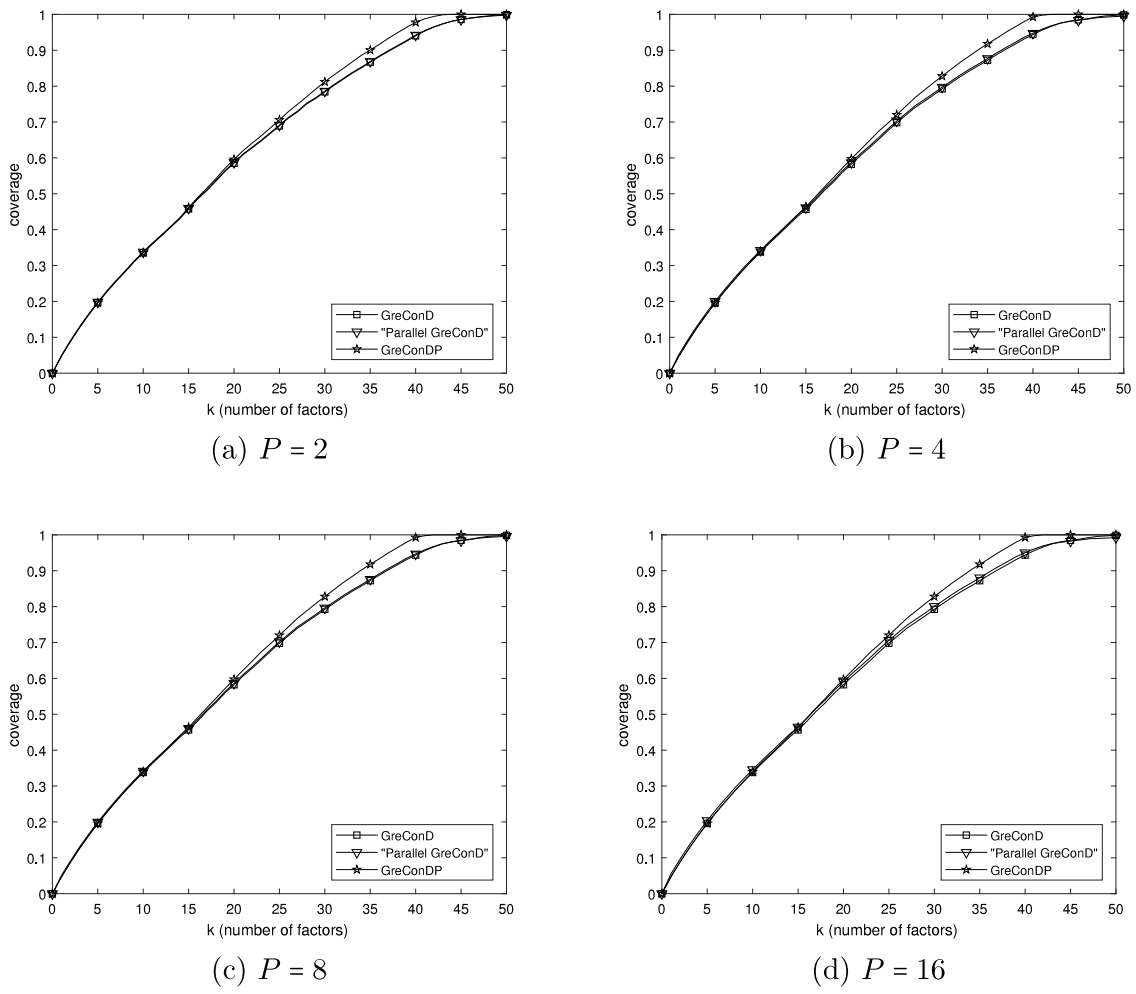


Fig. 2. Comparison of GreConDP with "parallel GreConD" and GreConD on synthetic data.

values are higher for a given number of factors and the difference grows with the number. 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. Also, for increasing number P of processes the difference between the algorithms slowly increases, when for particular larger values of P the amount of increase goes to zero (see Figs. 2a and 2d for $P = 8$ and $P = 16$). Of course, for $P = 1$ GreConDP produces the same results as GreConD.

For the real datasets we obtained similar results, see Figs. 3a–3f (this time without the trivial "parallel GreConD"), although the difference between the algorithms is not so significant (but still noticeable, from the point of view of the quality of decompositions). GreConDP, for $P = 4$, outperforms GreConD on Mushroom, Paleo and Zoo datasets. Here, the full coverage of data is obtained from both algorithms for the same number of factors but GreConDP gives higher coverage values for small values of k. In particular, this is quite noticeable on the Mushroom dataset, see Fig. 3d. On Emea, Fig. 3a, GreConDP produces better results than GreConD from the DBP point of view but from the AFP point of view it is outperformed. 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, see Table 1, 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 observed a similar behavior also on Firewall 1 and Domino datasets. 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 the decompositions produced by both algorithms are in this respect optimal.

4.3. Similarity of decompositions

As we saw in Section 3.2, the GreConDP algorithm produces P optimal decompositions 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 decompositions similar to each other? Likely for the trivial "parallel GreConD": How much are decompositions produced by the several altered copies of GreConD run in parallel similar to each other? For most of values of P up to 16 we obtained for all datasets from Table 1 similar results like those shown in Table 2 (for GreConDP) and Table 3 (for trivial "parallel GreConD"), both for $P = 8$. The numbers in the table refer to the degree of similarity of two decompositions defined as follows: a decomposition F_i in a row of the table and a decomposition 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 decompositions (i.e., in particular, if one of the decompositions is a permutation of the other one these are measured as equal). Note also that the idea of measuring similarity of decompositions appeared already in [2].

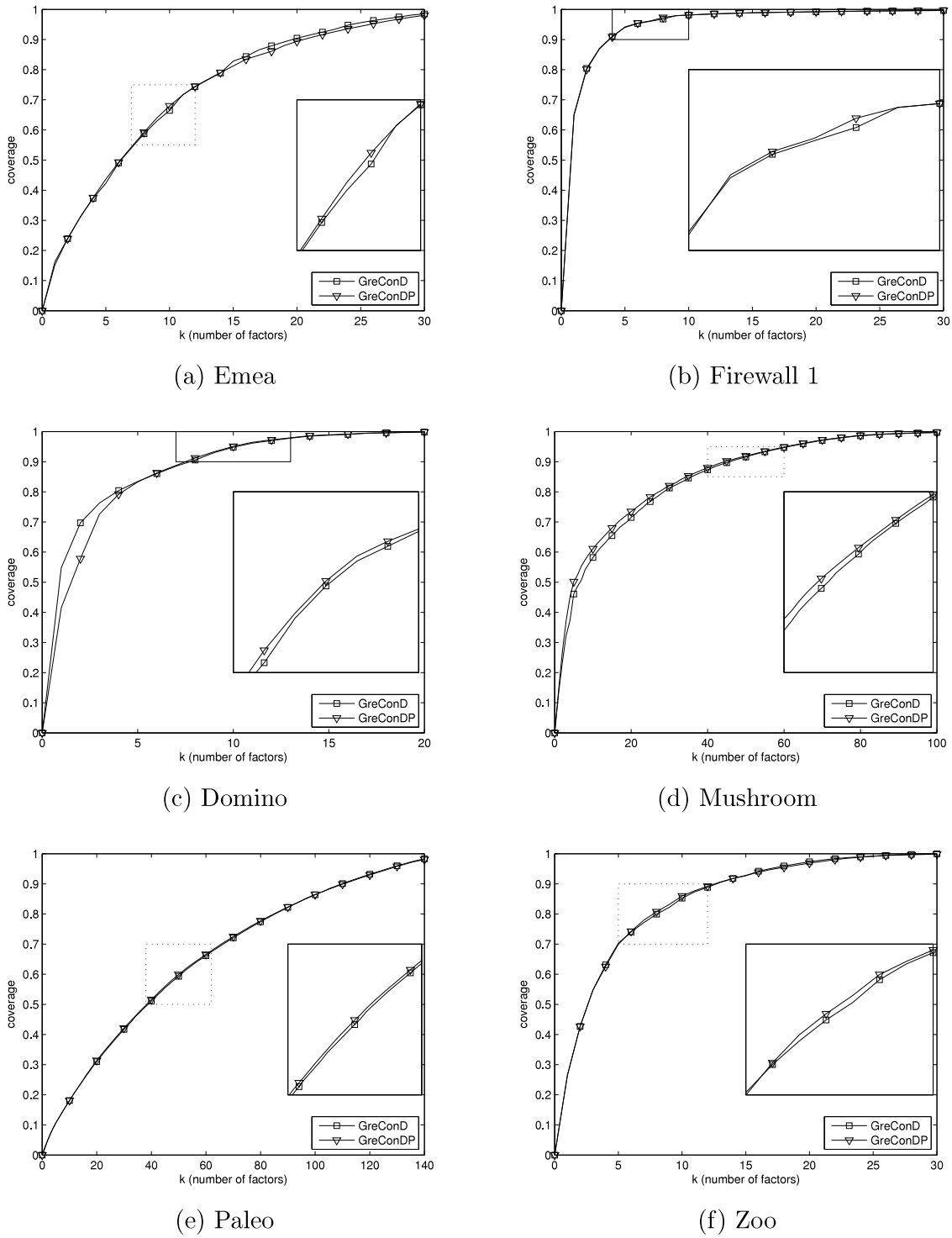


Fig. 3. Comparison of GreConDP with GreConD on real datasets.

We can see that the decompositions delivered 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 of GreConDP remains the same greedy heuristic which is used by GreConD. Factors in that strategy are sought to cover as many of still uncovered 1s in the input Boolean matrix as possible, hence a factor covering many 1s can be by GreConDP taken sooner or later, but it is taken. Contrary to that, decompositions produced in trivial "parallel GreConD" are less similar. The reason here is obvious: the decompositions are constructed independently by altered copies of GreConD run alone, while in GreConDP the decompositions are constructed jointly by concurrently run instances of GreConD and P best decompositions are selected repeatedly in each algorithm iteration.

Table 2
Similarity of the first eight decompositions delivered by GreConDP on Mushroom dataset.

	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 3
Similarity of the first eight decompositions produced in trivial "parallel GreCoND" on Mushroom dataset.

	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8
F_1	1.000	0.720	0.736	0.760	0.816	0.784	0.800	0.728
F_2	0.750	1.000	0.775	0.791	0.783	0.791	0.733	0.933
F_3	0.754	0.762	1.000	0.786	0.754	0.754	0.762	0.770
F_4	0.791	0.791	0.800	1.000	0.850	0.791	0.775	0.775
F_5	0.836	0.770	0.754	0.836	1.000	0.819	0.852	0.770
F_6	0.784	0.760	0.736	0.760	0.800	1.000	0.752	0.744
F_7	0.800	0.704	0.744	0.744	0.832	0.752	1.000	0.696
F_8	0.752	0.925	0.776	0.768	0.776	0.768	0.719	1.000

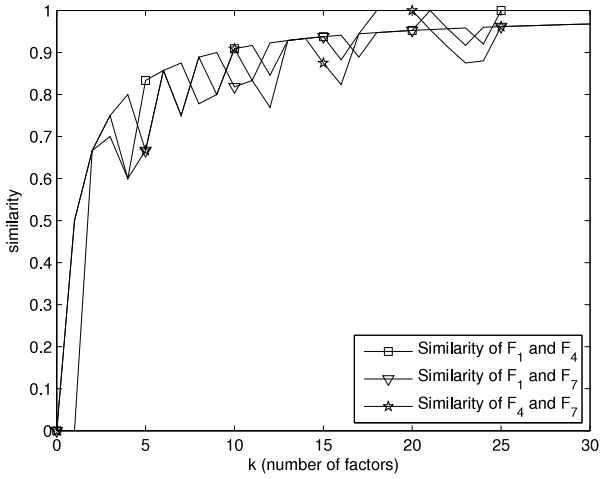


Fig. 4. Progress of similarities of decompositions of Mushroom dataset for the first 30 factors.

In addition, in Fig. 4 we can see the progress of similarities of decompositions F_1, F_4 and F_7 delivered by GreConDP for Mushroom dataset for the number of the first factors going from 1 to 30. As we can see, GreConDP starts with different factors (the decompositions are not similar at all) and with more factors the decompositions become more similar. That means that the algorithm finds the same final decompositions in different ways. Similar results were obtained also for the others decompositions and other datasets.

4.4. Running time

Usually, (theoretical asymptotic) time complexity of BMF algorithms is not a primary concern in the study of the algorithms, see e.g. [3]. Nevertheless, below we provide at least brief remarks on the observed running times of the GreConDP algorithm and trivial "parallel GreCoND" algorithm and compare them to time of the GreConD algorithm. Furthermore, we compare running times of different concurrently run processes in GreConDP.

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 from Table 1 was decomposed by all algorithms, on an ordinary PC, in order of minutes.[2] A slowdown of GreConDP to GreConD depends on the number P of processes run in GreConDP vs. the number of computer 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 and synchronization overhead (barriers and 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. The trivial "parallel GreConD", having no synchronization overhead, lies between GreConD and GreConDP, with a similar slowdown dependency on the number of processes to GreConD. Also, the slowdown of GreConDP is caused by additional required data structures in the implementation.

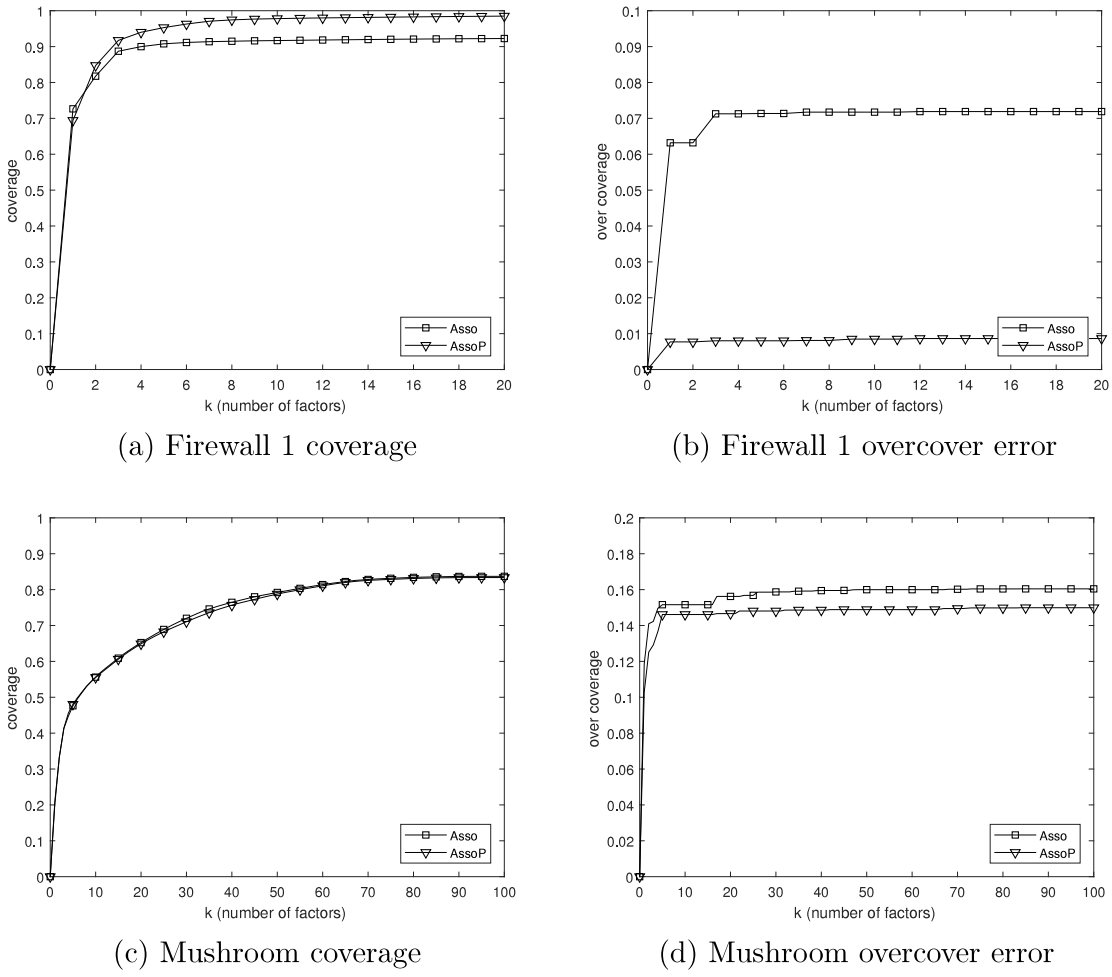
Having the synchronization barriers after each iteration of computing factors by processes run concurrently in GreConDP, where all processes have to wait for the slowest one, we were also interested in differences among running times of the processes in the iterations, i.e. between the barriers. To check the potential bottleneck of the GreConDP algorithm, namely a heavy-tail phenomenon according to which it takes (significantly) longer time to some concurrently running processes to solve a sub-problem (computing optimal factors in our case) than to other processes. In our measurements of computations on both synthetic and real datasets described above we observed only very small differences in the running times, with average variance 0.27%. This means than computing factors in iterations in GreConDP takes to all processes almost the same time.

5. General parallelization scheme

As we stated in Section 1, we do not propose only a parallelization of one particular BMF algorithm (GreConD), we propose a general parallelization scheme for any sequential heuristic BMF algorithm. To support this claim we add a short note on how to apply our scheme to another algorithm, namely the Asso algorithm [11]—a well known and widely used algorithm tailored for the DBP. Unlike the GreConD algorithm, however, Asso produces the so-called overcover error, that means some 0s in input Boolean matrix are covered by factors. We will not explain in details how the original Asso works – see [11] or [12] for thorough descriptions including a pseudocode – we just show the parallelization of it via our scheme, called AssoP. A pseudocode of the parallelization is depicted in Algorithms 4 and 5. Like the parallelization of GreConD, it is based on the same ideas and is quite straightforward as one can compare with pseudocode of the original Asso in [12]. Again, the main difference between AssoP and original Asso is that while Asso computes in each factor search iteration a single factor, based on single the so-called basis vector, AssoP computes (serially) in each process several basis vectors (in lines 2 to 11 in Algorithm 5) and factors in order to construct, in parallel, several final decompositions (in lines 12 to 24).

Selected results from experiments on real datasets are shown in Fig. 5. We can see that AssoP outperforms Asso in terms of both coverage and overcover error but, most importantly, it tends to produce smaller overcover error (see Figs. 5b and 5d). In case of synthetic data we observed similar behavior like in case of the GreConD algorithm, i.e. the coverage is higher and the difference increases with increasing number P of processes. The output decompositions of AssoP are quite similar like in case of GreConDP and we also have very similar observations regarding running times.

[2] An implementation of GreCoND in C optimized for speed decomposes the datasets on an ordinary PC in order of seconds, see e.g. [4].



(a) Firewall 1 coverage (b) Firewall 1 overcover error

(c) Mushroom coverage (d) Mushroom overcover error

Fig. 5. Comparison of AssoP with Asso on real datasets.

Algorithm 4: AssoP – Asso in parallel

Input: A Boolean matrix $I \in \{0,1\}^{n\times m}$, a number of factors $k > 0$, a threshold τ and weights $w+$ and $w-$
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

```
Q ← association matrix m × m
A_r ← empty n × 0 Boolean matrix
B_r ← empty 0 × m Boolean matrix
U_r ← 0 (r ∈ {1, ..., P})
AssoP-I(I, k, τ, w+, w−, Q, A_1, B_1, true)
for r = 1, ..., P do
    with process r
        AssoP-I(I, k, τ, w+, w−, Q, A_r, B_r, false)
    end
end
wait for all processes
return A_1 and B_1
```

6. Conclusions

We presented a general parallelization scheme for (sequential heuristic) Boolean matrix factorization (BMF) algorithms and also two new algorithms, GreConDP and AssoP, utilizing this scheme. The algorithms are based on well established algorithms for BMF, GreConD [4] and Asso [11]. We chose the GreConD algorithm as our base algorithm for presenting the proposed parallelization scheme due to its relative simplicity and high efficiency and the Asso algorithm to show that the scheme can be applied to any other sequential heuristic algorithm for BMF.

We evaluated properties of GreConDP and results delivered by it in various experiments involving synthetic (randomly generated) and real datasets. From presented results we saw that the algorithm outperforms in most cases the base algorithm, GreConD. Most importantly from the standpoint of quality of decomposition (commonly assessed by the coverage quality function measuring how well data are explained by factors) and second, at almost none or moderate computing time expenses (depending on the number of parallel instances vs. the number of available computer processor units). Those 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 are fully explained by considerably less number of factors. For the real datasets, at least for those we used, the data are, 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 the decomposition computation. The last is provided the numbers of equally locally optimal factors in decomposition constructions are not very high (not significantly higher than the number of parallel instances)—only then the utilization of more equally locally optimal factors is beneficial. Moreover, as expected and intended, GreConDP tends to produce better results than GreConD with the increasing number of parallel instances. What was also expected,

Algorithm 5: Procedure Asso-1

Input: A Boolean matrix $I \in \{0,1\}^{n\times m}$, a number of factors $k > 0$, a threshold τ, weights $w+$ and $w-$, an association matrix Q, 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, $r \in \{1,\ldots,P\}$, and a number $P \geq 1$ of processes

for $l = 1,\ldots,k$ **do**
  $D_r \leftarrow$ empty $1 \times m$ Boolean matrix, $e_r \leftarrow$ empty $n \times 1$ Boolean matrix, $V_r \leftarrow 0\,(r \in \{1,\ldots,P\})$
  for $j = 1,\ldots,m$ **do**
    $s \leftarrow P$
    while $s > 0$ and $cover\left(\begin{bmatrix} B_i \\ Q_{j\_} \end{bmatrix}, [A_i\ e], I, w^+, w^-\right) > V_s$ **do**
      if $s > 1$ and $V_{s-1} > 0$ and $Q_{j\_} = D_{s-1}$ **then break**
      if $s < P$ **then** $D_{s+1} \leftarrow D_s, e_{s+1} \leftarrow e_s, V_{s+1} \leftarrow V_s$
      $D_s \leftarrow Q_{j\_}, e_s \leftarrow e, V_s \leftarrow cover\left(\begin{bmatrix} B_i \\ Q_{j\_} \end{bmatrix}, [A_i\ e], I, w^+, w^-\right)$
      $s \leftarrow s - 1$
    end
  end
  $U_i \leftarrow 0$
  synchronization barrier
  for $r = 1,\ldots,P$ **do**
    $s \leftarrow P$
    begin critical section
      while $s > 0$ and $V_r > U_s$ **do**
        if $s < P$ **then** $A_{s+1} \leftarrow A_s, B_{s+1} \leftarrow B_s, U_{s+1} \leftarrow U_s$
        $A_s \leftarrow [A_i\ e_r], B_s \leftarrow \begin{bmatrix} B_i \\ D_r \end{bmatrix}, U_s \leftarrow V_r$
        $s \leftarrow s - 1$
      end
    end
  end
  synchronization barrier
  if *first* $=$ *true* **then break**
end

and has been observed, is that although producing rather similar final decompositions (especially the best ones), the algorithm starts with different factors in the decompositions. That is, the similar final decompositions are constructed in different ways. We observed similar results and behavior also for the second developed and evaluated algorithm, AssoP.

The observed results encourage us to the following future research directions. First, apply the proposed general parallelization scheme to further BMF algorithms. Especially to GreEss [3], which is based on a similar heuristic strategy like GreConD but exploits more theoretical foundations of formal concept analysis [7]. Second, study the properties of the equally locally optimal factors – the problem which every heuristic algorithm faces – in order to find further ways which lead to better factorizations.

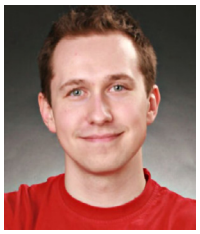
Acknowledgments

This work was supported by the Czech Science Foundation [grant number GA15-17899S]; and the IGA of Palacký University Olomouc [grant number PrF_2016_027]. We thank the anonymous reviewers for their valuable comments.

References

[1] K. Bache, M. Lichman, http://archive.ics.uci.edu/ml, University of California, School of Information and Computer Science, Irvine, CA, 2013.
[2] R. Belohlavek, M. Trnecka, Handling Noise in Boolean Matrix Factorization, in: Proceedings of the 26rd International Joint Conference on Artificial Intelligence (IJCAI-17), 2017, pp. 1433–1439.
[3] R. Belohlavek, M. Trnecka, From-below approximations in boolean matrix factorization: Geometry and new algorithm, J. Comput. System Sci. 81 (8) (2015) 1678–1697.
[4] R. Belohlavek, V. Vychodil, Discovery of optimal factors in binary data via a novel method of matrix decomposition, J. Comput. System Sci. 76 (1) (2010) 3–20.
[5] M.W. Berry, D. Mezher, B. Philippe, A. Sameh, Parallel algorithms for the singular value decomposition, in: E. Kontoghiorghes (Ed.), Handbook on Parallel Computing and Statistics, in: Stat. Textb. Monogr., vol. 184, Chapman & Hall/CRC, 2006, pp. 117–164.
[6] A. Ene, W. Horne, N. Milosavljevic, R. Rao. P, E.R. Tarjan, Fast exact and heuristic methods for role minimization problems, in: ACM SACMAT Proceedings of the 13th ACM Symposium on Access Control Models and Technologies, 2008, pp. 1–10.
[7] B. Ganter, R. Wille, Formal Concept Analysis: Mathematical Foundations, Springer, 1999.
[8] F. Geerts, B. Goethals, T. Mielikäinen, Tiling databases, in: Proc. Discovery Science, 2004, pp. 278–289.
[9] R. Kannan, G. Ballard, H. Park, A high-performance parallel algorithm for nonnegative matrix factorization, in: Proc. 21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'16), 2016.
[10] L. Lucchese, S. Orlando, R. Perego, Mining Top-K Patterns from Binary Datasets in presence of Noise, in: SIAM ICDM International Conference on Data Mining, 2010, pp. 165–176.
[11] P. Miettinen, T. Mielikäinen, A. Gionis, G. Das, H. Mannila, The discrete basis problem, IEEE Trans. Knowl. Data Eng. 20 (10) (2008) 1348–1362.
[12] J. Outrata, M. Trnecka, Evaluating association rules in boolean matrix factorization, in: Proc. 16th ITAT Conference Information Technologies–Applications and Theory (ITAT 2016), Workshop on Computational Intelligence and Data Mining (WCIDM 2016), 2016.
[13] L. Stockmeyer, The set basis problem is NP-complete, Tech. Rep. RC5431, IBM, Yorktown Heights, NY, USA, 1975.
[14] N. Tatti, T. Mielikainen, A. Gionis, H. Mannila, What is the dimension of your binary data? in: IEEE ICDM International Conference on Data Mining, 2006, pp. 603–612.
[15] Y. Xiang, R. Jin, D. Fuhry, F.F. Dragan, Summarizing transactional databases with overlapped hyperrectangles, Data Min. Knowl. Discov. 23 (2) (2011) 215–251.
[16] Y. Xue, S. Ermon, C.P. Gomes, B. Selman, Uncovering hidden structure through parallel problem decomposition for the set basis problem: application to materials discovery, in: Proceeding IJCAI 2015, 2015, pp. 146–154.

Jan Outrata is an associate professor of Computer Science at the Department of Computer Science, Faculty of Science, Palacký University Olomouc (Czech Republic). He obtained his M.Sc. in Computer Science in 2003 and Ph.D. in Mathematics in 2006, both from Palacký University Olomouc. His research interests include formal concept analysis and relational data analysis, classification and fuzzy relational systems. He has authored or co-authored over 30 papers in these areas in conference proceedings and journals including IEEE Trans. Fuzzy Systems, J. Computer and System Sciences, Int. J. General Systems, or Int. J. Uncertainty, Fuzziness and Knowledge-Based Systems. He is a member of the Steering Committee of the Int. Conf. on Concept Lattices and Their Applications (CLA).



Martin Trnecka is an assistant professor of Computer Science at the Department of Computer Science, Faculty of Science, Palacký University Olomouc (Czech Republic). He obtained his Ph.D. in Computer Science in 2017 from Palacký University Olomouc. His research interests are in data analysis, particularly in Boolean matrix decomposition, formal concept analysis and fuzzy relational equations. He published several papers in journals and conferences, including J. of Computer and System Sciences, Annals of Mathematics and Artificial Intelligence and IJCAI. He is a member of ACM.