Information Sciences 489 (2019) 37–49

Contents lists available at ScienceDirect

Information Sciences

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

Factorizing Boolean matrices using formal concepts and iterative usage of essential entries

Radim Belohlavek, Jan Outrata*, Martin Trnecka

Department of Computer Science, Palacký University Olomouc, Czech Republic

ARTICLE INFO

Article history:
Received 29 November 2017
Revised 1 March 2019
Accepted 2 March 2019
Available online 4 March 2019

Keywords:
Boolean matrices
Binary relation
Factorization
Formal concept
Concept lattice
Factor analysis

ABSTRACT

We present a new algorithm for factorization of Boolean matrices (binary relations), i.e. for extraction of factors from relational data, which is based on a new insight into the geometry of factorizations. The algorithm exploits in an iterative manner so-called essential entries in relational data and outperforms, sometimes significantly, the available algorithms for exact and almost exact factorizations of relational data. We describe the rationale for the new approach, present our algorithm, provide its experimental evaluation, and present open problems.

1. Introduction

1.1. Problem description in brief

Factorization of relational data represents an important problem which goes back to the 1970s. Its significance derives from two basic aspects. The first is the data-analytic aspect: The extracted factors reveal important information about the whole data and thus the factorization methods may be utilized in factor analyses of relational data in a similar vein that the classical factor-analytic methods are being utilized in understanding data with real-valued attributes. The second is the reduction-of-dimensionality aspect: The factors may be regarded as new, more fundamental variables (attributes) hidden in the data, which are revealed by factorization. The data may then be represented in the less dimensional space of factors rather than the higher-dimensional space of the original attributes.

In particular, we consider data describing a relationship between a set $X = \{1, \ldots, n\}$ of objects and a set $Y = \{1, \ldots, m\}$ of attributes, i.e. the most basic but also most fundamental form of relational data. Such data plays a central role in formal concept analysis (FCA, see [8]). We utilize FCA because the theoretical analysis of factorization we need is naturally described using notions of FCA. As is well known, such data may be represented as a binary relation I, in which case $\langle i, j\rangle \in I$ means that the object i has the attribute j (or, j applies to i) while $\langle i, j\rangle \notin I$ means that i does not have j; as an $n \times m$ Boolean matrix I, in which case the same is expressed by $I_{ij} = 1$ and $I_{ij} = 0$; or as a bipartite graph with sets of nodes X and Y, in which case i has j iff there is a directed edge from i to j. By abuse of notation, we do not distinguish between these representations for

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

https://doi.org/10.1016/j.ins.2019.03.001
0020-0255/

convenience. Thus, for instance, we speak of a relation I in which $I_{ij} = 1$, display relations as Boolean matrices, and denote the set of all binary relations between the above sets X and Y by $\{0, 1\}^{n\times m}$.

Basically, the factorization problem consists in the following (see Section 2 for details): Given an object-attribute relation $I \subseteq X \times Y$, find a set $Z = \{1, \ldots, k\}$ of factors, an object-factor relation $A \subseteq X \times Z$ and a factor-attribute relation $B \subseteq Z \times Y$ such that the number k of factors is as small as possible and

$$I = A \circ B,$$

where $\circ$ denotes the product (or, composition) of relations. That is, the original object-attribute relationship is explained via factors as follows: the object i has the attribute j (i.e. $I_{ij} = 1$) if and only if there exists a factor l such that l applies to i (i.e. $A_{il} = 1$) and j is one of the particular manifestations of l (i.e. $B_{lj} = 1$). Therefore, $(A \circ B)_{ij} = \max_{l=1}^{k} \min(A_{il}, B_{lj})$.

As is well-known, this factorization problem may equivalently be rephrased as a matrix decomposition problem in terms of Boolean matrices [11], in which case one refers to Boolean matrix factorization (BMF), as well as a problem of covering a bipartite graph by bicliques [20]. Since binary relations essentially coincide with so-called formal contexts in FCA, one also speaks of factorization of formal contexts [3,8]. Interestingly, the problem is also equivalent to finding the 2-dimension of a poset [8,25].

1.2. Our contribution

We propose a new algorithm for exact and high-precision factorization of formal contexts, i.e. factorization of binary relations, or, equivalently, for BMF. The algorithm is based on a new way of exploiting the recent notion of essential part of a binary relation. The entries in an essential part have a property greatly significant for factorization, namely the fact that coverage of essential part of a given relation by certain collection of factors implies coverage by these factors of the entire relation. We observe that the operator assigning to a given relation its essential part is not idempotent, i.e. one may form the essential part of the essential part, the essential part of the latter, and so on, until the essential part stops changing. We propose to utilize this property and regard it as a possibility to focus, in constructing a factorization, on increasingly smaller and at the same time increasingly more important parts of the given relation. We provide a theorem which shows how factorizations of the smaller parts may be used to construct factorizations of the larger ones, in an iterative manner, until a factorization of the given input relation is obtained. We propose a new factorization algorithm that is based on this theorem and constructs factorizations in an iterative manner. We demonstrate by experimental comparison that our new algorithm outperforms, sometimes significantly, the other available algorithms for exact and high-precision factorization.

1.3. Directly related work

As mentioned above, results relevant to decomposition problem which we examine have been presented in the literature on binary relations, ordered sets, formal concept analysis, Boolean matrices, and graph theory; see e.g. [5,8,11,20,22,25]. The first works were inspired by data analytic needs [18,19] and already contain two important foundational observations, both relevant to our considerations: It has been observed that the decomposition problem is NP-hard, due to NP-hardness of the set basis problem [23], and that the problem is significantly connected to lattice theory. As far as algorithms for exact or high-precision decompositions are concerned, we compare our algorithm to the following algorithms, which represent the main current approaches: Proximus [12], Tiling [9], GreConD [3], Hyper [27], and GreEss [4]. Let us also mention the graph-theoretic algorithm in [6], which we, nevertheless, do not include in our comparison because the algorithm is prohibitively slow even for middle-size data. Note that many algorithms have been proposed for various modifications of the decomposition problem, of which the most significant is the discrete basis problem [16]. Of these algorithms, let us mention the classic Asso [16], PaNDa+ [15], and Nassau [10], but we not include them because they perform poorly for exact and high-precision decompositions.

Most directly relevant to our paper are approaches in which formal concepts are used as factors for factorizing binary relations. Even though vaguely described, the first insights regarding this option are found in [18,19]. Various theoretical results regarding the role of formal concepts as factors of binary relations are presented in [3]. Importantly, Belohlavek and Vychodil [3] presents two factorization algorithms, which both use formal concepts of the input formal context, i.e. the input binary relation I, as factors. The first one, GreCon browses the whole concept lattice and selects formal concepts as factors in a greedy manner. The second one, GreConD, avoids the necessity to compute the possibly large concept lattice in advance. It is based on the idea of computing candidate formal concepts on demand during factorization. GreConD is much quicker than GreCon but still, delivers factorizations of comparable quality. Another algorithm, GreEss, utilizing formal concepts as factors is presented in [4]. The algorithm is based on results regarding geometry of factorizations and role of concept lattices presented in [4]. In particular, the algorithm utilizes essential parts of the input formal contexts: Instead of factorizing I directly, the algorithm first computes factors of the essential part of I and obtains the desired factorization of I from these factors according to the theorems on the geometry of factorization. The basic property of essential entries also appears in [7].

As regards applications of the present factorization problem, the first one was the analysis of biological data presented in the above-mentioned important papers [18,19]. A number of further applications exploiting the data-analytic facet of the factorization appeared in the literature on BMF and include, e.g., several variants of problems regarding role engineering,

mining information regarding access policy, identification of topics, data summarization, and analyses of various particular datasets, [6,13,14,16,24,26]. Applications of BMF in preprocessing Boolean data and reducing dimensionality of Boolean data from machine learning perspective is another significant direction of applications of BMF, see e.g. [2,21].

2. Basic notions

Let $X = \{1, \ldots, n\}$ and $Y = \{1, \ldots, m\}$ denote the set of objects and attributes, respectively. We consider the *approximate factorization problem* [3,4]:

> Given $I \in \{0, 1\}^{n\times m}$ and prescribed error $\varepsilon \geq 0$, find $A \in \{0, 1\}^{n\times k}$ and $B \in \{0, 1\}^{k\times m}$ with k as small as possible such that $||I - A \circ B|| \leq \varepsilon$, where $||I - A \circ B|| = \sum_{i,j=1}^{m,n} |I_{ij} - (A \circ B)_{ij}|$.

Notice that for $\varepsilon = 0$, we obtain the classical exact factorization problem as a special case.

We now present basic notions in FCA [8]. A *formal context* is a triplet $\langle X, Y, I\rangle$ in which I is a binary relation between X and Y, i.e. $I \in \{0, 1\}^{n\times m}$. Associate now to $I \in \{0, 1\}^{n\times m}$ the pair $\langle {}^{\uparrow_I}, {}^{\downarrow_I}\rangle$ of operators assigning to sets $C \subseteq X$ and $D \subseteq Y$ the sets

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

This pair, which we also denote just $\langle {}^{\uparrow}, {}^{\downarrow}\rangle$, forms a Galois connection and the set of its fixpoints is denoted $\mathcal{B}(I)$, i.e.

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

In formal concept analysis, $\mathcal{B}(I)$ is called the *concept lattice* of I, and its elements $\langle C, D\rangle$ are called *formal concepts* of I. $\mathcal{B}(I)$ is indeed a complete lattice in which the partial order $\leq$ (subconcept-superconcept hierarchy) is defined by $\langle C_1, D_1\rangle \leq \langle C_2, D_2\rangle$ iff $C_1 \subseteq C_2$ iff $D_1 \supseteq D_2$. The compound mappings ${}^{\uparrow\downarrow}$ and ${}^{\downarrow\uparrow}$ are closure operators in X and Y, respectively.

As proved for exact [3] and approximate [4] factorizations of I, one may restrict to factorizations of relations I which use formal concepts as factors, because optimal factorizations are achieved this way. In particular, one may restrict to factorizations I into the product of relations $A_{\mathcal{F}} \in \{0, 1\}^{n\times k}$ and $B_{\mathcal{F}} \in \{0, 1\}^{k\times m}$, where $\mathcal{F} = \{\langle C_1, D_1\rangle, \ldots, \langle C_k, D_k\rangle\} \subseteq \mathcal{B}(I)$ is an (indexed) set of formal concepts and $A_{\mathcal{F}}$ and $B_{\mathcal{F}}$ are defined as follows:

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

for $l = 1, \ldots, k$.

Consider now for every entry $\langle i, j\rangle \in I$, i.e. $I_{ij} = 1$, the interval

$$\mathcal{I}_{ij} = [\gamma(i), \mu(j)] = \{c \in \mathcal{B}(I) \mid \gamma(i) \leq c \leq \mu(j)\},$$

in the concept lattice $\mathcal{B}(I)$, whose lower and upper bounds are the formal concepts $\gamma(i) = \langle \{i\}^{\uparrow\downarrow}, \{i\}^{\uparrow}\rangle$ and $\mu(j) = \langle \{j\}^{\downarrow}, \{j\}^{\downarrow\uparrow}\rangle$. Entries $\langle i, j\rangle$ for which this interval is minimal w.r.t. set inclusion shall be called *essential* and play a crucial role for us. They constitute a new object-attribute relation, $\mathcal{E}(I) \in \{0, 1\}^{n\times m}$, called in [4] the *essential part of* I:

$$(\mathcal{E}(I))_{ij} = 1 \quad \text{iff} \quad \mathcal{I}_{ij} \text{ is non-empty and minimal w.r.t. } \subseteq, \tag{2}$$

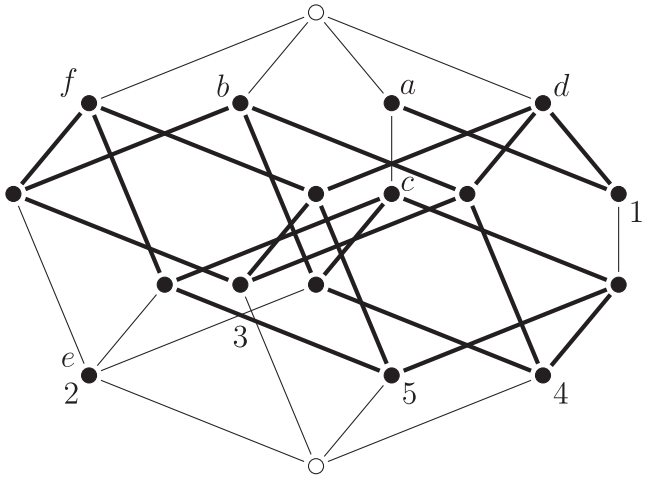
where $\subseteq$ denotes set inclusion.

Remark 1. The relation $\mathcal{E}(I)$ is easy to compute because $\mathcal{E}(I)_{ij} = 1$ iff the following two conditions are satisfied: (a) $I_{ij} = 1$; and (b) for every i' with $\{i'\}^{\uparrow} \subset \{i\}^{\uparrow}$ we have $I_{i'j} = 0$; for every j' with $\{j'\}^{\downarrow} \subset \{j\}^{\downarrow}$ we have $I_{ij'} = 0$. Note that the above condition (a) is equivalent to the fact that the interval $\mathcal{I}_{ij}$ is non-empty, and that, as is easily seen, (b) expresses the fact that $\mathcal{I}_{ij}$ is minimal w.r.t. $\subseteq$.

We now illustrate the concept of essential part by an example which we use as our running example in this paper.

Example 1. Consider the following matrix representing a relation I between the objects $1, \ldots, 5$ and the attributes $a, \ldots, f$. As we shall see, the underlined entries are just the essential 1s, i.e. those with $(\mathcal{E}(I))_{ij} = 1$:

	a	b	c	d	e	f
1	$\underline{1}$	0	0	$\underline{1}$	0	0
2	1	1	1	0	$\underline{1}$	1
3	0	$\underline{1}$	0	$\underline{1}$	0	$\underline{1}$
4	1	$\underline{1}$	$\underline{1}$	1	0	0
5	1	0	$\underline{1}$	1	0	$\underline{1}$



The diagram on the right displays the concept lattice $\mathcal{B}(I)$ of I, i.e. the picture is the labeled line diagram of $\mathcal{B}(I)$ with reduced labeling [8]. This means that the nodes and lines represent the formal concepts in $\mathcal{B}(I)$ and the partial order $\leq$ of $\mathcal{B}(I)$. Each object symbol $i \in \{1, 2, 3, 4, 5\}$ and attribute symbol $j \in \{a, b, c, d, e, f\}$ labels the concept $\gamma(i)$ and $\mu(j)$, respectively. Every node therefore represents a formal concept $\langle C, D\rangle$ where C and D consist of all objects and attributes whose labels lie on the paths below and above the node, respectively. For example, for the formal concept represented by the node labeled by 1 we have $C = \{1, 4, 5\}$ and $D = \{a, d\}$.

Let us now determine the essential part $\mathcal{E}(I)$ of I. Note that if $I_{ij} = 0$, one always has $(\mathcal{E}(I))_{ij} = 0$ since in this case, $\mathcal{I}_{ij}$ is empty, and hence the definition (2) of being an essential entry is not satisfied.

Now, whether a given entry $I_{ij} = 1$ is essential, i.e. whether one also has $(\mathcal{E}(I))_{ij} = 1$, is easily determined from the labeled line diagram of the concept lattice $\mathcal{B}(I)$. Consider, for example, the entry I_{1a}. It is clear from the labeled diagram that the corresponding interval $\mathcal{I}_{1a}$, i.e. the two-element interval bounded by the node with label 1 and the node with label a, is minimal w.r.t. set inclusion (no other interval of the form $\mathcal{I}_{ij}$ is included in $\mathcal{I}_{1a}$). Hence $(\mathcal{E}(I))_{1a} = 1$. In a similar manner, the entry I_{5c} is essential, i.e. $(\mathcal{E}(I))_{5c} = 1$, because the corresponding interval $\mathcal{I}_{5c}$, i.e. the four-element interval bounded from below and above by the nodes labels 5 and c, respectively, is minimal w.r.t. set inclusion. Hence $(\mathcal{E}(I))_{5c} = 1$. On the other hand, the entry I_{4a} is not essential, i.e. $(\mathcal{E}(I))_{4a} = 0$, because the corresponding interval $\mathcal{I}_{4a}$ is not minimal. Namely, the interval $\mathcal{I}_{4c}$ is included in $\mathcal{I}_{4a}$.

Let us now demonstrate how Remark 1 may be used to determine the essential entries without the need to inspect the concept lattice, which we did in the previous paragraph. Consider again the entry I_{1a}. Since $I_{1a} = 1$, we need to check condition (b) of Remark 1. This condition is satisfied: For one, there is no i' for which $\{i'\}^{\uparrow} \subset \{1\}^{\uparrow}$, hence the part of (b) involving i' is met; secondly, the only j' for which $\{j'\}^{\downarrow} \subset \{a\}^{\downarrow}$ are $j' = c$ and $j' = e$. However, neither c not e are incident with 1, i.e. $I_{1c} = 0$ and $I_{1e} = 0$, hence the part of (b) involving j' is met as well. Since (b) is verified, we conclude that $(\mathcal{E}(I))_{1a} = 1$. In a similar manner, we check $(\mathcal{E}(I))_{5c} = 1$. On the other hand, I_{4a} is not essential: Even though $I_{4a} = 1$, i.e. (a) is met, condition (b) is not met. Namely, putting $j' = c$ we get $\{j'\}^{\downarrow} \subset \{a\}^{\downarrow}$ but $I_{4j'} = 1$.

This way one may proceed for all entries in I to finally obtain $\mathcal{E}(I)$. The result is apparent from the above line diagram of $\mathcal{B}(I)$ and from the underlined 1s in the above matrix I: The bold part of the line diagram represents the intervals $\mathcal{I}_{ij}$ of $\mathcal{B}(I)$ that are nonempty and minimal w.r.t. set inclusion, and thus correspond to essential entries $\langle i, j\rangle$ of I. There are ten such intervals: $\mathcal{I}_{1a}$, $\mathcal{I}_{1d}$ (two-element intervals), $\mathcal{I}_{2e}$ (singleton), $\mathcal{I}_{3b}$, $\mathcal{I}_{3d}$, $\mathcal{I}_{3f}$, $\mathcal{I}_{4b}$, $\mathcal{I}_{4c}$, $\mathcal{I}_{5c}$ and $\mathcal{I}_{5f}$ (four-element intervals), whence the ten underlined entries in the matrix representing I.

In general, $\mathcal{E}(I) \subseteq I$ and in fact, $\mathcal{E}(I)$ tends to be significantly smaller than I. The significance of $\mathcal{E}(I)$ consists in the following assertion proved in [4]: In constructing a factorization of I, one may focus on the entries $\langle i, j\rangle \in \mathcal{E}(I)$, and ignore the (less important) entries in I that are not in $\mathcal{E}(I)$, because every factorization of I which covers all the entries $\langle i, j\rangle \in \mathcal{E}(I)$ is guaranteed to cover all entries contained in the original I. In this sense, $\mathcal{E}(I)$ consists just of the essential entries of the input I to which one may focus. The practical significance of $\mathcal{E}(I)$ is demonstrated by the algorithm GreEss developed in [4] which outperforms the existing algorithms.

3. IterEss: a BMF algorithm exploiting iterative essentials

While GreEss exploits essential entries represented by $\mathcal{E}(I)$ in that it looks for promising groupings of essential entries of

I, represented by $\mathcal{E}(I)$, and searches for factors of I in certain intervals in the lattice $\mathcal{B}(I)$ that correspond to these promising groupings, the way $\mathcal{E}(I)$ is utilized by our new algorithm is conceptually different and is based on two observations.

The first is an interesting property, not mentioned in [4], that the operator $\mathcal{E}(\cdot) : I \mapsto \mathcal{E}(I)$ is not idempotent, i.e. we might have $\mathcal{E}(\mathcal{E}(I)) \neq \mathcal{E}(I)$. In general, we thus consider

$$\mathcal{E}^0(I) = I \quad \text{and} \quad \mathcal{E}^{p+1}(I) = \mathcal{E}(\mathcal{E}^p(I)) \text{ for } p = 0, 1, 2, \ldots,$$

i.e.

$$\mathcal{E}^p(I) = \mathcal{E}(\mathcal{E}(\cdots(I)\cdots)),$$

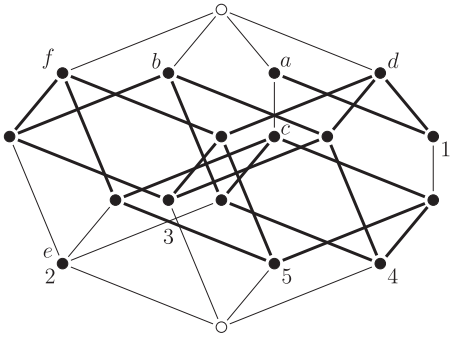
in which the operator $\mathcal{E}(\cdot)$ is applied p times.

Example 2. The following matrices demonstrate non-idempotency of the operator $\mathcal{E}(\cdot)$. In this case, $I \neq \mathcal{E}(I) \neq \mathcal{E}^2(I) \neq \mathcal{E}^3(I) \neq \mathcal{E}^4(I) = \mathcal{E}^5(I)$, i.e. the operator reaches its idempotent point after four iterations:

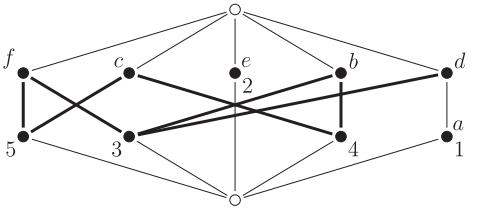
$$I = \begin{pmatrix} 1&0&0&1&0&0\\ 1&1&1&0&1&1\\ 0&1&0&1&0&1\\ 1&1&1&1&0&0\\ 1&0&1&1&0&1 \end{pmatrix}, \mathcal{E}(I) = \begin{pmatrix} 1&0&0&1&0&0\\ 0&0&0&0&1&0\\ 0&1&0&1&0&1\\ 0&1&1&0&0&0\\ 0&0&1&0&0&1 \end{pmatrix}, \mathcal{E}^2(I) = \begin{pmatrix} 1&0&0&0&0&0\\ 0&0&0&0&1&0\\ 0&1&0&1&0&1\\ 0&1&1&0&0&0\\ 0&0&1&0&0&1 \end{pmatrix}$$

$$\mathcal{E}^3(I) = \begin{pmatrix} 1&0&0&0&0&0\\ 0&0&0&0&1&0\\ 0&0&0&1&0&0\\ 0&1&1&0&0&0\\ 0&0&1&0&0&1 \end{pmatrix}, \mathcal{E}^4(I) = \begin{pmatrix} 1&0&0&0&0&0\\ 0&0&0&0&1&0\\ 0&0&0&1&0&0\\ 0&1&0&0&0&0\\ 0&0&0&0&0&1 \end{pmatrix}$$

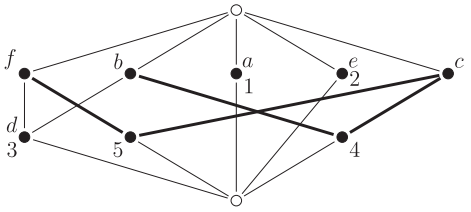
Note that computing from I its essential part $\mathcal{E}(I)$ was demonstrated in Example 1 and that computing the other matrices, $\mathcal{E}^2(I)$, $\mathcal{E}^3(I)$, and $\mathcal{E}^4(I)$, may be done in a similar manner. The corresponding concept lattices, in which the intervals corresponding to the essential entries are emphasized in bold, are:



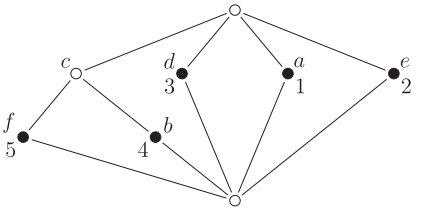
$\mathcal{B}(I)$



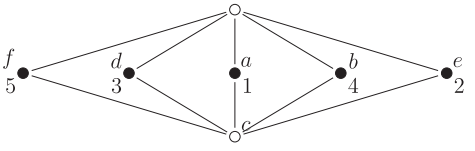
$\mathcal{B}(\mathcal{E}(I))$



$\mathcal{B}(\mathcal{E}^2(I))$



$\mathcal{B}(\mathcal{E}^3(I))$



$\mathcal{B}(\mathcal{E}^4(I))$

The second property is the observation, implicitly used in [4], that the promising groupings of essential elements in $\mathcal{E}(I)$ may, without loss of generality, be looked for in the form of rectangular areas in $\mathcal{E}(I)$ that exhaust all 1s in $\mathcal{E}(I)$. By a rectangular area in $\mathcal{E}(I)$ we mean a Cartesian subset of $\mathcal{E}(I)$, i.e. a Cartesian product $C \times D \subseteq \mathcal{E}(I)$ of some $C \subseteq \{1, \ldots, n\}$ and $D \subseteq \{1, \ldots, m\}$.

Now, since—as is well known—rectangular areas in any binary relation J are in fact possible factors of J, the problem to find promising groupings of essentials transforms to the problem to find a good factorization of matrix $\mathcal{E}(I)$. Therefore, the problem to factorize I leads to the problem to factorize $\mathcal{E}(I)$. Due to the non-idempotency of $\mathcal{E}(\cdot)$, factorizing $\mathcal{E}(I)$ may be performed the same way as for I: One computes the essential entries of $\mathcal{E}(I)$, i.e. computes $\mathcal{E}(\mathcal{E}(I))$, and attempts to factorize $\mathcal{E}(\mathcal{E}(I))$ with the prospect of finding promising seeds of essentials in $\mathcal{E}(\mathcal{E}(I))$ to be used to factorize $\mathcal{E}(I)$. The just described recursive descent may be performed until a desired depth p or until $\mathcal{E}^p(I)$ stops changing.

The particular basic property on which our new algorithm relies is the content of the following theorem. In this theorem and below, we denote for a given binary relation J and for $C \subseteq \{1, \ldots, n\}$, $D \subseteq \{1, \ldots, m\}$ by $\mathcal{I}_{C,D}$ the interval

$$\mathcal{I}_{C,D} = [\gamma(C), \mu(D)]$$

in the concept lattice $\mathcal{B}(J)$ where $\gamma(C) = \langle C^{\uparrow\downarrow}, C^{\uparrow}\rangle$ and $\mu(D) = \langle D^{\downarrow}, D^{\downarrow\uparrow}\rangle$ are the formal concepts induced by C and D, respectively ($^{\uparrow}$ and $^{\downarrow}$ are the operators induced by J).

Theorem 1. *For a given $I \in \{0, 1\}^{n\times m}$ denote by r the least integer for which $\mathcal{E}^r(I) = \mathcal{E}^{r+1}(I)$, i.e. $I \neq \mathcal{E}(I) \neq \cdots \neq \mathcal{E}^r(I) = \mathcal{E}^{r+1}(I)$. Let*

$$\mathcal{G}_r, \ldots, \mathcal{G}_1, \mathcal{G}_0 = \mathcal{F},$$

denote collections of formal concepts in the concept lattices $\mathcal{B}(\mathcal{E}^p(I))$, respectively, i.e. $\mathcal{G}_p \subseteq \mathcal{B}(\mathcal{E}^p(I))$, that satisfy the following properties:

1. *$\mathcal{G}_r$ factorizes $\mathcal{E}^r(I)$, i.e. $A_{\mathcal{G}_r} \circ B_{\mathcal{G}_r} = \mathcal{E}^r(I)$;*
2. *for every $p = r-1, r-2, \ldots, 1, 0$, and every $\langle C, D\rangle \in \mathcal{G}_{p+1}$, the set $\mathcal{G}_p$ contains at least one formal concept in the interval $\mathcal{I}_{C,D}$ in the lattice $\mathcal{B}(\mathcal{E}^p(I))$.*

Then $\mathcal{F}$ factorizes I, i.e. $A_{\mathcal{F}} \circ B_{\mathcal{F}} = I$.

Proof. The proof proceeds in an inductive manner. The inductive statement we prove is: For each $p = r, \ldots, 0$, the set $\mathcal{G}_p$ factorizes $\mathcal{E}^p(I)$. For $p = 0$, we obtain the desired claim, because $\mathcal{G}_0 = \mathcal{F}$ and $\mathcal{E}^0(I) = I$.

The base case is obviously satisfied because for $p = r$, we obtain assumption 1 of our theorem.

Inductive step: If $\mathcal{G}_p$ factorizes $\mathcal{E}^p(I)$ then $\mathcal{G}_{p-1}$ factorizes $\mathcal{E}^{p-1}(I)$. Denote for every formal concept $\langle C, D\rangle$ in $\mathcal{G}_p$ by $\langle E, F\rangle$ the corresponding formal concept in the interval $\mathcal{I}_{C,D}$ of $\mathcal{G}_{p-1}$. Note that $\langle E, F\rangle$ exists due to assumption 2 of our theorem. By virtue of [[4], Lemma 1 (a)], we have $C \subseteq E$ and $D \subseteq F$. Since this property is satisfied for each formal concept $\langle C, D\rangle$ in $\mathcal{G}_p$ and since relational composition $\circ$ is monotone, we obtain

$$A_{\mathcal{G}_p} \circ B_{\mathcal{G}_p} \subseteq A_{\mathcal{G}_{p-1}} \circ B_{\mathcal{G}_{p-1}}.$$

Since, due to the inductive hypothesis, $A_{\mathcal{G}_p} \circ B_{\mathcal{G}_p} = \mathcal{E}^p(I)$, we have

$$\mathcal{E}^p(I) \subseteq A_{\mathcal{G}_{p-1}} \circ B_{\mathcal{G}_{p-1}}.$$

Now, because $\mathcal{E}^p(I) = \mathcal{E}(\mathcal{E}^{p-1}(I))$, i.e. $\mathcal{E}^p(I)$ is the essential part of $\mathcal{E}^{p-1}(I)$, [[4], Theorem 2] yields $A_{\mathcal{G}_{p-1}} \circ B_{\mathcal{G}_{p-1}} = \mathcal{E}^{p-1}(I)$, i.e. $\mathcal{G}_{p-1}$ factorizes $\mathcal{E}^{p-1}(I)$. □

Remark 2. As one can check by inspecting the proof of Theorem 1, assumption 1 of the theorem may be replaced by a weaker condition requiring that the extensions $\langle C^{\uparrow\downarrow}, D^{\downarrow\uparrow}\rangle \in \mathcal{B}(\mathcal{E}^{r-1}(I))$ of formal concepts $\langle C, D\rangle \in \mathcal{B}(\mathcal{E}^r(I))$ cover the whole $\mathcal{E}^{r-1}(I)$ in that

$$\bigcup_{\langle C,D\rangle \in \mathcal{B}(\mathcal{E}^r(I))} C^{\uparrow\downarrow} \times D^{\downarrow\uparrow} = \{\langle i, j\rangle \mid \mathcal{E}^{r-1}(I)_{ij} = 1\}.$$

This observation may obviously be used to speed up factorization and we utilize it in the algorithm presented below.

These considerations represent the core of our new factorization algorithm, IterEss (Algorithm 1), which we now describe. We start by illustrating our algorithm with the following example.

Example 3. Let us consider again the binary relation I from Example 1. The least integer r for which $\mathcal{E}^r(I) = \mathcal{E}^{r+1}(I)$ is 4 (see Example 2). That is, we have the five-element chain

$$I \supset \mathcal{E}(I) \supset \mathcal{E}^2(I) \supset \mathcal{E}^3(I) \supset \mathcal{E}^4(I)$$

of relations, each followed by its essential part (it makes no sense to consider further iterations because $\mathcal{E}^k(I) = \mathcal{E}^4(I)$ for $k > 4$). In view of Theorem 1, a factorization of I may be obtained from a factorization of the simpler matrix $\mathcal{E}(I)$, which itself may be obtained from a factorization of the even simpler matrix $\mathcal{E}^2(I)$ and so on. Let us illustrate this process.

Algorithm 1: IterEss.

Input: Boolean matrix $I \in \{0,1\}^{n\times m}$, prescribed error $\varepsilon \geq 0$, integer p, Boolean flag *first*
Output: A set $\mathcal{F}$ of factors for which $||I - A_{\mathcal{F}} \circ B_{\mathcal{F}}|| \leq \varepsilon$

if *first* **then**
  $\mathcal{G} \leftarrow$ IterEss$(I, 0, p-1, false)$
  $U \leftarrow \{\langle i, j\rangle \mid I_{ij} = 1\}$
else
  $\mathcal{E} \leftarrow \mathcal{E}(I)$
  if $\mathcal{E} = I$ *or* $p = 0$ **then**
    $\mathcal{G} \leftarrow \{\langle \emptyset, \emptyset\rangle\}$
    if $\mathcal{E} = I$ **then return** $\mathcal{G}$
  else
    $\mathcal{G} \leftarrow$ IterEss$(\mathcal{E}, 0, p-1, false)$
  $U \leftarrow \{\langle i, j\rangle \mid \mathcal{E}_{ij} = 1\}$
  $\varepsilon \leftarrow 0$
$\mathcal{F} \leftarrow \emptyset$
while $|U| > \varepsilon$ **do**
  $s \leftarrow 0$
  foreach $\langle C, D\rangle \in \mathcal{G}$ **do**
    $J \leftarrow D^{\downarrow_I} \times C^{\uparrow_I} \cap I$
    if *first* **then** $\mathcal{E} \leftarrow J$
    $F \leftarrow \emptyset$; $s_{\langle C,D\rangle} \leftarrow 0$
    while exists $j \in C^{\uparrow_I} - F$ s.t. $|((F \cup \{j\})^{\downarrow_\varepsilon})^{\uparrow_J\downarrow_J} \times ((F \cup \{j\})^{\downarrow_\varepsilon\uparrow_\varepsilon})^{\downarrow_J\uparrow_J} \cap U| > s_{\langle C,D\rangle}$ **do**
      select j maximizing $|((F \cup \{j\})^{\downarrow_\varepsilon})^{\uparrow_J\downarrow_J} \times ((F \cup \{j\})^{\downarrow_\varepsilon\uparrow_\varepsilon})^{\downarrow_J\uparrow_J} \cap U|$
      $F \leftarrow (F \cup \{j\})^{\downarrow_\varepsilon\uparrow_\varepsilon}$; $E \leftarrow (F \cup \{j\})^{\downarrow_\varepsilon}$
      $s_{\langle C,D\rangle} \leftarrow |E^{\uparrow_J\downarrow_J} \times F^{\downarrow_J\uparrow_J} \cap U|$
    if $s_{\langle C,D\rangle} > s$ **then**
      $\langle E', F'\rangle \leftarrow \langle E, F\rangle$; $\langle C', D'\rangle \leftarrow \langle C, D\rangle$
      $s \leftarrow s_{\langle C,D\rangle}$
  add $\langle E', F'\rangle$ **to** $\mathcal{F}$
  if $\langle C', D'\rangle \neq \langle \emptyset, \emptyset\rangle$ **then remove** $\langle C', D'\rangle$ **from** $\mathcal{G}$
  $U \leftarrow U - E'^{\uparrow_J\downarrow_J} \times F'^{\downarrow_J\uparrow_J}$
return $\mathcal{F}$

First, factorizing $\mathcal{E}^4(I)$ is trivial. One can immediately see that

$$\mathcal{G}_4 = \{\langle\{1\},\{a\}\rangle, \langle\{4\},\{b\}\rangle, \langle\{3\},\{d\}\rangle, \langle\{2\},\{e\}\rangle, \langle\{5\},\{f\}\rangle\}$$

is a unique set of formal concepts factorizing $\mathcal{E}^4(I)$, whence this set is also produced by IterEss. Second, to obtain a factorization of $\mathcal{E}^3(I)$, Theorem 1 implies that it is sufficient to select for each formal concept $\langle C, D\rangle$ in the obtained factorization $\mathcal{G}_4$ a single formal concept in the interval $\mathcal{I}_{\langle C,D\rangle}$ in $\mathcal{B}(\mathcal{E}^3(I))$, i.e. one formal concept in each of the following intervals: $\mathcal{I}_{\langle\{1\},\{a\}\rangle}$, $\mathcal{I}_{\langle\{4\},\{b\}\rangle}$, $\mathcal{I}_{\langle\{3\},\{d\}\rangle}$, $\mathcal{I}_{\langle\{2\},\{e\}\rangle}$, and $\mathcal{I}_{\langle\{5\},\{f\}\rangle}$. Since these are one-element intervals, which correspond to the bold nodes in the line diagram of $\mathcal{B}(\mathcal{E}^3(I))$ in Fig. 1, our selection results in the set

$$\mathcal{G}_3 = \{\langle\{4\},\{b,c\}\rangle, \langle\{5\},\{c,f\}\rangle, \langle\{1\},\{a\}\rangle, \langle\{3\},\{d\}\rangle, \langle\{2\},\{e\}\rangle\},$$

which factorizes $\mathcal{E}^3(I)$. Analogously, to obtain a factorization of $\mathcal{E}^2(I)$, it is sufficient to obtain for every concept $\langle C, D\rangle \in \mathcal{G}_3$ a single formal concept in the interval $\mathcal{I}_{\langle C,D\rangle}$ in $\mathcal{B}(\mathcal{E}^2(I))$. Since these intervals, $\mathcal{I}_{\langle\{4\},\{b,c\}\rangle}$, $\mathcal{I}_{\langle\{5\},\{c,f\}\rangle}$, $\mathcal{I}_{\langle\{1\},\{a\}\rangle}$, $\mathcal{I}_{\langle\{3\},\{d\}\rangle}$, $\mathcal{I}_{\langle\{2\},\{e\}\rangle}$, are again one-element intervals, which in this case correspond to the bold nodes in the diagram of $\mathcal{B}(\mathcal{E}^2(I))$, the factorization is unique, namely

$$\mathcal{G}_2 = \{\langle\{3\},\{b,d,f\}\rangle, \langle\{4\},\{b,c\}\rangle, \langle\{5\},\{c,f\}\rangle, \langle\{1\},\{a\}\rangle, \langle\{2\},\{e\}\rangle\}.$$

By the same logic, for a factorization of $\mathcal{E}(I)$, one selects a single concept in the intervals $\mathcal{I}_{\langle C,D\rangle}$ in $\mathcal{B}(\mathcal{E}(I))$ for $\langle C, D\rangle \in \mathcal{G}_2$, and because these are again one-element intervals (see the bold nodes in the diagram of $\mathcal{B}(\mathcal{E}(I))$), one obtains the unique factorization

$$\mathcal{G}_1 = \{\langle\{3\},\{b,d,f\}\rangle, \langle\{4\},\{b,c\}\rangle, \langle\{5\},\{c,f\}\rangle, \langle\{1\},\{a,d\}\rangle, \langle\{2\},\{e\}\rangle\}$$

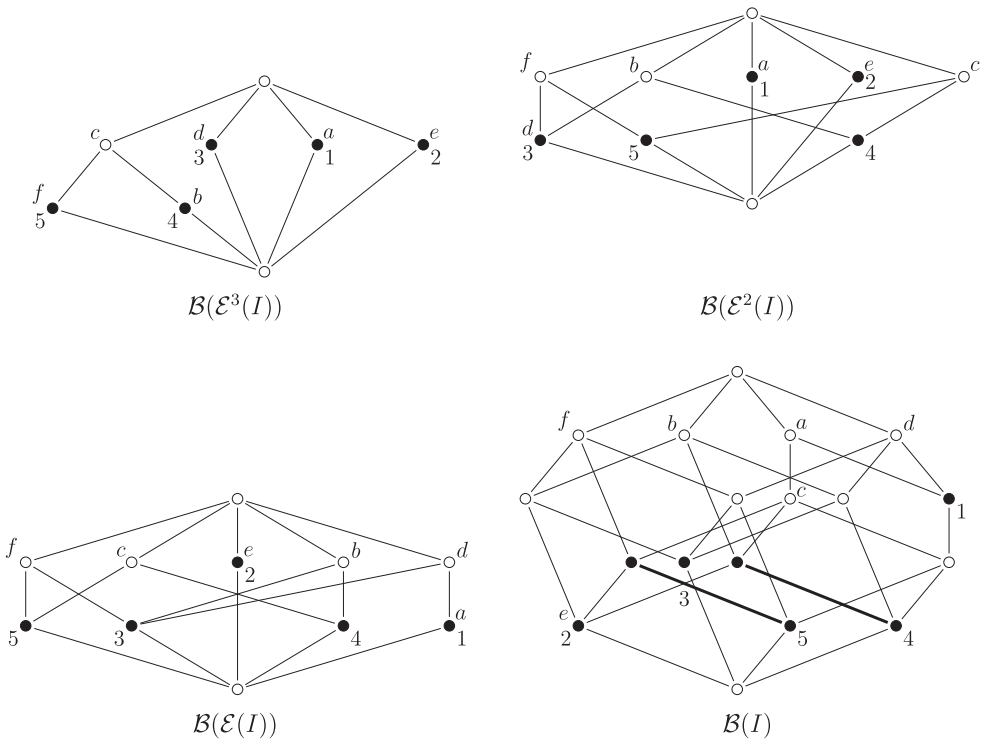


Fig. 1. Concept lattices from Example 3.

of $\mathcal{B}(\mathcal{E}(I))$. Now, to obtain finally a factorization of I, it is sufficient to select for each $\langle C, D\rangle \in \mathcal{G}_1$ one formal concept in the interval $\mathcal{I}_{\langle C,D\rangle}$ in $\mathcal{B}(I)$. In this case, the five intervals in $\mathcal{B}(I)$ are

$$\begin{aligned}
\mathcal{I}_{\langle\{3\},\{b,d,f\}\rangle} &= \{\langle\{3\},\{b,d,f\}\rangle\},\\
\mathcal{I}_{\langle\{4\},\{b,c\}\rangle} &= \{\langle\{4\},\{a,b,c,d\}\rangle, \langle\{2,4\},\{a,b,c\}\rangle\},\\
\mathcal{I}_{\langle\{5\},\{c,f\}\rangle} &= \{\langle\{5\},\{a,c,d,f\}\rangle, \langle\{2,5\},\{a,c,f\}\rangle\},\\
\mathcal{I}_{\langle\{1\},\{a,d\}\rangle} &= \{\langle\{1,4,5\},\{a,d\}\rangle\},\\
\mathcal{I}_{\langle\{2\},\{e\}\rangle} &= \{\langle\{2\},\{a,b,c,e,f\}\rangle\}.
\end{aligned}$$

These intervals are shown in bold in the line diagram of $\mathcal{B}(I)$ in Fig. 1. Using its greedy strategy described below, IterEss selects the formal concepts $\langle\{2, 4\}, \{a, b, c\}\rangle$, $\langle\{1, 4, 5\}, \{a, d\}\rangle$, $\langle\{3\}, \{b, d, f\}\rangle$, $\langle\{2, 5\}, \{a, c, f\}\rangle$, and $\langle\{2\}, \{a, b, c, e, f\}\rangle$, respectively, whence it returns the five-element set

$$\begin{aligned}
\mathcal{F} = \mathcal{G}_0 = \{&\langle\{2,4\},\{a,b,c\}\rangle, \langle\{1,4,5\},\{a,d\}\rangle, \langle\{3\},\{b,d,f\}\rangle,\\
&\langle\{2,5\},\{a,c,f\}\rangle, \langle\{2\},\{a,b,c,e,f\}\rangle\}.
\end{aligned}$$

of formal concepts that factorizes the input relation I. Note that this set is optimal because 5 is the Boolean rank of I, i.e. the least number of factors needed to decompose I.

Let us also note for completeness that the two related algorithms, GreEss and GreConD, obtain larger sets of factors for I, namely GreEss obtains the six-element set

$$\begin{aligned}
\mathcal{F} = \{&\langle\{1,4,5\},\{a,d\}\rangle, \langle\{2\},\{a,b,c,e,f\}\rangle, \langle\{3\},\{b,d,f\}\rangle,\\
&\langle\{2,4,5\},\{a,c\}\rangle, \langle\{2,3,4\},\{b\}\rangle, \langle\{3,5\},\{d,f\}\rangle\}
\end{aligned}$$

while GreConD obtains the seven-element set

$$\begin{aligned}
\mathcal{F} = \{&\langle\{2,4,5\},\{a,c\}\rangle, \langle\{1,3,4,5\},\{d\}\rangle, \langle\{2,3\},\{b,f\}\rangle,\\
&\langle\{1,2,4,5\},\{a\}\rangle, \langle\{2,3,4\},\{b\}\rangle, \langle\{2\},\{a,b,c,e,f\}\rangle,\\
&\langle\{2,3,5\},\{f\}\rangle\}.
\end{aligned}$$

Let us now describe in detail the pseudocode of IterEss (Algorithm 1). A factorization of I with a prescribed error $\varepsilon \geq 0$ is executed by calling IterEss$(I, \varepsilon, p, true)$. The value $first = true$ indicates a call of the recursive procedure IterEss in which the processed binary relation denoted I is the input binary relation, whose factorization is the ultimate goal, rather than

$\mathcal{E}^q(I)$ for some $q = 1, 2, \ldots$. In the subsequent calls of IterEss, one sets *first* $= false$ and $\varepsilon = 0$ to indicate that a factorization of $\mathcal{E}^q(I)$ is performed. In these subsequent calls, ε is set to 0 because when a factorization of $\mathcal{E}^q(I)$ is performed to group the essential entries, one needs to group (i.e. to cover) all the essential entries in $\mathcal{E}^q(I)$.

The algorithm has two parts. In the first part (l. 1–12), one obtains for the current I the groupings of its essential entries in $\mathcal{E}(I)$, and assigns to the variable U the entries in I to be covered by factors to be found in the second part (l. 13–30). Each such grouping is represented by a pair $\langle C, D\rangle$, where $C \subseteq \{1, \ldots, n\}$ and $D \subseteq \{1, \ldots, m\}$ are sets of objects and attributes, respectively. The set of all such groupings is denoted $\mathcal{G}$. The sets $\mathcal{G}$ are obtained either by a recursive call of IterEss (in the first call, l. 2; and also when further descent is desirable, l. 10) or as in l. 7 (when no further descent is desirable).

$\mathcal{G}$ represents intervals in the lattice $\mathcal{B}(I)$ in which factors of I are sought, namely $\langle C, D\rangle \in \mathcal{G}$ represents the interval $\mathcal{I}_{C,D} = [\langle C^{\uparrow\downarrow}, C^{\uparrow}\rangle, \langle D^{\downarrow}, D^{\downarrow\uparrow}\rangle]$. The factors in the intervals are sought in l. 16–26 by a greedy procedure inspired by Belohlavek and Vychodil [3]: One starts with the narrowest factors (i.e. formal concepts) in $\mathcal{I}_{C,D}$ and attempts to make them wider until the widening results in a drop in coverage by the constructed factor. The thus constructed factor $\langle E', F'\rangle$ from $\mathcal{I}_{C,D}$ is then added to the computed set $\mathcal{F}$ of factors. It follows from Theorem 1 that to obtain a factorization, it suffices to take one factor from every interval $\mathcal{I}_{C,D}$; an exception is the case $\mathcal{G} = \{\langle \emptyset, \emptyset\rangle\}$ (the limiting case of the recursion) when the interval equals the whole lattice $\mathcal{B}(I)$ and needs to be searched multiple times. This is why except for $\langle C', D'\rangle = \langle \emptyset, \emptyset\rangle$, every grouping $\langle C', D'\rangle \in \mathcal{G}$ for which a factor was already found in $\mathcal{I}_{C',D'}$ is removed from $\mathcal{G}$ in l. 28. Lastly, the so far uncovered entries in U that are now covered by the newly constructed factor $\langle E', F'\rangle$ are removed from U in l. 29. The set $\mathcal{F}$ of the constructed factors is then returned in l. 30.

Theorem 2. *IterEss is correct.*

Proof. We need to check that for every Boolean matrix I, a prescribed error ε, integer p, IterEss(I, ε, p, *true*) delivers a set $\mathcal{F}$ of factors for which $||I - A_{\mathcal{F}} \circ B_{\mathcal{F}}|| \leq \varepsilon$. As we have seen above, the algorithm is designed to follow the logic of Theorem 1, which serves as the principal argument for correctness of IterEss. As explained above, the algorithm computes factorizations in an iterative manner and distinguishes via the variable *first* whether the factorized matrix is the input matrix I or a matrix of the form $\mathcal{E}^q(I)$. In both cases, I and $\mathcal{E}^q(I)$, factorization proceeds as in the GreEss algorithm, with the provision that for *first* $= true$, factorizaton stops once the precision given by ε is reached, while for *first* $= false$, an exact factorization is computed, as required by Theorem 1. To obtain a factorization of I as well as $\mathcal{E}^q(I)$, the strategy adopted from GreEss demands a factorization of $\mathcal{E}(I)$ in the case of I and a factorization of $\mathcal{E}^{q+1}(I)$ in the case of $\mathcal{E}^q(I)$. These factorizations are obtained recursively, i.e. via IterEss again, except for the last step of the recursive descent, in which the factorization is obtained by a procedure inspired by Algorithm 2 of [3].

Correctness of IterEss therefore follows by the following inductive argument: The fact that a correct factorization is obtained in the last step of the recursive descent follows from correctness of Algorithm 2 of [3] (basis of induction). The fact that a correct factorization of $\mathcal{E}^q(I)$ is obtained from a factorization of $\mathcal{E}^{q+1}(I)$ follows from Theorem 1, correctness of GreEss and the above remarks regarding our adoption of IterEss (inductive step). □

Remark 3. An upper bound of worst-case computational complexity of IterEss may be derived as follows. When invoked with argument *first* $= true$, IterEss (Algorithm 1) computes a factorization of I by performing factorizations of $\mathcal{E}^q(I)$, for which purpose it calls itself recursively with *first* $= false$ at lines 2 and 10.

In each recursive invocation, $\mathcal{E}(\mathcal{E}^{q-1}(I))$ (or $\mathcal{E}(I)$) is computed (line 5), which may be done in time $O(||I||nm)$, where $||I||$ denotes the number of entries in I equal to 1. Namely, one verifies for every entry of $\mathcal{E}^{q-1}(I)$ (or I) the condition (b) in Remark 1 which takes time $O(nm) + O(mn) = O(nm)$ since computing $\{i\}^{\uparrow}$ and $\{j\}^{\downarrow}$ (and verifying $\subseteq$) takes time $O(m)$ and $O(n)$, respectively.

Then, in the last recursive invocation for which $\mathcal{E}^q(I) \neq \mathcal{E}^{q-1}(I)$ (or $\mathcal{E}(I) \neq I$) the factorization of $\mathcal{E}^q(I)$ is obtained by a procedure (ll. 13–30) inspired by Algorithm 2 of [3]. This procedure runs in time $O(||I||nm^3)$. Indeed, the loop in ll. 14–29 repeats at most $||\mathcal{E}^q(I)|| = O(||I||)$ times and the loop in ll. 16–26 proceeds in this last recursive invocation only once, since $\mathcal{G} = \{\langle \emptyset, \emptyset\rangle\}$. Inside the loop, the most expensive operation is computing $((F \cup \{j\})^{\downarrow_\varepsilon\uparrow_\varepsilon})^{\downarrow_J\uparrow_J}$, which takes time $O(nm)$, but which is performed inside the loop in ll. 20–23. That loop in fact consists of two loops, the outer 'while exists j' and the inner 'select j' (line 21), and the operation is performed inside the inner one. The outer one proceeds at most m times (no more attributes can be added to F) and the inner one is executed at most $m + 1 - j$ times within the jth iteration of the outer one (maximal number of remaining attributes to select from). Hence, the inner loop is executed $O(m^2)$ times. Putting together, the above-mentioned procedure inspired by Algorithm 2 of [3] indeed runs in time $O(||I||nm^3)$.

In all other invocations of the algorithm (i.e. recursive ones after computing $\mathcal{E}^q(I) \neq \mathcal{E}^{q-1}(I)$ (or $\mathcal{E}(I) \neq I$), and the first one for I), a factorization of $\mathcal{E}^q(I)$ or I is obtained as in the GreEss algorithm. This is done by the same procedure as the one above in ll. 13–30, except for that the loop in ll. 14–29 repeats at most $||I|| - \varepsilon = O(||I||)$ times and the loop in ll. 16–26 now repeats $||\mathcal{G}|| = O(||I||)$ times (the number of intervals in $\mathcal{G}$ cannot be greater than $||I||$). Hence the procedure now runs in time $O(||I||^2nm^3)$. Together with the preceding computation of $\mathcal{E}(\cdot)$, the body of each invocation of IterEss without the recursive calls runs in time $O(||I||nm) + \max(O(||I||nm^3), O(||I||^2nm^3)) = O(||I||^2nm^3)$; note that this coincides with the upper bound of computational complexity of the GreEss algorithm derived in [4]. Observe that our way of deriving the bound is rather loose in several steps; obtaining a tighter bound constitutes an interesting, rather combinatorial, problem involving a deeper insight into the behavior of the $\mathcal{E}(\cdot)$ operator.

Table 1
Real datasets used in experimental evaluation.

Dataset	Objects	Attributes	Density
Advertisement	3279	1557	0.884
Americas-small	3447	1587	0.019
Apj	2044	1164	0.003
Chess	3196	76	0.489
DNA	4590	392	0.015
Mushroom	8124	119	0.193
Paleo	501	139	0.051

Table 2
Datasets used in experimental evaluation.

dataset	$p_{\max}$	p_{opt}	$\frac{\lvert\mathcal{E}^1(I)\rvert}{\lvert I\rvert}$	$\frac{\lvert\mathcal{E}^2(I)\rvert}{\lvert I\rvert}$	$\frac{\lvert\mathcal{E}^3(I)\rvert}{\lvert I\rvert}$	$\frac{\lvert\mathcal{E}^4(I)\rvert}{\lvert I\rvert}$	$\frac{\lvert\mathcal{E}^5(I)\rvert}{\lvert I\rvert}$	$\frac{\lvert\mathcal{E}^6(I)\rvert}{\lvert I\rvert}$
Advertisement	5	3	0.227	0.151	0.134	0.132	0.132	–
Americas-small	4	2	0.487	0.479	0.475	0.475	–	–
Apj	2	2	0.436	0.431	–	–	–	–
Chess	5	3	0.603	0.532	0.518	0.518	0.518	–
DNA	3	2	0.064	0.059	0.059	–	–	–
Mushroom	6	3	0.444	0.418	0.382	0.379	0.369	0.367
Paleo	5	4	0.539	0.360	0.186	0.088	0.080	–

Finally, the algorithm recursively computes factorizations of $\mathcal{E}^q(I)$ until $\mathcal{E}^q(I) = \mathcal{E}^{q-1}(I)$ (or $\mathcal{E}(I) = I$) or the number of recursive invocations becomes greater that the argument p (l. 6). By this condition the number of invocations is bounded by $O(||I||)$ since $\mathcal{E}^q(I)$ and $\mathcal{E}^{q-1}(I)$ (or $\mathcal{E}(I)$ and I) differ in the worst case by a single entry only. Summing up, IterEss has a polynomial upper bound of worst-case computational complexity $O(||I||^3 nm^3)$. Note, however, that the bound $O(||I||)$ for the number of invocations of IterEss is a theoretical one, in practice the number is usually quite low (in comparison to $||I||$), as seen in experiments in the following section (Table 2, values of $p_{\max}$). The upper bound of the worst-case computational complexity of IterEss then becomes a constant factor times $O(||I||^2 nm^3)$, which is the same as the bound obtained for GreEss [4].

4. Experimental evaluation

We used several real datasets employed in benchmarks on factorization of relational data and report our results for the seven selected datasets in Table 1. Each dataset occupies a single row in this table, which contains the name, the number n of objects, the number m of attributes, and the density of the dataset, i.e. the number $\frac{||I||}{n \cdot m}$ where $||I||$ is the number of entries $\langle i, j \rangle$ with $I_{ij} = 1$.

The Advertisement data [1] describes a collection of possible advertisements (objects) on the Internet. The attributes represent the geometry of the images as well as the textual information in the advertisement. Both the Americas-small [6] and Apj [6] represent network access rules in use in the Hewlett Packard company to control connectivity of external business partners. The Chess data [1] describes chess games of certain kind described by attributes representing the positions of pieces on the board. The DNA data [17] describes DNA copy number amplifications. These activate oncogenes and are the hallmarks of almost all advanced tumors. The attributes represent chromosomal loci, i.e. positions on a chromosome. The Mushroom dataset [1] describes samples corresponding to 23 species of gilled mushrooms in the Agaricus and Lepiota family.[1] The Paleo dataset[2] describes fossil records per location (objects) by means of various properties such as size, genome, species, etc. (attributes).

In addition to IterEss, we employed five other algorithms in our comparison: GreEss [4], GreConD [3], Tiling [9], Hyper [27], and Proximus [12]. They represent the currently available competitive algorithms designed for producing exact and almost exact decompositions. In addition to these algorithms, we are aware of the graph-theoretic algorithm in [6], which is, nevertheless, prohibitively slow and we do not include it. Let us also mention that many algorithms have been proposed for various modifications of the decomposition problem (see Section 1.3). Since these algorithms perform poorly in computing exact and almost exact decompositions, we do not include them.

GreEss [4] is a direct predecessor of IterEss in that it uses formal concepts of the input binary relation as factors, and utilizes essential parts of relations. In particular, the algorithm exploits the basic property of essential parts, namely sufficiency for exact factorization of a given binary relation to cover by factors the essential part of the relation. The coverage of the essential part in GreEss is accomplished using a strategy inspired by the GreConD algorithm, which we describe

[1] The entries are taken from *The Audubon Society Field Guide to North American Mushrooms*, New York: Alfred A. Knopf, 1981.
[2] NOW public release 030717, available from `http://www.helsinki.fi/science/now/`.

Table 3
Numbers of factors needed for the prescribed coverage.

dataset	coverage	ITERESS	GREESS	GRECOND	TILING	HYPER	PROXIMUS
Advertisement	90%	268	266	265	265	1122	538
	95%	380	376	385	385	1324	620
	100%	708	721	767	767	1528	763
Americas-small	90%	15	14	14	14	296	136
	95%	32	33	30	30	483	162
	100%	184	190	197	197	1571	349
Apj	90%	269	268	271	271	647	443
	95%	327	328	332	332	823	495
	100%	453	455	464	465	1165	578
Chess	90%	32	33	33	33	53	47
	95%	43	45	47	47	60	54
	100%	98	113	124	123	90	75
DNA	90%	191	190	195	195	265	277
	95%	239	237	273	274	304	316
	100%	368	372	495	496	363	370
Mushroom	90%	45	47	46	47	57	63
	95%	57	61	62	62	70	82
	100%	99	105	120	120	123	113
Paleo	90%	107	106	110	112	107	113
	95%	122	122	127	128	122	127
	100%	139	145	151	156	139	139

below; the computation of a factorization of the given relation from the coverage of its essential part is performed using a particular greedy search in certain intervals in the concept lattice associated to the input relation.

GRECOND [[3], Algorithm 2] makes use of formal concepts as factors and implements a particular greedy search for these factors. This search improves the idea underlying the basic set cover greedy algorithm, in that it allows for computing the factor formal concepts "on demand", i.e. without the need to compute first the whole concept lattice of the input binary relation. This strategy leads to orders of magnitude time saving while retaining the quality of decomposition compared to the direct adoption of the set cover algorithm.

TILING [9] implements a greedy search for factors basically following the strategy of the basic greedy algorithm for set cover mentioned in the previous paragraph. Unlike GRECOND, it browses through all maximal Cartesian subrelations (called tiles in [9]) of a given binary relation, which makes it much slower than GRECOND. Note also that equivalent to TILING but considerably more efficient is the independently proposed Algorithm 1 in [3], which computes all the Cartesian subrelations in advance.

HYPER [27] is another algorithm that produces from a given binary relation I a set $\mathcal{F}$ of its Cartesian subrelations (called hyperrectangles by the authors) that provide exact decomposition of I. The aim is to find $\mathcal{F}$ with the smallest description length, thus employing the minimal description length principle (MDLP): The cost of a Cartesian subrelation $C \times D$ in $\mathcal{F}$, where C and D are the sets of objects and attributes that determine the Cartesian subrelation, is $|C| + |D|$, and the cost of $\mathcal{F}$ is the sum of the costs of the Cartesian closed relations in $\mathcal{F}$. The hyperrectangles are looked for using a particular greedy strategy which utilizes information regarding certain frequently occurring sets of attributes in the relation. A disadvantage of HYPER consists in that it very often produces narrow, column-like factors.

PROXIMUS [12] is a recursive factorization algorithm that makes use of a particularly modified semi-discrete decomposition method (SDD). The algorithm first computes a single factor in the given relation (the so-called rank-1 approximation) along with its corresponding residual relation. The recursive call with the residual relation as the new input relation then follows. The rank-1 approximation is determined via the SDD error function.

To illustrate the impact of utilizing for factorization the essential part of the input relation I, as opposed to computing the factorization directly from I, we include in Table 2 the following information. The first column displays the number $p_{\max}$ of iterations for which it makes sense to reduce I by means of $\mathcal{E}(\cdot)$, i.e. the least number p for which $\mathcal{E}^p(I) = \mathcal{E}^{p+1}(I)$, for each particular dataset. The second column shows the number p_{opt} for which ITERESS achieves the least number of factors. The next columns display the numbers $\frac{||\mathcal{E}^p(I)||}{||I||}$, for $p = 1, \ldots, p_{\max}$, which represent the reduction in size of the number of entries to be covered when factorizing the essential part $\mathcal{E}^p(I)$ compared to factorizing the input relation I. One may observe that the reduction is significant. For instance, the number 0.134 for the dataset Advertisement and $p = 3$ tells us that in order to factorize this dataset, one may restrict to the problem of factorizing the relation $\mathcal{E}^3(I)$ containing only 13.4% of all the 45 139 entries contained in the input relation I.

The performance of ITERESS in terms of its ability to factorize formal contexts (i.e. Boolean matrices) on the selected datasets and its comparison to the above algorithms is provided in Table 3. For this purpose, we employ the coverage function, c, which assigns to the first l factors computed by the algorithm, i.e. to the corresponding $n \times l$ and $l \times m$ matrices

Table 4
Running times in seconds of GreConD on selected datasets.

Advertisement	Americas-small	Apj	Chess	DNA	Mushroom	Paleo
5.32	0.58	1.06	1.18	0.86	1.11	0.04

$A(l)$ and $B(l)$, respectively, the number

$$c(l) = 1 - \frac{||I - A(l) \circ B(l)||}{||I||}. \tag{3}$$

The term "coverage" is derived from the fact that $c(l)$ may be regarded as the portion of input data I (more precisely, portion of the entries with 1 in I) that is correctly covered by the l factors. Since we intend to demonstrate performance of IterEss for exact and almost exact decompositions, we depict the coverage values for 90%, 95%, and 100%, i.e. for $c = 0.90$, $c = 0.95$, and $c = 1.00$. For each of these coverage values and each of the examined algorithms, the table displays the least number of factors needed to achieve the prescribed coverage. That is, the value 99 for the Mushroom and 100% coverage in the column for IterEss algorithm means that the algorithm needs to obtain 99 factors to achieve exact factorization of the Mushroom data.

As for the setting of the algorithms, we used the default setting of parameters for Proximus, as recommended by the authors. We ran IterEss for $p = 1, 2, \ldots,$ until $\mathcal{E}^p(I)$ stopped changing, and display the number of factors corresponding to the best value p_{opt} of p. Typically, this value is 2 or 3 for these and other real datasets we employed; see Table 2.

As one can see, IterEss produces the least numbers of factors on most datasets when exact or high-precision factorizations are required. The only exception is exact factorization of the Chess dataset for which Hyper and Proximus yield smaller numbers of factors. It should be noted, however, that both Hyper and Proximus tend to produce very narrow, column-like factors, which property leads in their poor performance for high-precision (almost exact but not exact) factorizations. As the table shows, the IterEss, as well as the other algorithms, need significantly smaller numbers of factors for these high-precision factorizations (see also [4] for a more detailed evaluation of Hyper). Note also that the numbers of factors needed by IterEss are sometimes significantly smaller, by about 5%, compared to the smallest numbers of factors reported in the literature. We also observed that as the required coverage drops to about 50–80%, IterEss gets outperformed by GreEss, Tiling, and GreConD. This may be interpreted as follows: Up to a certain precision, the purely greedy approaches win over IterEss. However, as the required precision reaches 90%, the more sophisticated strategy of IterEss delivers better results. Let us also note that for even smaller required precision, other algorithms are available, such as the classic Asso [16].

To sum up, IterEss turns out to be the best available algorithm for exact and high-precision factorizations. These are, naturally, useful factorizations. In some areas, they are the only factorizations of interest (examples are graph theory in which the factorization problem corresponds to the problem of covering bipartite graphs by bicliques, see e.g. [20]; the role mining problem [13]; or preprocessing classification data [21]).

Before concluding the paper, let us briefly comment on running times of the employed algorithms. Generally, in BMF an exact running time of an algorithm is not a primary concern. Thus, instead of the exact running times, we present a relative comparison of them, which is a standard practice for BMF algorithms. This comparison captures our experience with the algorithms not only on the presented datasets, but also on other real-word and synthetic data we used in our experiments. In particular, we present our comparison for the datasets in Table 1. The overall fastest algorithm is GreConD, which is not surprising because the algorithm does not perform any data preprocessing and utilizes a very fast heuristic for computing the factors. According to our evaluations and experience, GreConD implemented in the C language and optimized for speed factorizes datasets in Table 1 in order of seconds on an ordinary PC. GreEss, which first covers by factors the essential part $\mathcal{E}(I)$ of the given relation I and then using that coverage computes a factorization of I, is about $2\times$ slower than GreConD. The present algorithm, IterEss, is only about $1.5\times$ slower than GreEss, even though it factorizes a given relation I iteratively from a non-reducible relation $\mathcal{E}^r(I)$ of essential entries of I rather than from $\mathcal{E}(I)$ as GreEss does. Note that in the evaluation we used as r the maximal number of iterations for which it makes sense to reduce I by means of $\mathcal{E}(\cdot)$, i.e. the number $p_{\max}$ from Table 2, for each particular dataset. Hyper and Proximus are both about $5\times$ slower than GreConD. Note that the time consumed by Hyper depends on the size of the set of frequent itemsets which may be exponential in the number of attributes. Tiling is the slowest of the compared algorithms, on average about $400\times$ slower than GreConD. Similarly as Hyper, in selecting each tile Tiling browses the set of all maximal tiles which is usually very large and at worst may be exponential in terms of the minimum of the number of objects and the number of attributes. Note also that our observations agree with those in [4].

To put the relative performances in relation with actual running times, we present in Table 4 the running times of GreConD on selected datasets mentioned above. Note that the times correspond to a considerably optimized implementation of GreConD in the C language that was run on a desktop/server computer ($2\times$ CPU 3.2 GHz with 3GB of RAM running 32bit Linux operating system).

5. Conclusion

We presented a new algorithm for factorizing binary relations. The algorithm is based on a certain iterative usage of the notion of essential part of relation: A factorization of a given binary relation I is computed by first reducing I to a non-reducible relation $\mathcal{E}^r(I)$ of essential entries, which is as a rule considerably smaller than the input relation I, and then computing in an iterative manner a factorization of I from a factorization of the simpler relation $\mathcal{E}^r(I)$. The resulting algorithm outperforms, sometimes significantly, the current algorithms for exact and high-precision factorization. Our implementation of the algorithm can be downloaded from http://fcalgs.sourceforge.net.

In addition to presenting a new factorization algorithm, our purpose was to further demonstrate significance of obtaining theoretical insights into the factorization problem, which may naturally accompany the otherwise greedy strategies that are implied by the provable difficulty of the problem. We regard the following topics promising and worth further exploration. First, explore more the role of essential parts of relations in various other factorization algorithms and, from a broader perspective, in methods dealing with relational data. Secondly, even though essential parts of binary relations are theoretically understood to certain extent, we believe that a better understanding of certain important aspects will make it possible to further advance factorization algorithms. In particular, a description of a relationship of the concept lattice $\mathcal{B}(I)$ of original relation and the concept lattice $\mathcal{B}(\mathcal{E}(I))$ of the essential part of the original relation is of crucial importance in this regard. Third, exploit the reduction-of-dimensionality aspect of the new factorization method from machine learning perspective.

Acknowledgments

Supported by Grant No. 15-17899S 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 was co-financed by the European Social Fund and the state budget of the Czech Republic (the present research has been conducted in the follow-up period of this project). Support by Grant No. IGA_PrF_2018_030 of IGA of Palacký University is also acknowledged.

References

[1] K. Bache, M. Lichman, UCI machine learning repository, Irvine, CA: University of California, School of Information and Computer Science, 2013. http://archive.ics.uci.edu/ml.
[2] R. Belohlavek, J. Outrata, M. Trnecka, Impact of boolean factorization as preprocessing methods for classification of boolean data, Ann. Math. Artif. Intell. 72 (1–2) (2014) 3–22.
[3] R. Belohlavek, V. Vychodil, Discovery of optimal factors in binary data via a novel method of matrix decomposition, J. Comput. Syst. Sci. 76 (1) (2010) 3–20.
[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] R.A. Brualdi, H.J. Ryser, Combinatorial Matrix Theory, Cambridge University Press, 1991.
[6] A. Ene, et al., Fast exact and heuristic methods for role minimization problems, in: Proc. SACMAT, 2008, pp. 1–10.
[7] B. Ganter, C.V. Glodeanu, Ordinal factor analysis, Lect. Notes Comput. Sci. 7278 (2012) 128–139.
[8] B. Ganter, R. Wille, Formal Concept Analysis: Mathematical Foundations, Springer, Berlin, 1999.
[9] F. Geerts, B. Goethals, T. Mielikäinen, Tiling databases, in: Proc. Discovery Science, 2004, pp. 278–289.
[10] S. Karaev, P. Miettinen, J. Vreeken, Getting to know the unknown unknowns: destructive-noise resistant boolean matrix factorization, in: SDM, 2015, pp. 325–333.
[11] K.H. Kim, Boolean Matrix Theory and Applications, M. Dekker, NY, 1982.
[12] M. Koyuturk, A. Grama, PROXIMUS: a framework for analyzing very high dimensional discrete attributed datasets, in: ACM SIGKDD, 2003, pp. 147–156.
[13] H. Lu, J. Vaidya, V. Atluri, Optimal boolean matrix decomposition: application to role engineering, in: Proc. IEEE ICDE, 2008. 297–30.
[14] H. Lu, J. Vaidya, V. Atluri, Y. Hong, Constraint-aware role mining via extended boolean matrix decomposition, IEEE Trans. Dependable Secure Comp. 9 (5) (2012) 655–669.
[15] C. Lucchese, S. Orlando, R. Perego, Mining top-k patterns from binary datasets in presence of noise, in: SIAM DM, 2010, pp. 165–176.
[16] P. Miettinen, T. Mielikäinen, A. Gionis, G. Das, H. Mannila, The discrete basis problem, IEEE Trans. Knowledge and Data Eng. 20 (10) (2008) 1348–1362.
[17] S. Myllykangas, et al., DNA copy number amplification profiling of human neoplasms, Oncogene 25 (55) (2006) 7324–7332. 2006.
[18] D.S. Nau, Specificity covering, Tech. rep. cs-1976-7, Duke University, 1976.
[19] 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.
[20] J. Orlin, Contentment in graph theory: covering graphs with cliques, in: Proc. Kon. Neder. Akad. Wet., Amsterdam, ser. A, volume vol. 80, 1977.
[21] J. Outrata, Boolean factor analysis for data preprocessing in machine learning, in: Proc. ICMLA, 2010, pp. 899–902.
[22] G. Schmidt, Relational Mathematics, Cambridge University Press, 2011.
[23] L. Stockmeyer, The set basis problem is NP-complete, Tech. rep. rc5431, IBM, Yorktown Heights, NY, USA, 1975.
[24] N. Tatti, J. Vreeken, Comparing apples and oranges: measuring differences between exploratory data mining results, Data Min. Knowl. Discov. 25 (2012) 173–207.
[25] W.T. Trotter, Combinatorics and Partially Ordered Sets: Dimension Theory, Johns Hopkins University Press, Baltimore, MD, 1992.
[26] J. Vaidya, V. Atluri, Q. Guo, The role mining problem: finding a minimal descriptive set of roles, in: Proc. SACMAT, 2007, pp. 175–184.
[27] Y. Xiang, R. Jin, D. Fuhry, F.F. Dragan, Summarizing transactional databases with overlapped hyperrectangles, in: Data Mining and Knowl. Discovery, volume 23, 2011, pp. 215–251.