Knowledge-Based Systems 335 (2026) 115190

Contents lists available at ScienceDirect

Knowledge-Based Systems

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

Reduce before you factorize: A simple scheme for Boolean matrix factorization

Radim Belohlavek, Jakub Juracka [*]

Department of Computer Science, Palacký University Olomouc, Czech Republic

ARTICLE INFO

Keywords:
Boolean data
Factorization
Redundancy
Reduction
Sampling

ABSTRACT

We propose a simple idea that enables a speed-up of existing algorithms for Boolean matrix factorization. It consists in a straightforward redundancy-removing transformation of the input data and an appropriate modification of the factorization algorithm. Examination of real-world data used for benchmarking reveals that most are amenable to such a transformation, rendering the idea practically significant. Experimental evaluation confirms that our approach results in a significant speed-up of factorization algorithms. We also discuss the implications of our findings for factorization of large Boolean data and outline topics for future research.

1. Introduction

1.1. Problem setting and our contribution

In the past fifteen years or so, research in Boolean matrix factorization (BMF) has focused on developing new algorithms; see [1] for a recent overview. Various approaches have been proposed that are able to compute precise or approximate factorizations of a given input Boolean matrix. To cope with the NP-hardness of the proposed variants of BMF, the existing algorithms make use of various heuristics and compute the factorizations directly from the input Boolean matrices.

In this paper, we propose to utilize a scheme, not employed in the previous studies on BMF, that consists in transforming the input data, factorizing the transformed data, and retrieving from the computed factorization of the transformed data the resulting factorization of the original data. While such a scheme may result in various particular factorization methods depending on the kind of transformation of the input data and the particular factorization algorithm, we examine a particular transformation that removes a simple form of redundancy from the input data, namely redundant rows and columns from the input Boolean matrix. To factorize the thus transformed data, we employ a natural modification of a given factorization algorithm, which we demonstrate on the widely known Asso and GreConD algorithms.

Our experimental evaluation confirms a practical relevance of the proposed approach. Namely, an examination of real Boolean data used in the literature on BMF reveals that most data is considerably redundant in the above sense, which has not been observed in the previous studies on BMF. As a result, such data is amenable to the proposed approach, which results in a remarkable speed-up in factorizing the data. The proposed approach is also relevant for the possibility of sampling the input data, whose particular instance appeared in the literature. In view of our findings and additional experimental observations, we point out shortcomings of this existing work. In addition, we propose topics for future research and support them with preliminary experimental results.

1.2. Notation

Denote by $\{0,1\}^{n\times m}$ the set of all $n \times m$ Boolean matrices, i.e., matrices, denoted in our paper by I, that have n rows, m columns, and whose entries I_{ij} equal 0 or 1. Furthermore, the ith row and the jth column of I shall be denoted by $I_{i\_}$ and $I_{\_j}$, respectively, for $i = 1,\dots,n$ and $j = 1,\dots,m$. As a rule, the rows and the columns represent objects (exemplars, items) and attributes (properties, features), respectively, and $I_{ij} = 1$ indicates that the object represented by the row i, such as a particular organism or a particular product, has the attribute represented by the column j, such as "to breathe" or "to contain lithium."

The basic problem in BMF consists in finding for a given object×attribute matrix $I \in \{0,1\}^{n\times m}$ an object×factor matrix $A \in \{0,1\}^{n\times k}$ and a factor×attribute matrix $B \in \{0,1\}^{k\times m}$ such that

k is reasonably small and $I \approx A \circ B$,

i.e., I approximately equals the well-known Boolean matrix product $A \circ B$ defined by

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

[*] Corresponding author.
E-mail addresses: radim.belohlavek@acm.org (R. Belohlavek), jakub.juracka@upol.cz (J. Juracka).

https://doi.org/10.1016/j.knosys.2025.115190
Received 10 August 2024; Received in revised form 9 October 2025; Accepted 21 December 2025
Available online 26 December 2025

The approximate equality $I \approx A \circ B$ of I and $A \circ B$ is assessed by the metric $E(\cdot,\cdot)$ induced by the matrix L_1-norm [2], which turns into

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

The value $E(I, A\circ B)$ gets split into two conceptually different types of error,

$$E(I, A\circ B) = E_u(I, A\circ B) + E_o(I, A\circ B),$$

where the E_u and E_o are the numbers of entries of I that are uncovered by A and B and those that are overcovered, respectively, i.e.,

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

Both the exact and approximate factorization and the error of factorization are found in the supplementary material [3, example 1].

Two particular optimization problems result from the above basic problem [4,5]: the approximate factorization problem (AFP), in which a threshold ε is prescribed and the smallest number k of factors is sought for which $E(I, A\circ B) \leq \varepsilon$, and the discrete basis problem (DBP) in which a number k is prescribed and k factors are sought for which $E(I, A\circ B)$ is as small as possible. A number of BMF algorithms have been designed [1], of which we employ GreConD [6, Algorithm 2] and Asso [5], which are regarded as the representative algorithms for the AFP and DBP, respectively.

2. Redundancy in real Boolean data

In the literature on BMF, a number of real datasets have been explored both for the purpose of factor-analyzing them and utilizing them for benchmarking the BMF algorithms. In this section, we present the most popular of these datasets and examine them for redundancy, which is the crucial property for our purpose. The datasets along with their relevant characteristics are provided in Table 1. Each dataset, i.e., a Boolean matrix I, is represented by single table row, with the first four columns containing the name, the dimension $\dim(I)$, the number $||I||$ of 1s, and the density of the matrix I, respectively. The dimension $n \times m$ indicates that I has n rows and m columns, and the density is the ratio $\frac{||I||}{n\cdot m}$. The next two columns contain the dimension $\dim(\mathrm{Cla}(I))$ of the modified matrix denoted $\mathrm{Cla}(I)$, i.e., the one obtained from I by the removal of redundant rows and columns as described in the next section, and the redundancy defined as $(1 - \frac{n_c\cdot m_c}{n\cdot m})\cdot 100$, where n_c and m_c are the number of rows and columns of $\mathrm{Cla}(I)$. That is, a redundancy of 70 means that the modified matrix $\mathrm{Cla}(I)$ only contains $100 - 70 = 30\%$ of the entries of the original matrix I, i.e., 70% of I got removed.

We now provide a description of the involved datasets with details that are not commonly available in the literature.

- The Advertisement dataset [7], created in 1998 [8], comprises 3,279 internet images, each described by 1,555 binary attributes supplemented by three continuous ones (width, height and aspect ratio) and one class variable distinguishing advertisements from non-advertisements. Each of the binary attributes indicates the presence of keywords in text components of an advertisement such as URL.[1]
- Several datasets, including Americas Large/Small, APJ, Domino, Emea, Healthcare, and Customer, represent unspecified user-permission relationships and were initially utilized for addressing the Role Mining Problem [9]. The Americas Large and Americas Small datasets, collected from Cisco firewalls, contains 3,485 and 3,477 users, respectively, each described by 10,127 and 1,586 access rights to network services. The compact Healthcare dataset, obtained from the U.S. Veterans Administration, outlines 46 healthcare permissions that may be assigned to 46 certified providers.
- The Firewall 1 and Firewall 2 datasets [9] represent the outcomes of a checkpoint firewall analysis algorithm describing the reachability of specific service packets (such as `http`) from the source IP address range (row) to the destination range (column), with dimensions of 365×709 and 325×590, respectively.
- The DBLP dataset comes from the widely used computer science bibliographic database.[2] The dataset has been collected by Miettinen [10], and includes information about contributions to 19 selected premier conferences (columns) by 6,980 authors (rows) who contributed at least two papers to the selected conferences.
- The Chess dataset [7], known also as KRKPA7, is a well-known dataset, used for a classification of chess endgame scenarios. It employs a binary class label to indicate whether white can secure a win. The 3,196 unique board positions are described by 76 attributes determining the pieces' positions on the chessboard [11].
- The DNA dataset contains information about DNA copy number amplification profiles in human neoplasms extracted from a publicly accessible data collection [12]. It features a matrix with 4,590 tumor cases, each characterized by 392 chromosomal loci indicating the presence of amplifications as hallmarks of advanced tumors.
- Mushroom [7] is a well-known example of a classification dataset which represents 22 physical traits of 8,124 hypothetical gilled mushroom samples in the Agaricus and Lepiota families. These traits, such as size, color, or odor, are expressed by a total of 119 binary attributes.
- The NSF dataset was collected in [10] from the data accessible from the National Science Foundation in the form of bags of words of the abstracts of projects submitted for funding. The dataset represents 12,841 abstracts (rows) described by a set of 4,894 words (columns) resulting from a stemming process and excluding those appearing in more than 999 or fewer than 10 abstracts.
- Paleo[3] contains information on 501 fossils discovered in various paleontological sites across Europe. The data, collected in December 2008, has been processed to highlight 139 distinct characteristics of these fossils [10].
- Post (Post-Operative Patient) [7] outlines the health status of patients after surgery, detailing their condition through categorically assessed factors such as temperature, oxygen saturation, and blood pressure stability. Altogether, eight categorical health measurements, represented by 25 Boolean attributes, are used to evaluate the condition of 90 patients in a postoperative recovery area.
- Servo's description [7] is somewhat ambiguous. The dataset is supposed to represent a simulation involving four components of a servo system: An amplifier, a motor, a lead screw nut, and a sliding carriage. Each setup is characterized by a combination of two categorical and two integer-valued attributes, along with the system's rise time. Disregarding the final continuous attribute and converting the four attributes, which include three attributes with five unique values each and one with four unique values, results in a total of 19 Boolean attributes (columns) which represent 167 initial system configurations (rows).
- Shuttle [7] (Shuttle Landing Control) features categorical data with 15 rows corresponding to conditions under which autolanding would be preferable to manual control of a spacecraft. These are described by 7 numerical features with a total of 23 unique values represented by the columns of the corresponding Boolean matrix.
- The TicTacToe dataset captures 953 unique arrangements of circles and crosses on a 3×3 game board at the end of a tic-tac-toe game. Each configuration is detailed using 30 attributes, which specify whether a circle, cross, or blank space occupies each position on the board, along with a target variable indicating "secured win for

[1] The meaning of the two binary attributes (1,556th and 1,557th) is not clear.

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

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

Table 1
Real Boolean data used in BMF.

data	dim(I)	$\|\|I\|\|$	density	dim(CLA(I))	redundancy
Advertisement	3,279 × 1,557	45,139	0.009	1,989 × 763	70%
Americas Large	3,485 × 10,127	185,294	0.005	432 × 1,354	98%
Americas Small	3,477 × 1,586	105,205	0.019	259 × 349	98%
APJ	2,044 × 1,164	6841	0.003	564 × 578	86%
Customer	10,961 × 277	45,427	0.015	5,656 × 276	49%
DBLP	6,980 × 19	17,173	0.130	890 × 19	87%
DNA	4,590 × 392	26,527	0.015	1,316 × 371	73%
Domino	79 × 231	739	0.040	23 × 38	95%
Emea	35 × 3,046	7220	0.068	34 × 263	92%
Firewall 1	365 × 709	31,951	0.124	90 × 86	97%
Firewall 2	325 × 590	36,428	0.190	11 × 11	99%
Healthcare	46 × 46	1486	0.702	18 × 19	84%
Chess	3,196 × 76	118,252	0.487	3,196 × 76	0%
Mushroom	8,124 × 119	186,852	0.193	8,124 × 113	5%
NSF	12,841 × 4,894	564,462	0.009	12,658 × 4,893	1%
Paleo	501 × 139	3537	0.051	471 × 139	6%
Post	90 × 25	720	0.320	75 × 23	23%
Servo	167 × 19	668	0.211	167 × 19	0%
Shuttle	15 × 23	105	0.304	15 × 22	4%
Tic Tac Toe	958 × 30	9580	0.333	958 × 30	0%
Zoo	101 × 28	862	0.305	59 × 26	46%

cross." The last attribute is empty (full of zeros) with an unclear meaning.

- Zoo [7] represents a Boolean dataset describing 101 animals characterized by 14 Boolean and two numerical attributes. The Boolean attributes correspond to features such as presence of hair, presence of tail, or venomousness. The other attributes, describing a number of legs and representing animal classes such as mammals, birds or reptiles, have 14 unique values, resulting in 28 Boolean attributes in total.

3. New scheme for factorization

3.1. Clarification as the proposed reduction of input matrix

We employ a simple reduction which consists in removing duplicate rows and columns. This reduction is well known in formal concept analysis [13] where it is known as clarification, which term we use below. In particular, we employ a variant that clarifies an input Boolean matrix $I \in \{0,1\}^{n\times m}$ by removing from I all the duplicate rows and columns so that only the first occurrences are kept in the resulting matrix J.

More precisely, duplicity obviously induces equivalence relations $\equiv_X$ and $\equiv_Y$ on the sets

$$X = \{1,\dots,n\} \quad \text{and} \quad Y = \{1,\dots,m\}$$

of row and column indices of I, respectively, i.e.,

$$i_1 \equiv_X i_2 \text{ iff } I_{i_1\_} = I_{i_2\_} \quad \text{and} \quad j_1 \equiv_Y j_2 \text{ iff } I_{\_j_1} = I_{\_j_2},$$

for $i_1, i_2 \in X$ and $j_1, j_2 \in Y$. That is, $i_1 \equiv_X i_2$ means that rows i_1 and i_2 of I are equal, the equivalence class $[i]_{\equiv_X}$ consists of indices of all the rows equal to row i, and $\min[i]_{\equiv_X}$ is the index of first such row in I; the same holds for the columns. The equivalence classes $[i]_{\equiv_X}$ and $[j]_{\equiv_Y}$ hence correspond to the rows and the columns of the clarified matrix J, respectively, and represent the rows and the columns of I to be preserved: The numbers n_c and m_c of rows and columns of J thus equal the numbers of equivalence classes of $\equiv_X$ and $\equiv_Y$, respectively, i.e.,

$$n_c = |X/\equiv_X| \text{ and } m_c = |Y/\equiv_Y|.$$

In addition, the ordering of rows and columns in J coincides with the ordering of the first occurrences of their counterparts in I: If the first occurrence of row i_1 precedes the first occurrence of row i_2 in I, i.e., $\min[i_1]_{\equiv_X} < \min[i_2]_{\equiv_X}$, then the counterpart of i_1 precedes the counterpart of i_2 in J; the same applies to the columns. Moreover, it turns out useful for our purpose to denote for each row index $i = 1,\dots,n_c$ of J by $rpos(i)$ the index of the first occurrence of the counterpart of the row $J_{i\_}$ in I, i.e., the row position of I from which the row i of J originates; similarly for the column indices $j = 1,\dots,m_c$ and $cpos(j)$. Clarification is illustrated in the supplementary material [3, example 2].

Note that removing duplicate rows may be performed by a componentwise sorting of the rows followed by a single pass through the rows during which the duplicates are removed. The single pass may even be skipped when removing duplicity appropriately within the sorting process. This is a standard procedure in matrix computations and database query processing with a time complexity in $O(mn \log n)$; see [14] for details. Duplicate columns are treated dually.

3.2. Factorizing the clarified matrix and the need to modify a factorization algorithm

If $J \in \{0,1\}^{n_c\times m_c}$ is the clarified version of an input matrix $I \in \{0,1\}^{n\times m}$, as described in the previous section, one may consider computing first an exact or approximate factorization of the smaller matrix J, and "extend" it to obtain a factorization of the input matrix I. That is, if $J \approx C \circ D$, where $C \in \{0,1\}^{n_c\times k}$ and $D \in \{0,1\}^{k\times m_c}$ are the object-factor and factor-attribute matrices computed for J, respectively, 1 aims to obtain from C and D matrices $A = \text{ext}(C)$ and $B = \text{ext}(D)$ satisfying $I \approx A \circ B$.

While this procedure may be considered even for a more general kind of reduction of the input matrix, in the case of clarification, one may utilize straightforward extensions

$$C \in \{0,1\}^{n_c\times k} \mapsto \text{ext}(C) \in \{0,1\}^{n\times k} \quad \text{and}$$
$$D \in \{0,1\}^{k\times m_c} \mapsto \text{ext}(D) \in \{0,1\}^{k\times m},$$

that consist in taking for each row $\text{ext}(C)_{i\_}$ of $\text{ext}(C)$ the corresponding row $C_{i^*\_}$ of C, and for each column $\text{ext}(D)_{\_j}$ of $\text{ext}(D)$ the corresponding column $D_{\_j^*}$ of D. That is, we consider the extensions defined for rows $i = 1,\dots,n$ and columns $j = 1,\dots,m$ by

$$\text{ext}(C)_{il} = C_{i^*l} \text{ where } i^* = rpos^{-1}(\min[i]_{\equiv_X}), \tag{1}$$

$$\text{ext}(D)_{lj} = D_{lj^*} \text{ where } j^* = cpos^{-1}(\min[j]_{\equiv_Y}), \tag{2}$$

for every $l = 1,\dots,k$. Note that the definition of i^* in (1) says that i^* is the index of the row of J to which the row i of I (and all rows equivalent to row i) got reduced by the considered clarification; symmetrically, for j^* and the columns.

Since each factor $l = 1,\dots,k$ behind the factorization $J \approx C \circ D$ may be identified with a pair consisting of the column $C_{\_l}$ of C and the row

$D_{l\_}$ of D (Section 1.2), the extensions described in (1) and (2) may be understood as follows: Each factor of J, i.e., column $C_{\_l}$ and row $D_{l\_}$, gets expanded to a possible factor of I by copying each value C_{i^*l} to the corresponding positions in the column $\text{ext}(C)_{\_l}$, i.e., to the positions $i \in [rpos(i^*)]$ in $\text{ext}(C)_{\_l}$, and dually for D. For an example demonstrating the preceding extensions, see the supplementary material [3, example 3].

One can easily check that in the just mentioned example, $\text{ext}(C)\circ\text{ext}(D)$ is the exact decomposition of I, i.e., $\text{ext}(C)\circ\text{ext}(D) = I$. This is always the case when $J = C\circ D$, as shown below in corollary 1. This corollary follows from the next theorem in which the error of the resulting factorization of I by $\text{ext}(C)$ and $\text{ext}(D)$ is derived in a general case:

Theorem 1. *Let $I \in \{0,1\}^{n\times m}$ and consider the clarified matrix $J \in \{0,1\}^{n_c\times m_c}$ obtained from I as described in Section 3.1. For any $C \in \{0,1\}^{n_c\times k}$ and $D \in \{0,1\}^{k\times m_c}$, and the corresponding extended $\text{ext}(C) \in \{0,1\}^{n\times k}$ and $\text{ext}(D) \in \{0,1\}^{k\times m}$ we have*

$$E(I,\text{ext}(C)\circ\text{ext}(D)) = \sum_{\substack{i,j=1;\\ J_{ij}\neq(C\circ D)_{ij}}}^{n_c,m_c} |[rpos(i)]_{\equiv_X}|\cdot|[cpos(j)]_{\equiv_Y}| \tag{3}$$

Proof. The proof is straightforward, the basic argument being that an error in $E(J,C\circ D)$ caused by the entry $\langle i,j\rangle$ in J, i.e., $J_{ij} \neq (C\circ D)_{ij}$, gets multiplied by the factor $|[rpos(i)]_{\equiv_X}|\cdot|[cpos(j)]_{\equiv_Y}|$ as regards the contribution to the error $E(I,\text{ext}(C)\circ\text{ext}(D))$. Namely, due to the construction of $\text{ext}(C)$ and $\text{ext}(D)$, the same configuration at the entry $\langle i,j\rangle$ in J, i.e., $J_{ij} \neq (C\circ D)_{ij}$, appears in all the entries of I corresponding to the rows equivalent to $rpos(i)$ and the columns equivalent to $cpos(j)$. □

Corollary 1. *With the same assumptions as in theorem 1,*

$J = C\circ D$ implies $I = \text{ext}(C)\circ\text{ext}(D)$.

Proof. Immediate from theorem 1 since $J = C\circ D$ and $I = \text{ext}(C)\circ\text{ext}(D)$ mean $E(J,C\circ D) = 0$ and $E(I,\text{ext}(C)\circ\text{ext}(D)) = 0$, and since $E(J,C\circ D) = 0$ implies that there is no summand in (3). □

Analogous relationships are easily obtained for E_u and E_o; for instance,

$$E_o(I,\text{ext}(C)\circ\text{ext}(D)) = \sum_{\substack{i,j=1;\\ J_{ij}<(C\circ D)_{ij}}}^{n_c,m_c} |[rpos(i)]_{\equiv_X}|\cdot|[cpos(j)]_{\equiv_Y}|.$$

The meaning of the theorem 1 is illustrated in the supplementary material [3, example 4]. Note also that it directly follows from corollary 1 and the involved considerations that the Boolean ranks of the input matrix I and the reduced matrix J are equal (recall that a Boolean rank of I is the smallest number k of factors for which an exact decomposition $I = A\circ B$ exists for some $A \in \{0,1\}^{n\times k}$ and $B \in \{0,1\}^{k\times m}$).

The above discussion suggests the following procedure for factorizing a Boolean matrix I:

1. Compute a clarified J from I as described in Section 3.1;
2. Compute an exact or approximate factorization of J into C and D using an established BMF algorithm Alg;
3. Return $\text{ext}(C)$ and $\text{ext}(D)$ given by (1) and (2).

This procedure has two convenient properties: First, it is faster than a direct factorization of I, if I contains redundant rows and columns. Second, $I = \text{ext}(C)\circ\text{ext}(D)$ whenever $J = C\circ D$.

However, the procedure has a significant shortcoming: For one, it may deliver a factorization that is different from the one obtained by a direct factorization of I by Alg, hence the procedure may not be regarded as speeding up Alg. In addition, and more importantly, it may deliver a factorization that is considerably worse in terms of coverage of data by factors compared to the factorization obtained by Alg. That the described shortcoming indeed materializes on real datasets is demonstrated in detail in Section 4.3.1.

It nevertheless turns out that the shortcoming may be eliminated by employing a modified version of Alg in step 2 of the above scheme. This is worked out in Sections 3.3 and 3.4, in which we present the modified schemes for the two prototypical algorithms for the AFP and the DBP problems, namely GreConD and Asso.

3.3. Extended GreConD

Since the original GreConD [6, Algorithm 2] utilizes formal concepts associated to the input Boolean matrix I, we need to recall the notions involved. Let $X = \{1,\dots,n\}$ and $Y = \{1,\dots,m\}$ denote the set of objects and attributes, respectively. Each $n\times m$ Boolean matrix I induces the so-called concept forming operators ${}^{\uparrow_I} : 2^X \to 2^Y$ and ${}^{\downarrow_I} : 2^Y \to 2^X$, defined for $A \subseteq X$ and $B \subseteq Y$ by

$$A^{\uparrow_I} = \{j \in Y \mid I_{ij} \text{ for each } i \in A\} \text{ and } B^{\downarrow_I} = \{i \in X \mid I_{ij} \text{ for each } j \in B\}.$$

That is, $A^{\uparrow_I}$ is the set of all attributes shared by all the objects in A, and $B^{\downarrow_I}$ consists of all objects sharing all the attributes in B. A pair $\langle A,B\rangle$ of $A \subseteq X$ and $B \subseteq Y$ satisfying $A^{\uparrow_I} = B$ and $B^{\downarrow_I} = A$ is called a formal concept of I, and the set of all formal concepts, denoted $\mathcal{B}(X,Y,I)$, is called the concept lattice of I.[4] Each set $\mathcal{F} = \{\langle C_1,D_1\rangle,\dots,\langle C_k,D_k\rangle\}$ of formal concepts of I (with a fixed ordering given by the indices $1,\dots,k$) naturally induces two Boolean matrices, $A_\mathcal{F} \in \{0,1\}^{n\times k}$ and $B_\mathcal{F} \in \{0,1\}^{k\times m}$: The columns of $A_\mathcal{F}$ are the characteristic vectors of the sets $C_1,\dots,C_k$, and the rows of $B_\mathcal{F}$ are the characteristic vectors of $D_1,\dots,D_k$. In the illustrative example 1 in the supplementary material [3], the pair $\langle C_1,D_1\rangle$ with $C_1 = \{2,3\}$ and $D_1 = \{1,2,3\}$ is a formal concept of I and so are the pairs $\langle C_2,D_2\rangle = \langle\{3,4\},\{3,4,5\}\rangle$ and $\langle C_3,D_3\rangle = \langle\{1,3,4\},\{4,5\}\rangle$. These are just the formal concepts corresponding to the three column-row pairs used in that example, and hence the matrices $A_\mathcal{F}$ and $B_\mathcal{F}$ induced by $\mathcal{F} = \{\langle C_1,D_1\rangle,\langle C_2,D_2\rangle,\langle C_3,D_3\rangle\}$ are just the matrices A and B of the example.

Now, the original GreConD algorithm computes for a given matrix $I \in \{0,1\}^{n\times m}$ a set $\mathcal{F}$ of formal concepts of I such that $I = A_\mathcal{F}\circ B_\mathcal{F}$ using a particular greedy search method. Basically, our new algorithm, Algorithm 1, extends the original GreConD as follows. First, the algorithm computes a clarified matrix $J = \mathsf{cla}(I)$ from I. Second, it computes a factorization of J, i.e., a set $\mathcal{G}$ of formal concepts of J for which $J = A_\mathcal{G}\circ B_\mathcal{G}$, in a way similar to that used by GreConD, but with a different approach to coverage as explained below. Third, a set $\mathcal{F}$ of formal concepts of I is obtained from $\mathcal{G}$ such that $I = A_\mathcal{F}\circ B_\mathcal{F}$, i.e., the algorithm finishes with a factorization of I.

In detail, ll. 1–4 initialize the variables used. In addition to $\mathcal{F}$ and $\mathcal{G}$, explained in the previous paragraph, these include n_c, m_c, $rpos$, and $cpos$ which are returned by the clarification procedure Cla; see Section 3.1. Moreover, the algorithm involves matrix U of the same dimension as J which contains information about the entries containing 1 that are not covered by the factors of $\mathcal{G}$ computed in the previous iterations. Unlike the original GreConD, which stores in U the values 0 and 1 indicating "not covered" and "covered" by $\mathcal{G}$, U_{ij} is now a non-negative integer. While $U_{ij} = 0$ still indicates that the entry J_{ij} has been covered by the previously computed factors, $U_{ij} > 0$ means that $J_{ij} = 1$ and that if the entry J_{ij} gets covered by a formal concept $\langle C,D\rangle$ of J, then its extension $\langle E,F\rangle$ computed in ll. 20–21 shall cover U_{ij} entries of the input matrix I that contain 1. The rationale behind l. 7 is explained in the proof of theorem 2. Next, the difference from GreConD in computing $\mathcal{G}$ from J in ll. 8–18 consists in storing different information in matrix U indicating which entries of J have not been covered by the factors in $\mathcal{G}$ computed in the previous iterations. The loop in ll. 8–18 basically agrees with the original GreConD when applied to the clarified matrix J, with the

[4] These are the basic notions of formal concept analysis (FCA) [13]. Note that FCA is based on a formalism of sets rather than Boolean matrices used in BMF. Thus, instead of a Boolean matrix I, FCA assumes a binary relation between X and Y, i.e. a subset of $X\times Y$. The correspondence of sets and relations used in FCA, on the one hand, and vectors and matrices used in BMF, is straightforward, and one can switch between the two formalisms: $I_{ij} = 1$ indicates that the pair $\langle i,j\rangle$ is in the corresponding relation.

provision that the improvement in coverage in the extension of D by the attribute J is computed according to

$$D \oplus^J j = \sum_{x \in (D \cup \{j\})^{\downarrow_J}} \sum_{y \in (D \cup \{j\})^{\downarrow_J \uparrow_J}} U_{xy}, \tag{4}$$

i.e., taking into account the number of uncovered entries in I. In ll. 19–22, the set $\mathcal{F}$ is computed from $\mathcal{G}$ by a simple extension: For each $\langle C, D\rangle \in \mathcal{G}$, one puts to $\mathcal{F}$ the pair $\langle E, F\rangle$ in which E contains along with each row index $i \in C$ all the indices $i' \in X$ of the rows $\equiv_X$-equivalent to i, and F contains with each column index $j \in D$ all the $j' \in Y$ $\equiv_Y$-equivalent to j.

Algorithm 1 Extended GreConD.

```
Input: $I \in \{0,1\}^{n \times m}$
Output: $\mathcal{F}$

1: $\mathcal{F} \leftarrow \emptyset$
2: $J, n_c, m_c, rpos, cpos \leftarrow$ Cla$(I)$
3: $\mathcal{G} \leftarrow \emptyset$
4: $U \leftarrow J$
5: for $i = 1, \dots, n_c$ do
6:     for $j = 1, \dots, m_c$ do
7:         $U_{ij} = U_{ij} \cdot |[rpos(i)]_{\equiv_X}| \cdot |[cpos(j)]_{\equiv_Y}|$
8: while $U \neq 0^{n_c \times m_c}$ do
9:     $D \leftarrow \emptyset$
10:    $V \leftarrow 0$
11:    while there is $j \notin D$ such that $(D \oplus^J j) > V$ do
12:        select $j \notin D$ that maximizes $D \oplus^J j$
13:        $D \leftarrow (D \cup \{j\})^{\downarrow_J \uparrow_J}$
14:        $V \leftarrow D \oplus^J j$
15:    $C \leftarrow D^{\downarrow_J}$
16:    add $\langle C, D\rangle$ to $\mathcal{G}$
17:    for $\langle i, j\rangle \in C \times D$ do
18:        $U_{ij} \leftarrow 0$
19: for $\langle C, D\rangle \in \mathcal{G}$ do
20:    $E \leftarrow \bigcup_{i \in C} [rpos(i)]_{\equiv_X}$
21:    $F \leftarrow \bigcup_{j \in D} [cpos(j)]_{\equiv_Y}$
22:    add $\langle E, F\rangle$ to $\mathcal{F}$
23: return $\mathcal{F}$
```

Theorem 2. *Algorithm 1 computes the same formal concepts of I in the same order as the original GreConD for any input matrix I. In particular, it computes a set $\mathcal{F}$ of formal concepts of I for which $I = A_{\mathcal{F}} \circ B_{\mathcal{F}}$.*

Proof. We verify that the extensions $\langle E, F\rangle$ of the factors $\langle C, D\rangle$ coincide and are being added to the output set $\mathcal{F}$ in the same order as the formal concepts produced by the original GreConD when applied to I. As mentioned in the description of the algorithm, ll. 8–18 perform the original GreConD on the clarified matrix J, but using the modified matrix U. That is, starting with $D = \emptyset$, it constructs the formal concept $\langle C, D\rangle$ to be added to $\mathcal{G}$ by an incremental extension by attributes j until such an extension stops improving coverage of 1s. The attribute j is the one corresponding to a best possible extension of D. Here, best" means with respect to the improvement in coverage by the constructed formal concept; see ll. 11–14. Now, due to (4) and the definition of U_{ij}, $D \oplus^J j$ equals the improvement in the number of the 1s in I uncovered so-far, i.e., uncovered by the extensions $\langle E, F\rangle$ of the formal concepts $\langle C, D\rangle$ in $\mathcal{G}$ obtained in the previous iterations. A moment's reflection reveals that since the orderings of rows and columns in J respect those in I (Section 3.1), the loop in ll. 11–14 results in a formal concept $\langle C, D\rangle$ to be added to $\mathcal{G}$ whose extension $\langle E, F\rangle$, later computed in ll. 20–21, is just the formal concept selected in the corresponding step of the original GreConD when run on the input matrix I.

The second part follows the original GreConD's output set $\mathcal{F}$ satisfies $I = A_{\mathcal{F}} \circ B_{\mathcal{F}}$. □

A detailed demonstration of the extended GreConD algorithm is found in the supplementary material [3, example 5].

Remark 1. As with the original GreConD, algorithm 1 can easily be modified to compute approximate factorizations by changing the stopping condition from $U \neq 0^{n_c \times m_c}$ to $\sum_{i,j} U_{ij} \geq \varepsilon$. □

Remark 2. As we shall see in Section 4, the extended GreConD enjoys a considerable speed-up on real-world data compared to the original GreConD. Yet, its asymptotic worst-case time complexity remains the same as that of the original GreConD in most scenarios. Namely, recall first that the worst-case time complexity of GreConD is in $O(||I||nm^3)$ [4] (this bound follows from loose estimations; a tighter bound is an open problem). In the worst case, i.e., with no duplicate rows and columns, the reduced matrix J coincides with the input matrix I. The extended algorithm first executes the reduction procedure on line 2, which runs in time $O(mn \log n)$; see the end of Section 3.1. The algorithm then follows the logic of the original GreConD with modified updates of the auxiliary data structures which clearly do not affect the overall complexity. Hence, the overall worst-case time complexity of the extended GreConD is $O(mn \log n + ||I||nm^3)$. Now, if $mn \log n \in O(||I||nm^3)$, which is true in realistic situations (notice that for this to be true, it suffices that I contains at least $\log n/m^2$ entries containing 1), the overal complexity of the extended algorithm is $O(||I||nm^3)$, i.e., that of the original GreConD.

3.4. Extended Asso

The Asso algorithm [15] has been designed to solve the DBP by making use of the so-called association rules among the attributes, i.e., matrix columns. Since the description in [15] is somewhat incomplete as regards some details essential for our extension, we start by a description of the original algorithm suitable for our purpose. Let again $X = \{1, \dots, n\}$ and $Y = \{1, \dots, m\}$. For an input matrix $I \in \{0,1\}^{n \times m}$ and a non-negative integer $k \leq \min(m, n)$, Asso attempts to find matrices $S \in \{0,1\}^{n \times k}$ and $B \in \{0,1\}^{k \times m}$ such that $E(I, S \circ B)$ is minimal. Asso first computes the so-called association matrix $A \in \{0,1\}^{m \times m}$ from I and a user-defined parameter $\tau \in [0,1]$, which is defined by

$$A_{j_1 j_2} = 1 \text{ iff } \operatorname{conf}(j_1 \Rightarrow j_2) \geq \tau$$

where

$$\operatorname{conf}(j_1 \Rightarrow j_2) = \frac{|\{i\,;\, I_{ij_1} = 1 \text{ and } I_{ij_2} = 1\}|}{|\{i\,;\, I_{ij_1} = 1\}|} = \frac{\sum_{l=1}^n I_{lj_1} \cdot I_{lj_2}}{\sum_{l=1}^n I_{lj_1}}$$

is the so-called confidence of the association rule $j_1 \Rightarrow j_2$ in the input matrix I. Finally, the matrices S and B are computed in a greedy manner to minimize the error

$$E(I, S \circ B) = E_u(I, S \circ B) + E_o(I, S \circ B)$$

by maximizing the value of

$$\begin{aligned}\text{cover}(S, B, I, w^+, w^-) = w^+ \cdot |\{\langle i, j\rangle\,;\, I_{ij} = 1, (S \circ B)_{ij} = 1\}| - \\ w^- \cdot |\{\langle i, j\rangle\,;\, I_{ij} = 0, (S \circ B)_{ij} = 1\}|\end{aligned} \tag{5}$$

where w^+ and w^- are user-defined non-negative weights to reward covering and penalize overcovering of I, respectively. Note that value of cover can be evaluated iteratively for each factor $l = 1, \dots, k$ as

$$\text{cover}(S, B, I, w^+, w^-) = \sum_{l=1}^{k} \text{cover}^l(S_{\_l}, B_{l\_}, U, O, w^+, w^-), \tag{6}$$

where the matrices U and O maintain information about the current coverage state by the factors $1, \dots, l-1$. Both U and O need to be updated to avoid repeated rewarding/penalizing for covering the same entries I_{ij}.

Our extension of Asso (Algorithm 2) aims to simulate the steps of the original Asso algorithm on a clarified matrix resulting from I. First, one computes from I the clarified matrix $\text{Cla}(I) = J \in \{0,1\}^{n_c \times m_c}$. Second, an association matrix A^J is computed from J using slightly modified confidence rule $\overline{\text{conf}}(j_1 \Rightarrow j_2)$ to take into account the rows removed within clarification. Third, the matrices U and O are identified as the essential part for the accurate calculation of the modified cover function as explained below. Finally, an approximate factorization $J \approx S^J \circ B^J$ is iteratively computed and the factorization of $I \approx S \circ B$ is found by extending S^J and B^J as described by (1) and (2), respectively. The algorithm finishes with the exactly same factorization of I as the original Asso.

In detail, matrix I is clarified on line 1 along with the initialization of the other used variables on ll. 2–3. Note that Cla returns, in addition to J, values n_c, m_c, $rpos$, and $cpos$ as described in Section 3.1. The association matrix $A^J \in \{0,1\}^{m_c \times m_c}$ of J is computed in ll. 4–6 utilizing a modified confidence function

$$\overline{\text{conf}}(j_1 \Rightarrow j_2) = \frac{\sum_{i=1}^{n_c} J_{ij_1} \cdot J_{ij_2} \cdot |[rpos(i)]_{\equiv_X}|}{\sum_{i=1}^{n_c} J_{ij_1} \cdot |[rpos(i)]_{\equiv_X}|}.$$

This modification includes information about the number of reduced rows. The following theorem shows, that $\overline{\text{conf}}(j_1 \Rightarrow j_2)$ indeed reflects the degree of association between the corresponding columns $cpos(j_1)$ and $cpos(j_2)$ in I:

Theorem 3. *Let $I \in \{0,1\}^{n \times m}$ be binary matrix and $J \in \{0,1\}^{n_c \times m_c}$ be the corresponding clarified matrix of I. For each $j_1, j_2 \in \{1, \dots, m_c\}$, we have*

$$\overline{\text{conf}}(j_1 \Rightarrow j_2) = \text{conf}(cpos(j_1) \Rightarrow cpos(j_2)).$$

Proof.

For each column j_1 of J and its corresponding column $cpos(j_1)$ in I we can compute the number of rows $i \in \{1, \dots, n\}$ such that $I_{i,cpos(j_1)} = 1$ as

$$\sum_{i=1}^{n} I_{i,cpos(j_1)} = \sum_{S \in X/\equiv_X} I_{min(S)cpos(j_1)} \cdot |S| = \sum_{i=1}^{n_c} J_{ij_1} \cdot |[rpos(i)]_{\equiv_X}|.$$

Indeed, since $X/\equiv_X$ contains the sets S_i of identical rows (see Section 3.1), we can utilize them for the sum instead of adding each row separately. The first equality follows immediately. Additionaly, the clarification procedure preserves only the index $\min(S)$ of the first row for each $S \in X/\equiv_X$ and since for each $i \in \{1, \dots, n_c\}$ there is exactly one $S_i \in X/\equiv_X$ such that $rpos(i) = \min(S_i)$ and $|S_i| = |[rpos(i)]_{\equiv_X}|$, the second equality follows.

Now, we can easily extend the expression by second attribute j_2 to prove the equality

$$\sum_{i=1}^{n} I_{icpos(j_1)} \cdot I_{icpos(j_2)} = \sum_{i=1}^{n_c} J_{ij_1} \cdot J_{ij_2} \cdot |[rpos(i)]_{\equiv_X}|.$$

Finally, combining the first and the second equality, we obtain

$$\frac{\sum_{i=1}^{n} I_{icpos(j_1)} \cdot I_{icpos(j_2)}}{\sum_{i=1}^{n} I_{icpos(j_1)}} = \frac{\sum_{i=1}^{n_c} J_{ij_1} \cdot J_{ij_2} \cdot |[rpos(i)]_{\equiv_X}|}{\sum_{i=1}^{n_c} J_{ij_1} \cdot |[rpos(i)]_{\equiv_X}|}$$

finishing the proof. $\square$

Corollary 2. *With the same assumptions as in theorem 3, for the confidence matrices A of I and A^J of J holds*

$$\text{Cla}(A) = \text{Cla}(A^J).$$

Proof. Theorem 3 implies that for each $j_1, j_2 \in \{1, \dots, m_c\}$ we have

$$A^J_{j_1 j_2} = A_{cpos(j_1)cpos(j_2)}.$$

Algorithm 2 Extended Asso.

```
    Input: $I \in \{0,1\}^{n \times m}, k < \min(m,n), \tau \in [0,1]$ and $w^+, w^- \in \mathbb{R}$
    Output: $S \in \{0,1\}^{n \times k}, B \in \{0,1\}^{k \times m}$

 1: $J, n_c, m_c, rpos, cpos \leftarrow \text{Cla}(I)$
 2: $S^J \leftarrow 0^{n_c \times k}, B^J \leftarrow 0^{k \times m_c}$
 3: $A^J \leftarrow 0^{m_c \times m_c}$
 4: for $\langle i,j \rangle \in \{1, \dots, m_c\} \times \{1, \dots, m_c\}$ do
 5:     if $\overline{\text{conf}}(i \Rightarrow j) \geq \tau$ then
 6:         $A^J_{ij} = 1$
 7: $U \leftarrow J$
 8: for $\langle i,j \rangle \in \{1, \dots, n_c\} \times \{1, \dots, m_c\}$ do
 9:     $U_{ij} = U_{ij} \cdot |[rpos(i)]_{\equiv_X}| \cdot |[cpos(j)]_{\equiv_Y}|$
10:     $O_{ij} = |J_{ij} - 1| \cdot |[rpos(i)]_{\equiv_X}| \cdot |[cpos(j)]_{\equiv_Y}|$
11: for $l = 1, \dots, k$ do
12:     select $j \in \{1, \dots, m_c\}$ maximizing $\overline{\text{COVER}}(s^j, A_{j\_}, U, O, w^+, w^-)$
13:     $B^J_{l\_} \leftarrow A^J_{j\_}$ and $S^J_{\_l} \leftarrow \text{GENERATES}(A^J_{j\_}, U, O, w^+, w^-)$
14:     for $\langle i,j \rangle \in \{1, \dots, n_c\} \times \{1, \dots, m_c\}$ do
15:         $U_{ij} \leftarrow U_{ij} \cdot |\min(S^J_{il}, B^J_{lj}) - 1|$
16:         $O_{ij} \leftarrow O_{ij} \cdot |\min(S^J_{il}, B^J_{lj}) - 1|$
17: $S = 0^{n \times k}, B = 0^{k \times m}$
18: for $l = 1, \dots, k$ do
19:     for $i = 1, \dots, n$ do
20:         $S_{il} \leftarrow S^J_{rpos^{-1}(\min[i]_{\equiv_X}),l}$
21:     for $j = 1, \dots, m$ do
22:         $B_{lj} \leftarrow B^J_{l,cpos^{-1}(\min[j]_{\equiv_Y})}$
23: return $S, B$
```

The matrix A^J is then just a submatrix of A, such that the rows and columns of A^J correspond to the columns left in J after the clarification of I. In the other words, since for each $j_1, j_2 \in \{1, \dots, m\}$ we clearly have

$$\text{if} \quad I_{\_j_1} = I_{\_j_2} \quad \text{then} \quad A_{j_1\_} = A_{j_2\_} \text{ and } A_{\_j_1} = A_{\_j_2},$$

we have $\text{Cla}(A) = \text{Cla}(A^J)$. Note that in general, $\text{Cla}(A) \neq A^J$, because two non-identical columns $j_3, j_4 \in \{1, \dots, m_c\}$ in J, i.e., $J_{\_j_3} \neq J_{\_j_4}$, can clearly have identical rows (and columns) $A^J_{j_3\_} = A^J_{j_4\_}$ in A^J and then $A^J \neq \text{Cla}(A^J)$; see example 6 in the supplementary material [3]. $\square$

The search for a decomposition of J is carried out with regard to the resulting decomposition I, which requires a proper adjustment of the cover function. We introduce (ll. 7–10) the matrices $U \in \mathbb{N}_0^{n_c \times m_c}$ (uncovered) and $O \in \mathbb{N}_0^{n_c \times m_c}$ (overcovered), which maintain information about the number of as yet uncovered and non-overcovered elements in I, respectively. In particular, the meaning of $U_{ij} > 0$ is: If $J_{ij} = 1$ gets covered, then extending $S^J \circ B^J$ to $S \circ B$ results in covering a total of U_{ij} entries of I that contain 1. Similarly, $O_{ij} > 0$ indicates that a decomposition of J with $(S^J \circ B^J)_{ij} = 1$, followed by the extension to a decompositon $S \circ B$ of I, would result in overcovering O_{ij} entries of I that contain 0. In each iteration l (ll. 11–16), the algorithm selects a row $A^J_{j\_}$ and a vector s^j described below, maximizing value of the revised COVER^l function

$$\overline{\text{COVER}}^l(s^j, A^J_{j\_}, U, O, w^+, w^-) = \sum_{i=1}^{n_c} \sum_{k=1}^{m_c} s^j_i \cdot A^J_{jk} \cdot (w^+ \cdot U_{ik} - w^- \cdot O_{ik});$$

and, in addition, the tuple $\langle A^J_{j\_}, s^j \rangle$ with the highest score is selected as a new factor, i.e. $S^J_{\_l} = s^j$ and $B^J_{l\_} = A^J_{j\_}$. The value $s^j_i \cdot A^J_{jk} = 1$ indicates a coverage of J_{ik} entry by the current candidate tuple j and the expression in parentheses reflects the contribution to the overall score for the covering/overcovering of J_{ik}. Note, that in each iteration at most one of the values U_{ik} or O_{ik} is nonzero.

Algorithm 3 GENERATES.

```
Input: $a \in \{0,1\}^{m_c}, U \in \mathbb{N}_0^{n_c \times m_c}, O \in \mathbb{N}_0^{n_c \times m_c}$, and $w^+, w^- \in \mathbb{R}$
Output: $s \in \{0,1\}^{n_c}$

1: $s \leftarrow 0^{n_c}$
2: for $i = 1, \ldots, n_c$ do
3:     $score^i = w^+ \cdot \sum_{j=1}^{m_c} U_{ij} \cdot a_j - w^- \cdot \sum_{j=1}^{m_c} O_{ij} \cdot a_j$
4:     if $score^i > 0$ then
5:         $s_i \leftarrow 1$
6: return $s$
```

Let us now describe the algorithm (Algorithm 3) for finding the vector s^J that maximizes the $\overline{\text{COVER}}^l$ function. We have

$$s^J = \text{GENERATES}(A^J_{j\_}, U, O, w^+, w^-)$$

as a greedy selection of vector s^J for candidate $A^J_{j\_}$. Our implementation[5] initializes zero vector $s^J = 0^{n_c}$. Then, each s^J_i is evaluated in a greedy manner to determine whether it is better to set $s^J_i = 0$ or $s^J_i = 1$, selecting the value resulting in the largest increase of $\overline{\text{COVER}}^l$. The selected pair of $A^J_{j\_}$ and s^j is then stored as a new factor for J (line 14) and the newly covered elements are updated accordingly in the matrix U and O (ll. 14–16).

Finally, a decomposition of I is obtained by extending the matrix S^J to S according to (1) and extending B^J to B according to (2).

Theorem 4. *Algorithm 2 computes the same decomposition of I as the original Asso for any input matrix I.*

Proof.

Denote the matrices returned by original Asso algorithm for input I by S^* and B^*. To prove our theorem, we verify that finding factors on ll. 11–16 copy the behaviour of original Asso, and thus for each factor l it holds

$$S^*_{\_l} = S_{\_l} \text{ and } B^*_{l\_} = B_{l\_}$$

where S and B are the matrices returned by Algorithm 2.

The selection of $S^J_{\_l}$ and $B^J_{l\_}$ in iteration l is based on the value of $\overline{\text{COVER}}^l$, and analogously for $S^*_{\_l}$, $B^*_{l\_}$, and COVER^l. We have

$$\text{COVER}^l(S^*_{\_l}, B^*_{l\_}, U^*, O^*, w^+, w^-) = w^+|\{I_{ij}; U^*_{ij} = 1, (S^*_{\_l} \circ B^*_{l\_})_{ij} = 1\}| - w^-|\{I_{ij}; O^*_{ij} = 1, (S^*_{\_l} \circ B^*_{l\_})_{ij} = 1\}|,$$

where U^* and O^* are the matrices representing the uncovered and non-overcovered entries of I by factors $1, \ldots, l-1$. Since $s^j_i \cdot A^J_{jk} = (s^j \circ A^J_{j\_})_{ik}$, we obtain

$$\begin{aligned}\overline{\text{COVER}}^l(s^j, A^J_{j\_}, U, O, w^+, w^-) &= \sum_{i=1}^{n_c} \sum_{k=1}^{m_c} s^j_i \cdot A^J_{jk} \cdot (w^+ \cdot U_{ik} - w^- \cdot O_{ik}) \\ &= \sum_{i=1}^{n_c} \sum_{k=1}^{m_c} w^+ \cdot (s^j \circ A^J_{j\_})_{ik} \cdot U_{ik} - w^- \cdot (s^j \circ A^J_{j\_})_{ik} \cdot O_{ik} \\ &= \text{COVER}^l(\text{ext}(s^j), \text{ext}(A^J_{j\_}), U^*, O^*, w^+, w^-),\end{aligned}$$

for $\text{ext}(Z)$ defined as in (1) and (2). Theorem 3 and corollary 2 yield that the matrices A and A^J maintain the same information. It is evident that if the extended algorithm chooses a tuple $\langle s^j, A^J_{j\_}\rangle$, with the maximal value of $\overline{\text{COVER}}^l$ as a new factor, then in the matrix A there exists a corresponding $cpos(j)$ with the same score COVER^l, and vice versa. Both algorithms hence clearly select the same factor in each iteration l. □

[5] Note that finding the vector s^j is not detailed in the paper on the original Asso algorithm [15], hence our description.

A detailed demonstration of the extended Asso algorithm can be found in the supplementary material [3, example 6].

Remark 3. While the extended Asso algorithm achieves a considerable speed-up on real-world data (Section 4), its asymptotic worst-case time complexity is the same as that of the original Asso algorithm in realistic scenarios. In detail, note first that the time complexity of the original Asso is $O(km^2n)$, computing the association matrix in time $O(m^2n)$ and each of the k factors also in time $O(m^2n)$; see [5]. In the worst case, i.e., with no redundancy, the reduced matrix J coincides with the input matrix I. The extended algorithm first runs the reduction procedure on line 1, which runs in time $O(mn \log n)$; see Section 3.1. Then the extended algorithm follows the logic of the original Asso, extending it by bookkeeping steps to ensure correct updates of the auxiliary matrices. It is immediate to see that none of these bookkeeping steps negatively affects the overall complexity of $O(km^2n)$. Therefore, the overall worst-case time complexity of the extended Asso is $O(mn \log n + km^2n)$. Moreover, if $\log n \leq m$—which may be considered a rather realistic scenario—the time complexity of the extended Asso remains in $O(km^2n)$.

4. Experimental evaluation

In this section, we demonstrate that the reduction scheme proposed in our paper results in a significant speed-up of factorization algorithms. In doing so, we provide various results for GreConD and Asso, i.e., the two basic algorithms for the AFP and DBP problems whose extensions we developed in our paper. We focus on the aspects relevant to the purpose of the proposed extension and omit evaluation of other aspects, such as coverage graphs of the data by the computed factors or precision of the computed factorizations which are described in detail elsewhere in the literature on BMF. This is possible due to our main theorems according to which both the extended GreConD and the extended Asso compute exactly the same factorizations as their well-known ordinary counterparts.

In particular, we focus on the dependence of the speed-up resulting from the proposed transformation of the input data and employment of the extended algorithms on the redundancy of the input data. We also explore other relevant questions such as the impact of row redundancy and column redundancy, and examine further relevant topics. For this purpose we utilize both the well-known real-world datasets (Section 4.1) and synthetic datasets (Section 4.2).

4.1. Real-world data

We use the well-known, commonly used benchmark datasets described in Section 2. Since all the presented algorithms are deterministic, the measured runtimes have a small standard deviation for each dataset. The experiments were hence repeated 10 times only to obtain average runtimes for both the original and modified versions of the algorithms.

The results regarding the observed speed-up of a running time are summarized in Fig. 1. Note that the datasets with no redundancy as well as small datasets which get factorized in a fraction of a second, and hence with a speed-up possibly affected by other factors, are omitted. The bars on the left depict the redundancy of the dataset, as defined in section Section 2. The bars on the right present the measured speed-up, i.e., the ratio

$$\frac{\text{runtime of the original algorithm}}{\text{runtime of the extended algorithm}},$$

which is depicted on a logarithmic scale. Observe that the highest speed-up occurs for the large datasets Americas Large and Americas Small with the redundancy around 98%. For instance, with Americas Large, the extended GreConD runs approximately 48 times faster, while Asso

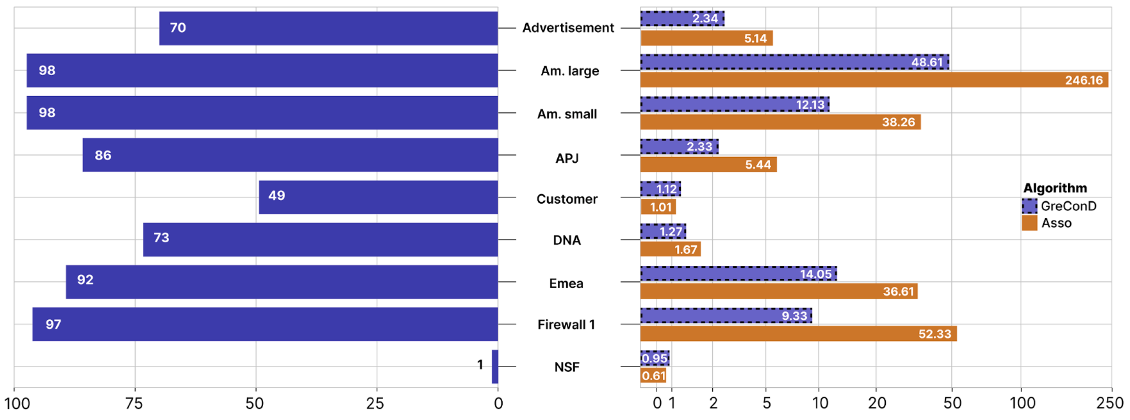
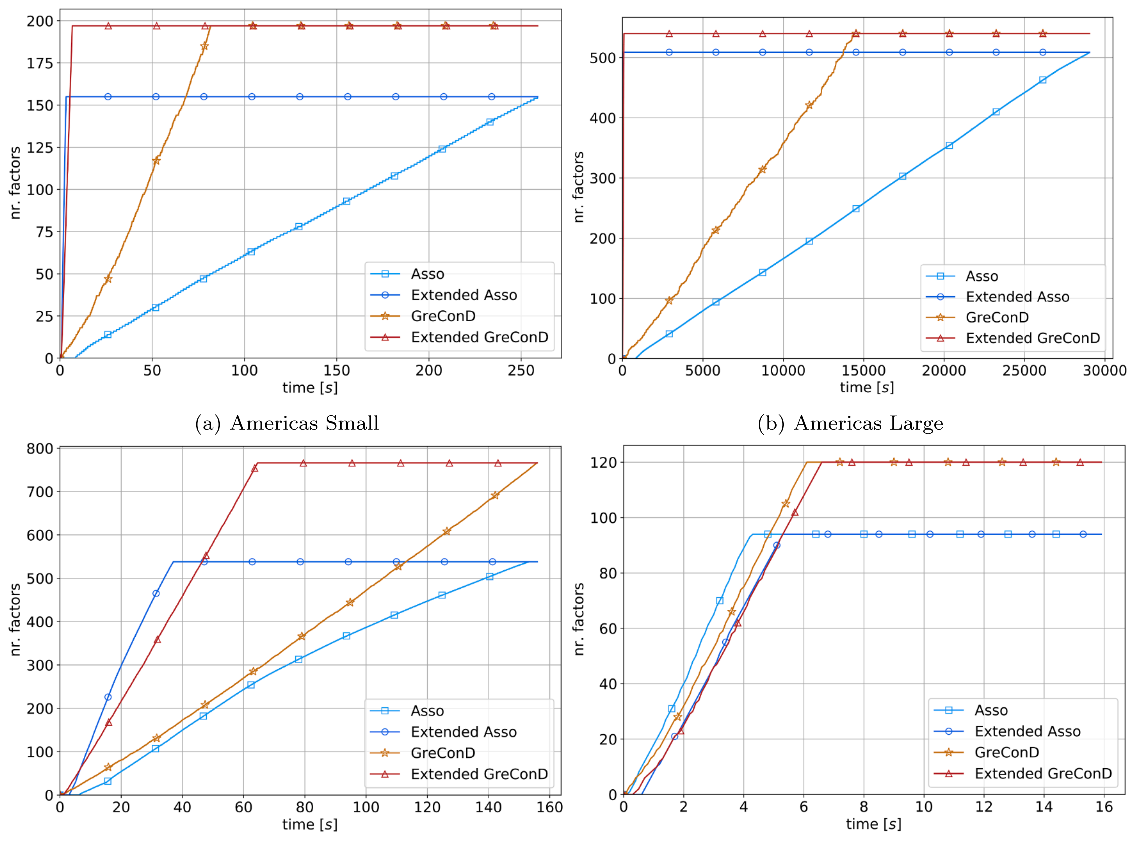


Fig. 1. Comparison of a redundancy and speed-up on benchmark datasets.



(a) Americas Small

(b) Americas Large

(c) Advertisement

(d) Mushroom

Fig. 2. Comparison of the number of factors (vertical axes) computed over time (horizontal axes) between the original and extended GRECOND and ASSO. Both versions, the original and the extended, compute the same factorizations but their speeds clearly differ; the point where a line stops rising corresponds to the time when the respective algorithm finishes computation.

runs more than 245 times faster compared to the original algorithms. In practical terms, this means that the runtime dropped from hours to seconds. One may also observe a phenomenon we examine in more detail in the next section, namely, a larger impact of column-redundancy compared to row-redundancy on the speed-up. In particular, the Customer dataset with a considerable 49% redundancy has a very small column-redundancy (see Table 1), which results in a very small speed-up for both GRECOND and ASSO, which also applies to the DNA dataset. We also include the considerably large NSF dataset with a rather small 1% redundancy, resulting in a slow-down of the running time for both GRECOND and ASSO, which is expected because the extended algorithms naturally involve overhead.

In this context, let us also note that another factor affecting the speed-up consists in that while the original algorithms may use binary matrices to represent coverage information, the extended versions need to use integer matrices, which slows down particularly the repeatedly performed matrix summations. The overall results imply that this slow-down is amply compensated by the overall speed-up. Further evaluations of the speed-up is provided in the online supplementary data [3, S1].

Fig. 2 demonstrates in more detail a single computation of the individual factors by the original vs the extended algorithms in dependence on the running time. As explained above, both versions, the original and the extended, compute the same factorizations but their speeds differ.

The points where the lines stop rising indicate when the respective algorithm stops. Recall at this point that Asso rarely achieves an exact factorization and typically ends up with an approximate factorization; in contrast to that, GreConD always produces an exact factorization which sometimes results in a longer runtime. The Americas Large and Small, and the Advertisement datasets enhance the results presented in Fig. 1; the Mushroom dataset is an example of a zero-redundancy dataset and displays a slight slow-down caused by the factors explained in the previous paragraph. The detailed results provided in the online supplementary data [3, S1], as well as the results for synthetic data, presented in the following section, show that the slow-down is small in most cases, and thus does not significantly affect the overall runtime. Generally, data reduction appears to have a more significant impact on the speed of the Asso algorithm compared to GreConD.

4.2. Synthetic data

To provide further experimental results, in particular to assess difference between the impacts of row- and column-redundancy on the runtime, a synthetic data generator has been designed to produce a symmetric square matrices to suppress a possible bias. As is clear from the description of both algorithms, they focus on columns when computing factorizations: GreConD constructs the factors from the best columns selected in a greedy manner; as for Asso, it constructs its association matrix from the columns. The column-redundancy is thus expected to have a larger impact on speed-up compared to row-redundancy.

To produce natural random matrices for factorization, we employed the following process. For a given dimension n and a prescribed density c (see Section 2), matrices $A \in \{0,1\}^{n\times 1}$ and $B \in \{0,1\}^{1\times n}$ are randomly initialized. Until the density of a matrix $C = A \circ B$ reaches c, we add randomly generated vectors $A_i, B_i \in \{0,1\}^n$ as a new factor to A and B, respectively. Finally, the square matrix $C \in \{0,1\}^{n\times n}$ is made symmetric by flipping the upper diagonal to the lower one.

In the experiments presented below, we used $n = 1000, 1500, 2000, 2500, 3000$, and $c = 0.1$, since 0.1 appears to represent a common density in the real-wold datasets. We then added redundancy of $r = 0, 0.05, 0.10, 0.25, 0.50, 0.75$ by adding duplicit rows and columns in a way to maintain symmetry of the resulting matrix C. For each n and r, we generated 20 input matrices this way, and computed their factorizations via the original and the extended algorithms. Finally, a speed-up was computed for each run, and the average speed-up was stored.

The results are summarized in Table 2. For each redundancy level r, the value dim in the second column stands for the average dimension of the input matrix after the addition of redundancy. For both the GreConD and Asso algorithms, the mean speed-up is presented in three columns corresponding to three kinds to redundancy removal: row-redundancy only (sp-up^R), column-redundancy only (sp-up^C), and both row- and column-redundancy (sp-up^{R+C}) which is equivalent to our modification.

For data without redundancy, i.e., for $r = 0.0$, the removal of redundancy, i.e., clarification, does not do anything to the input data, hence the extended algorithms essentially proceeds as the original ones. In this case, only the most time-consuming experiment involving removal of redundant rows and columns is reported in the table. Overall, the results suggest that even though the removal of column-redundancy does not achieve the effect of the row- and column-redundancy removal, i.e., the effect of clarification, it saves more time compared to row-redundancy removal (see also similar observations in Section 4.1 for the Customer and DNA datasets).

Table 2
Impact of removal of row-redundancy, column-redundancy, and both row- and column-redundancy on speed-up (sp-up).

		GreConD			Asso		
r	dim	sp-up^R	sp-up^C	sp-up^{R+C}	sp-up^R	sp-up^C	sp-up^{R+C}
0.0	1000	–	–	1.00	–	–	0.95
	1500	–	–	0.99	–	–	0.92
	2000	–	–	1.00	–	–	0.91
	2500	–	–	1.00	–	–	0.84
	3000	–	–	0.99	–	–	0.87
0.05	1053	0.95	1.02	1.04	1.02	1.17	1.13
	1579	0.96	1.02	1.04	1.02	1.20	1.11
	2106	0.98	1.03	1.06	1.09	1.23	1.25
	2632	1.00	1.06	1.09	0.93	1.05	1.05
	3150	1.02	1.08	1.16	0.89	1.08	1.07
0.10	1112	0.96	1.11	1.16	1.05	1.18	1.41
	1667	0.98	1.15	1.20	1.06	1.41	1.41
	2223	1.02	1.16	1.25	1.06	1.39	1.52
	2778	1.06	1.19	1.25	0.93	1.20	1.25
	3300	1.14	1.32	1.52	0.85	1.48	1.65
0.25	1334	1.05	1.42	1.61	1.22	2.27	2.56
	2000	1.09	1.59	1.82	1.20	2.13	2.70
	2666	1.19	1.67	1.96	1.22	2.00	2.63
	3334	1.37	1.68	2.43	1.19	1.92	2.38
	4000	1.32	1.79	2.27	1.14	1.82	2.10
0.50	2000	1.37	2.86	3.70	1.79	5.56	8.34
	3000	1.52	3.33	4.77	1.92	4.35	10.00
	4000	1.75	3.70	5.88	1.89	3.85	8.33
	5000	2.78	4.17	10.05	1.89	5.56	10.04
	6000	2.12	3.22	9.09	1.91	3.45	6.45
0.75	4000	2.56	10.02	19.78	3.56	13.67	52.19
	6000	3.57	14.29	33.33	3.93	12.82	77.33
	8000	4.54	11.12	52.14	4.11	14.56	75.13
	10,000	6.09	14.53	78.52	5.20	17.41	70.41
	12,000	7.19	15.52	84.12	5.14	15.16	72.88

4.3. Further topics

4.3.1. The need to modify the original algorithms

In Section 3.2, we described a simple, alternative factorization approach that utilizes removal of duplicity but employs the unmodified version of the given factorization algorithm, such as GreConD and Asso, and described the shortcomings of this alternative approach. The purpose of this section is to demonstrate that these shortcomings indeed materialize.

The shortcomings are demonstrated in Fig. 3. The graphs, which display a typical behavior of both GreConD and Asso, represent the coverage $c(l)$ of the input data by the first $l = 1, 2, 3, \ldots$ computed factors for two selected datasets $I \in \{0,1\}^{n\times m}$. Note that $c(l)$ is defined [16] by

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

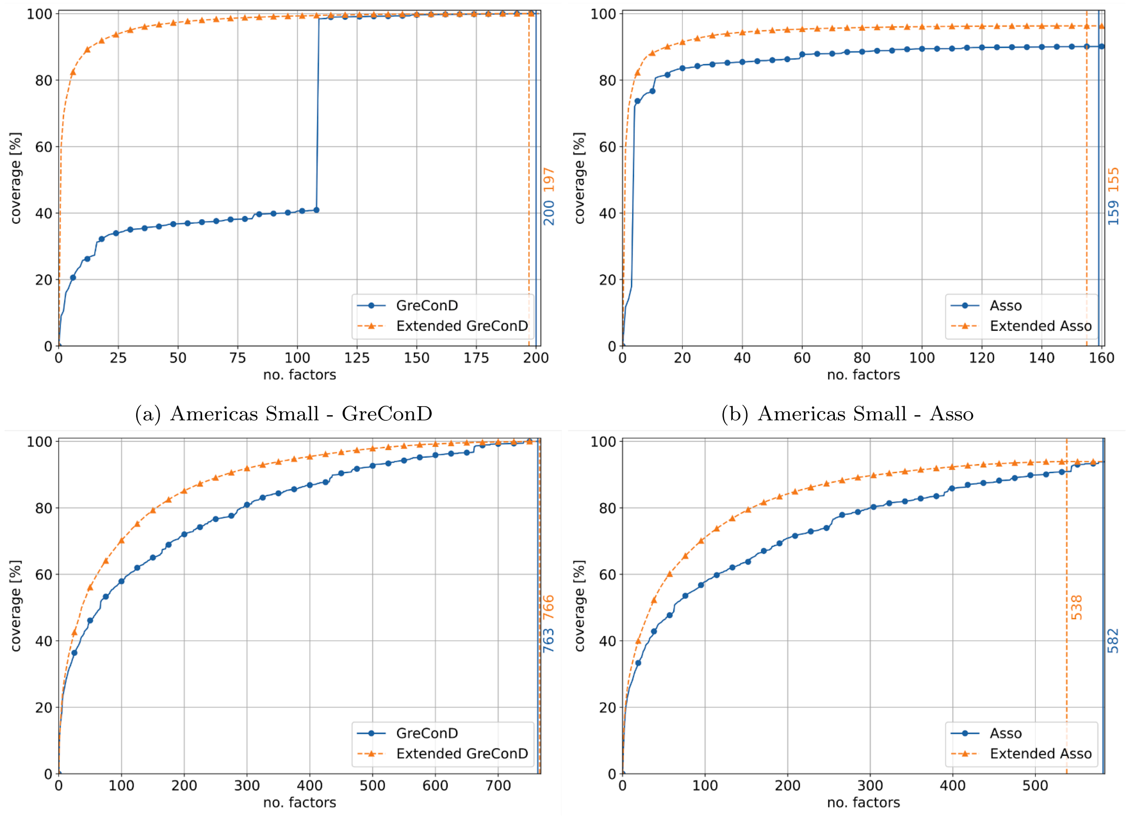
where $A(l)$ and $B(l)$ denote the $n \times l$ and $l \times m$ matrices corresponding of the first l of the computed factors. It is apparent that while the coverage graphs of the modified algorithms display the desired shape, i.e., are steeply increasing for small l, the graphs corresponding to the alternative approach involving the unmodified algorithms, both of which run on the clarified data, do not have this desired shape and display jumps. In particular, the graphs demonstrate that the unmodified algorithms deliver different factorizations compared to those computed by the modified algorithms.

In addition, as is evident from the graph for Americas Small, the approach involving the unmodified Asso achieves smaller coverage, i.e., smaller precision, compared to the approach involving a modified Asso as proposed in our paper. For the Advertisement dataset, the unmodified Asso algorithm required 44 more factors to reach the coverage achieved by the modified Asso.

In conclusion, the approach based on the proposed transformation of input data and employment of the original, unmodified algorithms suffers from the discussed shortcomings. More results for the other real-world datasets are found in the supplementary material [3, S2].

4.3.2. Possible limitation

While our method proves useful in the scenario assumed in our study, its utilization may be limited in a scenario in which the analyzed data



(a) Americas Small - GreConD (b) Americas Small - Asso

(c) Advertisement - GreConD (d) Advertisement - Asso

Fig. 3. Shortcomings of the unmodified algorithms and comparison with their extended versions on representative datasets Advertisement and Americas Small. The graphs depict the coverage of input data by the first k factors in the computed factorizations. Vertical lines indicate the final number of factors identified by each algorithm.

is subject to noise [17,18], i.e., a scenario in which "true data" gets distorted to some extent before it undergoes a Boolean matrix factorization. It appears obvious that even a very small extent of noise (or error) in Boolean data reduces or even destroys the amount of redundancy of rows and columns that may naturally be present in noise-free data. A simple quantitative indication consists in realizing that if a noise of level p% is applied to two equal rows of a Boolean matrix with m columns,[6] the probability that these rows remain equal is $[(1-\frac{p}{100})^2+(\frac{p}{100})^2]^m$. This reveals that redundancy in the "true data" gets significantly reduced as the noise level and the matrix dimensions get larger. Consequently, the time efficiency of our method would be impaired.

An experimental evidence of the reduction of redundancy as a result of noise and growing matrix dimension is apparent from Table 3, whose entries display the redundancy of selected datasets introduced in Section 2 to which a random noise of level p is added as explained above (the displayed redundancies represent average values over 20 repetitions).

It needs to be emphasized that as demonstrated above, many real Boolean datasets are highly redundant (Table 1), and hence, do not presumably contain noise. Therefore, the possible limitations discussed in this section do not apply, which justifies the relevance of our method. Nevertheless, the present considerations regarding noise call for an extension of our method or for an alternative approach to handle redundancy in situations with noise.

[6] In the sense that each matrix entry is flipped with the probability of $\frac{p}{100}$, independently of the other entries.

Table 3
Reduction of redundancy in selected datasets of Table 1, displayed in percents as a result of added noise of level p and matrix dimension dim(I).

data	dim(I)	$p=0$	$p=.1$	$p=1$	$p=2$
Americas Small	3,477 × 1,586	98%	24%	0%	0%
APJ	2,044 × 1,164	86%	18%	0%	0%
Customer	10,961 × 277	49%	37%	2%	0%
DBLP	6,980 × 19	87%	87%	82%	77%
Zoo	101 × 28	46%	44%	30%	21%

5. Conclusions

We observe that many of the benchmark datasets used in the literature on Boolean matrix factorization are redundant and propose a simple scheme that makes use of it. The scheme consists in transforming the input data to remove redundancy, applying an appropriately modified BMF algorithm, and use the computed, interim factors to restore from them the factors for the original input data. We developed an implementation of our scheme for two basic BMF algorithms, GreConD and Asso, and provided theorems justifying the proposed modifications of the original algorithms required by our scheme. Our experimental evaluation proves the new scheme efficient in terms of speed-up of the running time which is considerable as redundancy increases.

Reducing the size of an input Boolean matrix I and subsequently factorizing the resulting smaller matrix to obtain a reasonable factorization of the original matrix I in a shorter time represents a broad research topic which may be approached from several perspectives and requires further research. We propose the following directions:

Table 4
Speed-up of modified Tiling in comparison to original Tiling algorithm on selected datasets from Table 1.

data	dim(I)	redundancy	speed-up
Advertisement	3,279 × 1,557	70%	3.4
Americas Large	3,485 × 10,127	98%	118.9
Americas Small	3,477 × 1,586	98%	50.0
APJ	2,044 × 1,164	86%	3.3
Customer	10,961 × 277	49%	1.6
DNA	4,590 × 392	73%	2.2
Emea	35 × 3,046	92%	33.3
Firewall 1	365 × 709	97%	12.5
Mushroom	8,124 × 119	5%	1.0
Paleo	501 × 139	6%	1.1
Tic Tac Toe	958 × 30	0%	1.0

- In our paper, utilization of redundancy is illustrated on GreConD and Asso, i.e., for two primary algorithms designed for the AFP and the DBP problems (Section 1.2), respectively. Explorations to utilize redundancy, including redundancy in a broader perspective as described in this section below, for a number of other available BMF algorithms [1] remain a topic for future research.

  As mentioned above, utilization of redundancy reduction to speed up factorization may be implemented either in the straightforward scenario described in steps 1.–3. at the end of Section 3.2, or in the improved scenario which alleviates a shortcoming of the first scenario. The improved scenario requires an appropriate modification of the considered factorization algorithm, which we provided for GreConD and Asso in Sections 3.3 and 3.4 along with proofs of correctness of the proposed modifications. A correct modification of a factorization algorithm is clearly a non-trivial step but appears feasible due to the kind of redundancy we explore. As a preliminary step toward exploration of other algorithms, we examined such a modification for Tiling—another well-known factorization algorithm [19].[7] While Tiling uses a different strategy from that of GreConD, it also employs as factors rectangular areas of an input matrix I that are full of 1. Similar ideas to those on which our modification of GreConD is based may hence be used to obtain a modified Tiling and its proof of correctness. Table 4 presents the speed-up of the thus modified algorithm for selected benchmark datasets. While the modified Tiling does not slow down factorization of data with low redundancy, its speed-up is comparable and in most cases larger than that of GreConD on redundant data (cf. Fig. 1).
- In addition to removing duplicate rows and columns of the input matrix I, one may utilize the so-called reduction of I—a different method of reducing the size of I employed by formal concept analysis [13, pp. 24–34]. The basic idea consists in removing the rows and columns of I that may be obtained as intersections of other rows and columns, respectively. The resulting reduced matrix J preserves important structural information of the original matrix I [13]. Most significantly, the concept lattice of J is isomorphic to the concept lattice of I, and the formal concepts of I, i.e., the potential factors of I, can be restored from those of J. According to our preliminary results, factorizing reduced matrices and an appropriate extension of the factors of J to obtain factors of I results in factorizations whose quality in terms of the coverage of data by the computed factors is comparable to those obtained by factorizing clarified matrices in most cases. Note also that for most of the real-world datasets used in our study, reduction results in a removal of a much smaller number of rows compared to clarification, i.e., the real-world datasets seem not greatly amenable to reduction. In some cases, factorizing a reduced matrix lead to a larger number of computed factors compared to factorizing the original matrix. Also note that modifying a factorization algorithm so that it delivers a factorization of the reduced matrix that equals the factorization of the original data seems considerably more complex compared to clarification. Due to the significance of reduction in processing Boolean data, questions related to utilizing reduction for the purpose of factorization need further exploration.
- A rather general approach to factorization via size reduction of the input matrix derives from the idea of sampling. In particular, one may attempt to select only a certain percentage of rows of I (and possibly also columns) to get a smaller matrix J, factorize J, and extend the factors of J to obtain a factorization of I. From this perspective, both the approach studied in this paper and the approach described in the previous paragraph may be regarded as particular cases of sampling. In the first case, the non-selected rows are the duplicate ones; in the latter case, they are the reducible ones. In general, sampling requires a heuristic or theoretically justified method of row selection. This method is supposed to select rows that are representative of I in that the factors computed form the sample obtained from I, i.e., from the smaller matrix, provide a good factorization of I. In our preliminary exploration, we used a probability-based selection of rows with a uniform distribution of probability. Such a simple random sampling yields reasonable results for the datasets used in this paper in that selection of 20% of the rows results in a considerably faster and still rather precise factorization of the input matrix. On the other hand, the simple random sampling does not utilize any insight into the factorization problem, and hence is likely to be outperformed by better sampling methods that need to be explored.

  Note in this context that while sampling methods for Boolean matrix factorization have not been studied in the past, a recent work published in this journal [21] is an exception. In this work, the authors propose a method to select rows of the input matrix that is based on the so-called essential entries of the input matrix [4]. The basic idea is to select rows containing a large number of essential entries. The authors demonstrate that the proposed way of reducing the input matrix leads to promising results. Nevertheless, our examination revealed a notable shortcoming of [21]. Namely, reversing the logic of row selection, i.e., preferring the rows with a small number of essential entries, results in a comparable performance. In addition, the simple random sampling described in the previous paragraph, which ignores the property of essentiality at all, leads to comparable and mostly even better results. In fact, it turns out that rather than being justified by the preference of rows with a large number of essential entries, the seemingly promising experimental results in [21] are a consequence of the fact that the involved datasets contain a considerable duplicity of rows and the fact that the unintended consequence of the method in [21] is removal of duplicate rows—a phenomenon studied in our paper. An analysis of the method devised in [21] is a subject of our forthcoming note.
- The idea of removing duplicate, i.e., identical, rows and columns suggests a more general idea of considering an appropriately defined similarity of rows and columns and collapsing highly similar rows and columns for the purpose of reducing the input data instead of collapsing identical rows and columns. Such an approach would likely result in the lost of the possibility to compute exact factorizations of the input data, but could lead to yet faster computation of approximate factorizations. Considering a more general kind of redundancy is also relevant in scenarios in which the Boolean data is subject to noise (or error), mentioned in Section 4.3.2.

CRediT authorship contribution statement

Radim Belohlavek: Writing – review & editing, Writing – original draft, Validation, Supervision, Resources, Project administration, Methodology, Investigation, Formal analysis, Conceptualization; **Jakub Juracka:** Writing – review & editing, Writing – original draft, Visualiza-

[7] While Tiling is devised for the problem of tiling Boolean databases in [19], it essentially coincides with the GreCon algorithm; see also [20] for an efficient implementation of this algorithm.

tion, Validation, Software, Methodology, Investigation, Formal analysis, Data curation, Conceptualization.

Data availability

Data will be made available on request.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgement

This work was supported by grant No. IGA_PrF_2025_018 of Palacký University Olomouc.

References

[1] P. Miettinen, S. Neumann, Recent developments in Boolean matrix factorization, in: Proc. IJCAI, IJCAI, (2020), pp. 4922–4928.
[2] K.H. Kim, Boolean Matrix Theory and Applications. M. Dekker, NY, 1982.
[3] R. Belohlavek, J. Juracka, Reduce before you factorize, in: Mendeley Data V2, (2024) https://doi.org/10.17632/d73jhrxdx3.2
[4] R. Belohlavek, M. Trnecka, From-below approximations in Boolean matrix factorization: geometry and new algorithm, J. Comput. Syst. Sci. 81 (8) (2015) 1678–1697.
[5] P. Miettinen, T. Mielikainen, A. Gionis, G. Das, H. Mannila, The discrete basis problem, IEEE Trans. Knowl. Data Eng. 20 (10) (2008) 1348–1362.
[6] R. Belohlavek, V. Vychodil, Discovery of optimal factors in binary data via a novel method of matrix decomposition, Proc. SCIS ISCIS 76 (2006) 3–20.
[7] M. Kelly, R. Longjohn, K. Nottingham, The UCI machine learning repository, http://archive.ics.uci.edu/ml.
[8] N. Kushmerick, Learning to remove Internet advertisments, AGENTS 99: Proc. 3rd International Conference on Autonomous Agents, Washington, Seattle, USA, 1999, pp. 175–181.
[9] A. Ene, et al., Fast exact and heuristic methods for role minimization problems, in: Proc. SACMAT, SACMAT, 2008, pp. 1–10.
[10] P. Miettinen, Matrix Decomposition Methods for Data Mining: Computational Complexity and Algorithms, Technical Report, PhD thesis, 2009.
[11] A. Shapiro, The Role of Structured Induction in Expert Systems, Technical Report, Ph.D. Thesis, 1983.
[12] S. Myllykangas, et al., DNA copy number amplification profiling of human neoplasms, Oncogene 25 (55) (2006) 7324–7332.
[13] B. Ganter, R. Wille, Formal Concept Analysis: Mathematical Foundations, Berlin, Springer, 1999.
[14] T. Do, G. Graefe, J. Naughton, Efficient sorting, duplicate removal, grouping, and aggregation, ACM Trans. Database Syst. 47 (4) (2023) 1–35.
[15] P. Miettinen, The Discrete Basis Problem, Technical Report, MSc thesis, 2005.
[16] R. Belohlavek, J. Outrata, M. Trnecka, Toward quality assessment of Boolean matrix factorizations, Inf. Sci. 459 (2018) 71–85.
[17] R. Belohlavek, M. Trnecka, Handling noise in Boolean matrix factorization, Int. J. Approx. Reason 96 (2018) 78–94.
[18] S. Karaev, P. Miettinen, J. Vreeken, Getting to know the unknown unknowns: destructive-noise resistant Boolean matrix factorization, in: Proceedings of the 2015 SIAM International Conference on Data Mining, the 2015 SIAM International Conference on Data Mining, 2015, pp. 325–333.
[19] F. Geerts, B. Goethals, T. Mielikainen, Tiling databases, Proc. DS 2004 3245 (2004) 278–289.
[20] M. Trnecka, R. Vyjidacek, Revisiting the GreCon algorithm for Boolean matrix factorization, Knowl. Based Syst. 249 (2022) 108895.
[21] M. Trnecka, M. Trneckova, Data reduction for Boolean matrix factorization algorithms based on formal concept analysis, Knowl. Based Syst. 158 (2018) 75–80.