The 8M Algorithm from Today's Perspective

RADIM BELOHLAVEK and MARTIN TRNECKA, Palacký University Olomouc

We provide a detailed analysis and a first complete description of 8M—an old but virtually unknown algorithm for Boolean matrix factorization. Even though the algorithm uses a rather limited insight into the factorization problem from today's perspective, we demonstrate that its performance is reasonably good compared to the currently available algorithms. Our analysis reveals that this is due to certain concepts employed by 8M that are not exploited by the current algorithms. We discuss the prospect of these concepts, utilize them to improve two well-known current factorization algorithms, and, furthermore, propose an improvement of 8M itself, which significantly enhances the performance of the original 8M. Our findings are illustrated by experimental evaluation.

CCS Concepts: • **Computing methodologies** → **Factorization methods**; • **Theory of computation** → Design and analysis of algorithms; • **Information systems** → Information extraction;

Additional Key Words and Phrases: Boolean matrix factorization, algorithms

ACM Reference format:
Radim Belohlavek and Martin Trnecka. 2020. The 8M Algorithm from Today's Perspective. *ACM Trans. Knowl. Discov. Data* 15, 2, Article 22 (December 2020), 23 pages.
https://doi.org/10.1145/3428078

1 INTRODUCTION

1.1 Content in Brief

Boolean matrix factorization (BMF), or decomposition, has been subject of extensive research during the recent past. Even though the first applications and fundamental results regarding the complexity of factorization appeared already in the 1970s [21, 23], a more comprehensive exploration of BMF, which led to development of various new methods for analysis and processing of Boolean data and improved our understanding of Boolean data as regards foundational aspects, has only been performed during the past decade or so. Most of the respective contributions have been devoted to the design of new factorization algorithms. To name some of the best-known currently available factorization algorithms, let us recall Tiling [11], the nowadays classic Asso [18], GreConD [6], Hyper [25], PaNDa [15], GreEss [4], as well as various modifications of these algorithms and variants of the factorization problems discussed in the above-mentioned papers as well as in, e.g., [3, 12, 14, 16, 17, 19, 24].

This study is supported by the grant JG 2020 of Palacký University Olomouc, No. JG_2020_003. R. Belohlavek acknowledges support by grants IGA 2019 of Palacký University Olomouc, No. IGA_PrF_2019_034 and IGA 2020, No. IGA_PrF_2020_019.
Authors' addresses: R. Belohlavek and M. Trnecka, Department of Computer Science, Faculty of Science, Palacký University Olomouc, 17. listopadu 12, 771 46 Olomouc, Czech Republic; emails: radim.belohlavek@acm.org, martin.trnecka@gmail.com.
Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2020 Association for Computing Machinery.
1556-4681/2020/12-ART22 $15.00
https://doi.org/10.1145/3428078

In view of the extensive research on algorithms, it comes as a surprising fact that there exists an old BMF algorithm, namely, the 8M algorithm, which is virtually unknown in the present research on BMF. This fact is remarkable particularly in view of our experimental evaluations which demonstrate that the 8M algorithm performs reasonably well even from today's perspective, as well as in view of the ideas upon which 8M is based and which are not exploited by the current BMF algorithms.

We learned about this algorithm from Hana Řezanková who used it in her various works on comparison of various clustering and factorization methods. A list of such papers may be found e.g., in the references of [2]. Even though the performance of 8M may be partially assessed from those works, the principles of 8M have never been discussed in the literature. In addition, only parts of 8M's description are available, namely, in the BMDP manual mentioned below.

The goals of our article are as follows. First, we provide a complete description of the 8M algorithm, including its pseudo-code, and the description of its principles from today's perspective. Second, we propose an improvement of the 8M algorithm, which is based on current knowledge of BMF and which improves the performance of 8M considerably. Third, we utilize one of the principles of 8M, namely, revisiting previously generated factors, to enhance the performance of two standard algorithms, namely, Asso and GreConD. Note at this point that we discussed the 8M algorithm in our recent article [5], in which we were solely interested in one particular property of this algorithm which we exploited in [5].

Our article is organized as follows. Notation and preliminaries are summarized in Section 1.2. Section 2 provides a detailed description of 8M and our comments on its design from today's perspective. Sections 3 describes our proposed improvements to 8M (Section 3.1) and to the standard GreConD and Asso algorithms (Sections 3.2 and 3.3) which are inspired by 8M. Section 4 provides a detailed experimental evaluation of the basic 8M algorithm, its improvement we proposed, as well as the improvements of Asso and GreConD. Section 5 concludes the article and outlines topics to be explored in the future.

1.2 Notation and Preliminaries

By a Boolean matrix we mean a matrix whose entries are either 0 or 1. Such matrices appear in several contexts. Basically, a Boolean matrix represents a relationship between the entities represented by the rows and the entities represented by the columns. A Boolean matrix may equivalently be represented by a binary relation between a set representing the rows and a set representing the columns, or by a bipartite graph between two sets of nodes (again representing rows and columns). In addition to the literature on Boolean matrices themselves, relevant results appear in the literature on binary relations and graph theory. A classic text on Boolean matrices is the book [13]; see also [7] and, for a relationally-oriented view of aspects related to factorization of Boolean matrices, see [10, 22].

We shall denote the particular Boolean matrices by I, J, and the like, and shall denote the set of all $n \times m$ Boolean matrices, i.e., matrices with n rows and m columns, by $\{0,1\}^{n\times m}$. The ith row and jth column vector of I is denoted by $I_{i\_}$ and $I_{\_j}$, respectively.

An input matrix I shall primarily be interpreted as an object-attribute incidence matrix (hence the symbol I): The entry I_{ij} corresponding to the row i and the column j equals 1, i.e., $I_{ij} = 1$, or 0, i.e., $I_{ij} = 0$, depending on whether the object i does or does not have the attribute j, respectively. This is the basic interpretation used in applications of Boolean matrices.

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

$$I \approx A \circ B, \tag{1}$$

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

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

and $\approx$ denotes equality or an appropriate approximate equality.

Note that a decomposition of I into $A \circ B$ may be interpreted as a discovery of k factors that exactly or approximately explain the data. Interpreting I, A, and B as object-attribute, object-factor, and factor-attribute matrices, model (1) reads: The object i has the attribute j if and only if there exists factor l such that l applies to i and j is one of the particular manifestations of l. The least k for which an exact decomposition $I = A \circ B$ exists is the well-known Boolean rank of I. The approximate equality $\approx$ in (1) is commonly assessed in BMF by the metric $E(\cdot, \cdot)$ defined for matrices $C, D \in \{0,1\}^{n \times m}$ by

$$E(C, D) = \sum_{i,j=1}^{m,n} |C_{ij} - D_{ij}|. \tag{2}$$

Note that E is just the metric induced by the matrix L_1-norm.

The above, somewhat ambiguously described goal of BMF to find an approximate decomposition $I \approx A \circ B$ naturally leads to two concrete variants of the BMF problem, which are considered in the literature. First, one prescribes the number k of factors and attempts to find the most precise decomposition using at most k factors. This problem is called the

> ***Discrete Basis Problem*** **(DBP)** [18]: 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 $E(I, A \circ B)$.

The second problem results by prescribing a required precision ε and attempting to find a decomposition involving as few factors as possible which is at least ε-precise. This is called the

> ***Approximate Factorization Problem*** **(AFP)** [6]: Given I 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 $E(I, A \circ B) \leq \varepsilon$.

The DBP and the AFP reflect two important views of BMF. While the DBP emphasizes the importance of the first few (presumably most important) factors, the AFP emphasizes the need to account for (and thus to explain) a prescribed portion of data (prescribed by ε).

The following view of the committed error, $E(I, A \circ B)$, is important for our considerations below: $E(I, A \circ B)$ has two parts, namely,

$$E(I, A \circ B) = E_u(I, A \circ B) + E_o(I, A \circ B), \tag{3}$$

where $E_u(I, A \circ B) = |\{\langle i, j\rangle \mid I_{ij} = 1 \text{ and } (A \circ B)_{ij} = 0\}|$ and $E_o(I, A \circ B) = |\{\langle i, j\rangle \mid I_{ij} = 0 \text{ and } (A \circ B)_{ij} = 1\}|$ are the so-called undercovering error and overcovering error, respectively.

Example 1.1. In our running example, we shall use the following 5×5 Boolean matrix:

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

One easily observes that the matrix decomposes exactly into the product of the following matrices A and B:

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad B = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \end{pmatrix},$$

i.e., $I = A \circ B$. In fact, this decomposition, which uses 4 factors, has been found by the GreConD algorithm mentioned above.

2 DESCRIPTION OF THE 8M ALGORITHM

2.1 History of 8M

The 8M method is one of the many data analysis methods available in an old and widely used statistical software package known as BMDP. The acronym "BMDP" stands for "Bio-Medical Data Package" (some sources say "BioMeDical Package").[1] The package was developed primarily for biomedical applications since the 1960s at the University of California in Los Angeles (UCLA) under the leadership of W. J. Dixon.[2] BMDP was originally available for free, later through BMDP Statistical Software, Inc., and then by its subsidiary, Statistical Solutions Ltd. As of 2017, BMDP is no longer available.[3]

BMDP and its methods are described in several editions of manuals, starting with a 1961 manual of BMD, a direct predecessor of BMDP. In our description of 8M, we use the 1992 edition [8], which accompanies release 7 of BMDP. There, 8M is described in chapter "Boolean factor analysis" on pp. 933–945, written by M.R. Mickey, L. Engelman, and P. Mundle, and in Appendix B.11 on pp. 1401–1403. The name "8M" is just a catalogue code of the method in BMDP (for instance, "4M" denotes classical factor analysis in BMDP).

The 8M method has been added to BMDP in the late 1970s: it was not part of the 1979 manual but it is part along with other new methods in the next version, whose revised printing appeared in 1983. In addition to 8M, the 1983 manual contains 41 other methods of BMDP. According to this edition, 8M is based on research done by the statistician M. Ray Mickey of the UCLA, was designed by Mickey with contributions from Laszlo Engelman, and was programmed by Peter Mundle and Engelman.[4]

2.2 Description of 8M

The description of 8M in the manual [8] is reasonably detailed. Nevertheless, the description is aimed at the users of the method. It is far from offering a possibility to implement the method, does not provide a comprehensible rationale of the method, and some parts of the algorithm are missing in the description. As to the procedural details, we therefore examined the step-by-step program behavior on various data to figure out the unclear parts until our own implementation of 8M yielded the same results as the implementation of 8M in the BMDP software, which we

[1]The package grew out from an older computer program BIMED, which was developed for biomedical applications, and was first called BMD. Since the implemented methods allowed a parameterized format, the letter "P" was added. Later, "P" was interpreted as standing for "Package."

[2]Wilfrid Joseph Dixon (1915–2008) was an American mathematician and statistician who made notable contributions to nonparametric statistics.

[3]We crosschecked our implementation against the version of BMDP we purchased in 2015 from Statistical Solutions Ltd.

[4]The references of the BMDP manual include some papers by Mickey but none of them concerns 8M and Boolean factor analysis.

purchased from from Statistical Solutions Ltd. in 2015. As to the rationale of 8M, we provide our explanation of the particular steps of 8M below.

Basic Idea. The basic idea of 8M may be described as follows. The algorithm takes as its input four parameters: An $n \times m$ Boolean matrix I (the object-attribute matrix to be decomposed), a number k of desired factors, and two auxiliary parameters, a number *init* of initial factors, and a number *cost* used to refine the factors being computed. The desired output consists of $n \times k$ and $k \times m$ Boolean matrices A (object-factor matrix) and B (factor-attribute matrix).

The algorithm starts by computing the initial factors, the number of the initial factors is *init*. Then the algorithm iteratively computes new factors until k desired factors are obtained. The way 8M computes the factors is very different from the current BMF algorithms in two respects. The first consists in the way the new factors are generated. The second, more significant difference consists in the fact that the previously generated factors are revisited and some of them are possibly dropped. The corresponding procedures are described in detail below.

Remark 2.1. 8M's revisiting of the previously generated factors is an interesting property. Even though the way 8M revisits the factors is rather straightforward and—as we shall see in Section 4—may considerably be improved, it directly relates to a topic of fundamental importance that is not properly addressed in the existing papers, namely the behavior of the parts E_u and E_o of the error function E; see (3).

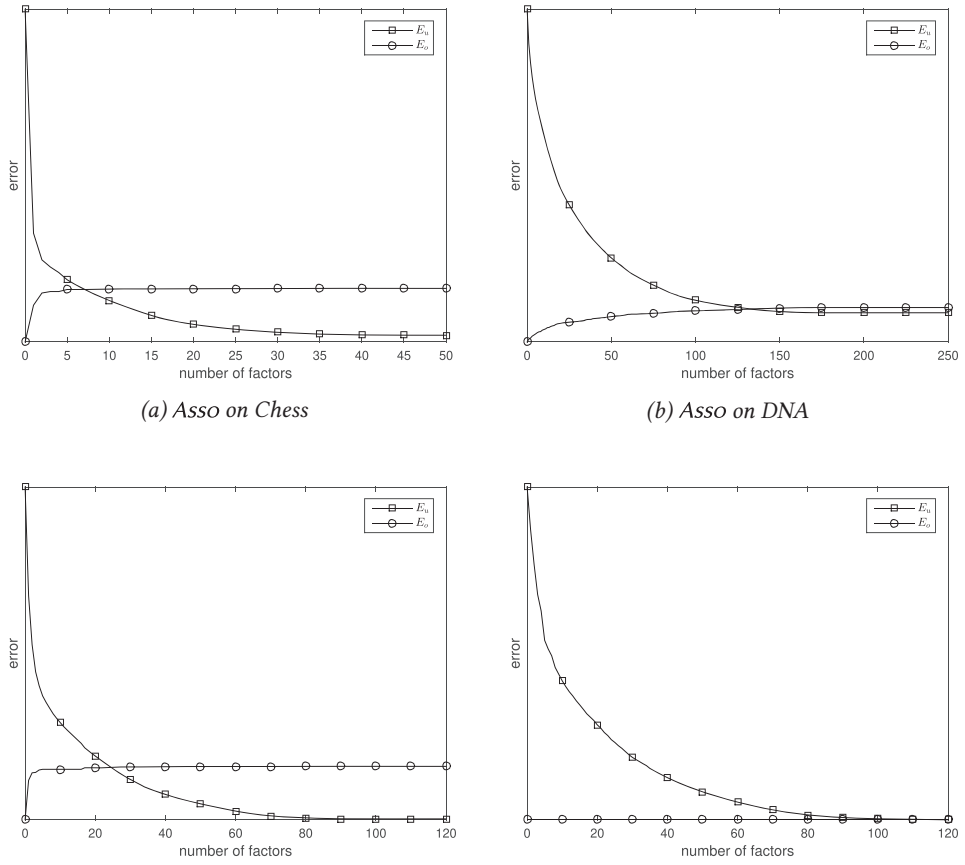
In particular, while the undercovering and overcovering error, E_u and E_o, seem symmetric, they are not symmetric from the viewpoint of BMF algorithm design. Namely, due to the NP-hardness of the various variants of the decomposition problem [23], most of the current factorization algorithms are heuristic approximation algorithms computing the factors one-by-one until a satisfactory factorization is obtained.

It is immediate from the definitions that E_u is nonincreasing and E_o is nondecreasing in that as new factors are successively computed by a given algorithm, E_u may only decrease or have the same value and E_o may only increase or have the same value. Now, by computing a certain number, say k, factors, an algorithm commits a certain undercover error E_u and certain overcover error E_o. It thus follows from the properties just mentioned that while committing the E_u error may be repaired by adding further factors, committing the E_o error will never be repaired by adding further factors and must thus be carefully considered. Parts (a), (b), and (c) of Figure 1 illustrate this property: They display graphs of E_u and E_o in dependence on the number of factors successively generated by the classic Asso algorithm [18] when factorizing the Chess, DNA, and Mushroom data (see Section 4). Notice also part (d) of Figure 1 which displays such a graph for GreConD. As is well known, GreConD along with several other algorithms does not commit E_o at all due to its design. As we show below in Section 3.3, revisiting and dropping factors makes sense even in this case.

It is clear from this perspective that revisiting and possibly dropping previously generated factors is a natural procedure to cope with the problem of nondecreasing E_o error. It is therefore interesting that while the current algorithms do not use any kind of revisiting, the old 8M already used this idea.

Detailed Description and Pseudocode of 8M *(algorithm 1).* To compute $n \times k$ and $k \times m$ Boolean matrices A and B given the input $n \times m$ Boolean matrix I, the prescribed number *init* of initial factors, the desired number k of factors, and the parameter *cost*, the algorithm 8M (algorithm 1) proceeds as follows.

Note first that the computed factors are represented by the Boolean matrices A and B in the following way, which is standard in BMF: every factor, say factor l, is represented by the column l



(a) Asso on Chess *(b) Asso on DNA*

(c) Asso on Mushroom *(d) GreConD on Mushroom*

Fig. 1. Typical behavior of E_u and E_o.

of matrix A and the row l of matrix B. As new factors are computed and added to matrices A and B, the matrices A and B grow in their number of columns and rows, respectively. At the end, after the algorithm finishes its computation of the final k factors, the algorithm therefore outputs an $n \times k$ matrix A and a $k \times m$ matrix B.

To start the process of computing and adding new factors, the algorithm needs to initialize the matrices A and B. This is done by computing *init* initial factors. These factors are used to initialize A and B on l. 1 as follows: A is set to the $n \times init$ matrix $\mathbf{0}_{n \times init}$ full of 0s; and B is set to a $init \times m$ matrix computed by the procedure ComputeInitialFactors, which is described below. Note at this point that by default, $init = k - 2$, but in general, *init* is set by the user. The variable f storing the number of the currently computed factors is set accordingly (l. 2). The matrices A and B are then refined (l. 3) by the procedure RefineMatricesAB described below. The algorithm then enters a loop (l. 5–17) whose purpose is to add new factors and remove some of the previously generated ones until the desired number k of factors is reached for the second time or all 1s in I are covered by $A \circ B$, i.e., we have $I_{ij} \leq (A \circ B)_{ij}$ for all i, j (l. 5).

A new factor is computed and added to the previously computed factors in l. 6–8 of 8M by computing first the positive part Δ^+ of the discrepancy matrix $\Delta = I - A \circ B$: One adds to A as

ALGORITHM 1: 8M

Input: Boolean $n \times m$ matrix I, desired number of factors k, number *init* of initial factors, number *cost*
Output: Boolean matrices A and B

```
1  B ← ComputeInitialFactors(init); A ← 0_{n×init}
2  f ← init
3  RefineMatricesAB(A, B, I, cost)
4  kReached ← 0
5  while kReached < 2 or I ≤ A ∘ B do
6      foreach ⟨i, j⟩ do if I_ij > (A ∘ B)_ij then Δ+_ij ← 1 else Δ+_ij ← 0
7      add column j of Δ+ with the largest count of 1s as new column to A
8      add row of 0s as new row to B and set entry j of this row to 1
9      f ← f + 1
10     RefineMatricesAB(A, B, I, cost)
11     if another two new factors were added then
12         remove column A_{_(f−2)} from A and row B_{(f−2)_} from B
13         f ← f − 1
14         RefineMatricesAB(A, B, I, cost)
15     end
16     if f = k then kReached ← kReached + 1
17 end
18 return A, B
```

new column the column j of Δ^+ containing the largest number of 1s, and adds to B a row of 0s with 1 at position j in this row. The variable f is then incremented in l. 9.

The addition of new factor described in the previous paragraph ensures that all 1s in the column j of I that are not covered by the factors computed so far get covered by the new factor; since the column j contains the largest number of such uncovered 1s, the selection of a new factor represents a greedy strategy.

Whenever a new factor is added or a previously computed factor is removed, A and B are refined (in l. 10 after addition, in l. 14 after removal). Adding and removing factors is performed according to the following scheme. One starts with $f = \mathit{init}$ factors, adds two factors so that $f + 2$ factors are obtained, then removes the factor generated two steps back, i.e., the fth factor, adds another two factors, removes a factor generated two steps back, and so on. Hence, starting with $\mathit{init} = 2$ factors, one successively obtains

$$2, 3, 4, 3, 4, 5, 4, 5, 6, 5, 6, 7, 6, 7, 8, \ldots$$

factors. One stops when the desired number k of factors is obtained the second time. For instance, if $k = 6$ one computes the sequence of

$$2, 3, 4, 3, 4, 5, 4, 5, 6, 5, 6$$

factors, and the last six factors are the final factors output by the algorithm (provided the algorithm does not stop due to the second condition in l. 5).

The initial factors are computed by ComputeInitialFactors (Algorithm 2) as follows. First, an $m \times m$ matrix C is computed in which $C_{ij} = 1$ iff column i is included in column j in I (i.e., $I_{qi} \leq I_{qj}$ for each q) and column i is not empty (i.e., contains at least one 1). Second, one removes duplicate rows from C: If C contains several identical rows, one keeps just the first of them and removes the others. One then goes through the rows i of C, $i = 1, 2, \ldots$, and adds them as new rows of B

ALGORITHM 2: ComputeInitialFactors

Input: $n \times m$ Boolean matrix I and the number ofinitial factors *init*
Output: *init* $\times m$ Boolean matrix B

```
1  C ← m × m Boolean matrix with all entries equal to 0
2  foreach C_ij do
3  |  if I_i ≤ I_j and |I_i| > 0 then
4  |  |  C_ij ← 1
5  |  end
6  end
7  remove all duplicate and empty rows from C
8  f ← 0
9  foreach row i ∈ 1, . . . , m of matrix C do
10 |  if C_ki = 0 for all k ≠ i then
11 |  |  f ← f + 1
12 |  |  add row C_i_ as a new row to B
13 |  end
14 |  if f = init then
15 |  |  return B
16 |  end
17 end
```

until *init* rows have been added: row i of C is added to B provided no other row of C contains 1 at position i.

Remark 2.2. Initialization of the factors is an important step in 8M in that the quality of the computed factorization depends on it. As we shall see in Section 4.1, the original initialization of 8M may significantly be improved. A new initialization is part of our improvement of 8M.

Let us point out an interesting observation. Computing the association matrix in the classic Asso algorithm [18] is a kind of initialization. In particular, the vectors of the association matrix serve as the candidate B-parts of factors. Now, it is easy to observe that to select the rows of the association matrix, Asso uses basically the same strategy as 8M, except that its strategy is more general in the following sense (we disregard the technical difference in how 8M and Asso handle empty columns): Where 8M tests inclusion of columns i and j (l. 3 of Algorithm 2), Asso tests partial inclusion rather than (full) inclusion. More precisely, Asso tests whether the degree of partial inclusion of column i in column j, defined by

$$incl(i,j) = \frac{|\{k \,;\, I_{ki} = 1 \text{ and } I_{kj} = 1\}|}{|\{k \mid I_{ki} = 1\}|}, \tag{4}$$

is greater than or equal to a user-specified threshold τ (equivalently, in terms of association rules, whether the confidence of the association rule $\{i\} \Rightarrow \{j\}$ is $\geq \tau$). For $\tau = 1$, partial inclusion becomes (full) inclusion, hence setting $\tau = 1$ would yield the same vectors in the association matrix of Asso as what 8M uses as the rows in matrix C, i.e., as the B-parts of potential initial factors. We demonstrate in Section 4.1 that the more general initialization strategy using partial inclusion yields better results (both for Asso and 8M).

Refining of A and B by RefineMatricesAB (Algorithm 3) consists in performing a cycle until A and B do not change with the additional condition that the cycle is run at most three times. In each cycle, A is computed from I, B, and the parameter *cost* by a so-called Boolean regression

ALGORITHM 3: RefineMatricesAB

Input: Boolean matrices A, B, I, number *cost*

1 **repeat**
2 RefineMatrixA($A, B, I, cost$)
3 RefineMatrixB($A, B, I, cost$)
4 **until** *loop executed 3 times or A and B did not change*

described in RefineMatrixA (Algorithm 4), followed by computing symmetrically B from I, A, and *cost* using RefineMatrixB (Algorithm 5).

We now describe the so-called Boolean regression employed in 8M, and explain how this procedure is implemented in RefineMatrixA (Algorithm 4); RefineMatrixB is symmetric. The essential idea of Boolean regression consists in that given matrices I and B, and the parameter *cost*, one determines the matrix A in such a way that a desired measure of quality gets maximized. For this measure, one considers

$$\sum_{i=1,j=1}^{n,m} M_{ij} = \sum_{i=1,j=1}^{n,m} \left(I_{ij} \max_{l=1}^{f} A_{il} \cdot B_{lj} - cost \cdot (1 - I_{ij}) \cdot \max_{l=1}^{f} A_{il} \cdot B_{lj} \right).$$

Since $\max_{l=1}^{f} A_{il} \cdot B_{lj} = 1$ iff $(A \circ B)_{ij} = 1$, one easily observes that $\sum_{i=1,j=1}^{n,m} M_{ij}$ is the number of entries $\langle i, j \rangle$ for which $I_{ij} = 1$ and $(A \circ B)_{ij} = 1$ (the factorization model, represented by A and B, makes a correct prediction of existing incidence $I_{ij} = 1$) minus *cost* times the number of entries $\langle i, j \rangle$ for which $I_{ij} = 0$ and $(A \circ B)_{ij} = 1$ (the model makes a wrong prediction of non-existing incidence $I_{ij} = 0$). To determine A, one proceeds row by row (l. 1 in RefineMatrixA). Notice that since B and I are fixed, one attempts to set the row i of A so that the measure $\sum_{i=1,j=1}^{n,m} M_{ij}$, or equivalently—since only row i is considered at this point—that the measure $M_i = \sum_{j=1}^{m} M_{ij}$ is maximized. Appropriate initializations are made in l. 2: y is set to the ith row of I, Z is set to B, and all entries in the ith row $A_{i\_}$ of A are set to 0. In the loop in l. 3–16, one attempts to select the best entry p in the row $A_{i\_}$ to be set to 1, i.e., $A_{ip} = 1$ (l. 9), until no such setting brings improvement in M_i (l. 16). Note that m_p denotes the best value M_i possible by such setting and that the candidate values m_l are obtained in l. 5 according to the rationale described above. After p is determined and A_{ip} set to 1, one updates Z and y in the loop in l. 10–14 to make possible the choice of the next best entry (keeping Z and y would prevent us from finding another entry p).

Notice that this form of regression may be modified according to one's preference regarding the significance of the four possible possibilities of prediction by the model, namely $I_{ij} = 0$ or $I_{ij} = 1$ and $(A \circ B)_{ij} = 0$ and $(A \circ B)_{ij} = 1$.

2.3 Illustrative Example

Consider again the matrix I in Example 1.1. Suppose we run 8M with I as the matrix to be decomposed with the requirement that the number of desired factors be $k = 4$, the number of initial factors be $init = 2$, and the parameter $cost = 1$.

To compute the initial factors, the algorithm first computes the auxiliary matrix C, which in this case turns to be

$$C = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

ALGORITHM 4: RefineMatrixA

Input: Boolean matrices A, B, I and *cost*
Output: Refined matrix A

foreach *row* $i \in \{1, \ldots, n\}$ **do**
  $y \leftarrow I_{i\_}$; $Z \leftarrow B$; $A_{i\_} \leftarrow \mathbf{0}$
  repeat
    foreach *factor* $l \in 1, \ldots, f$ **do**
      $m_l \leftarrow \sum_{j=1}^{m} y_j \cdot Z_{lj} - cost \cdot \sum_{j=1}^{m} (1 - y_j) \cdot Z_{lj}$
    end
    select p for which $m_p = \max_l m_l$
    if $m_p > 0$ **then**
      $A_{ip} \leftarrow 1$
      foreach $j \in \{1, \ldots, m\}$ **do**
        if $Z_{pj} = 1$ **then**
          $Z_{\_j} \leftarrow \mathbf{0}$; $y_j \leftarrow 0$
        end
      end
    end
  until $m_p \leq 0$
end

ALGORITHM 5: RefineMatrixB

Input: Boolean matrices A, B, I and *cost*
Output: Refined matrix B

foreach *column* $i \in \{1, \ldots, m\}$ **do**
  $y \leftarrow I_{\_i}$; $Z \leftarrow A$; $B_{\_i} \leftarrow 0$
  repeat
    foreach *factor* $l \in 1, \ldots, f$ **do**
      $m_l \leftarrow \sum_{j=1}^{n} y_j \cdot Z_{jl} - cost \cdot \sum_{j=1}^{n} (1 - y_j) \cdot Z_{jl}$
    end
    select p for which $m_p = \max_l m_l$
    if $m_p > 0$ **then**
      $B_{pi} \leftarrow 1$
      foreach $j \in \{1, \ldots, n\}$ **do**
        if $Z_{jp} = 1$ **then**
          $Z_{j\_} \leftarrow 0$; $y_{j\_} \leftarrow 0$
        end
      end
    end
  until $m_p \leq 0$
end

Recall that $C_{ij} = 1$ iff column i is included in column j in the input matrix I. We therefore have, e.g., $C_{31} = 1$, because the third column in I is included in the first. Since C does not contain any duplicate rows or empty rows, the algorithm proceeds by computing the matrix B representing the initial factors, and obtains

$$B = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}.$$

Note that B consists of the third and the fourth rows of C; the first two rows of C are not selected as rows of B because the entries j with 1 in these rows contain 1 in the third row as well.

Given this matrix B and the matrix A full of zeros (l. 1 in Algorithm 1), one proceeds with refining matrices A and B in l. 3. This proceeds in an iterative manner by computing first the matrix

$$A = \begin{pmatrix} 0 & 1 \\ 1 & 0 \\ 0 & 1 \\ 1 & 1 \\ 0 & 0 \end{pmatrix},$$

then computing

$$B = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix},$$

followed by

$$A = \begin{pmatrix} 0 & 1 \\ 1 & 0 \\ 0 & 0 \\ 1 & 1 \\ 0 & 0 \end{pmatrix},$$

and

$$B = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix},$$

after which the refinement ends. A new factor is added to A and B based on Δ^+, resulting in

$$A = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \quad B = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

A refinement of these matrices results in

$$A = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \quad B = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{pmatrix}.$$

Adding a new factor one obtains

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad B = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix}.$$

At this point, $f = 4$, hence the required number of factors is reached for the first time. Subsequently, the second factor (namely, $2 = f - 2$ at this point) is removed from A and B and the matrices are refined again, resulting in

$$A = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}, \quad B = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 \end{pmatrix}.$$

A new factor is added and refinement is performed, resulting in

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \text{and} \quad B = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \end{pmatrix}.$$

Since the algorithm reached the required number of factors for the second time, it ends. Notice that in this case, 8M obtained an exact decomposition, i.e., $I = A \circ B$.

3 IMPROVEMENTS OF ALGORITHMS BASED ON 8M

3.1 Improving 8M

Initialization. Our first improvement of the 8M algorithm consists in replacing the 8M's computation of initial factors in l. 1 of Algorithm 1. As mentioned in remark 2.2, the matrix C computed as part of the initialization in 8M may be regarded as a particular case of what Asso uses as its association matrix, namely, a case with the user-specified parameter τ equals 1. Our experience with Asso shows that setting $\tau = 1$, i.e., refraining to full subsethood in computing associations of attributes, does not yield particularly good results compared to when τ is set to lower values such as around 0.8. A similar trend is observed when the matrix C in 8M's original initialization is computed using partial inclusion in dependence on a parameter τ, i.e., when the entry C_{ij} is set according to

$$C_{ij} = \begin{cases} 1 & \text{if } incl(i,j) \geq \tau \\ 0 & \text{if } incl(i,j) < \tau \end{cases}$$

where $incl(i,j)$ is the degree of partial inclusion of the column i in the column j of the input matrix I defined by (4). This behavior is illustrated in Section 4.1.

Nevertheless, instead of computing the *init* initial factors by the procedure ComputeInitialFactors, as done by the original 8M, or the above-described improved version via partial inclusion, one may compute the first *init* initial factors by the simple and efficient GreConD algorithm, which results in an even better factorization. This enhanced version of 8M shall be denoted 8M+ here and in Section 4, in which we provide experimental comparison of 8M and 8M+. The initial factors are computed by GreConD in time that does not impair the overall computation time of the resulting algorithm considerably. Importantly, since the first factors computed by GreConD are generally the most important ones (of all the factors computed by GreConD) due to the design of GreConD, and since these factors appear to be good factors in terms of their coverage of the input data (see e.g., [4, 6]), using these factors as initial factors provide a better choice compared to the initialization of the original 8M, which is somewhat

simplistic. Another advantageous property of the initial factors obtained by GreConD consists in the fact that these factors commit no overcovering error E_o; see (3).

Revisiting previously generated factors. In Section 4.4, we moreover examine yet another modification of 8M, which consists in employing a different strategy of dropping the previously generated factors. This modification is inspired by the fact that the strategy used by 8M appears somewhat rigid (after computing two new factors, one factor must always be removed) and, moreover, 8M completely ignores the quality of the factor to be removed and its relationship to the other computed factors (when a factor is to be removed, it is always the one computed two steps back).

3.2 Improving Asso

As was mentioned above, the idea of revisiting and possibly dropping the previously generated factors, which is utilized in a particular way by 8M, is not employed by the current BMF algorithms. This idea appears rather natural in view of remark 2.1, particularly for algorithms which generate factors one-by-one and commit the overcovering error E_o, such as Asso: Since E_o never decreases by adding further factors, it makes sense to inspect, at every moment a new factor is generated, previous factors and remove one whose significance is small in view of all the factors generated up to the moment. In the worst case, there may exist a previously generated factor which contributes to the overcovering error by a large extent and which does not contribute to a decrease of the undercover error E_u at all (the latter occurs when all the ones in the factorized matrix I are covered by the other factors). Such a bad factor is a candidate for being removed from the set of the factors computed so far.

The pseudocode of the modified Asso algorithm is detailed in Algorithm 6 along with the auxiliary Algorithm 7. Since the algorithm is generally known, see e.g., [18], we restrict to noting that the procedure of revisiting and possibly dropping the computed factors is included in l. 19–27. Note that a factor l is removed if the condition on l. 25 is satisfied. This requires that the increase in error, $E(I, X \circ Y) - E(I, S \circ B)$, which results by dropping the factor, does not exceed $p \cdot |C|$, i.e., does not exceed the proportion of the number $|C|$ of 1s in the input matrix C given by the parameter p. Note also that as usual in considerations on Asso, the input $n \times m$ Boolean matrix is denoted C, the association matrix by A, and the matrices to which C is to be decomposed by S and B, and that by $\neg$ we denote the binary complement, i.e., $\neg 0 = 1$ and $\neg 1 = 0$.

3.3 Improving GreConD

A similar modification to the one described in the previous section for Asso may be applied to the GreConD algorithm. Again, since GreConD is described and widely used in the literature, see e.g., [4], we restrict our description of the modified algorithm (Algorithm 8 along with Algorithm 9). The modification consists in adding in l. 15–23 a loop in which the previously generated factors are revisited and those satisfying the condition on l. 20 are removed, according to an analogous logic used for Asso in the previous section. A difference compared to our modification of Asso consists in that the factors are revisited only when all of them are computed, rather then after computation of every particular factor. Revisiting the factors each time a new factor would also be possible but as it yields comparable results, we do not discuss it further.

4 EXPERIMENTAL EVALUATION

Our evaluation involves the real-world datasets Apj [9], DNA [20], Emea [9], Chess [1], Firewall 1 [9], Firewall 2 [9], Mushroom [1], and Paleo[5]. These datasets are well known and commonly used

[5]NOWpublic release 030717, available from http://www.helsinki.fi/science/now/.

ALGORITHM 6: Modified Asso algorithm

Input: $n \times m$ Boolean matrix C, a desired number of factors k, a threshold value $\tau \in (0, 1]$, real-valued weights w^+, w^- and a parameter ϵ
Output: Boolean matrices S and B

```
for i = 1, ..., m do
    for j = 1, ..., m do
        if confidence of association rule {i} ⇒ {j} is ≥ τ then
            A_ij = 1
        else
            A_ij = 0
        end
    end
end
S ← empty matrix
B ← empty matrix
for l = 1, ..., k do
    foreach row A_l_ in A do
        ⟨c_l, e_l⟩ ← Cover(A_l_, C, S, B, w+, w−)
    end
    select i for which c_i is maximal
    add column e_i as new column to S
    add row A_i_ as new row to B
    foreach factor j = l, ..., 1 do
        X ← S
        Y ← B
        remove column X_j from X
        remove row Y_j_ from Y
        if E(I, X ∘ Y) − E(I, S ∘ B) ≤ p · |C| then
            remove column S_j from S and row B_j_ from B
        end
    end
end
return S and B
```

in the literature on BMF. Their characteristics are provided in Table 1. Note that in Table 1, size refers to the number of objects × number of attributes and that density is the percentage of the entries with 1 of the dataset, i.e., the ratio $\frac{|I|}{n \cdot m}$, in which

$$|I| = |\{\langle i, j\rangle \mid I_{ij} = 1\}| \tag{5}$$

is the number of 1s in the matrix I. Moreover, we used two collections, $X1$ and $X2$, of synthetic datasets. Each collection includes 1,000 randomly generated matrices obtained as Boolean products $A \circ B$ of $1,000 \times 40$ and 40×500 matrices A and B which are randomly generated. The average densities of datasets included in $X1$ is 0.15. In the case of $X2$, the average densities are 0.2.

We use the algorithms mentioned in Section 1, namely, Tiling, Asso, GreConD, Hyper, and PaNDa, and we compare them with the original 8M algorithm.

To assess the quality of the algorithms, we use the commonly employed coverage c of the input data I by the first l computed factors, i.e., by the $n \times l$ and $l \times m$ matrices A and B, defined by

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

ALGORITHM 7: Cover function of Asso algorithm

Input: A candidate row *row*, Boolean matrices C, S, B, $w+$, w^-
Output: A pair $\langle cover, s\rangle$ where *cover* is the value of cover function for candidate *row* and column s computed for the candidate

$uncovered \leftarrow \mathbf{1}_{n\times m}$
$s \leftarrow \mathbf{0}_{n\times 1}$
foreach $\langle i, j\rangle$ **do**
  if $(S \circ B)_{ij} = 1$ **then**
    $uncovered_{ij} \leftarrow 0$
  end
end
foreach *row* $C_{i\_}$ *in* C **do**
  $cost_i = w^+ \sum_{j=1}^{m} row_j \cdot uncovered_{ij} - w^- \sum_{j=1}^{m} \neg row_j \cdot uncovered_{ij}$
  if $cost_i < 0$ **then**
    $cost_i \leftarrow 0$
  else
    $s_j \leftarrow 1$
  end
end
$cover = \sum_{i=1}^{n} cost_i$
return $\langle cover, s\rangle$

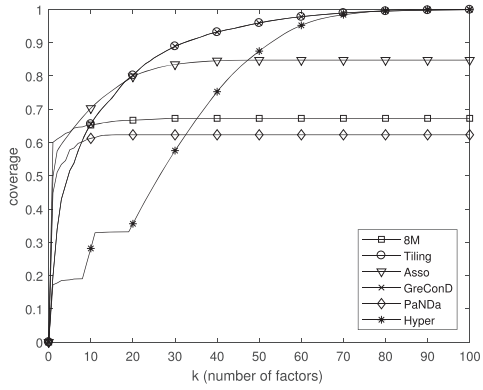
in which $|I|$ is the number of 1s in I (5). For 8M we used the default recommendation $cost = 1$ and used various values for *init*.

4.1 Evaluating 8M and Its Improvement 8M+

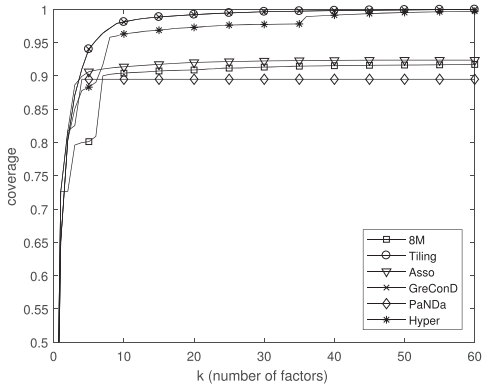
8M *vs. other BMF algorithms.* We first consider how the original 8M algorithm compares with the selected existing BMF algorithms described above in this section. The results are displayed in Figure 2. The graphs depict the coverage $c(l)$ of the first l factors generated by the algorithms. One may observe that 8M compares fairly well with the current algorithms. It even outperforms PaNDa on all datasets and on most of those we experimented with. On some data, 8M outperforms Asso and very often it outperforms Hyper in its coverage by the first few factors.

Initialization of 8M *via partial inclusion.* Next, we illustrate the role of full inclusion and partial inclusion in initialization of 8M as described in remark 2.2 and the first part of Section 3.1. The left part of Figure 3 illustrates the effect of τ on the coverage by factors of Asso; the right part illustrates such an effect of τ on 8M. The observed behavior is typical and appears in a fairly similar manner on other datasets as well. It is seen that $\tau = 1$ represents the worst choice of the parameter τ. For 8M, this means that the generalized intitialization via partial inclusion indeed represents a natural improvement of the original algorithm.

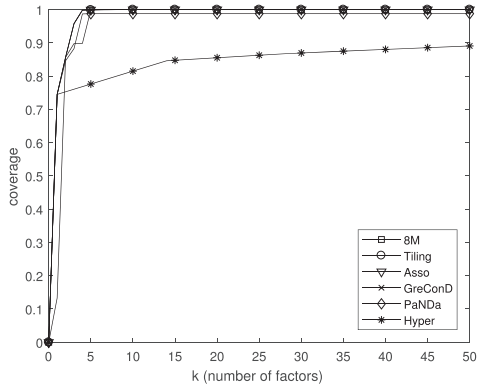
Evaluation of 8M *+.* Next we compare the original 8M to 8M+, i.e., to our improvement based on the initialization inspired by GreConD, as described in Section 3.1. This initialization fares even better than the one based on partial inclusion examined in the previous paragraph. The results are displayed in Figure 4. One may observe from the graphs that the improvement is significant. The 8M+ algorithm delivers factors with larger coverage for the smaller numbers of factors than 8M. Additionally, the overall coverage of input data obtained via 8M+ is (sometimes significantly) bigger than coverage in case of 8M. Moreover, taking into account Figure 2, one can see that this improvement makes the new algorithm an interesting rival to the current algorithms.



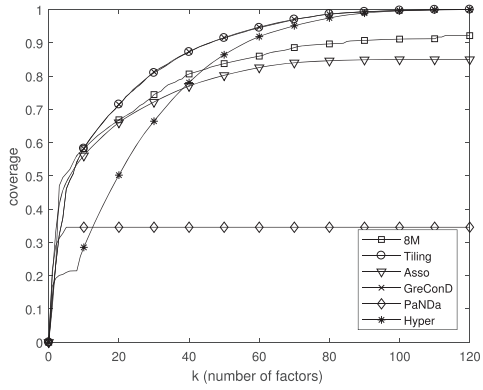
(a) Chess



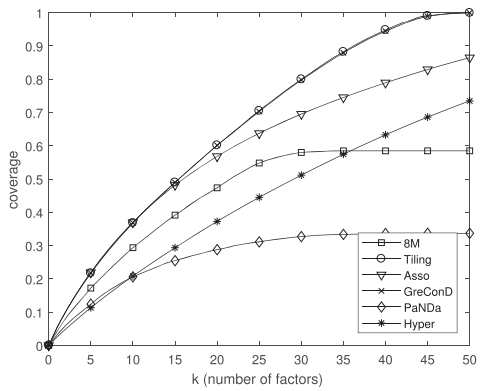
(b) Firewall 1



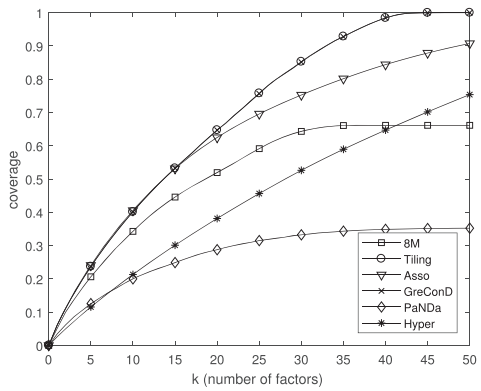
(c) Firewall 2



(d) Mushroom



(e) Set X1



(f) Set X2

Fig. 2. Coverage quality of the first l factors on real and synthetic data.

ALGORITHM 8: Modified GreConD algorithm

Input: $n \times m$ Boolean matrix I and parameter ϵ
Output: Boolean matrices A and B

$A \leftarrow$ empty matrix
$B \leftarrow$ empty matrix
while $|I - A \circ B| > 0$ **do**
  $D \leftarrow \mathbf{0}_{1\times m}$
  $v \leftarrow 0$
  while *there is index* j *such that* $D_j = 0$ *and* $\textsc{Cover}(D, j, I, A, B) > v$ **do**
    $j \leftarrow \arg\max_{j, D_j=0} \textsc{Cover}(D, j, I, A, B)$
    $D_j \leftarrow 1$
    $D \leftarrow D^{\downarrow\uparrow}$
    $v \leftarrow \textsc{Cover}(D, j, I, A, B)$
  end
  add column $D^{\downarrow}$ as new column **to** A
  add row D as new row **to** B
end
foreach *factor* $l = k, \ldots, 1$ **do**
  $X \leftarrow A$
  $Y \leftarrow B$
  remove column $X_{\_l}$ **from** X
  remove row $Y_{l\_}$ **from** Y
  if $E(I, X \circ Y) - E(I, S \circ B) \leq p \cdot |C|$ **then**
    remove column $A_{\_l}$ **from** A and row $B_{l\_}$ **from** B
  end
end
return A and B

ALGORITHM 9: Cover function of GreConD algorithm

Input: A candidate row D, an index j, Boolean matrices I, A, B
Output: A number of elements in I covered by $A \circ B$

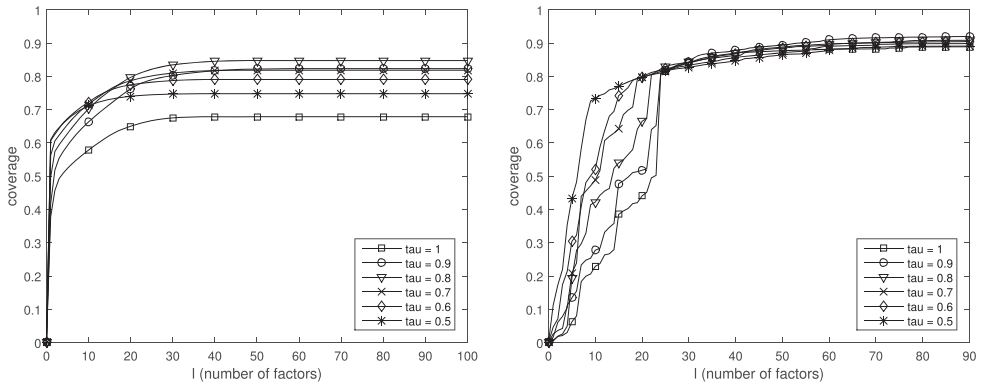
$D_j \leftarrow 1$
add column $D^{\downarrow}$ as new column **to** A
add row D as new row **to** B
$cover \leftarrow \sum_{i=1,j=1}^{n,m} (A \circ B)_{ij}$
return *cover*

4.2 Evaluating the Improved Asso Algorithm

We now consider the improvement to the Asso algorithm inspired by 8M, as described in Section 3.2. The results are presented in Table 2. The columns represent the results for the original Asso and the improved Asso for various values of parameter p, namely, $p = 0, 0.01, \ldots, 0.05$ (see Section 3.2). The rows labeled "k" represent the number of factors obtained by the particular algorithm on the given dataset. The number k is the number of factors for which the algorithm obtains the most precise factorization (i.e., the smallest error E, or, equivalently, the largest coverage c). Therefore, due to the logic of the algorithms, k is the number of factors after which the algorithm stops producing further factors. The rows labeled "c" contain the coverage of the computed fac-

Table 1. Datasets and Their Characteristics

dataset	size	density
Apj	2,044 × 1,164	0.003
Chess	3,196 × 76	0.487
DNA	4,590 × 392	0.015
Emea	3,046 × 35	0.068
Firewall 1	365 × 709	0.124
Firewall 2	325 × 590	0.190
Mushroom	8,124 × 119	0.193
Paleo	501 × 139	0.051

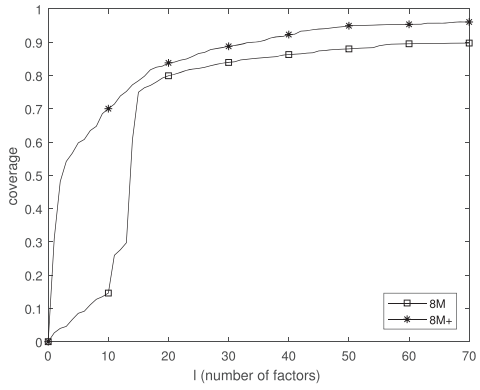


(a) Asso (b) 8M with initialization via partial inclusion

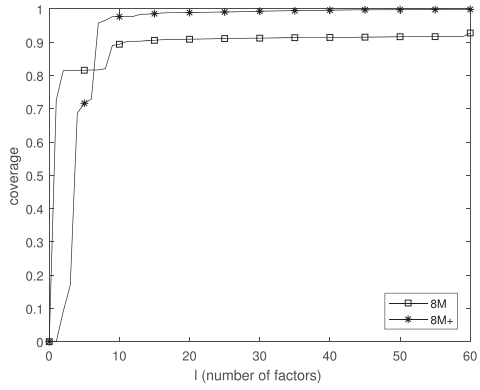
Fig. 3. Factorization of the Chess dataset for selected values of τ.

Table 2. Improvements to the Asso Algorithm

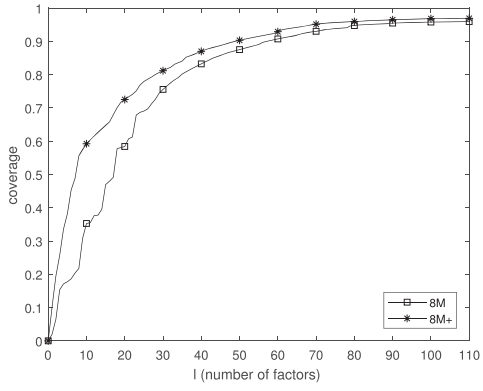
Dataset		orig. Asso	improved Asso with parameter p					
			$p = 0$	$p = 0.01$	$p = 0.02$	$p = 0.03$	$p = 0.04$	$p = 0.05$
Apj	k	455	454	408	380	362	344	327
	c	0.975	0.975	0.965	0.955	0.945	0.935	0.925
Chess	k	53	53	33	28	25	22	21
	c	0.848	0.848	0.839	0.831	0.821	0.809	0.804
DNA	k	280	280	213	186	167	153	141
	c	0.912	0.912	0.902	0.892	0.882	0.872	0.862
Emea	k	32	32	27	25	24	23	22
	c	0.972	0.972	0.965	0.956	0.949	0.942	0.933
Firewall 1	k	48	47	11	5	4	3	3
	c	0.924	0.924	0.914	0.906	0.899	0.887	0.887
Firewall 2	k	8	8	4	4	4	4	3
	c	0.999	0.999	0.998	0.998	0.998	0.998	0.958
Mushroom	k	95	94	68	62	57	51	47
	c	0.837	0.837	0.827	0.818	0.808	0.845	0.831
Paleo	k	130	130	127	123	120	117	114
	c	0.975	0.975	0.966	0.955	0.945	0.936	0.926



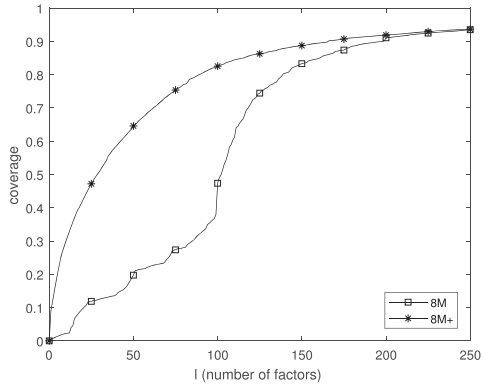
(a) Chess



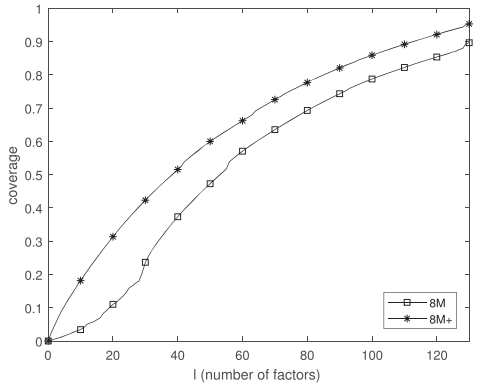
(b) Firewall 1



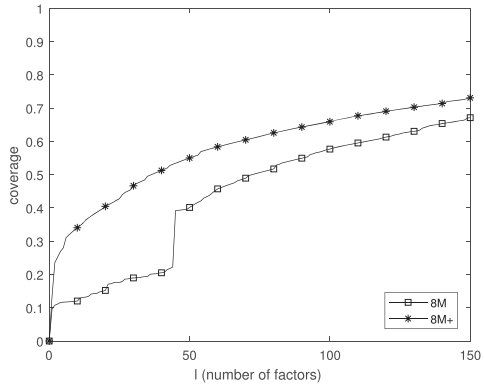
(c) Mushroom



(d) DNA

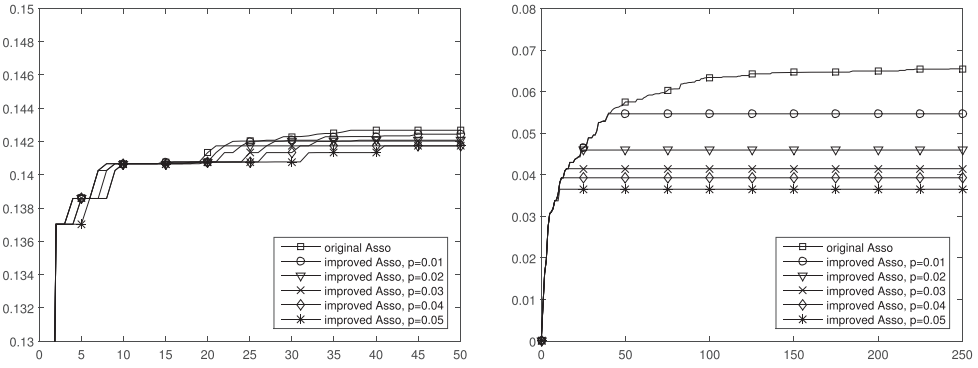


(e) Paleo



(f) Apj

Fig. 4. Coverage quality of the first l factors on real data: 8M vs. 8M+.



(a) Chess (b) DNA

Fig. 5. Normalized overcover error of Asso and its improvement.

torization using k factors. Thus, for instance, when factorizing the Emea data, the original Asso needs 32 factors to obtain a very precise factorization, namely, with $c = 0.972$. Our modification with $p = 0.02$ requires only 25 factors for computing a factorization which is highly accurate as well, namely one with coverage $c = 0.956$. From Table 2, one may easily observe that the reduction of factors is significant while the overall coverage is only slightly affected. These results confirm that the idea of revisiting and dropping factors, which is inspired by 8M, is natural for the reasons described in Section 3.2, and that the idea is also significant from a practical point of view.

Since the properties of the the E_o-part of the overall error may be regarded as the principal justification of revisiting the previously generated factors (see remark 2.1), Figure 5 displays (a zoom of) the graph of E_o on the Chess and the DNA data of the original Asso algorithm and its improvement for seleced values of the parameter p (in fact, the graph depicts the values of the normalized overcover error, $\frac{E_o}{|I|}$). As is apparent, E_o is smaller for the improved Asso than for the original Asso, and, as expected, larger values of p lead to smaller E_o.

4.3 Evaluating the Improved GreConD Algorithm

We now present the results of the improvement of the original GreConD algorithm described in Section 3.3. Table 3 provides a comparison of the original GreConD algorithm and its modified version in a similar manner we did for the improved Asso algorithm in the previous section. Note that GreConD always attains an exact factorization of its input matrix. We can see in the table that, for instance, when factorizing the Mushroom data, the original GreConD needs 120 factors to obtain an exact factorization. Our modification with $p = 0$ requires only 113 factors for exact factorization and only 61 factors for computing a highly accurate factorization, namely, with coverage $c = 0.951$. Table 3 demonstrates that the reduction in the number of factors is in most cases quite significant even for small values of p. As in the case of Asso, revisiting of the previously generated factors turns out to be useful from a practical viewpoint.

4.4 Improvement of 8M+ via New Revisiting Strategy

The key idea, borrowed from the 8M algorithm, in the improvements of Asso and GreConD consists in revisiting and possibly dropping the already computed factors. However, the original 8M as well as 8M+ utilize only a limited instance of this idea. Namely, the number of revisited factors is limited to three and, in addition, one always drops a factor that has been computed two steps back.

Table 3. Improvements to the GreConD Algorithm

Dataset		orig. GreConD	improved GreConD with parameter p					
			$p = 0$	$p = 0.01$	$p = 0.02$	$p = 0.03$	$p = 0.04$	$p = 0.05$
Apj	k	464	464	413	384	363	346	330
	c	1.000	1.000	0.990	0.980	0.970	0.960	0.950
Chess	k	124	119	72	62	55	51	47
	c	1.000	1.000	0.991	0.981	0.970	0.962	0.952
DNA	k	496	496	341	313	282	264	250
	c	1.000	1.000	0.990	0.980	0.970	0.960	0.950
Emea	k	42	34	29	26	25	24	23
	c	1.000	1.000	0.992	0.981	0.975	0.963	0.956
Firewall 1	k	66	65	17	10	8	7	6
	c	1.000	1.000	0.990	0.981	0.972	0.964	0.953
Firewall 2	k	10	10	4	4	4	4	3
	c	1.000	1.000	0.998	0.998	0.998	0.998	0.958
Mushroom	k	120	113	81	73	69	65	61
	c	1.000	1.000	0.990	0.980	0.970	0.960	0.951
Paleo	k	151	139	137	135	131	128	124
	c	1.000	1.000	0.991	0.981	0.970	0.961	0.951

Table 4. Improvements to the 8M+ Algorithm

Dataset		orig. 8M+	improved 8M+ with parameter p					
			$p = 0$	$p = 0.01$	$p = 0.02$	$p = 0.03$	$p = 0.04$	$p = 0.05$
Apj	k	150	150	147	141	136	130	124
	c	0.731	0.731	0.725	0.717	0.710	0.703	0.695
Chess	k	67	67	57	49	43	40	38
	c	0.965	0.965	0.955	0.946	0.936	0.926	0.916
DNA	k	350	350	282	243	213	192	173
	c	0.955	0.955	0.945	0.935	0.925	0.916	0.906
Emea	k	30	30	28	26	25	24	27
	c	0.970	0.970	0.962	0.954	0.942	0.936	0.923
Firewall 1	k	57	57	30	11	8	10	7
	c	0.999	0.999	0.990	0.980	0.970	0.960	0.955
Firewall 2	k	9	8	4	4	4	4	3
	c	1.000	1.000	0.992	0.992	0.992	0.964	0.958
Mushroom	k	114	114	93	82	70	65	60
	c	0.981	0.981	0.971	0.961	0.952	0.942	0.933
Paleo	k	130	130	128	125	121	118	115
	c	0.953	0.953	0.943	0.936	0.925	0.915	0.907

We therefore modified 8M+ in a similar vein we did for Asso and GreConD, i.e., we allowed for revisiting of all previously generated factors. A comparison of 8M+ and the present modification is presented in Table 4. One may see, like in the cases of Asso and GreConD that the new algorithm is capable of achieving almost the same coverage with a significantly smaller number of factors.

5 CONCLUSIONS

We provided a complete pseudocode and a detailed description of the 8M algorithm along with an experimental evaluation of 8M and its comparison to selected algorithms for BMF. Our analysis, which was long overdue, revealed ideas utilized by 8M but not used by the current factorization algorithms. The most interesting is the idea of revisiting and possibly dropping the previously computed factors. This idea is appealing particularly for algorithms performing general factorization, i.e., those committing overcovering error. Namely, unlike the symmetric undercovering error, overcovering may only increase in the course of computation of factors if the algorithm does not revisit and modify the previously computed factors. We used this idea to improve the Asso and the GreConD algorithms, resulting in new algorithms with a significantly better performance. Revisiting of the previously generated factors is a natural general idea that may be applied to any factorization algorithm which computes factors in a consecutive manner. A further exploration of this idea and its application to the design of factorization algorithms remain a promising future topic.

ACKNOWLEDGMENT

The article is an extended version of Belohlavek, R., Trnecka, M., The 8M algorithm from today's perspective. Proc. CLA 2018, pp. 167–178.

REFERENCES

[1] K. Bache and M. Lichman. 2013. UCI Machine Learning Repository. Irvine, CA: University of California, School of Information and Computer Science. Retrieved from http://archive.ics.uci.edu/ml.

[2] E. Bartl, R. Belohlavek, P. Osicka, and H. Řezanková. 2012. Dimensionality reduction in boolean data: Comparison of four BMF methods. In *Proceedings of the International Workshop on Clustering High-Dimensional Data*. 118–133.

[3] R. Belohlavek, J. Outrata, and M. Trnecka. 2014. Impact of Boolean factorization as preprocessing methods for classification of Boolean data. *Annals of Mathematics and Artificial Intelligence* 72, 1–2 (2014), 3–22.

[4] R. Belohlavek and M. Trnecka. 2015. From-below approximations in Boolean matrix factorization: Geometry and new algorithm. *Journal of Computer and System Sciences* 81, 8 (2015), 1678–1697.

[5] R. Belohlavek and M. Trnecka. 2015. A new algorithm for Boolean matrix factorization which admits overcovering. *Discrete Applied Mathematics* 249 (2018), 36–52.

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

[7] R. A. Brualdi and H. J. Ryser. 1991. *Combinatorial Matrix Theory*. Cambridge University Press.

[8] W. J. Dixon (ed.). 1992. *BMDP Statistical Software Manual*. University of California Press, Berkeley, CA.

[9] A. Ene, W. Horne, N. Milosavljevic, P. Rao, R. Schreiber, and R. E. Tarjan. 2008. Fast exact and heuristic methods for role minimization problems. In *Proceedings of the 13th ACM Symposium on Access Control Models and Technologies*. 1–10.

[10] B. Ganter and R. Wille. 1991. *Formal Concept Analysis: Mathematical Foundations*. Springer, Berlin.

[11] F. Geerts, B. Goethals, and T. Mielikäinen. 2004. Tiling databases. In *Proceedings of the 2004 International Conference on Discovery Science*. 278–289.

[12] S. Karaev, P. Miettinen, and J. Vreeken. 2015. Getting to know the unknown unknowns: Destructive-noise resistant Boolean matrix factorization. In *Proceedings of the 2015 SIAM International Conference on Data Mining*. 325–333.

[13] K. H. Kim. 1982. *Boolean Matrix Theory and Applications*. M. Dekker, NY.

[14] H. Lu, J. Vaidya, V. Atluri, and Y. Hong, 2012. Constraint-aware role mining via extended Boolean matrix decomposition. *IEEE Transactions on Dependable and Secure Computing* 9, 5 (2012), 655–669.

[15] C. Lucchese, S. Orlando, and R. Perego. 2010. Mining top-k patterns from binary datasets in presence of noise. In *Proceedings of the 2010 SIAM International Conference on Data Mining*. 165–176.

[16] C. Lucchese, S. Orlando, and R. Perego. 2014. A unifying framework for mining approximate top-k binary patterns. *IEEE Transactions on Knowledge and Data Engineering* 26, 12 (2014), 2900–2913.

[17] P. Miettinen. 2010. Sparse Boolean matrix factorizations. In *Proceedings of the 2010 IEEE International Conference on Data Mining*. 935–940.

[18] P. Miettinen, T. Mielikäinen, A. Gionis, G. Das, and H. Mannila, 2008. The discrete basis problem. *IEEE Transactions on Knowledge and Data Engineering* 20, 10 (2008), 1348–1362.

[19] P. Miettinen and J. Vreeken. 2011. Model order selection for Boolean matrix factorization. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 51–59.
[20] S. Myllykangas, J. Himberg, T. Böhling, and B. Nagy. 2006. DNA copy number amplification profiling of human neoplasms. *Oncogene* 25, 55 (2006), 7324–7332.
[21] D. S. Nau, G. Markowsky, M. A. Woodbury, and D. B. Amos. 1978. A mathematical analysis of human leukocyte antigen serology. *Mathematical Biosciences* 40, 3–4 (1978), 243–270.
[22] G. Schmidt. 2011. *Relational Mathematics*. Cambridge University Press.
[23] L. Stockmeyer. 1975. *The Set Basis Problem is NP-complete*. Technical Report No. RC5431, IBM, Yorktown Heights, NY.
[24] J. Vaidya, V. Atluri, and Q. Guo. 2007. The role mining problem: Finding a minimal descriptive set of roles. In *Proceedings of the 12th ACM Symposium on Access Control Models and Technologies*. 175–184.
[25] Y. Xiang, R. Jin, D. Fuhry, and F. F. Dragan. 2011. Summarizing transactional databases with overlapped hyperrectangles. *Data Mining and Knowledge Discovery* 23 (2011), 215–251.

Received December 2019; revised September 2020; accepted October 2020