Discrete Applied Mathematics 249 (2018) 36–52

Contents lists available at ScienceDirect

Discrete Applied Mathematics

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

A new algorithm for Boolean matrix factorization which admits overcovering

Radim Belohlavek, Martin Trnecka *

Department Computer Science, Palacký University, Olomouc, Czech Republic

ARTICLE INFO

Article history:
Received 13 February 2016
Received in revised form 13 July 2017
Accepted 18 December 2017
Available online 16 August 2018

Keywords:
Boolean matrix factorization
Formal concept analysis
Algorithms

ABSTRACT

We present a new algorithm for general Boolean matrix factorization. The algorithm is based on two key ideas. First, it utilizes formal concepts of the factorized matrix as crucial components of constructed factors. Second, it performs steps back during the construction of factors to see if some of the already constructed factors may be improved or even eliminated in view of the subsequently added factors. The second idea is inspired by 8M—an old, previously incompletely described and virtually unknown factorization algorithm, which we analyze and describe in detail. We provide experimental evaluation of the new algorithm and compare it to 8M and two other well-known algorithms. The results demonstrate that our algorithm outperforms these algorithms in terms of quality of the decompositions as well in its robustness with respect to small changes in data.

1. Problem description

1.1. Problem in brief

Research in Boolean matrix factorization (BMF), or Boolean matrix decomposition, has resulted in various new methods of analysis and processing of data and has also contributed to our understanding of Boolean (binary, yes/no) data as regards foundational aspects. While the developments of practical methods and theoretical foundations are clearly connected, most of the current BMF methods use limited theoretical insight. Building upon our previous research [4], we developed in our recent paper [3] an efficient BMF algorithm utilizing a better understanding of the geometry of BMF, which we also developed in [3]. The understanding, provided in terms of Galois connections, concept lattices, and other structures underlying formal concept analysis (FCA [11]), as well as the algorithm, are primarily developed for exact Boolean matrix factorizations and employ formal concepts as factors. As such, the constructed factorizations are limited (in that they never commit overcovering, see below). Such limitation presents no restriction when exact factorizations are desired. Moreover, even though computing restricted type of decompositions, the algorithm outperforms the other BMF algorithms also when approximate factorizations are needed with a prescribed precision [3]. Nevertheless, there are situations in which general factorizations are desirable, hence the limitation described above may indeed prove restrictive. In the present paper, we extend our previous approach to BMF and develop a new algorithm that computes general factorizations.

1.2. Basic notions and rationale for computing general BMFs

We denote by I an $n \times m$ Boolean matrix and interpret primarily as an object–attribute incidence matrix (hence the symbol I). That is, the entry I_{ij} corresponding to the row i and the column j is either 1 or 0, indicating that the object i does

* Corresponding author.
E-mail addresses: radim.belohlavek@acm.org (R. Belohlavek), martin.trnecka@gmail.com (M. Trnecka).

https://doi.org/10.1016/j.dam.2017.12.044
0166-218X/

or does not have the attribute j, respectively. The set of all $n \times m$ Boolean matrices is denoted $\{0, 1\}^{n\times m}$. The ith row and jth column vector of I is denoted by $I_{i\_}$ and $I_{\_j}$, respectively. In BMF, one generally attempts to find for a given $I \in \{0, 1\}^{n\times m}$ 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. $(A \circ B)_{ij} = \max_{l=1}^{k} \min(A_{il}, B_{lj})$. 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 called the *Boolean rank* (or Schein rank) of I.

The approximate equality in (1) is commonly assessed in BMF by means of the L_1-norm (Hamming weight in case of Boolean matrices) $\|\cdot\|$ and the corresponding metric $E(\cdot, \cdot)$, defined for $C, D \in \{0, 1\}^{n\times m}$ by

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

The following particular variants of the BMF problem, relevant to this paper, are considered in the literature.

- *Discrete Basis Problem* (DBP, [21]):
  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 $\|I - A \circ B\|$.
- *Approximate Factorization Problem* (AFP, [4]):
  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 $\|I - A \circ B\| \leq \varepsilon$.

These problems reflect two important views of BMF: DBP emphasizes the importance of the first few (presumably most important) factors; AFP emphasizes the need to account for (and thus to explain) a prescribed portion of data.

A useful view of BMF is provided in terms of rectangles [4,3]: $J \in \{0, 1\}^{n\times m}$ is called *rectangular* (a rectangle, for short) if $J = C \circ D$ for some $C \in \{0, 1\}^{n\times 1}$ (column) and $D \in \{0, 1\}^{1\times m}$ (row); this implies that upon suitable permutations of columns and rows, the 1s in J form a rectangular area. We say that J (or, the pair $\langle C, D\rangle$ for which $J = C \circ D$) *covers* $\langle i, j\rangle$ if $J_{ij} = 1$ (equivalently, $C_i = 1$ and $D_j = 1$). For matrices J_1 and J_2, we put

$$J_1 \leq J_2 \ (J_1 \text{ is contained in } J_2) \text{ iff } \quad (J_1)_{ij} \leq (J_2)_{ij} \text{ for every } i, j. \tag{3}$$

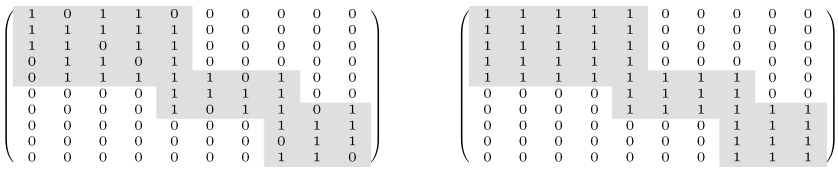
The following observation shows that a Boolean matrix product may be considered as a $\vee$-superposition of (or a coverage by) rectangles (see e.g. [3]):

Observation 1. *The following conditions are equivalent for any $I \in \{0, 1\}^{n\times m}$.*

(a) *$I = A \circ B$ for some $A \in \{0, 1\}^{n\times k}$ and $B \in \{0, 1\}^{k\times m}$.*
(b) *There exist rectangles $J_1, \ldots, J_k \in \{0, 1\}^{n\times m}$ such that $I = J_1 \vee \cdots \vee J_k$, i.e. $I_{ij} = \max_{l=1}^{k} (J_l)_{ij}$.*
(c) *There exist rectangles $J_1, \ldots, J_k \in \{0, 1\}^{n\times m}$ such that $I_{ij} = 1$ if and only if $\langle i, j\rangle$ is covered by some J_l.*

In particular, if A and B are the matrices from Observation 1(a), then one may put $J_l = A_{\_l} \circ B_{l\_}$ ($l = 1, \ldots, k$), i.e. J_l is the product of the lth column of A and the lth row of B, to obtain the rectangles in (b) and (c). Conversely, if $J_1 = C_1 \circ D_1, \ldots, J_k = C_k \circ D_k$ are the rectangles in (b) or (c) then the matrices A and B in which the lth column and lth row are C_l and D_l, respectively, satisfy (a). As a result, computing an exact factorization of I with a small number k of factors is equivalent to computing k rectangles contained in I that cover all the 1s in I. Since maximal rectangles in I correspond to formal concepts of I [11], the above observation led to the employment of formal concepts as factors in [4,3]. Clearly, one may utilize rectangles in I to cover not necessarily all 1s in I and thus to solve AFP. Even though such approach to AFP, as demonstrated by the algorithms in [4,3], is considerably successful, the resulting approximate factorizations $I \approx A \circ B$, which are called *from-below approximations* of I in [3], are restricted: While it may happen that $I_{ij} = 1$ and $(A \circ B)_{ij} = 0$ (undercovering), it never happens that $I_{ij} = 0$ and $(A \circ B)_{ij} = 1$ (overcovering).

That the lack of possible overcovering may be severely limiting is apparent from the following examples. Consider first the matrices I in Fig. 1a and J in Fig. 1b. Let I represent the observed data. One clearly recognizes three rectangles in I, the union of which forms the gray area, even though some of the entries inside the area contain 0 rather than 1. A natural view of I is that it results from the true data, represented by J, due to error. For instance, there might be insufficient evidence for the presence of some attributes on some objects, i.e. for the presence of 1s in certain entries, which is a plausible explanation of this kind of situation. From this viewpoint, one is interested in discovering from the observed data I the three factors behind the true data J, i.e. in view of the above observation in discovering from I the 10×3 and 3×10 matrices A and B for which $A \circ B = J$. But even if I represented true data, one may be interested in the decomposition into the above A and B because it



(a) Observed data (b) Hypothetical true data

Fig. 1. Three factors of hypothetical true data explaining observed data.

provides an approximate concise model of I. Clearly, such matrices are

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

Crucial for our argument is the fact that none of the three rectangles involved in the approximate decomposition of I into A and B is a rectangle contained in I, because I contains 0s (is not full of 1s) in the corresponding areas. Hence, the approximate factorization of I into A and B is not a from-below approximation and as such is not discoverable by from-below approximation algorithms such as those in [4,3]. In particular, the algorithm GreConD from [4] computes from I a decomposition involving 9 small factors. On the other hand, the algorithm GreConD+ developed in this paper computes precisely the approximate decomposition of I into A and B and thus discovers the three factors in question, even though the change of I w.r.t. J is relatively large.

As a second example, consider the $n \times n$ inequality matrix $I_{\neq}$. For $n = 6$,

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

It may be shown that the least number of rectangles covering $I_{\neq}$ (and thus contained in $I_{\neq}$) is n and that the largest of the rectangles contained in $I_{\neq}$ covers $\lceil n/2 \rceil \cdot \lfloor n/2 \rfloor$ entries. Therefore, when restricting to the from-below factorizations and to the rectangles in $I_{\neq}$, one has to do with considerable fragmentation—a relatively large number of small rectangles. On the other hand, when general approximate factorizations are considered, the $n \times n$ matrix $\mathbf{1}_n$ full of 1s may be regarded as a single large approximate factor, committing a relative error $\frac{E(I, \mathbf{1}_n)}{n^2} = \Theta(\frac{1}{n})$.

1.3. Related work

Due to a limited scope, we focus on the directly relevant work. Except for Boolean matrix theory itself [14], results relevant to BMF are traditionally presented in the literature on binary relations and graph theory, see e.g. [7,28]. A particularly important related area is formal concept analysis (FCA) [11], in which Boolean matrices are represented by so-called formal contexts. FCA provides solid lattice-theoretical foundations which we utilize.

Among the first works on applications of BMF in data analysis are [25,26], in which the authors have already been aware of the NP-hardness (i.e. provable computational difficulty) of the decomposition problem due to the NP-hardness of the set basis problem [29]. An early but currently virtually unknown is the 8M algorithm which we present below. Interest in BMF in current data mining is primarily due to the work of Miettinen et al. In particular, the DBP, the corresponding complexity results, and the Asso algorithm discussed below appeared in [21]. In [4], we showed that formal concepts (i.e. fixpoints of Galois connections) are natural factors of Boolean matrices, proved their optimality for exact factorizations, described transformations between attribute and factor spaces, and proposed the GreConD algorithm discussed below. [3] presents a deeper insight into from-below approximations and a new algorithm based on it, as well as some observations regarding general BMF which we use.

Let us also mention some other BMF algorithms which appeared in the recent data mining literature, in particular Hyper [31], PaNDa [17], PaNDa$^+$ [18], Nassau [13], as well as further work on various aspects of the decomposition

problem and applications of BMF, including [10,16,19,20,22,23,27,30]. Due to its direct relevance to the idea behind our new algorithm, we include PANDA$^+$ in our experimental evaluation. The other algorithms are not included in our evaluation for space reasons. However, comparisons of these algorithms to Asso, which we use in our evaluation, on data similar to our data are available in the mentioned literature; in particular, [3] uses almost the same data we use in this paper.

2. General BMF, Asso, PANDA$^+$, and 8M

2.1. Non-symmetry of committed error

Crucial for the design of BMF algorithms is the following observation. The distance (error) function E as defined by (2) is a sum of two components, E_u corresponding to 1s in I that are 0s (and hence *uncovered*) in $A \circ B$ and E_o corresponding to 0s in I that are 1s (and hence *overcovered*) in $A \circ B$:

$$E(I, A \circ B) = E_u(I, A \circ B) + E_o(I, A \circ B), \text{ where}$$
$$E_u(I, A \circ B) = |\{\langle i, j\rangle \,;\, I_{ij} = 1, (A \circ B)_{ij} = 0\}|,$$
$$E_o(I, A \circ B) = |\{\langle i, j\rangle \,;\, I_{ij} = 0, (A \circ B)_{ij} = 1\}|.$$

Even though E_u and E_o look symmetric, they have a highly non-symmetric role in BMF. The non-symmetry is apparent from the following observation, which says that as we add new factors to the already established ones (i.e., add columns and rows to A and B, respectively), E_u may only decrease while E_o may only increase:

Observation 2. *Let $A' \in \{0, 1\}^{n\times(k+1)}$ and $B' \in \{0, 1\}^{(k+1)\times m}$ result by adding to A and B a single column and row, respectively. Then*

$$E_u(I, A' \circ B') \leq E_u(I, A \circ B) \text{ and } E_o(I, A' \circ B') \geq E_o(I, A \circ B).$$

This observation has not been explicitly presented in the literature until [3]. Nevertheless, the two components, E_u and E_o, are implicitly used in the Asso algorithm [21] and are treated non-symmetrically by function *cover* using two different weights (see Section 2.2).

Observation 2 provides a clear warning for the design of BMF algorithms. Namely, due to the provable hardness of the BMF related problems, such as DBP or AFP [4,21], it is reasonable to assume that conceivable algorithms compute one factor after another (this is indeed the case of all of the existing algorithms). The warning reads: One needs to be careful with committing E_o error because E_o never decreases by adding further factors.

2.2. Asso

This algorithm, proposed in [21], primarily for DBP, is nowadays a classic algorithm for general BMF. It first computes from a given $n \times m$ matrix I an $m \times m$ Boolean matrix C in which $C_{ij} = 1$ if the confidence of rule $\{i\} \Rightarrow \{j\}$ – the fraction of objects having j among those having i – exceeds a user-specified parameter τ. The rows of C are then used as candidates for the rows of the factor–attribute matrix. The actual rows are selected in a greedy manner using function *cover* that rewards with weight w^+ the decrease of E_u and penalizes with weight w^- the increase of E_o that is due to a given row of C. Asso thus commits both types of errors, E_u and E_o. The once committed error E_o never decreases and thus the only way to control it in Asso is via the weights w^+ and w^-; notwithstanding the non-symmetry, the authors recommend a default setting $w^+ = w^- = 1$.

2.3. PANDA$^+$

PANDA$^+$ [18] is basically an algorithmic framework based on the algorithm PANDA [17]. It employs a general cost function which makes it possible to impose various criteria. The algorithm aims to extract a set $\mathcal{F}$ of pairs $\langle C, D\rangle$ (C is a set of objects, D is a set of attributes) that minimize the cost function. Every $\langle C, D\rangle$ in $\mathcal{F}$ is computed by first computing its core and then extending the core to $\langle C, D\rangle$. A core is, in fact, a rectangle contained in I and is computed by adding columns from a sorted list (sorting by several criteria is proposed). Extension to $\langle C, D\rangle$ is performed by adding further columns and rows to a core until such addition does not help in minimizing the cost. This idea is similar in principle to that of our GRECOND+, as described below. As the cost function, we use the function proposed by the authors, which is the sum of the description complexity of all $\langle C, D\rangle$s in $\mathcal{F}$, i.e. $\sum_{\langle C,D\rangle\in\mathcal{F}}(|C| + |D|)$, and the error committed by $\mathcal{F}$, i.e. $E(I, A_\mathcal{F} \circ B_\mathcal{F})$ where $A_\mathcal{F}$ and $B_\mathcal{F}$ are the matrices corresponding to the patterns in $\mathcal{F}$, see (4) below. To make our comparison fair with respect to the goals of the algorithms, we also employ as the cost function solely the error $E(I, A_\mathcal{F} \circ B_\mathcal{F})$, and denote the corresponding algorithm PANDA$^+$EO ("EO" stands for "error only").

2.4. 8M: A forgotten factorization method

The 8M method is one of the methods available in BMDP—a statistical software package which was being developed primarily for biomedical applications since the 1960s at UCLA under the leadership of W. J. Dixon. 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 [9], 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.

Even though the description of 8M in [9] is fairly detailed, some parts such as the initialization step are somewhat unclear. We therefore observed in detail the program behavior on data to figure out the unclear parts until our own implementation yielded the same results as the software which we purchased from Statistical Solutions Ltd. 8M 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. According to this edition, 8M is based on research done by M. R. Mickey, was designed by Mickey with contributions from L. Engelman, and was programmed by P. Mundle and Engelman.

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 $B \leftarrow$ ComputeInitialFactors(*init*); $A \leftarrow \mathbf{0}_{n \times init}$
2 $f \leftarrow init$
3 RefineMatricesAB(A, B, I, *cost*)
4 *kReached* $\leftarrow 0$
5 **while** *kReached* < 2 *or* $I \leq A \circ B$ **do**
6 **foreach** $\langle i, j \rangle$ **do if** $I_{ij} > (A \circ B)_{ij}$ **then** $\Delta^+_{ij} \leftarrow 1$ **else** $\Delta^+_{ij} \leftarrow 0$
7
8 **add** column j of Δ^+ with the largest count of 1s as new column **to** A
9 **add** row of 0s as new row **to** B and **set** entry j of this row **to** 1
10 $f \leftarrow f + 1$
11 RefineMatricesAB(A, B, I, *cost*)
12 **if** *another two new factors were added* **then**
13 **remove** column $A_{\_(f-2)}$ **from** A and row $B_{(f-2)\_}$ **from** B
14 $f \leftarrow f - 1$
15 RefineMatricesAB(A, B, I, *cost*)
16 **end**
17 **if** f=k **then** *kReached* $\leftarrow$ *kReached* $+ 1$
18 **end**
19 **return** A, B

To compute – for a given $n \times m$ Boolean matrix I and a prescribed number *init* of initial factors – $n \times k$ and a $k \times m$ Boolean matrices A and B where k is a given desired number of factors, the algorithm 8M (Algorithm 1) works as follows. First, *init* initial factors are computed (l. 1, see below for details) and the number f 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. A loop is then entered (l. 5–17) in which new factors are being added and deleted until the desired number k of factors is reached for the second time or all 1s in I are covered by $A \circ B$ (conditions on l. 5). Whenever a factor is added or removed, A and B are refined. Adding and removing factors is as follows. One starts with $f = init$ factors, adds another 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 $init = 2$ factors, one successively obtains 2, 3, 4, 3, 4, 5, 4, 5 etc. factors. Since one stops when the desired number k of factors is obtained the second time, one computes 2, 3, 4, 3, 4, 5, 4, 5, 6, 7, 6 factors in the case $k = 6$, and the last six factors are the final factors output by the algorithm (provided the second condition in l. 5 is not met). By default, $init = k - 2$ but *init* is generally set by the user.

The initial factors are computed by ComputeInitialFactors 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). One then goes through the rows i of C, $i = 1, 2, \ldots$, and adds them as new rows of B until *init* rows have been added: row i of C is added to B provided there exists j with $C_{ij} = 1$ such that no row previously added to B contains 1 at position j.

Refining of A and B by RefineMatricesAB (Algorithm 2) consists in performing a cycle until A and B do not change but at most three times, in which A is computed from I, B, and the parameter *cost* by a so-called Boolean regression described in RefineMatrixA (Algorithm 3), followed by computing symmetrically B using RefineMatrixB. A new factor is computed 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 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.

Let us now comment on 8M. An interesting feature of this old algorithm, which is not present in the currently known algorithms such as Asso, is its continuous update and even removal of the previously generated factors. The rationale for this "rethinking of previous factors" – which is, however, not mentioned in the description of 8M in [9] – is apparent from our observations in Section 2.1: Since the overcover-part E_o of the error function only increases by adding further factors, it is reasonable to look back if what had been added previously may be improved in view of the subsequently added factors. Furthermore, let us mention the similarity between Asso's computing of its candidate basis vectors and 8M's computing of the initial factors stored in B. This deserves further exploration, particularly in view of the comments in [9] suggesting

that not only the full but also approximate inclusion of columns – i.e. exactly the confidence used by Asso –may be used in computing the initial factors.

Algorithm 2: RefineMatricesAB

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

```
repeat
    RefineMatrixA(A, B, I, cost)
    RefineMatrixB(A, B, I, cost)
until loop executed 3 times or A and B did not change
```

Algorithm 3: RefineMatrixA

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

```
foreach row i ∈ {1, ..., n} do
    y ← I_i_; Z ← B; A_i_ ← 0
    repeat
        foreach factor l ∈ 1, ..., f do
            m_l ← Σ_{j=1}^{m} y_j · Z_lj − cost · Σ_{j=1}^{m} (1 − y_j) · Z_lj
        end
        select p for which m_p = max_l m_l
        if m_p > 0 then
            A_ip ← 1
            foreach j ∈ {1, ..., m} do
                if Z_pj = 1 then
                    Z_j ← 0; y_j ← 0
                end
            end
        end
    until m_p > 0
end
```

3. GreConD+: A new algorithm

Since GreConD+ is conveniently described using basic notions of formal concept analysis (FCA [11]), we now review these notions. To every Boolean matrix $I \in \{0,1\}^{n\times m}$, one may associate the pair $\langle{}^{\uparrow}, {}^{\downarrow}\rangle$ of operators assigning to sets $C \subseteq X = \{1,\ldots,n\}$ and $D \subseteq Y = \{1,\ldots,m\}$ the sets

$$C^{\uparrow} = \{j \in Y \mid \forall i \in C : I_{ij} = 1\} \text{ and } D^{\downarrow} = \{i \in X \mid \forall j \in D : I_{ij} = 1\}.$$

That is, $C^{\uparrow}$ is the set of all attributes (columns) shared by all objects (rows) in C and $D^{\downarrow}$ is the set of all objects sharing all attributes in D. The set

$$\mathcal{B}(I) = \{\langle C, D\rangle \mid C \subseteq X, D \subseteq Y, C^{\uparrow} = D, D^{\downarrow} = C\}$$

is called the *concept lattice* of I. It consists of all $\langle{}^{\uparrow}, {}^{\downarrow}\rangle$-closed pairs $\langle C, D\rangle$, called the *formal concepts* of I, with C and D called the *extent* and the *intent*. Concept lattices are the basic structures in FCA; see [8,11] for details. Note also that several polynomial-time delay algorithms exist for computing $\mathcal{B}(I)$ [15].

An important link between BMF and formal concepts consists in the following facts. First, in view of Observation 1, rectangles contained in I are the building blocks of decompositions of I. Clearly, most efficient are rectangles maximal w.r.t. containment $\leq$ defined by (3). As is well known, maximal rectangles contained in I correspond to formal concepts in $\mathcal{B}(I)$ in that J is a maximal rectangle in I if and only if there exists a formal concept $\langle C, D\rangle \in \mathcal{B}(I)$ such that $J_{ij} = 1$ is equivalent to $i \in C$ and $j \in D$. This link is utilized in [4,3] in the algorithms computing from-below factorizations of Boolean matrices.

Rather than two matrices, A and B, our algorithm computes from the matrix I a factorization in the form of a set $\mathcal{F} = \{\langle C_1, D_1\rangle, \ldots, \langle C_k, D_k\rangle\}$ of pairs of $C_l \subseteq \{1,\ldots,n\}$ (objects/rows) and $D_l \subseteq \{1,\ldots,m\}$ (attributes/columns). This is convenient due to the logic of the algorithm; it also connects naturally to our previous papers [4,3]. A straightforward relationship to factorizations represented by two matrices, A and B, is the following. Given such a set $\mathcal{F}$ with a fixed indexing of the pairs $\langle C_l, D_l\rangle$, define the $n \times k$ and $k \times m$ Boolean matrices $A_{\mathcal{F}}$ and $B_{\mathcal{F}}$ by

$$(A_{\mathcal{F}})_{il} = \begin{cases} 1 & \text{if } i \in C_l, \\ 0 & \text{if } i \notin C_l, \end{cases} \quad \text{and} \quad (B_{\mathcal{F}})_{lj} = \begin{cases} 1 & \text{if } j \in D_l, \\ 0 & \text{if } j \notin D_l, \end{cases} \tag{4}$$

for $l = 1,\ldots,k$. That is, the lth column and lth row of $A_{\mathcal{F}}$ and $B_{\mathcal{F}}$ are the characteristic vectors of C_l and D_l, respectively. Clearly, given $n\times k$ and $k\times m$ Boolean matrices A and B, there exists $\mathcal{F}$ for which $A = A_{\mathcal{F}}$ and $B = B_{\mathcal{F}}$. Whether one represents the factors by $\mathcal{F}$ or by A and B is thus a matter of one's preference.

Our new algorithm for general BMF, GreConD+, (Algorithm 4) is based on the following ideas. The non-symmetry of error and the observations discussed in Section 2.1 lead us to construct new factors by attempting first to cover as many uncovered 1s in I as possible, committing at the same time no E_o error. For this purpose, we employ the idea from our previous algorithm, GreConD [4]. Every such factor, which represents a formal concept in I, is then expanded in a greedy manner by further columns and rows for which the gain due to decrease of E_u is considerably larger than the loss due to committing of E_o. After a new factor is computed, the previously computed factors are examined and modified or even removed in order to decrease E_o in view of the subsequently computed factors. Revisiting previous factors is inspired by 8M, but our method of revisiting is very different from that of 8M. For simplicity, the pseudocodes in Algorithms 4 and 5 contain a version in which the expansion as well as modification of factors involves only columns. A full version is obtained easily by inserting the corresponding symmetric parts regarding rows. Note, however, that even this simplified version proved useful, particularly for data with a much larger number of rows than columns which feature is typical for the current benchmark data. We now provide details.

Algorithm 4: GreConD+

Input: Boolean $n \times m$ matrix I, number w
Output: set $\mathcal{F}$ of factors

```
1  U ← {⟨i, j⟩ | I_ij = 1}; F ← ∅
2  while U ≠ ∅ do
3      D ← ∅; V ← 0
4      while exists j ∉ D such that |D ⊕ j| > V do
5          select j ∉ D that maximizes |D ⊕ j|
6          D ← (D ∪ {j})^↓↑
7          V ← |(D^↓ × D) ∩ U|
8      end
9      C ← D^↓
10     ⟨E, F⟩ ← Expansion(⟨C, D⟩ , w)
11     add ⟨C ∪ E, D ∪ F⟩ to F
12     U ← U − (C ∪ E) × (D ∪ F)
13     foreach factor ⟨A, B⟩ ∈ F do
14         if for each ⟨i, j⟩ ∈ A × B with I_ij = 1 there is ⟨G, H⟩ ∈ F − {A, B} with ⟨i, j⟩ ∈ G × H then
15             remove ⟨A, B⟩ from F
16         else
17             foreach j ∈ B − nucleus(B) do
18                 if for each ⟨i, j⟩ ∈ A × B with I_ij = 1 there is ⟨G, H⟩ ∈ F − {A, B} with ⟨i, j⟩ ∈ G × H then
19                     remove j from B
20                 end
21             end
22         end
23     end
24 end
25 return F
```

The variable $\mathcal{U}$, initialized in l. 1, represents in each step of the algorithm the 1s in I that are uncovered by the set $\mathcal{F}$ of factors computed so far, i.e. $\langle i, j\rangle \in \mathcal{U}$ iff $I_{ij} = 1$ and there is no $\langle C, D\rangle \in \mathcal{F}$ with $i \in C$ and $j \in D$. The computation of factors (l. 2–24) proceeds until all 1s in I are covered. Clearly, an alternative stopping criterion is possible, such as stopping after a prescribed number of factors is computed, which corresponds to the DBP problem, or after the overall error does not exceed ε, which corresponds to AFP. In l. 3–8, a formal concept in I covering a large part of $\mathcal{U}$ is selected in a greedy manner: D, set initially to $\emptyset$, represents the constructed concept's intent and V the currently largest number of previously uncovered $\mathcal{U}$ that are covered by the concept corresponding to D; while improvement in coverage is possible (l. 4), one selects the attribute j whose addition to D results in the largest increase in coverage, i.e. in the largest number of elements in $D \oplus j = (D \cup \{y\})^{\downarrow} \times (D \cup \{y\})^{\downarrow\uparrow} \cap \mathcal{U}$ (l. 5), and updates D and V (l. 6–7). The thus resulting concept $\langle C, D\rangle$ is taken as a *nucleus* for which its *expansion* $\langle E, F\rangle$ is then computed by Expansion (Algorithm 5): We add columns with positive gain until no j with positive gain exists. The gain of a candidate column is assessed by taking into account the positive effect of covering 1s, i.e. decrease in E_u, and the negative effect of overcovering 0s, i.e. increase in E_o, amplified by penalty w. The thus expanded factor $\langle C \cup E, D \cup F\rangle$ is then added to $\mathcal{F}$ and the covered entries are removed from $\mathcal{U}$ (l. 11–12). The loop in l. 13–23 attempts to modify existing factors, from the first one to the last: A factor is removed (l. 14–15) if all the 1s covered by it are covered by the other factors; if this is not possible, one removes all columns $j \in B - \mathit{nucleus}(B)$ for which the covered 1s are covered by the remaining factors, thus possibly reducing the part of E_o committed by the column j ($\mathit{nucleus}(B)$ is the intent of the nucleus of $\langle A, B\rangle$).

Algorithm 5: EXPANSION

Input: pair $\langle C, D\rangle$ of $C \subseteq \{1, \ldots, n\}$ and $D \subseteq \{1, \ldots, m\}$, number w
Output: expansion $\langle E, F\rangle$ of $\langle C, D\rangle$

```
1 O ← {⟨i, j⟩ | I_ij = 0}
2 E ← ∅; F ← ∅
3 repeat
4 |   select column j ∉ D ∪ F maximizing gain(j) defined as |(C × (D ∪ F ∪ {j})) ∩ U| − w · |(C × (D ∪ F ∪ {j})) ∩ O)|
5 |   if gain(j) > 0 then
6 |   |   add j to F
7 |   end
8 until F did not change
9 return ⟨E, F⟩
```

4. Evaluation of algorithms

We now provide experimental evaluation of Asso, PANDA$^+$, its variant PANDA$^+$EO (see Section 2.3), 8M, and our new algorithm GRECOND+, which extends our previous algorithm GRECOND [4] for from-below BMF to the realm of general BMF. Due to lack of space, we include Asso, PANDA$^+$, and PANDA$^+$EO as the only representatives of the existing BMF algorithms. Our choice may partly be justified by the fact that Asso represents a nowadays classic BMF algorithm whose comparison to the other existing algorithms on data similar to ours is available in the literature (see the end of Section 1.3) and the fact that PANDA$^+$ is directly relevant to GRECOND+ due to its logic.

4.1. General remarks on the algorithms

In terms of quality of the computed factors, which we assess by the coverage function explained below, the four algorithms compare, put briefly, as follows. Asso delivers a good coverage of data by the first few factors, which corresponds to its being designed for the DBP problem. On the other hand, it often ends up in a state in which a considerable amount of data is left unexplained, i.e. with a considerable error. In contrast, 8M very often has a considerably worse coverage by the first few factors compared to Asso, except for very dense data on which it compares to Asso. On the other hand, 8M tends to perform slightly better than Asso in its capability to eventually generate factors explaining a large portion of data. PANDA$^+$ as well as PANDA$^+$EO as a rule exhibit small coverage compared to the other algorithms and commits an overcoverage error smaller than Asso but larger than the other two algorithms. It also seems that PANDA$^+$ performs better on dense rather than sparse data. GRECOND+ generally outperforms Asso, PANDA$^+$, and 8M in either of the two criteria embodied by DBP and AFP, i.e. good coverage by the first few factors and eventually a good coverage by the whole set of computed factors. The details regarding the quality of delivered factors as well as other characteristics of the algorithms are provided below.

Worth noting is also the following observation which lets us regard GRECOND+ as a parameterized (by w) version of GRECOND:

Observation 3. *With $w \geq n$, where n is the number of objects, and the removal of previous factors (l. 14) omitted,* GRECOND+ *delivers the same output as* GRECOND.

The observation follows from the fact that a weight $w \geq n$ makes the gain of each column non-positive (see above) and thus prohibits any additional column. When expansion includes both columns and rows, the same holds true for $w \geq \max(n, m)$. The role of w is further examined in Section 4.3.

4.2. Datasets

Synthetic data. We use the datasets described in Table 1, which are generated in a scenario similar to [3]. Every Set X*i* consists of 1000 matrices of size 1000×500, obtained as Boolean products $A \circ B$ of $1000 \times k$ and $k \times 500$ matrices A and B which are randomly generated with prescribed densities dens A and dens B and the corresponding varying k. The column dens I contains the average density of the resulting matrix I.

Real data. We used the datasets Mushroom [1], DBLP,[1] Chess [1], DNA [24], Americas-small, Apj, and Firewall 1 [10], described in Table 2, most of which are well known and used as benchmark data in the literature on BMF.

4.3. Performance of algorithms

Even though quality of the delivered decompositions is arguably the most important aspect, we start with brief observations on relative time complexity of the algorithms.

[1] http://www.informatik.uni-trier.de/~ley/db/.

Table 1
Synthetic data.

Dataset	k	dens A	dens B	dens I
Set A1	20	0.052	0.05	0.05
Set A2	20	0.11	0.05	0.10
Set A3	20	0.10	0.08	0.15
Set A4	20	0.11	0.10	0.20
Set B1	30	0.343	0.04	0.05
Set B2	30	0.07	0.05	0.10
Set B3	30	0.08	0.07	0.15
Set B4	30	0.11	0.07	0.20
Set C1	40	0.043	0.03	0.05
Set C2	40	0.07	0.04	0.10
Set C3	40	0.07	0.06	0.15
Set C4	40	0.11	0.05	0.20

Table 2
Real data.

Dataset	Size	$\|I\|$
Americas-small	3477 × 1587	105 205
Apj	2044 × 1164	6 841
DBLP	19 × 6980	40 637
DNA	4590 × 392	26 527
Chess	3196 × 76	118 252
Firewall 1	365 × 709	31 951
Mushroom	8124 × 119	186 852

Time complexity. The fastest of the four algorithms is GreConD+, which is about 1.5 times slower than GreConD—the nowadays fastest BMF algorithm [3]. This is because compared to GreConD, GreConD+ performs an additional expansion of factors and revisits previous factors. To get a concrete idea, note that GreConD implemented in the C language factorizes the Mushroom dataset in the order of seconds on a current ordinary PC. Second fastest is Asso, which is about 3 times slower than GreConD+. A little slower than Asso are PaNDa$^+$ and PaNDa$^+$Eo, which are both about 3.5 times slower than GreConD+. Most time demanding – basically due to its refinement of factors – is 8M, which is about 10 times slower than Asso. Time complexity of the algorithms was never a limiting factor in our evaluation, in which the scenario and size of data represent the ones commonly used in current BMF studies.

Quality of decompositions. To assess quality of decompositions delivered by the algorithms, we employ the following function of $A \in \{0, 1\}^{n\times l}$ and $B \in \{0, 1\}^{l\times m}$ representing the *coverage quality* of the first l factors delivered by the particular algorithm:

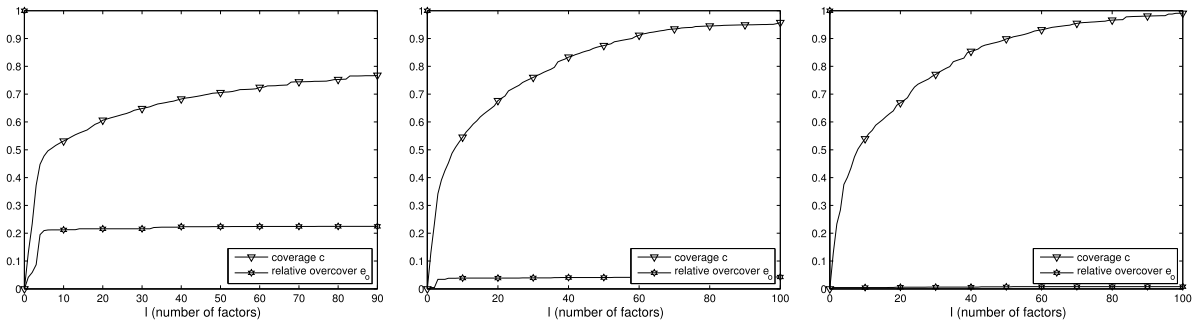
$$c(l) = 1 - E(I, A \circ B)/\|I\|, \tag{5}$$

cf. (2) and Section 2.1. Similar functions are used in [4,3,12,21]. We observe the values of c for $l = 0, \ldots, k$, where k is the number of factors delivered by a particular algorithm. Since we are also interested in the overcover part E_o of E, we observe explicitly the relative error

$$e_o(l) = E_o(I, A \circ B)/\|I\| \tag{6}$$

as well, from which one easily determines the dual part $e_u = E_u(I, A \circ B)/\|I\|$, because $c = 1 - E/\|I\| = 1 - (E_u + E_o)/\|I\| = 1 - e_u - e_o$. In terms of graphs we use, the value of c represents the overall coverage of data by the particular number l of observed factors and e_o represents the part of the residuum $1 - c$ that is due to overcovering. Clearly, for $l = 0$ (no factors added, A and B are "empty") we have $c = 0$ and $e_o = 0$. In accordance with the above requirements, for a good factorization algorithm c should be increasing in l, should have relatively large values even for small l, and it is desirable that for $l = k$ we have $E(I, A \circ B)$ equal or close to 0, i.e. the data is fully or almost fully explained by the k factors computed, in which case c is equal or close to 1.

We set the parameters in a manner generally best suited for each algorithm. In particular, for Asso, we used τ around 0.8, and set w^+ and w^- to 1, which also corresponds to the recommendation by [21]. For PaNDa$^+$, we set its parameters controlling the addition of rows and columns around $\varepsilon_r = \varepsilon_c = 0.1$. For 8M, we observed that small values of *init* such as 3 to 5, which we used, lead to a better performance than the default value $k - 2$ with k being the required number of factors, which observation is in correspondence to how *init* is set in the examples in the BMDP manual; we set *cost* to 1. For GreConD+, we set w to 4, which is a compromise between small values of w resulting in a large overcover E_o and large values of w for which the algorithm tends to commit no E_o at all and tends to behave like GreConD, cf. Observation 3. The impact of w on the behavior of GreConD+ is illustrated in Fig. 2. For each of the selected values of w, i.e. $w = 0.5$, $w = 1$,



(a) $w = 0.5$ (b) $w = 1$ (c) $w = 4$

Fig. 2. Impact of w on GreConD+ for Mushroom dataset.

Table 3
Values of standard deviation for set C4.

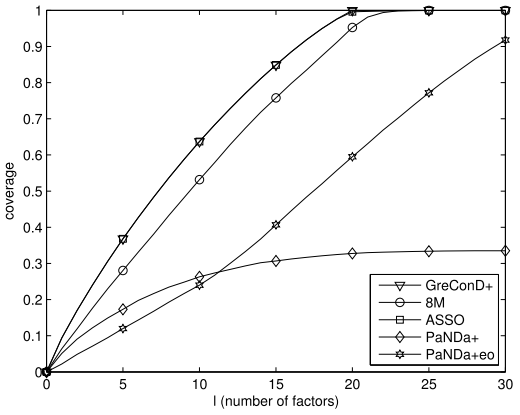
	5	10	15	20	25	30	35	40	45
GreConD+	0.0373	0.0315	0.0268	0.0238	0.0227	0.0206	0.0148	0.0071	0.0000
Asso	0.0373	0.0314	0.0268	0.0238	0.0227	0.0205	0.0135	0.0031	0.0008
8M	0.0372	0.0327	0.0284	0.0257	0.0233	0.0194	0.0137	0.0131	0.0080
PaNDa$^+$	0.0371	0.0315	0.0270	0.0240	0.0229	0.0266	0.0362	0.0376	0.0356
PaNDa$^+$eo	0.0481	0.0480	0.0495	0.0506	0.0501	0.0501	0.0519	0.0500	0.0504

and $w = 4$, we depict there the graphs of c, i.e. coverage (5), and e_o, i.e. relative overcover (6), as functions of the number l of factors. That is, we let GreConD+ compute the set of all factors and then depict the values $c(l)$ corresponding to the first l factors computed; similarly for e_o.

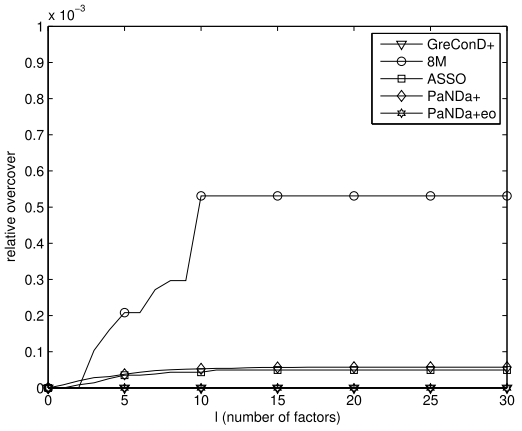
The results of computed decompositions in terms of coverage c and relative overcover e_o for the synthetic data are depicted in Figs. 3–6. For each Set Xi we depict the graph of c and e_o as a function of l, i.e. the first l computed factors, as above. The values are obtained as averages over the 1000 matrices comprised by Xi. One can see that 8M has poorer coverage by the first few factors compared to both Asso and GreConD+, but with increasing density, the gap between 8M and these two algorithms narrows. Note also that for very dense data (density exceeding 0.4), 8M even outperforms Asso; we do not present the graphs because such densities do not appear in real datasets. Comparing Asso and GreConD+, one observes that these algorithms have about the same coverage for the first few factors and that generally, with the increasing number of factors and thus increasing coverage, GreConD+ outperforms Asso. Note also that for the recommended setting of w [21], Asso commits a significantly larger overcover than GreConD+, particularly for denser data. As is clear from the graphs, the worst in terms of coverage of the algorithms is PaNDa$^+$, which is due to the fact that it is not primarily designed for the criteria employed in DBP and AFP problems. PaNDa$^+$ generally produces less factors compared to the other algorithms, the coverage by these factors is smaller than the coverage by the same number of factors of the other algorithms. Particular for dense data, PaNDa$^+$ is usually not capable of achieving a higher coverage. Coverage by PaNDa$^+$eo is better than that by PaNDa$^+$ which is expected due to the cost functions of these two algorithms, but still the coverage by PaNDa$^+$eo is in general considerably worse than those by the other algorithms.

In addition to the average values of coverage $c(l)$ and the relative overcover $e_o(l)$, we also observed their standard deviations, which values we do not include due to lack of space. The smallest values of standard deviations are exhibited by GreConD+ and Asso, which are both comparable in this respect. 8M yields slightly higher values of the observed deviations. Larger values of standard deviations were obtained with PaNDa$^+$ and in particular PaNDa$^+$eo. For illustration, the values of standard deviation for set C4 are shown in Table 3.

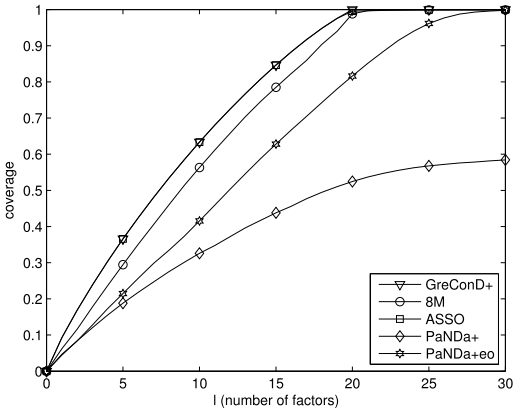
The results for real data, depicted in Table 4, confirm that GreConD+ usually outperforms the other algorithms. The rows in the table represent, for each particular dataset, a given prescribed value of coverage in the first column (in fact, 100c is used instead of c). The next columns show, for each of the algorithms (we do not include PaNDa$^+$eo for space reasons), the number of factors computed that are needed to achieve the prescribed coverage along with the relative overcover e_o committed when the prescribed coverage is reached. Since full coverage, $c = 1$, which corresponds to finding an exact decomposition, is rarely achieved by the BMF algorithms committing overcover, we do not include a row for 100%. We nevertheless include rows for 90% and 95% to examine high coverage. One may observe that regarding the ability to achieve high coverage, GreConD+ clearly outperforms the other algorithms. For coverage by a few first factors, GreConD+ is comparable to Asso and mostly better than 8M and PaNDa$^+$. GreConD+ does not perform well on DNA for reasons unknown to us. Like for synthetic data, Asso commits by far the largest overcover of the algorithms. DBLP is an exception to this rule but DBLP itself is a very specific dataset which is also apparent from the zero overcover committed by Asso on this data. The results confirm that PaNDa$^+$ is, as a rule, not capable of achieving high coverage.



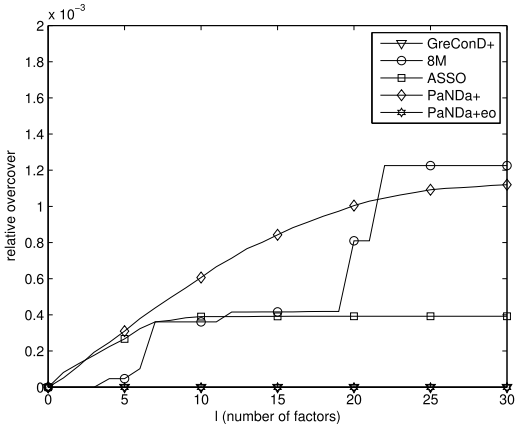
(a) Set A1: coverage c



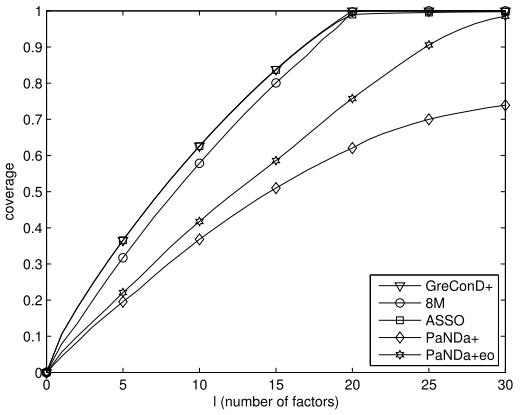
(b) Set A1: relative overcover e_o



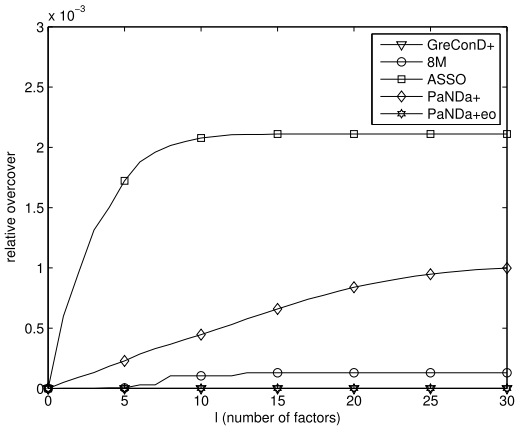
(c) Set A2: coverage c



(d) Set A2: relative overcover e_o

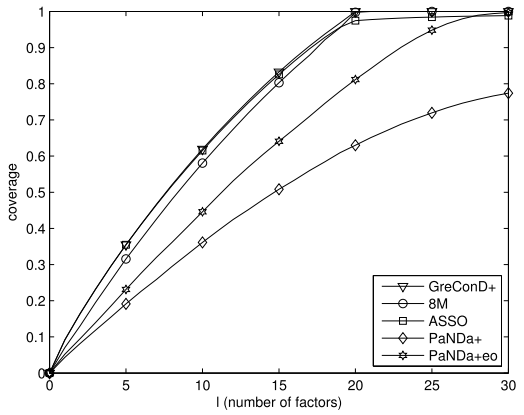


(e) Set A3: coverage c

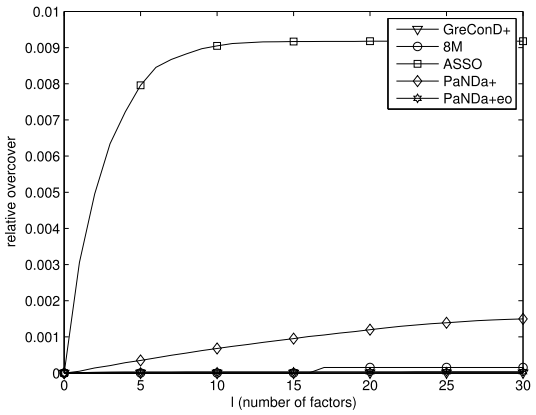


(f) Set A3: relative overcover e_o

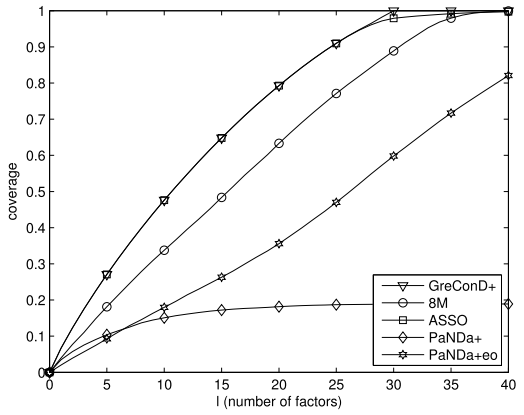
Fig. 3. Factorization of synthetic data (part 1).



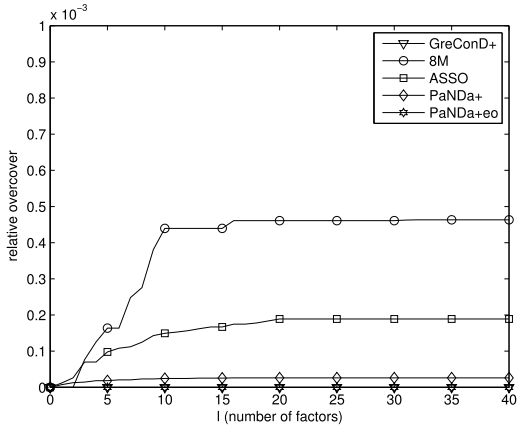
(a) Set A4: coverage c



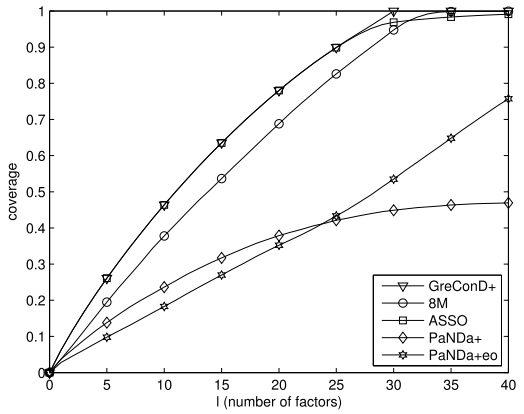
(b) Set A4: relative overcover e_o



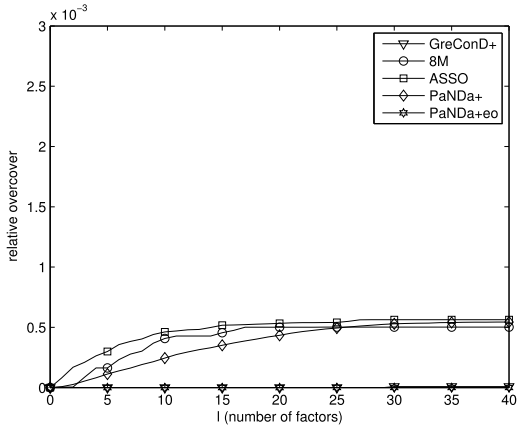
(c) Set B1: coverage c



(d) Set B1: relative overcover e_o



(e) Set B2: coverage c

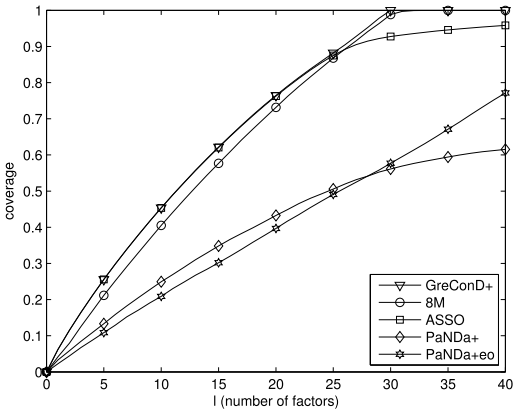


(f) Set B2: relative overcover e_o

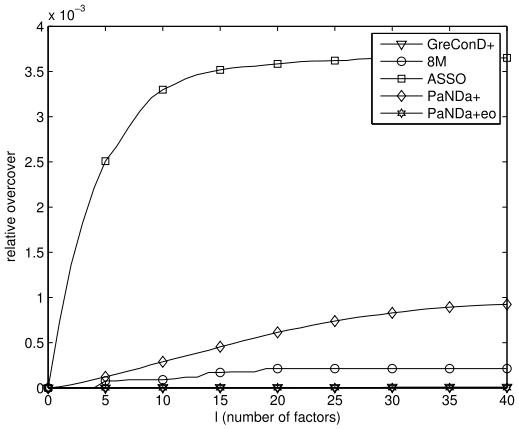
Fig. 4. Factorization of synthetic data (part 2).

5. Robustness of algorithms w.r.t. small error/noise

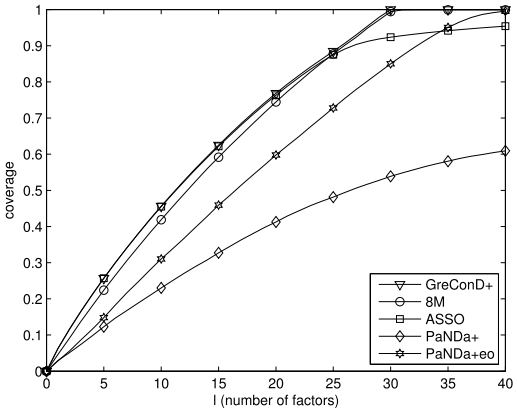
In this part we address the following important problem and present an initial approach to it (a thorough examination of this phenomenon as well as of the ability of algorithms to discover ground truth shall be a subject of a forthcoming paper).



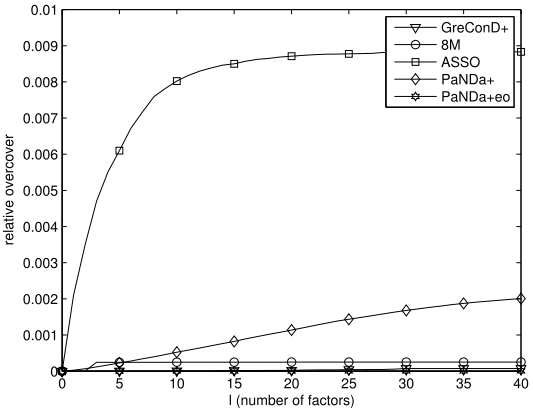
(a) Set B3: coverage c



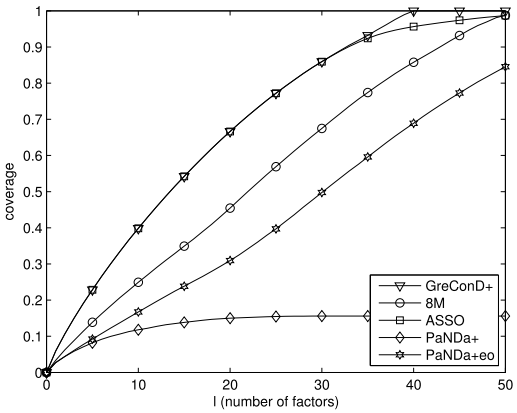
(b) Set B3: relative overcover e_o



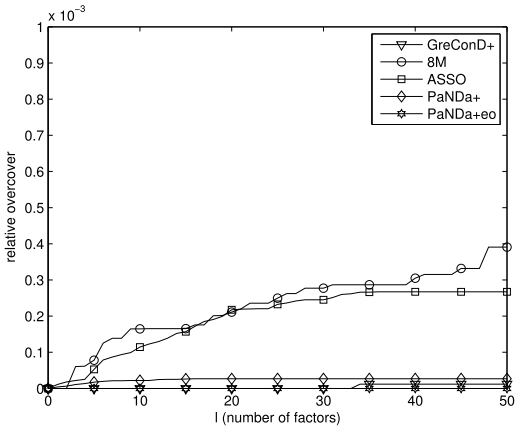
(c) Set B4: coverage c



(d) Set B4: relative overcover e_o



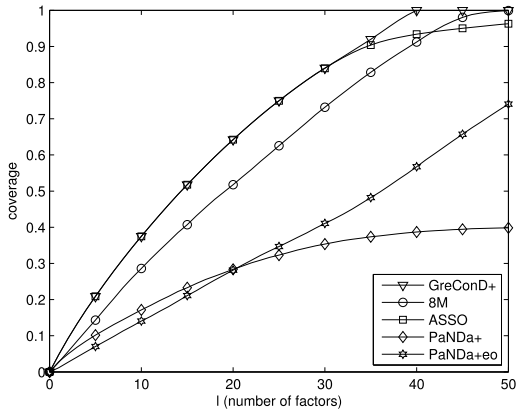
(e) Set C1: coverage c



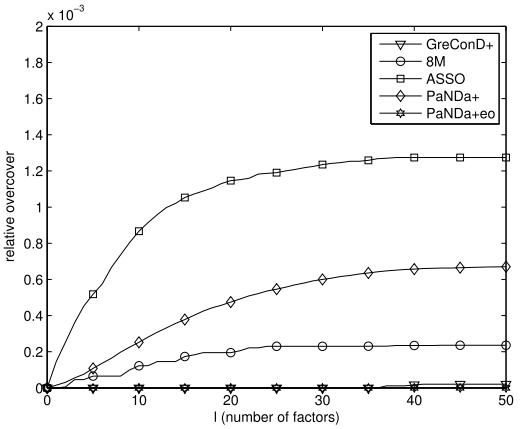
(f) Set C1: relative overcover e_o

Fig. 5. Factorization of synthetic data (part 3).

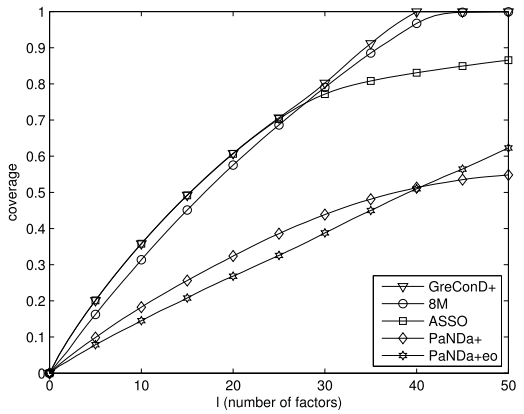
With respect to the rationale for the algorithms computing general BMF presented in Section 1.2, we ask whether and how one may assess the capability of a BMF algorithm to find reasonable factors in data after the data changes slightly. In more detail, suppose a BMF algorithm computes a set $\mathcal{F}$ of factors and the corresponding matrices A and B from a given matrix



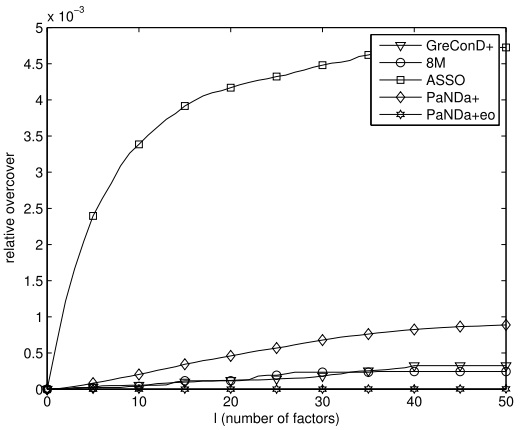
(a) Set C2: coverage c



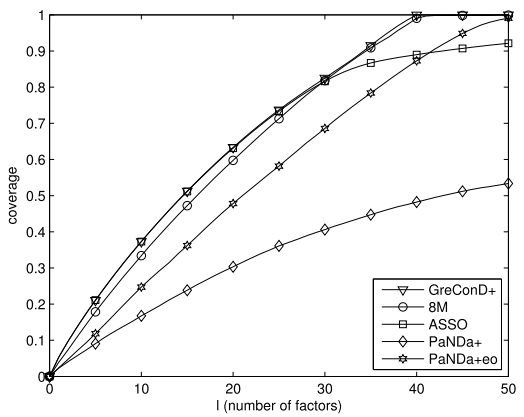
(b) Set C2: relative overcover e_o



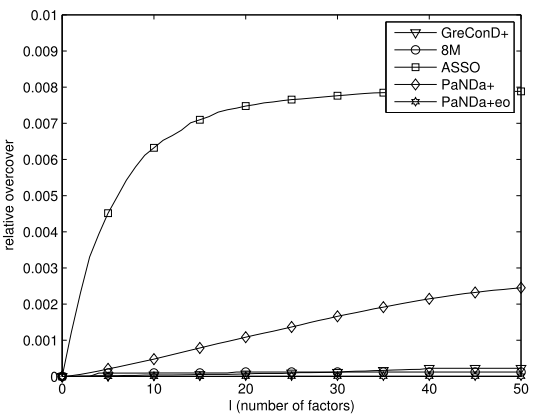
(c) Set C3: coverage c



(d) Set C3: relative overcover e_o



(e) Set C4: coverage c



(f) Set C4: relative overcover e_o

Fig. 6. Factorization of synthetic data (part 4).

I. We change I slightly to a new matrix, I', and let the algorithm compute, this time from I', a new set of factors, $\mathcal{F}'$, and the corresponding matrices A' and B'. We ask whether and to what extent are the two sets of factors, $\mathcal{F}$ and $\mathcal{F}'$, similar. The reason is that in reality, the matrix we analyze may not be the true matrix I but, due to some small error, the modified I'. We

Table 4
Factorization of real data.

Dataset	Coverage (100c%)	GreConD+		8M		Asso		PaNDa+	
		no. fac.	e_o	no. fac.	e_o	no. fac.	e_o	no. fac.	e_o
Amer.-small	25	1	0.00	2	0.49	1	0.00	1	0.00
	50	1	0.00	2	0.49	1	0.00	1	0.00
	75	4	0.05	4	0.68	3	2.54	5	0.08
	90	16	0.07	19	1.19	16	3.77	NA	NA
	95	34	0.08	45	1.46	63	3.96	NA	NA
Apj	25	3	0.00	15	0.42	2	2.00	2	0.00
	50	43	0.45	54	1.08	37	2.12	NA	NA
	75	158	0.89	182	2.82	147	2.31	NA	NA
	90	294	1.10	323	3.55	293	2.32	NA	NA
	95	369	1.21	397	3.64	370	2.32	NA	NA
DBLP	25	3	0.00	3	10.88	3	0.00	NA	NA
	50	7	0.00	7	10.88	6	0.00	NA	NA
	75	11	0.00	17	10.88	11	0.00	NA	NA
	90	16	0.00	NA	NA	15	0.00	NA	NA
	95	18	0.00	NA	NA	17	0.00	NA	NA
DNA	25	19	0.93	10	1.52	7	3.19	7	3.19
	50	65	1.12	27	3.35	27	4.73	27	4.73
	75	174	1.34	84	5.65	78	6.06	78	6.06
	90	269	1.35	205	6.99	207	6.50	207	6.50
	95	315	1.35	NA	NA	NA	NA	NA	NA
Chess	25	2	0.58	1	15.79	1	4.80	1	0.53
	50	4	0.75	1	15.79	2	14.32	4	1.90
	75	16	0.76	17	20.93	19	16.71	20	4.20
	90	33	0.77	NA	NA	NA	NA	NA	NA
	95	47	0.77	NA	NA	NA	NA	NA	NA
Firewall1	25	1	0.00	2	6.28	1	6.32	1	0.00
	50	1	0.00	2	6.28	1	6.32	1	0.00
	75	2	0.00	3	7.08	2	6.32	2	0.02
	90	4	0.02	7	7.16	5	6.99	6	0.03
	95	8	0.02	NA	NA	NA	NA	6	0.03
Mushroom	25	3	0.46	3	9.40	2	14.10	2	0.28
	50	8	0.46	5	11.55	7	15.16	9	0.52
	75	28	0.64	33	14.43	36	15.92	NA	NA
	90	51	0.74	NA	NA	NA	NA	NA	NA
	95	69	0.81	NA	NA	NA	NA	NA	NA

thus want to know whether the algorithm is capable of discovering from I' factors similar to those it would have discovered from I, i.e. whether the algorithm is robust in this sense.

Note that the issue at stake is related to what is often discussed as the problem of noise in data in the BMF literature. However, we find the current treatment of this problem rather unsatisfactory. In particular, the experimental scenario described in this section, which in our view directly addresses the problem discussed, does not appear in the BMF literature.

To assess similarity of the sets $\mathcal{F}$ and $\mathcal{F}'$ of factors, we used both a subjective judgment and a simple quantitative measure *Sim* defined next; both confirm the findings described below. We employ the similarity measure $Sim(\mathcal{F}, \mathcal{F}')$ of two sets of factors, $\mathcal{F}$ and $\mathcal{F}'$, defined as follows. Note first that in accordance with Section 3, we view a factor as represented by a pair $\langle C, D\rangle$ of sets $C \subseteq \{1, \ldots, n\}$ and $D \subseteq \{1, \ldots, m\}$, cf. also Observation 1. The similarity $sim(\langle C_1, D_1\rangle, \langle C_2, D_2\rangle)$ of two factors is defined as the well-known Jaccard index of their corresponding rectangles $C_1 \times D_1$ and $C_2 \times D_2$, i.e. as

$$sim(\langle C_1, D_1\rangle, \langle C_2, D_2\rangle) = \frac{|C_1 \times D_1 \cap C_2 \times D_2|}{|C_1 \times D_1 \cup C_2 \times D_2|}.$$

Finally, we put

$$Sim(\mathcal{F}, \mathcal{F}') = \min\left(\frac{\sum_{c\in\mathcal{F}} \max_{c'\in\mathcal{F}'} sim(c, c')}{|\mathcal{F}|}, \frac{\sum_{c'\in\mathcal{F}'} \max_{c\in\mathcal{F}} sim(c, c')}{|\mathcal{F}'|}\right).$$

That is, $Sim(\mathcal{F}, \mathcal{F}')$ may be regarded as the minimum of two numbers, one expressing the average similarity of a factor in $\mathcal{F}$ and its most similar factor in $\mathcal{F}'$, the other expressing the symmetric value with $\mathcal{F}$ and $\mathcal{F}'$ exchanged. Analogous measures are possible but led to similar conclusions regarding the robustness of algorithms.

For simplicity, we consider general noise only, i.e. both types of changes in the input data are allowed, from 1 to 0 (subtractive) as well as from 0 to 1 (additive). The results assessing robustness of algorithms in the above sense are depicted in Table 5, which contains the results for the well-known Domino dataset [10], a dataset of dimensions 231 × 79. For each particular value of $k = 5, 10, 15$, we obtained a set $\mathcal{F}$ of k factors from the dataset I using a particular algorithm

Table 5
Robustness of algorithms w.r.t. changes in data.

k	Change (%)	GreConD	GreConD+	8M	Asso	PaNDa+
5	0.10	0.146 ± 0.009	0.996 ± 0.006	0.894 ± 0.159	0.996 ± 0.006	0.903 ± 0.162
	0.50	0.074 ± 0.006	0.937 ± 0.088	0.590 ± 0.205	0.962 ± 0.060	0.671 ± 0.206
	1.00	0.056 ± 0.005	0.861 ± 0.100	0.606 ± 0.201	0.876 ± 0.094	0.629 ± 0.186
	1.50	0.051 ± 0.004	0.812 ± 0.081	0.447 ± 0.124	0.712 ± 0.091	0.543 ± 0.128
	2.00	0.047 ± 0.004	0.764 ± 0.080	0.539 ± 0.168	0.634 ± 0.116	0.499 ± 0.135
	3.00	0.043 ± 0.005	0.696 ± 0.091	0.413 ± 0.101	0.546 ± 0.068	0.467 ± 0.130
	5.00	0.041 ± 0.003	0.645 ± 0.070	0.334 ± 0.065	0.411 ± 0.075	0.379 ± 0.084
	10.00	0.031 ± 0.004	0.427 ± 0.099	0.250 ± 0.046	0.219 ± 0.058	0.235 ± 0.040
10	0.10	0.297 ± 0.015	0.997 ± 0.003	0.795 ± 0.143	0.993 ± 0.008	0.937 ± 0.129
	0.50	0.152 ± 0.007	0.953 ± 0.047	0.616 ± 0.080	0.913 ± 0.061	0.768 ± 0.227
	1.00	0.111 ± 0.005	0.866 ± 0.057	0.503 ± 0.090	0.735 ± 0.085	0.520 ± 0.141
	1.50	0.106 ± 0.006	0.878 ± 0.047	0.393 ± 0.060	0.604 ± 0.101	0.592 ± 0.178
	2.00	0.098 ± 0.007	0.820 ± 0.070	0.344 ± 0.061	0.481 ± 0.051	0.587 ± 0.148
	3.00	0.082 ± 0.005	0.676 ± 0.051	0.307 ± 0.045	0.394 ± 0.064	0.416 ± 0.083
	5.00	0.068 ± 0.004	0.496 ± 0.051	0.228 ± 0.033	0.283 ± 0.015	0.330 ± 0.059
	10.00	0.057 ± 0.003	0.415 ± 0.040	0.134 ± 0.015	0.193 ± 0.023	0.226 ± 0.058
15	0.10	0.433 ± 0.023	0.987 ± 0.024	0.752 ± 0.098	0.885 ± 0.060	0.874 ± 0.156
	0.50	0.216 ± 0.012	0.904 ± 0.030	0.516 ± 0.045	0.761 ± 0.042	0.685 ± 0.243
	1.00	0.161 ± 0.011	0.833 ± 0.052	0.433 ± 0.039	0.612 ± 0.061	0.698 ± 0.184
	1.50	0.144 ± 0.009	0.786 ± 0.036	0.338 ± 0.055	0.447 ± 0.053	0.538 ± 0.183
	2.00	0.138 ± 0.009	0.762 ± 0.054	0.291 ± 0.044	0.395 ± 0.044	0.474 ± 0.114
	3.00	0.124 ± 0.007	0.689 ± 0.050	0.231 ± 0.033	0.323 ± 0.023	0.539 ± 0.209
	5.00	0.102 ± 0.009	0.544 ± 0.054	0.147 ± 0.020	0.243 ± 0.028	0.339 ± 0.112
	10.00	0.076 ± 0.005	0.389 ± 0.021	0.112 ± 0.015	0.148 ± 0.016	0.213 ± 0.064

(we used the five algorithms shown in the table). We then added the general type of error (see above) whose value in % is shown in the second column, obtained a new dataset I' from I this way, and computed a set $\mathcal{F}'$ of k factors from I' by the respective algorithm. The entries in the columns corresponding to the particular algorithms contain the values $Sim(\mathcal{F}, \mathcal{F}')$ $\pm$ the standard deviation over 1000 iterations of adding error. As was intuitively expected, GreConD which is restricted to from-below factorizations, does not prove robust. On the other hand, GreConD+ seems the best of the four remaining algorithms, followed by Asso and PaNDa$^+$. For smaller levels of error, Asso outperforms PaNDa$^+$; for large amounts of error, PaNDa$^+$ outperforms Asso, but note that these levels may justly be considered unrealistically high. Of the four algorithms performing general factorizations, 8M proved least robust. Due to restricted space, we do not include results for other datasets but the presented results are representative as far as robustness of the four algorithms is considered.

The approach and our findings presented in this section are to be understood as representing the first steps in pursuing the important problem of robustness of BMF algorithms. Nevertheless, we consider the findings interesting enough to include them in this paper. Let us also note that robustness is but one of the characteristics of a particular algorithm and needs to be considered jointly with other characteristics such as the algorithm's ability to achieve a reasonable coverage in the sense examined in the previous section.

6. Conclusions

We presented a new algorithm, GreConD+, for computing general Boolean matrix factorizations, i.e. factorizations committing both of the two possible kinds of errors. Our aim was to utilize the principal virtue of our previous algorithm for a special type of factorizations, namely its very quick and effective search for factors that commit no E_o-error at all (and thus form a from-below factorization), and extend it to general BMF. In view of the non-symmetry of the E_u- and E_o-parts of the error function (Section 2.1), we construct factors by finding large nuclei of factors with zero E_o-error (thus decreasing significantly E_u), which are then expanded to general factors in such a way that an additional decrease in E_u ensues at the expense of committing reasonable E_o which is emphasized via a penalty weight w for E_o. Furthermore, the already constructed factors are being revisited and modified or even deleted in view of the subsequently constructed factors, which results in a decrease in E_o. This step is motivated by 8M, an old but nowadays virtually unknown BMF algorithm whose complete description we also provide, even though our revisiting of factors is very different from 8M's. GreConD+ outperforms 8M as well as the well-known BMF algorithms Asso and PaNDa$^+$ on most of the datasets employed in our experimental evaluation. Importantly, it tends to be more robust than Asso, PaNDa$^+$, and 8M in that a factorization obtained from a given matrix tends to be similar to a factorization obtained from a slightly modified version of that matrix. This is an important feature because robustness in the above sense is one of the main motivations for exploring general BMF algorithms.

Further research includes the following topics. First, a detailed experimental evaluation of the presented and other existing BMF algorithms, including a study of the impact of the parameters involved. Second, examination of further strategies to obtain good general BMF algorithms that follow our basic idea to concentrate first on finding factors committing

no overcovering, then extending them carefully to factors with a possible overcovering, and at the same time revisiting and modifying the previously generated factors in view of the new ones. Furthermore, a study in more detail of the phenomenon of robustness of BMF algorithms, including the design of appropriate measures of similarity of factors and sets of factors as well as a thorough experimental study. Last but not least, a study of general matrix decompositions with ordinal data [2,5,6], i.e. decompositions not restricted to from-below approximations.

Acknowledgments

Supported by Grant No. GA15-17899S of the Czech Science Foundation. R. Belohlavek acknowledges support by the ECOP (Education for Competitiveness Operational Programme) project no. CZ.1.07/2.3.00/20.0059, which is co-financed by the European Social Fund and the state budget of the Czech Republic. The present work was done during the sustainability period of this project. Martin Trnecka also acknowledges partial support by Grant No. PrF_2016_027 of IGA of Palacký University Olomouc.

References

[1] K. Bache, M. Lichman, UCI Machine Learning Repository [http://archive.ics.uci.edu/ml], Irvine, CA: University of California, School of Information and Computer Science, 2013.
[2] R. Belohlavek, Optimal decompositions of matrices with entries from residuated lattices, J. Logic Comput. 22 (6) (2012) 1405–1425.
[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 (preliminary version in Proc. SCIS & ISCIS 2006).
[5] R. Belohlavek, V. Vychodil, Attribute dependencies for data with grades I, Int. J. Gen. Syst. 45 (7-8) (2016) 864–888.
[6] R. Belohlavek, V. Vychodil, Attribute dependencies for data with grades II, Int. J. Gen. Syst. 46 (1) (2017) 66–92.
[7] R.A. Brualdi, H.J. Ryser, Combinatorial Matrix Theory, Cambridge University Press, 1991.
[8] B.A. Davey, H.A. Priestley, Introduction to Lattices and Order, second ed., Cambridge University Press, 2002.
[9] W.J. Dixon (Ed.), BMDP Statistical Software Manual, University of California Press, Berkeley, CA, 1992.
[10] A. Ene, W. Horne, N. Milosavljevic, P. Rao, R. Schreiber, R.E. Tarjan, Fast exact and heuristic methods for role minimization problems, in: Proc. SACMAT 2008, pp. 1–10.
[11] B. Ganter, R. Wille, Formal Concept Analysis: Mathematical Foundations, Springer, Berlin, 1999.
[12] F. Geerts, B. Goethals, T. Mielikäinen, Tiling databases, in: Proc. Discovery Science 2004, pp. 278–289.
[13] S. Karaev, P. Miettinen, J. Vreeken, Getting to know the unknown unknowns: destructive-noise resistant Boolean matrix factorization, in: Proc. SIAM DM 2015, pp. 325–333.
[14] K.H. Kim, Boolean Matrix Theory and Applications, M. Dekker, NY, 1982.
[15] S.O. Kuznetsov, S. Obiedkov, Comparing performance of algorithms for generating concept lattices, J. Exp. and Theor. Artif. Intell. 14 (2002) 189–216.
[16] H. Lu, J. Vaidya, V. Atluri, Y. Hong, Constraint-aware role mining via extended Boolean matrix decomposition, IEEE Trans. Dependable Secure Comput. 9 (5) (2012) 655–669.
[17] C. Lucchese, S. Orlando, R. Perego, Mining top-K patterns from binary datasets in presence of noise, in: Proc. SIAM DM 2010, pp. 165–176.
[18] C. Lucchese, S. Orlando, R. Perego, A Unifying framework for mining approximate top-k binary patterns, IEEE Trans. Knowl. Data Eng. 26 (12) (2014) 2900–2913.
[19] P. Miettinen, The Boolean column and column-row matrix decompositions, Data Min. Knowl. Discov. 17 (2008) 39–56.
[20] P. Miettinen, Sparse Boolean matrix factorizations, in: Proc. IEEE ICDM 2010, pp. 935–940.
[21] P. Miettinen, T. Mielikäinen, A. Gionis, G. Das, H. Mannila, The discrete basis problem, IEEE Trans. Knowl. Data Eng. 20 (10) (2008) 1348–1362.
[22] P. Miettinen, J. Vreeken, Model order selection for Boolean matrix factorization, in: Proc. ACM SIGKDD 2011, pp. 51–59.
[23] S.D. Monson, S. Pullman, R. Rees, A survey of clique and biclique coverings and factorizations of (0,1)- matrices, Bull. ICA 14 (1995) 17–86.
[24] S. Myllykangas, et al., DNA copy number amplification profiling of human neoplasms, Oncogene 25 (55) (2006) 7324–7332.
[25] D.S. Nau, Specificity covering, Tech. Rep. CS-1976-7, Duke University, 1976.
[26] D.S. Nau, G. Markowsky, M.A. Woodbury, D.B. Amos, A mathematical analysis of human leukocyte antigen serology, Math. Biosci. 40 (1978) 243–270.
[27] J. Outrata, Boolean factor analysis for data preprocessing in machine learning, in: Proc. ICMLA 2010, pp. 899–902.
[28] G. Schmidt, Relational Mathematics, Cambridge University Press, 2011.
[29] L. Stockmeyer, The set basis problem is NP-complete, Tech. Rep. RC5431, IBM, Yorktown Heights, NY, USA, 1975.
[30] J. Vaidya, V. Atluri, Q. Guo, The role mining problem: finding a minimal descriptive set of roles, in: Proc. SACMAT 2007, pp. 175–184.
[31] Y. Xiang, R. Jin, D. Fuhry, F.F. Dragan, Summarizing transactional databases with overlapped hyperrectangles, Data Min. Knowl. Discov. 23 (2011) 215–251.