MDL4BMF: Minimum Description Length for Boolean Matrix Factorization

PAULI MIETTINEN, Max-Planck Institute for Informatics
JILLES VREEKEN, Max-Planck Institute for Informatics, Saarland University, University of Antwerp

Matrix factorizations—where a given data matrix is approximated by a product of two or more factor matrices—are powerful data mining tools. Among other tasks, matrix factorizations are often used to separate global structure from noise. This, however, requires solving the "model order selection problem" of determining the proper rank of the factorization, that is, to answer where fine-grained structure stops, and where noise starts.

Boolean Matrix Factorization (BMF)—where data, factors, and matrix product are Boolean—has in recent years received increased attention from the data mining community. The technique has desirable properties, such as high interpretability and natural sparsity. Yet, so far no method for selecting the correct model order for BMF has been available. In this article, we propose the use of the Minimum Description Length (MDL) principle for this task. Besides solving the problem, this well-founded approach has numerous benefits; for example, it is automatic, does not require a likelihood function, is fast, and, as experiments show, is highly accurate.

We formulate the description length function for BMF in general—making it applicable for any BMF algorithm. We discuss how to construct an appropriate encoding: starting from a simple and intuitive approach, we arrive at a highly efficient data-to-model–based encoding for BMF. We extend an existing algorithm for BMF to use MDL to identify the best Boolean matrix factorization, analyze the complexity of the problem, and perform an extensive experimental evaluation to study its behavior.

Categories and Subject Descriptors: H.2.8 [**Database Management**]: Database Applications—*Data mining*

General Terms: Theory, Algorithms, Experimentation

Additional Key Words and Phrases: Boolean matrix factorization, model order selection, model selection, Boolean rank, pattern sets, summarization, minimum description length principle, MDL, parameter free

ACM Reference Format:
Pauli Miettinen and Jilles Vreeken. 2014. MDL4BMF: Minimum description length for Boolean matrix factorization. ACM Trans. Knowl. Discov. Data 8, 4, Article 18 (October 2014), 31 pages.
DOI: http://dx.doi.org/10.1145/2601437

1. INTRODUCTION

A typical task in data mining is to find observations and variables that behave similarly. Consider, for instance, the standard example of supermarket basket data. We are given

Jilles Vreeken is supported by the Cluster of Excellence "Multimodal Computing and Interaction" within the Excellence Initiative of the German Federal Government, and by a Post-Doctoral Fellowship of the Research Foundation—Flanders (FWO).
Authors' address: P. Miettinen, Max-Planck-Institute for Informatics, Campus E1 4, D-66123 Saarbrücken, Germany; email: pauli.miettinen@mpi-inf.mpg.de; J. Vreeken, Max-Planck Institute for Informatics, and Saarland University, Campus E1 7, D-66123, Saarbrücken, Germany; and Department Wiskunde en Informatica, Universiteit Antwerpen, Middelheimlaan 1, 2020 Antwerp, Belgium; email: jilles@mpi-inf.mpg.de.

transactions over items, and we want to find groups of transactions and groups of items such that we can (approximately) represent our data in terms of these groups, instead of the original records. Such a representation is called a *low-dimensional representation of the data* and is usually obtained using some form of matrix factorization.

In matrix factorizations, the input data (represented as a matrix) is decomposed into two (or more) factor matrices. Usually the aim is to have low-dimensional factor matrices whose product approximates the original matrix well. By imposing different constraints, one obtains different factorizations. Perhaps the two best-known factorizations are Singular Value Decomposition (SVD), closely related to Principal Component Analysis (PCA), and Nonnegative Matrix Factorization (NMF). SVD and PCA restrict the factor matrices to be orthogonal, whereas NMF requires the data and the factor matrices to be nonnegative.

When the input data is Boolean (i.e., contains only 0s and 1s, as is typical with supermarket basket data), one can apply Boolean Matrix Factorization (BMF). Similarly to NMF, it restricts the factor matrices for added interpretability and sparsity. In BMF, the factor matrices are required to be Boolean (i.e., contain only 0s and 1s). Also, the matrix product is changed, from normal to Boolean. As a consequence, it is possible that BMF obtains smaller reconstruction error than SVD for the same decomposition size—something that NMF, by definition, cannot do [Miettinen 2009]. Furthermore, it can be shown that for sparse Boolean matrices, there is always a sparse exact factorization [Miettinen 2010].

But no matter what factorization method one applies, one always has to solve the model order selection problem: What is the correct number of latent dimensions? In some situations the answer is obvious, for example, if a user wants to have a three-dimensional representation of the data (say, for visualization). But when the user wants a good description of the structure in the data, selecting the number of latent dimensions comes down to the question: What is the correct rank of the latent structure? That is, what is structure and what is noise?

Whereas various methods have been proposed to answer this question for non-Boolean matrix factorizations, varying from statistical methods based on likelihood scores (such as the Bayesian Information Criterion, BIC [Schwarz 1978]) to subjective analysis of error (so-called “elbow methods”), there is no known applicable method for selecting the model order for BMF (other than visual analysis of errors).

In this article, we study the model order selection problem in the framework of BMF. To that end, we merge two orthogonal lines of research, namely those of Boolean matrix factorizations and Minimum Description Length (MDL) principle [Rissanen 1978]. We formulate description length functions that can be used for model order selection with any BMF algorithm. We then extend an existing algorithm for BMF to use MDL in an effective way, and via extensive experimental evaluation, we show that using our description length formulation, the algorithm is able to identify the correct number of latent dimensions in synthetic and real-world BMF tasks.

Besides studying how we can apply MDL to the end of model order selection for BMF, a specific goal of this article is to construct a good MDL encoding for BMF. This is not a trivial task, as there are no known Universal Codes for BMF models; these are encodings for which the expected lengths of the code words are within a constant factor of the true probability distribution underlying the data, regardless of what this distribution actually is [Grünwald 2007]. As such, we will have to device a two-part MDL encoding that rewards structure and punishes noise. This involves a number of choices, which by MDL, we can make in a principled manner: fewer bits are better. We will start by considering simple and intuitive encodings, which we will incrementally improve through established information theoretic insights. We will empirically explore the quality of these encodings and show the necessity of using more refined insights to obtain good model order estimates.

This article is an extended version of a preliminary conference version [Miettinen and Vreeken 2011], improving it in a number of ways. First, we develop our encodings further. We introduce two new BMF encodings that, by employing highly efficient data-to-model codes, have superior noise/structure separation quality, which is reflected by the model order estimate accuracies. Second, we present a proof that optimizing a nontrivial encoding for BMF is NP-hard. To the best of our knowledge, this is the first proof showing this kind of hardness results for the use of MDL in pattern mining or Boolean matrix factorizations. Third, we provide much more extensive experimental evaluation of the proposed encodings on both synthetic and real data, as well as a comparison to two recent proposals, that is, the PaNDa [Lucchese et al. 2010] algorithm and minimum transfer cost principle [Frank et al. 2011].

The remainder of this article is organized as follows. In Section 2, we discuss related work. In Section 3, we give the preliminaries and notation used throughout the article. Section 4 gives a short primer on both Boolean matrix factorization, as well as the Asso algorithm for finding low-error BMF. In Section 5, we give a quick introduction to the Minimum Description Length principle, develop description length functions for BMF, and subsequently discuss the computational complexity of the proposed minimization problem. We empirically evaluate our encodings on real and synthetic data in Section 6, including a comparison to competing methods. In Section 7, we discuss our methods, and we conclude the article in Section 8.

2. RELATED WORK

In this section, we discuss related work. We start by discussing matrix factorizations in general, and Boolean matrix factorizations in particular before discussing model order selection for matrix factorization, as well as earlier applications of MDL in data mining. We end this section with a discussion about pattern mining–based summarization.

2.1. Matrix Factorization

Matrix factorization methods such as SVD [Golub and Van Loan 1996] or NMF [Paatero and Tapper 1994] are ubiquitous in data mining and machine learning. Two of the most popular uses for matrix factorizations are separating structure and noise, and learning missing values of matrices. For a comprehensive study of the former, see Skillicorn [2007] and references therein.

Whereas standard matrix factorization methods expect real-valued data and return real-valued factor matrices, various other methods either require discrete data, return discrete factors, or both. Logistic PCA [Schein et al. 2003] is an example of a method of the first kind: the data is binary, but the factor matrices are continuous, their product defining a multivariate probability distribution. Semidiscrete decomposition [O'leary and Peleg 1983] can be seen as an example of the second type, expecting continuous input data, but restricting the factor matrices so that two of them assume values from $\{-1, 0, 1\}$ while the third is a real-valued diagonal scaling matrix. Binary matrix factorization [Zhang et al. 2010] is an example of a factorization of the third type: both the input and factor matrices are binary.

2.2. Boolean Matrix Factorization

The difference between binary matrix factorization and BMF is that the former operates under the normal algebra, whereas the latter uses the Boolean algebra. BMFs have been studied extensively in combinatorics (see, e.g., Monson et al. [1995] and references therein). The use of Boolean factorizations in data mining was proposed by

Miettinen et al. [2008], although they had earlier been used to analyse biological [Nau et al. 1978] and psychometrical data [De Boeck and Rosenberg 1988]. There exist many data mining problems and techniques related to BMF. We give an overview in the following text, but refer to Miettinen [2009] for a more extensive discussion on methods related to BMF. Outside data mining, Boolean factorizations have found application in, for example, finding roles for access control [Vaidya et al. 2007; Streich et al. 2009].

The ASSO algorithm to solve BMF was proposed by Miettinen et al. [2008]. Later, Lu et al. [2008] proposed a heuristic based on a mixed integer programming formulation. Independently, Belohlavek and Vychodil [2010] gave an algorithm for computing the Boolean rank of a matrix based on solving the Set Cover problem. In the worst case, this algorithm can take exponential time, but recently it was shown that with certain sparsity constraints, the algorithm runs in polynomial time and provides a logarithmic approximation guarantee [Miettinen 2010].

2.3. Model Order Selection

Matrix factorizations have a long history in various fields of science. SVD and its close relative PCA have been of particular importance. Hence, it is no surprise that many methods for model order selection for these two decompositions have been proposed. One of the earliest suggestions was the Guttman–Kaiser criterion, dating back to the 1950s (see Yeomans and Golder [1982]). In that criterion, one selects those principal vectors that have corresponding principal value greater than 1. It is perhaps not surprising that this simple criterion has shown to perform poorly [Yeomans and Golder 1982]. Another often-used method is Cattell's scree test [Cattell 1966], where one selects the point where the ratio between two consecutive singular values (in descending order) is high. Usually, this is done by visual analysis, but automated methods have also been proposed (e.g., Zhu and Ghodsi [2006]).

Since these two classical methods, researchers have proposed many alternative approaches. For example, in a probabilistic framework one can use Bayesian model selection (e.g., Minka [2001] and Schmidt et al. [2009]). For BMF, however, it would be very hard, if not impossible, to construct a good likelihood function, as it is unclear which probability distribution to use. Yet another approach is to use cross-validation. While this is perhaps mostly used when learning missing values of the matrix, it can also be applied to the noise removal. The assumption is that when the model order is too high, the factors start to specialize to noise, and hence, the cross-validation error increases. Normally, a holdout set would contain either rows or columns, but not both, and the test error is computed against using optimal (or as good as possible) combination of row factors for each row in the test set. This, however, yields to severe overfitting problems, as there is no penalty associated with having more factors.

To overcome this, Owen and Perry [2009] proposed a method to leave out a submatrix. The method is based on the assumption that the remaining matrix has the same rank as the original matrix, as this is needed to fit the factors to the test data. The method of Owen and Perry [2009] unfortunately does not work with BMF, as it requires operations not available in Boolean matrix algebra.

Recently Frank et al. [2011] proposed a method to apply cross-validation to BMF (among others) they call *Minimum Transfer Cost Principle*. In their approach, the holdout set consist of rows (or columns) of the data matrix withhold from the algorithm. To compute the test error, they map each row in the holdout set into the training data row that is closest to it and the test error for the row is computed using exactly the same row factors as were used with the mapped data row.

The concept of intrinsic dimensionality of the data is related to the model order. While often the intrinsic dimensionality refers to the number of variables needed to explain the data completely (e.g., the rank of a matrix), also noise-invariant approaches have been studied [Pestov 2008]. Tatti et al. [2006] defined intrinsic dimensionality to Boolean data based on fractal dimensions.

2.4. MDL in Data Mining

As discussed by Faloutsos and Megalooikonomou [2007], Kolomogorov Complexity, or its practical implementation, the Minimum Description Length principle [Rissanen 1978; Grünwald 2007], are powerful, well-founded, and natural approaches to data mining, as they allow us to clearly identify the most succinct and least redundant model for a dataset. As such, MDL has been successfully employed for a wide range of data mining tasks, including, for example, discretization [Fayyad and Irani 1993], imputation [Vreeken and Siebes 2008], classification [Quinlan and Rivest 1989], transfer learning [Shao et al. 2013], and clustering [Cilibrasi and Vitányi 2005; Van Leeuwen et al. 2009].

2.5. Pattern-Based Summarization

Basically, a BMF returns a group of patterns (the left-hand matrix), and the occurrences per pattern (the right-hand matrix). As such, BMF essentially describes the data with a *set of patterns*. Therefore, pattern set mining techniques are related.

2.5.1. Summarizing Pattern Collections. There are two main approaches in pattern set mining. The first aims at summarizing a collection of patterns. Well-known examples include closed frequent itemsets [Pasquier et al. 1999], maximal frequent itemsets [Bayardo 1998], nonderivable itemsets [Calders and Goethals 2007], contour patterns [Jin et al. 2009], and margin-closed itemsets [Moerchen et al. 2011].

In BMF, the goal is not to summarize a collection of patterns (or, factors), but instead to find a set of patterns that together summarize the data well—which is the second main school in pattern set mining.

2.5.2. Summarizing Data. Wang and Karypis [2004] propose to find summary sets, sets of itemsets such that every transaction is (at least partially) covered by the largest itemset that is frequent. Chandola and Kumar [2007] extended this approach by allowing wildcards, while taking the information loss of the cover with respect to the data into account. Different from these approaches, in BMF, we do not require every row to be modeled by at least one factor.

KRIMP [Siebes et al. 2006; Vreeken et al. 2011] employs MDL for identifying that group of frequent itemsets that describes the data best. The related LESS [Heikinheimo et al. 2009] and PACK [Tatti and Vreeken 2008] algorithms follow similar approaches to describe data in terms of, respectively, low-entropy sets and decision trees. A major difference with BMF is that these methods only cover rows using subsets of that row, and approximate matches are not allowed. Further, KRIMP and LESS do not allow overlap between patterns covering the same row. All typically return more, and in particular more specific patterns than BMF.

Wang and Parthasarathy [2006] and Mampaey et al. [2012] propose algorithms for summarizing data with sets of itemsets and frequencies. To this end, they construct a probabilistic model for the rows of the data by the maximum entropy principle, and iteratively mine itemsets that maximize the likelihood of the data under the model, while controlling complexity through BIC or MDL scores. In contrast to BMF, these methods only consider how often itemset occur in the data, not where; that is, they ignore the right-hand factor matrix.

2.5.3. Tiling Databases. More closely related to BMF is Tiling [Geerts et al. 2004]. Essentially Tiling is the well-known greedy set-cover algorithm, iteratively finding that itemset that covers the most uncovered 1s in the data. Unlike BMF, however, it covers the data exactly, and by focusing on area, for sparse data, it tends to return full rows or columns.

Tiling was independently discovered by Xiang et al. [2008], who later extended the algorithm to also mine noisy tiles—that is allowing 0s in the area of the data a tile covers [Xiang et al. 2010]. To this end, the HYPER algorithm starts by mining a collection of (closed) frequent itemsets, and subsequently iteratively selects that pair of tiles for which the merger covers the fewest 0s of the data. The process stops when either the user-specified number of k tiles are found, or when the ratio of 0s surpasses a user-specified threshold.

Earlier, Lakshmanan et al. [2002] defined a method for summarizing a dataset by means of hyper rectangles, where the method is allowed to ignore w 0s in the data. In our approach, by the MDL principle, we define a lossless encoding by encoding the false positive and false negatives in an error matrix. This allows our method to automatically decide how many cells to "ignore," as well as to detect differences in the distributions of false positive and false negatives.

Kontonasios and De Bie [2010] iteratively discover the most interesting "noisy tile," where they define interestingness through a local MDL score. Unlike our situation, it does not return a model for the data, but rather orders a given collection of itemsets.

Perhaps closest to our work is the PANDA algorithm proposed by Lucchese et al. [2010]. PANDA performs B, but instead of minimizing only the error, it tries to minimize the sum of errors and number of 1s in the factors. There are few important differences to present work. First, PANDA uses a different approach toward factorization. Their approach is to start by finding a good core tile, (i.e., a submatrix full of 1s). This core is then extended to also include rows and columns that contain 0s. After the extension, next core is found, excluding all those 1s already covered by the previously found factors. Another difference between PANDA and the present work is that PANDA uses only very coarse measure of the complexity of the model, namely, the number of 1s in the factors and in the error—that is, a single error is as expensive as a 1 in the factor matrices. Our approach differs from this in two major ways: we count the actual number of bits needed to encode the factors, and the error allowing us to use sophisticated encodings to actually minimize the encoding length.[1]

3. NOTATION

Before we introduce the theory behind our approach, we introduce the notation we will use throughout this article.

We identify datasets as Boolean matrices. Matrices are denoted by upper-case bold letters ($\mathbf{A}$). Vectors are lower-case bold letters ($\mathbf{a}$). If $\mathbf{A}$ is an n-by-m Boolean matrix, $|\mathbf{A}|$ denotes the number of 1s in it (i.e., $|\mathbf{A}| = \sum_{i,j} a_{ij}$). We extend the same notation to Boolean vectors. The scalar product of two vectors $\mathbf{x}$ and $\mathbf{y}$ is denoted as $\langle \mathbf{x}, \mathbf{y} \rangle$.

If $\mathbf{X}$ and $\mathbf{Y}$ are two n-by-m Boolean matrices, we have the following element-wise matrix operations. The *Boolean sum* $\mathbf{X} \vee \mathbf{Y}$ is the normal matrix sum with addition defined as $1 + 1 = 1$. The *Boolean subtraction* $\mathbf{X} \ominus \mathbf{Y}$ is the normal element-wise subtraction with $0 - 1 = 0$. Notice that this does not define an inverse of Boolean sum, as $1 + 1 - 1 = 0$. The *Boolean element-wise product* $\mathbf{X} \wedge \mathbf{Y}$ is defined as normal element-wise matrix product. The *exclusive or* $\mathbf{X} \oplus \mathbf{Y}$ is the normal matrix sum with addition defined as $1 + 1 = 0$ (i.e., addition is done over the field $\mathbb{Z}_2$).

Let $\mathbf{X}$ be n-by-k and $\mathbf{Y}$ be k-by-m Boolean matrices (i.e., $\mathbf{X}$ and $\mathbf{Y}$ take values from $\{0, 1\}$). Their *Boolean matrix product*, $\mathbf{X} \circ \mathbf{Y}$, is the Boolean matrix $\mathbf{Z}$ with

[1]After the acceptance of this article, Lucchese et al. [2014] published PANDA$^+$ that can also directly minimize the description length.

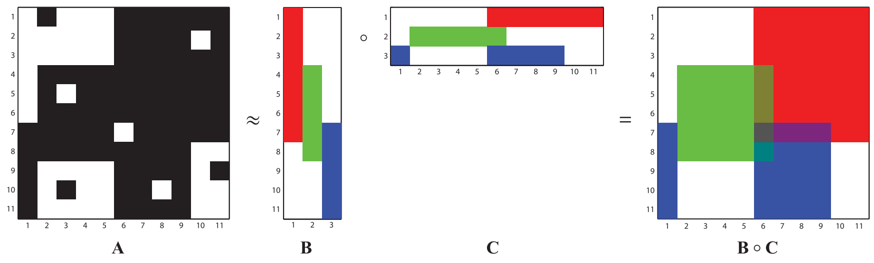

Fig. 1. An approximate BMF of an 11-by-11 matrix **A** (left) into Boolean rank-3 factor matrices **B** and **C** (middle), and the Boolean matrix product **B** ∘ **C** (right). The colors denote the different factors; white matrix elements are 0 and nonwhite elements are 1.

$z_{ij} = \bigvee_{l=1}^{k} x_{il} y_{lj}$, that is, Boolean matrix product is the normal matrix product using the Boolean addition.

The *Boolean rank* of an n-by-m Boolean matrix $\mathbf{A}$, $\text{rank}_B(\mathbf{A})$, is the least integer k such that there exists an n-by-k Boolean matrix $\mathbf{B}$ and a k-by-m Boolean matrix $\mathbf{C}$ for which $\mathbf{A} = \mathbf{B} \circ \mathbf{C}$. Matrices $\mathbf{B}$ and $\mathbf{C}$ are the *factor matrices* of $\mathbf{A}$, and the pair $(\mathbf{B}, \mathbf{C})$ is the (exact) *Boolean factorization* of $\mathbf{A}$. If $\mathbf{A} \neq \mathbf{B} \circ \mathbf{C}$ (but the dimensions match), the factorization is approximate.

Further, all logarithms are of base 2, and we employ the usual convention that $0 \log 0 = 0$.

4. BOOLEAN MATRIX FACTORIZATION

In this section we introduce Boolean matrix factorization (BMF) and give a short description of Asso, one of the existing algorithms for Boolean matrix factorization.

4.1. BMF, A Brief Primer

In Boolean matrix factorization, the goal is to (approximately) represent a Boolean matrix as the Boolean product of two Boolean matrices. The crux is the Boolean product: as the product is not over a field, but over semiring $(0, 1, \vee, \wedge)$, Boolean matrix factorizations have some unique properties. For example, the Boolean rank of a matrix $\mathbf{A}$ can be only a logarithm of the normal matrix rank of $\mathbf{A}$ [Monson et al. 1995]. As a consequence, Boolean factorizations can yield smaller reconstruction error than factorizations of same size done under the normal arithmetic. Unfortunately, unlike normal rank, computing the Boolean rank is NP-hard [Nau et al. 1978], and even approximation is hard [Miettinen et al. 2008] (although recent work shows that logarithmic approximations can be obtained by assuming sparsity [Miettinen 2010]).

But even assuming we could compute the Boolean rank efficiently, this is rarely what we actually want. Similarly to normal rank, one would assume that most of the real-world data matrices have full or almost full Boolean rank, due to noise; instead, we often want to have a low-rank approximation of a matrix. Such approximation is usually interpreted to contain the latent structure of the data, while the error it causes is regarded as the noise. When the target rank is given, we have the Boolean matrix factorization problem:

PROBLEM 1 (BMF). *Given n-by-m Boolean matrix* $\mathbf{A}$ *and integer k, find n-by-k Boolean matrix* $\mathbf{B}$ *and k-by-m Boolean matrix* $\mathbf{C}$ *such that* $\mathbf{B}$ *and* $\mathbf{C}$ *minimize*

$$|\boldsymbol{A} \oplus (\boldsymbol{B} \circ \boldsymbol{C})|. \tag{1}$$

Example 4.1. Figure 1 shows an approximate BMF an 11-by-11 matrix $\mathbf{A}$ (left) into 11-by-3 and 3-by-11 matrices $\mathbf{B}$ and $\mathbf{C}$ (middle), and the Boolean matrix product $\mathbf{B} \circ \mathbf{C}$

(right). Using normal matrix product, $\mathbf{BC}$ would become nonbinary. For example, row 7, column 6 of $\mathbf{BC}$ would be 3, not 1. With this decomposition, the error $|\mathbf{A} \oplus (\mathbf{B} \circ \mathbf{C})| = 7$.

Unsurprisingly, also this optimization problem is NP-hard, and has strong inapproximability results in terms of multiplicative and additive errors (see Miettinen [2009]). But there is also another, more fundamental problem: the formulation of the BMF problem requires us to have *a priori* knowledge on k, the Boolean rank of the decomposition. With this structure/noise interpretation, this means that we have to have *a priori* knowledge of the dimensionality of the latent structure—something we in practice are most likely not to have.

This problem is, by no means, unique to BMF. Indeed, the same issue underlies any matrix factorization method. And also in clustering, for example, we have to deal with the same problem, known as the model order selection problem. The main contribution of this article is to provide a method to (approximately) solve the model order selection problem in the BMF framework.

4.2. The Asso Algorithm

Knowing the latent dimensionality of the data is usually not enough—we also want to know the latent factors (i.e., we want to solve the BMF problem). As the problem is NP-hard, even to approximate well, we will solve it using a heuristic approach. We have opted to use an existing algorithm for BMF, called Asso [Miettinen et al. 2008]. We chose to use Asso as previous studies have shown it performs reasonably well [Miettinen et al. 2008; Streich et al. 2009; Miettinen 2010], and because the algorithm is hierarchical, that is, the rank-$(k-1)$ decomposition gives the first $k-1$ columns of $\mathbf{B}$ (and the first $k-1$ rows of $\mathbf{C}$) of rank-k decomposition. The latter property is particularly useful when doing the model order selection, as we will see later. We emphasize, though, that the proposed model order selection method is not bound to any specific algorithm for BMF.

For the sake of completeness, we provide a quick overview of how Asso works. For more detailed explanation, see Miettinen et al. [2008]. The name of Asso stems from the algorithm using pairwise association accuracies to generate so-called candidate columns. More precisely, Asso generates an n-by-n matrix $\mathbf{X} = (x_{ij})$ with $x_{ij} = \langle \mathbf{a}_i, \mathbf{a}_j \rangle / \langle \mathbf{a}_j, \mathbf{a}_j \rangle$, where $\mathbf{a}_i$ is the jth row of $\mathbf{A}$. That is, x_{ij} is the association accuracy for rule $\mathbf{a}_j \Rightarrow \mathbf{a}_i$. Matrix $\mathbf{X}$ is then rounded to have Boolean values. The rounding is done from a user-specified threshold $\tau \in (0, 1]$.

The columns of $\mathbf{B}$ are selected from the columns of $\mathbf{X}$. The selection of columns of $\mathbf{B}$ happens in a greedy fashion: each unused column of rounded $\mathbf{X}$ is tried, and the selected column is the one that maximizes the gain, defined being the number of newly covered 1s of $\mathbf{A}$ minus the number of newly covered 0s of $\mathbf{A}$. Element a_{ij} is newly covered if $(\mathbf{B} \circ \mathbf{C})_{ij} = 0$ before adding the new column to $\mathbf{B}$. The row of $\mathbf{C}$ corresponding to the column of $\mathbf{B}$ is build using the same technique: if the gain of using the new column of $\mathbf{B}$ to cover a column of $\mathbf{A}$ is positive, then the corresponding element of the new row of $\mathbf{C}$ is set to 1; otherwise, it is 0. The gain is computed by the function cover:

$$\mathsf{cover}(\mathbf{A}, \mathbf{B}, \mathbf{C}) = |\{(i,j) : a_{ij} = 1 \wedge (\mathbf{B} \circ \mathbf{C})_{ij} = 1\}| - |\{(i,j) : a_{ij} = 0 \wedge (\mathbf{B} \circ \mathbf{C})_{ij} = 1\}|. \quad (2)$$

The pseudocode for Asso is given in Algorithm 1.

As Asso never tracks back its decisions, it clearly has the desired hierarchical property. But Asso also requires the user to set an extra parameter: the rounding threshold τ. Selecting this parameter can be daunting, as it is hard to anticipate the difference it makes to the factorization. To solve this problem, we will use our model order selection mechanism to slightly larger question of *model selection* and in addition to selecting the best k, we also select the best τ.

ALGORITHM 1: An algorithm for the BMF using association rules (ASSO)

```
Input: A matrix $\mathbf{A} \in \{0,1\}^{n\times m}$ for data,a positive integer $k$, and a threshold value $\tau \in (0,1]$.
Output: Matrices $\mathbf{B} \in \{0,1\}^{n\times k}$ and $\mathbf{C} \in \{0,1\}^{k\times m}$ such that $\mathbf{B} \circ \mathbf{C}$ approximates $\mathbf{A}$.
 1: function Asso($\mathbf{A}, k, \tau, w$)
 2:     $\mathbf{X} = (x_{ij}), \quad x_{ij} = \mathbf{1}(\langle \mathbf{a}_i, \mathbf{a}_j\rangle / \langle \mathbf{a}_j, \mathbf{a}_j\rangle \geq \tau)$
 3:     $\mathbf{B} \leftarrow [\,], \mathbf{C} \leftarrow [\,]$                ▷ B and C are empty matrices.
 4:     for $l = 1, \ldots, k$ do                ▷ Select the $k$ basis vectors from X.
 5:         $(i, \mathbf{c}) \leftarrow \arg\max\{\text{cover}(\mathbf{A}, [\mathbf{B}\ \mathbf{x}^i], \left[\begin{smallmatrix}\mathbf{C}\\ \mathbf{c}\end{smallmatrix}\right]) : i \in \{1,\ldots,n\}, \mathbf{c} \in \{0,1\}^{1\times m}\}$
 6:         $\mathbf{B} \leftarrow [\mathbf{B}\ \mathbf{x}^i]$
 7:         $\mathbf{C} \leftarrow \left[\begin{smallmatrix}\mathbf{C}\\ \mathbf{c}\end{smallmatrix}\right]$
 8:     end for
 9:     return B and C
10: end function
```

5. MDL FOR BMF

In this section, we give our approach for selecting model orders for BMF by the MDL principle.

5.1. MDL: A Brief Primer

The MDL [Rissanen 1978; Grünwald 2007] principle, like its close cousin Minimum Message Length (MML) [Wallace 2005], is a practical version of Kolmogorov complexity [Li and Vitányi 1993]. All three embrace the slogan *induction by compression*. For MDL, this principle can be roughly described as follows.

Given a set of models[2] $\mathcal{H}$, the best model $H \in \mathcal{H}$ is the one that minimizes

$$L(H) + L(D|H)$$

in which $L(H)$ is the length, in bits, of the description of H, and $L(D \mid H)$ is the length, in bits, of the description of the data when encoded with H.

This is called *two-part MDL*, or *crude MDL*. As opposed to *refined* MDL, where model and data are encoded together [Grünwald 2007]. We use two-part MDL because we are specifically interested in the compressor: the factorization that yields the best compression. Further, although refined MDL has stronger theoretical foundations, it cannot be computed except for some special cases. Note that MDL requires the compression to be *lossless* in order to allow for fair comparison between different $H \in \mathcal{H}$.

To use MDL, we have to define what our models $\mathcal{H}$ are, how a $H \in \mathcal{H}$ describes a database, and how all of this is encoded in bits.

Note that with MDL we are only interested in the length of the description, not in the encoded data itself. That is, we are only concerned with *optimal* code lengths—not with their length of actual materialized codes. Following, in order to keep maximal differentiability between models, we should *not* round up code lengths: this is only required when actual codes have to be constructed. Moreover, in practice, this rounding can also be avoided by using arithmetic coding [Cover and Thomas 2006].

5.2. Encoding BMF

We now proceed to define how we can use MDL to identify the best Boolean factorization $(\mathbf{B}, \mathbf{C})$ for a given dataset $\mathbf{A}$.

Recall that an essential requirement of MDL is that the encoding is lossless. That is, regardless of whether a factorization $(\mathbf{B}, \mathbf{C}) \in \mathcal{H}$ for $\mathbf{A}$ is exact, we need to be able to reconstruct $\mathbf{A}$ without loss. We do this by explicitly encoding the difference, or error,

[2]MDL-theorists talk about *hypothesis* in this context, hence the $\mathcal{H}$.

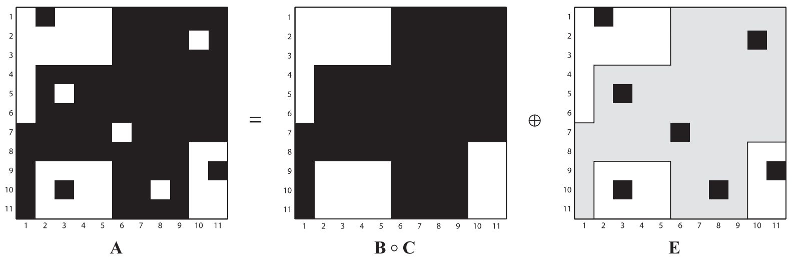

Fig. 2. Example of a Boolean matrix **A**, and a lossless representation thereof by the exclusive-OR between an approximation of **A** by the Boolean product of two factor matrices, $\mathbf{B} \circ \mathbf{C}$, and the error matrix or residual **E**. Note that **E** consists of both additive and subtractive errors, respectively, 1s that exist in **A**, but not in $\mathbf{B} \circ \mathbf{C}$, and vice-versa.

between the original data **A** and its approximation as given by the Boolean product of its factor matrices **B** and **C** (i.e., $\mathbf{B} \circ \mathbf{C}$). That is, we define error matrix **E** for **A**, **B**, and **C**, to be the unique Boolean matrix of dimensions n-by-m such that

$$\mathbf{E} = \mathbf{A} \oplus (\mathbf{B} \circ \mathbf{C}) . \tag{3}$$

Vice versa, when we are given matrices **B**, **C**, and **E**, we can reconstruct **A** without loss.

Example 5.1. Figure 2 shows matrix **E** for the example Boolean matrix factorization we discussed in Example 4.1. From left to right, we have our Boolean matrix **A**, the Boolean product of two factor matrices $\mathbf{B} \circ \mathbf{C}$, and the resulting error matrix **E**. If we transmit **B**, **C**, and **E**, the recipient can reconstruct **A** without loss by taking the exclusive-OR between $\mathbf{B} \circ \mathbf{C}$ and **E**.

The approximation of **A** by our example factorization leads to seven errors. Four of these errors are *subtractive*, ones covered by the approximation but not present in **A**; the black squares in **E** within the grey outline of $\mathbf{B} \circ \mathbf{C}$. The remaining three errors are *additive*, ones in **A** that are not covered by the approximation.

Now, we can define the total compressed size $L(\mathbf{A}, H)$, in bits, for a Boolean dataset **A** and a Boolean matrix factorization $H = (\mathbf{B}, \mathbf{C})$, with $H \in \mathcal{H}$, as

$$L(\mathbf{A}, H) = L(H) + L(\mathbf{E}) , \tag{4}$$

where **E** follows from **A** and H, using Equation (3). Following the MDL principle, the best factorization for **A** is found by minimizing Equation (4). As we will discuss later, this is not as simple as it sounds. But let us first discuss how we encode H and **E**, or most importantly, how many bits this requires.

We start by defining how to compute the number of bits required for a factorization $H = (\mathbf{B}, \mathbf{C})$, of dimensions n-by-k and k-by-m, for **B** and **C,** respectively, as

$$L(H) = L_{\mathbb{N}}(n) + L_{\mathbb{N}}(m) + L(k) + L(\mathbf{B}) + L(\mathbf{C}) . \tag{5}$$

That is, we encode the dimensions n, m, k, and then the content of the two factor matrices. By explicitly encoding the dimensions of the matrices, we can subsequently encode matrices **B** and **C** using an optimal prefix code [Cover and Thomas 2006].

To encode m and n, we use $L_{\mathbb{N}}$, the MDL optimal universal code for integers [Rissanen 1983]. A universal code is a code that can be decoded unambiguously without requiring the decoder to have any background information, but for which the expected length of the code words are within a constant factor of the true optimal code [Grünwald 2007]. With this encoding, $L_{\mathbb{N}}$, the number of bits required to encode an integer $n \geq 1$

is defined as

$$L_{\mathbb{N}}(n) = \log^*(n) + \log(c_0)\,, \tag{6}$$

where $\log^*$ is defined as $\log^*(n) = \log(n) + \log\log(n) + \cdots$, where only the positive terms are included in the sum. To make $L_{\mathbb{N}}$ a valid encoding, c_0 is chosen as $c_0 = \sum_{j\geq 1} 2^{-L_{\mathbb{N}}(j)} \approx 2.865064$ such that the Kraft inequality is satisfied—that is, ensure that this is a valid encoding by having all probabilities sum to 1.

Note that, as desired, $L_{\mathbb{N}}(n) + L_{\mathbb{N}}(m)$ is constant between any $(\mathbf{B}, \mathbf{C}) \in \mathcal{H}$ with $\mathbf{B} \circ \mathbf{C}$ of dimensions n-by-m (which is the case when we regard Boolean matrix factorizations of an n-by-m matrix $\mathbf{A}$.)

As we want the selection for the best factorization to depend strictly on the structure of the factorization and the error it introduces—and do not want to introduce any bias to small k by how we encode its value—we do not encode k using $L_{\mathbb{N}}$ (Equation (6)). Instead, we use a fixed number of bits (i.e., block-encoding), which gives us

$$L(k) = \log(\min(m, n))\,. \tag{7}$$

This gives us a number of bits in which we can encode values for k up to the minimum of m and n. Note that larger values do not make sense in BMF, as for $k = \min(m, n)$, there already is a trivial factorization $(\mathbf{B}, \mathbf{C})$ of $\mathbf{A}$ with $\mathbf{B} = \mathbf{A}$ and $\mathbf{C}$ the identity-matrix, or vice versa.

With the dimensions encoded, we continue by encoding the factor matrices $\mathbf{B}$ and $\mathbf{C}$. To not introduce bias between different factors, these are encoded per factor. That is, we encode $\mathbf{B}$ per column and $\mathbf{C}$ per row. For transmitting the Boolean values of each factor, we use an optimal prefix code, for which Shannon entropy, $-\log P(x)$, gives us the optimal code lengths. In order to use this optimal code, we need to first encode the probability $p_1^{\mathbf{b}_i} = P(1 \mid \mathbf{b}_i) = |\mathbf{b}_i|/n$ of encountering 1 in column i of $\mathbf{B}$. As the maximum number of 1s in a column of $\mathbf{B}$ is n, we need k times $\log(n)$ bits to encode these probabilities. This gives us, for $\mathbf{B}$ of n-by-k,

$$L(\mathbf{B}) = k\log(n) + \sum_{i=1}^{k} \left(|\mathbf{b}_i| l_1^{\mathbf{b}_i} + (n - |\mathbf{b}_i|) l_0^{\mathbf{b}_i}\right) \tag{8}$$

in which $k\log(n)$ is the number of bits to transmit the number of 1s in each column vector $\mathbf{b}_i$, and

$$l_1^{\mathbf{b}_i} = -\log\left(p_1^{\mathbf{b}_i}\right) = -\log\left(\frac{|\mathbf{b}_i|}{n}\right) \quad \text{and} \quad l_0^{\mathbf{b}_i} = -\log\left(p_0^{\mathbf{b}_i}\right) = -\log\left(\frac{n - |\mathbf{b}_i|}{n}\right)$$

are the optimal prefix code lengths for 1 and 0, respectively, for vector $\mathbf{b}_i$ corresponding to the ith column of $\mathbf{B}$.

Analogously, we encode the k-by-m matrix $\mathbf{C}$ per row and have

$$L(\mathbf{C}) = k\log(m) + \sum_{j=1}^{k} \left(|\mathbf{c}_j| l_1^{\mathbf{c}_j} + (m - |\mathbf{c}_j|) l_0^{\mathbf{c}_j}\right) \tag{9}$$

with

$$l_1^{\mathbf{c}_j} = -\log\left(p_1^{\mathbf{c}_j}\right) = -\log\left(\frac{|\mathbf{c}_j|}{m}\right) \quad \text{and} \quad l_0^{\mathbf{c}_j} = -\log\left(p_0^{\mathbf{c}_j}\right) = -\log\left(\frac{m - |\mathbf{c}_j|}{m}\right),$$

where $\mathbf{c}_j$ corresponds the jth row of $\mathbf{C}$.

With the aforementioned definitions, we now have all elements to calculate $L(H)$. By H, the receiver knows $\mathbf{B}$ and $\mathbf{C}$, and only needs $\mathbf{E}$ to be able to lossless reconstruct

A. We will now discuss four increasingly involved alternatives for encoding **E**; we will explore their quality experimentally in Section 6.

Example 5.2. As an example, let us compute $L(H)$ for the factor matrices **B** and **C** of Examples 4.1 and 5.1. Encoding the values of n and m, here 11 for both, takes $L_{\mathbb{N}}(n) = L_{\mathbb{N}}(m) = 6.09 + 1.519 = 7.609$ bits each. To encode k, we use $L(k) = \log(\min(m, n)) = \log(11) = 3.459$ bits.

Next, we consider $L(\mathbf{B})$. To encode the number of 1s in each of the factors, we require $k \log(n) = 3 \log 11 = 10.378$ bits. The first column of **B** consists of seven 1s and four 0s. Hence, as probabilities, we have $p_1^{\mathbf{b}_1} = 7/11$ and $p_0^{\mathbf{b}_1} = 4/11$, respectively. The corresponding code lengths to encode a value then are $l_1^{\mathbf{b}_1} = -\log(p_1^{\mathbf{b}_1}) = 0.652$ bits and $l_0^{\mathbf{b}_1} = 1.459$ bits, respectively. Encoding the values of the first column of **B** hence takes $7 \times 0.652 + 4 \times 1.459 = 10.40$ bits. Skipping the details for the second and third column, we have for **B** a description length of $L(\mathbf{B}) = 42.65$ bits. For $L(\mathbf{C})$ we arrive at 43.18 bits.

In total, for our example, the encoded cost of the model is $L(H) = 2 \times 7.609 + 3.459 + 42.65 + 43.18 = 104.51$ bits.

5.2.1. Encoding **E***: Naïve Factors.* As we are scoring Boolean matrix factorizations, it would be natural to also encode the entries of **E** as a factorization, for example, such that $\mathbf{E} = \mathbf{F} \circ \mathbf{G}$. Of course, we have to keep in mind that **B** and **C** encode the structure in **A**, whereas **E** encodes the noise, and noise by definition is unstructured. Hence, we should not encode full rows or columns of **E** into one factor—as then we are assuming structure. Instead, we have to encode each 1 in **E** in a separate factor, that is, separate columns/rows in **F** and **G**. The amount of bits this requires is given by

$$L_{\mathrm{NF}}(\mathbf{E}) = \log(mn) + |\mathbf{E}|\Bigg(-(n-1)\log\left(\frac{n-1}{n}\right) - \log\left(\frac{1}{n}\right) - (m-1)\log\left(\frac{m-1}{m}\right) - \log\left(\frac{1}{m}\right)\Bigg) \tag{10}$$

in which we essentially use the same encoding for the factors as in Equations (8) and (9) (but with the extra knowledge that every row (column, respectively) in the factor matrices contains exactly one 1). We refer to this encoding as the Naïve Factors encoding for **E**, or NF for short.

Example 5.3. Continuing from Example 5.2, we here encode **E** using NF. There are seven errors to be encoded. Each error requires us to describe a factor in **F** and one in **G**, each of which we know contains exactly one 1. Using the aforementioned encoding, encoding one such factor of length 11 in which only one 1 occurs, requires $(10 \times 0.137) + (1 \times 3.459) = 4.834$ bits. For $L_{\mathrm{NF}}(\mathbf{E})$, we then have $\log(11 \times 11) + 7(2 \times 4.834) = 6.919 + 67.676 = 74.595$ bits. Putting this together with the $L(H)$ as we obtained in Example 5.2, for the NF error encoding, we have a total encoded size, $L(\mathbf{A}, H)$ of 179.11 bits.

Quick analysis of this encoding tells us it is monotonically increasing for larger error, which is good, but also that we are spending too many bits, as we essentially encode full rows and columns, whereas we only want to transmit the locations of the 1s in **E**.

5.2.2. Encoding **E***: Naïve Indices.* This observation suggests that we should simply transmit the coordinates of the 1s in **E**. Clearly, this takes $\log m + \log n$ bits per entry. Then,

$$L_{\mathrm{NI}}(\mathbf{E}) = \log(mn) + |\mathbf{E}|\,(\log m + \log n), \tag{11}$$

gives us the total cost for transmitting $\mathbf{E}$. We refer to this encoding as NI, for Naïve Indices.

Example 5.4. Continuing from Example 5.3, we here encode $\mathbf{E}$ using NI. Encoding one error, one 1 in $\mathbf{E}$, takes $\log 11 + \log 11 = 6.919$ bits. For $L_{\text{NI}}(\mathbf{E})$, we then have $\log(11 \times 11) + 7(6.919) = 8(6.919) = 55.35$. Putting this together with the $L(H)$ as we obtained in Example 5.2, for Naïve Indices we have a total encoded size, $L(\mathbf{A}, H)$ of 159.86 bits—which is much cheaper than we obtained using Naïve Factors in Example 5.3.

NI saves bits compared to NF and hence by MDL it is a better encoding. Further, it is monotonically increasing with the amount of 1s in $\mathbf{E}$.

5.2.3. Encoding $\mathbf{E}$: Naïve Exclusive-Or. Although perhaps counterintuitive, we can encode $\mathbf{E}$ more efficiently, more succinctly, if we transmit the *whole* matrix instead of just the 1s. That is, we can save bits by transmitting, in a fixed order, and using an optimal prefix code, not only the 1s but also the 0s.

We do this by first transmitting the number of 1s in $\mathbf{E}$, that is, $|\mathbf{E}|$, in $\log mn$ bits, which allows the receiver to calculate the probability $p_1^{\mathbf{E}} = |\mathbf{E}|/mn$, and hence, the optimal prefix codes for 1 and 0. Then, using these codes, and in a fixed order, we transmit the value of every cell of $\mathbf{E}$. The total number of bits required by this approach, which we refer to as NX for Naïve XOR, is given by

$$L_{\text{NX}}(\mathbf{E}) = \log(mn) + |\mathbf{E}|l_1 + (mn - |\mathbf{E}|)l_0, \tag{12}$$

where the lengths of the codes for 1 and 0, respectively, are

$$l_1 = -\log p_1^{\mathbf{E}} \quad \text{and} \quad l_0 = -\log\left(1 - p_1^{\mathbf{E}}\right).$$

By this approach, we consider every cell in $\mathbf{E}$ independently, yet, importantly, with regard to $p_1^{\mathbf{E}}$. This means that L_n is not strictly monotonic with regard to the number of 1s in $\mathbf{E}$, as once $p_1^{\mathbf{E}} > (1 - p_0^{\mathbf{E}})$, adding 1s to $\mathbf{E}$ decreases its cost. In practice, however, this is not a problem, as it only occurs if $\mathbf{A}$ is both extremely large and extremely dense, yet contains so little structure that encoding it in factors costs more bits than one gains. Besides a pathological case, this situation could be avoided by spending 1 extra bit to indicate whether we are encoding $\mathbf{E}$ or its complement—a cost that is dwarfed by the total number of bits to describe $\mathbf{E}$. We will refer to this encoding as NX.

Example 5.5. Continuing from Example 5.4, we here instead encode $\mathbf{E}$ using NX. The code length for a 1 in $\mathbf{E}$ is $-\log(7/121) = 4.11$ bits, while encoding a 0 only takes 0.09 bits. For $L_{\text{NX}}(\mathbf{E})$, we then have $\log(11 \times 11) + 7 \times 4.11 + 114 \times 0.09 = 45.50$. Putting this together with the $L(H)$ as we obtained in Example 5.2, for Naïve XOR, we have a total encoded size, $L(\mathbf{A}, H)$ of 150.01 bits—which in turn is cheaper than with NI.

In general, the gain of NX over NF and NI is substantial.

5.2.4. Encoding $\mathbf{E}$: Typed Exclusive-Or. Our final refinement in what to encode is to differentiate between noise in the part of $\mathbf{E}$ that falls within the modeled part of $\mathbf{A}$, that is, the 1s we have modeled but do not occur in $\mathbf{A}$, $\mathbf{E}^- = \mathbf{E} \wedge (\mathbf{B} \circ \mathbf{C})$, and those 1s that are part of $\mathbf{A}$ but not included in the model (i.e., $\mathbf{E}^+ = \mathbf{E} \ominus (\mathbf{B} \circ \mathbf{C})$). Trivially, this gives us $\mathbf{E} = \mathbf{E}^+ \vee \mathbf{E}^-$. We refer to this approach as TX.

We encode each of these two parts analogously to NX but can transmit the number of 1s in the additive and subtractive parts in, respectively, $\log(mn - |\mathbf{B} \circ \mathbf{C}|)$ and $\log |\mathbf{B} \circ \mathbf{C}|$ bits.

We define the probability of a 1 in $\mathbf{E}^+$ as $p_1^+ = |\mathbf{E}^+|/(mn - |\mathbf{B} \circ \mathbf{C}|)$, and similarly for $\mathbf{E}^-$, $p_1^- = |\mathbf{E}^-|/|\mathbf{B} \circ \mathbf{C}|$. When we combine this, we can calculate the number of bits required to encode **E** by the Typed XOR encoding as

$$L_{\text{TX}}(\mathbf{E}) = L_{\text{TX}}(\mathbf{E}^+) + L_{\text{TX}}(\mathbf{E}^-), \tag{13}$$

where

$$\begin{aligned} L_{\text{TX}}(\mathbf{E}^+) = {} & \log(mn - |\mathbf{B} \circ \mathbf{C}|) + |\mathbf{E}^+| l_1^+ \\ & + (mn - |\mathbf{B} \circ \mathbf{C}| - |\mathbf{E}^+|) l_0^+ \end{aligned}$$

and

$$L_{\text{TX}}(\mathbf{E}^-) = \log(|\mathbf{B} \circ \mathbf{C}|) + |\mathbf{E}^-| l_1^- + (|\mathbf{B} \circ \mathbf{C}| - |\mathbf{E}^-|) l_0^-,$$

respectively, give the number of bits to encode the additive and subtractive parts of **E**. We calculate l_1^+, l_0^+, l_1^-, l_1^- analogous to how we calculate l for NX.

Example 5.6. Continuing from Example 5.5, we here encode **E** using TX. In Figure 2, the area of **E** corresponding to $\mathbf{E}^+$ is depicted in grey, while the area of $\mathbf{E}^-$ has a white background.

As the calculation of the encoded lengths for, respectively, $\mathbf{E}^+$ and $\mathbf{E}^-$ follow that of NX, we skip the details. In total, for encoding **E** by TX, we require $L_{\text{TX}}(\mathbf{E}) = 49.89$ bits. Note that this is more than NX; this is because the distribution of errors within $\mathbf{E}^+$ and $\mathbf{E}^-$ here are very similar. As such, TX cannot gain much by encoding these separately, while having the additional cost of encoding the number of errors within each part. As we will see in the experiments, in practice these distributions typically do strongly vary, and TX obtains large gains over NX. For TX, we here have a total encoded size, $L(\mathbf{A}, H)$ of 154.4 bits.

Like NX, in general, this encoding is monotonically increasing for larger error $|\mathbf{E}|$. However, it is more efficient than its naïve cousin when the noise is not uniformly distributed over the modeled and not modeled parts of **A**. That is, when the probability of a 1 being part of a true pattern being recorded as a 0 is not equal to the probability of a true 0 being recorded as a 1. Clearly, in many situations, this will be the case.

5.2.5. Encoding by Data to Model Codes. Although increasingly involved, so far, the techniques we proposed to encode **B**, **C**, and **E** are fairly straightforward: we use explicit codes for the individual values. By using optimal prefix codes, we know the encoded lengths of the entries follow the observed empirical probability distributions. However, it is important to note this style of encoding is only optimal for encoding a *random* entry drawn from that distribution. When encoding a stream of unknown length and of fixed distribution, that is naturally satisfied, and hence these codes are optimal.

However, in our setting, we know more: the total number of entries we will have to decode. Armed with this knowledge, we do not have to use fixed code lengths per entry, as we did earlier. Instead, we know the number of 1s and 0s we will receive, we can derive the codes optimal for the next entry. After receiving that code, we can unambiguously decode it. Moreover, we know we can decrease the number of codes we will receive for that value; by which we obtain a new coding distribution, which is optimal for the new situation. This is called *adaptive* coding [Grünwald 2007].

Instead of calculating new codes at every step, but using the same rationale and obtaining the same total encoded length, we can encode *all* entries in one *code*. This is called *data-to-model (DtM) encoding* [Vereshchagin and Vitanyi 2004]. The basic idea is that, in an abstract way, we can enumerate all possible strings that satisfy the given information. Assuming each of these strings in the enumeration are equally likely, we can identify an individual string by its index in the enumeration. The encoded length of this index is then simply $\log |\mathcal{D}|$, where $\mathcal{D}$ is the number of possible data strings.

In our case, the enumeration consists of all possible binary strings of a particular length and a given number of 1s that occur in it. The number of possible strings can be calculated by the binomial $\binom{v}{w}$, where v is the length of the string, and w the number of 1s (or 0s, as it is symmetric). Following, the encoded length of an index over this enumeration is $\log\binom{v}{w}$ bits.

Next, we formalize a DtM encoding for the factor matrices, and the error matrix. We start with the factor matrices. As before, we assume independence between factors, and hence will encode the factors independently of each other. This gives us

$$L^{dtm}(\mathbf{B}) = k\log(n) + \sum_{i=1}^{k} \log\binom{n}{|\mathbf{b}_i|}$$

and

$$L^{dtm}(\mathbf{C}) = k\log(m) + \sum_{j=1}^{k} \log\binom{m}{|\mathbf{c}_j|}$$

for the **B** and **C** factor matrices, respectively. We will use this encoding for both of the two strategies for encoding **E**.

Example 5.7. Continuing from Example 5.6, we now calculate the description length of our running example model H by the DtM approach (i.e. $L^{dtm}(H)$).

For **B**, we identify how many 1s each factor contains, taking $\log 11$ bits each. For the first factor, there are $\binom{11}{7} = 330$ ways to distribute seven 1s over 11 (n) locations. Identifying one takes $\log 330 = 8.37$ bits. Analog for the other factors, we need $L^{dtm}(\mathbf{B}) = 36.45$ bits to describe **B**. Analog for **C**, we require $L^{dtm}(\mathbf{C}) = 36.93$ bits. Wrapping things up, we have $L(H) = 92.06$ when using the DtM encoding for the factors. This is more than 12 bits cheaper than we needed using the straightforward individual-codes encoding $L(H)$, as used in Example 5.2.

For the error matrix **E**, we construct encodings alike to the Exclusive-Or encodings above, by which we have

$$L_{\text{NXD}}(\mathbf{E}) = \log(mn) + \log\binom{mn}{|\mathbf{E}|}$$

for the DtM equivalent of NX. We will refer to this encoding as NXD for Naïve XOR DtM.

Next, for the Typed Exclusive-Or encoding, when we follow a DtM approach, we have

$$L_{\text{TXD}}(\mathbf{E}) = L_{\text{TXD}}(\mathbf{E}^+) + L_{\text{TXD}}(\mathbf{E}^-),$$

where

$$L_{\text{TXD}}(\mathbf{E}^+) = \log(mn - |\mathbf{B}\circ\mathbf{C}|) + \log\binom{mn - |\mathbf{B}\circ\mathbf{C}|}{|\mathbf{E}^+|}$$

and

$$L_{\text{TXD}}(\mathbf{E}^-) = \log(|\mathbf{B}\circ\mathbf{C}|) + \log\binom{|\mathbf{B}\circ\mathbf{C}|}{|\mathbf{E}^-|}.$$

We refer to this encoding as TXD for Typed XOR DtM.

Example 5.8. Continuing from Example 5.7, we can now calculate for our running example the encoded size of **E** using NXD and TXD. For $L_{\text{NXD}}(\mathbf{E},)$ we arrive at a cost of 42.80 bits, and for TXD, we require $L_{\text{TXD}}(\mathbf{E}) = 45.47$ bits.

For the total encoded sizes, $L(\mathbf{A}, H)$, using the DtM encoding for H, we then require 134.86 and 137.54 bits for NXD and TXD, respectively.

We see that even in this toy example, the DtM encodings are much more efficient than their standard XOR counterparts. Here, the difference is already $\sim$15 bits, over 10%, for encoding exactly the same model. This means the DtM encodings make much better use of the provided information, and hence that by these encodings, we are much better able to measure small differences between different models.

As in Example 5.6, we see that for our running example the typed encoding does not provide a gain, although the difference between NXD and TXD is already smaller. As we will see in the experiments, for real data, TXD does obtain a strong lead over NXD.

By making a choice for one of these six encoding strategies, we can now calculate the total compressed size $L(\mathbf{A}, H)$ for a dataset $\mathbf{A}$ and its factorization. As out of the six strategies, TXD is the most efficient approach for encoding both the model H and error matrix $\mathbf{E}$ given the available information, we expect it to be the best choice for identifying the correct model order. In Section 6, we will empirically evaluate performance, but first we discuss the complexity of finding the factorization that minimizes $L(\mathbf{A}, H)$.

5.3. Computational Complexity

Finding the minimum description length Boolean matrix factorization is a computationally hard task. To start with, the shortest encoding corresponds to the Kolmogorov complexity, which is noncomputable. But even when we try to minimize some given encoding, like one of those described earlier, the problem does not necessarily become easy. In particular, we cannot have a polynomial-time algorithm that minimizes the description length in *any* given encoding.

PROPOSITION 5.9. *Unless* P = NP*, there exists no polynomial-time algorithm that, given an encoding length function L and an n-by-m Boolean matrix $\mathbf{A}$, finds Boolean matrices $\mathbf{B}$ and $\mathbf{E}$ such that $L(\mathbf{A}, (\mathbf{B}, \mathbf{C}))$ is minimized.*

Proving Proposition 5.9 is straightforward: we only need to note that if L is such that encoding even a single bit of error takes more space than *any* possible factorization and any factorization with k factors is always cheaper than any other factorization with $k + 1$ factors, provided they cause the same amount of error, then any decomposition minimizing L must find exact decomposition with least number of factors. As such encoding length functions obviously exist, and minimizing them is equivalent to finding the Boolean rank of the input matrix—an NP-complete problem—Proposition 5.9 must hold.

This, however, does not answer to the question whether the practical encodings are hard, or are all of the hard cases just some specially crafted tautological examples one would not use in any case. We answer this problem, at least partially, by showing that the Naïve Indices encoding (Section 5.2.2) is indeed NP-hard to minimize when one factor matrix is given. For that, we need the following result:

THEOREM 5.10 ([MIETTINEN 2008, 2009]). *Given an n-by-m Boolean matrix $\mathbf{A}$ and an n-by-k Boolean matrix $\mathbf{B}$, it is NP-hard to find a k-by-m Boolean matrix $\mathbf{C}$ such that $|\mathbf{A} \oplus (\mathbf{B} \circ \mathbf{C})|$ is minimized.*

We can now prove the following theorem.

THEOREM 5.11. *Given an n-by-m Boolean matrix $\mathbf{A}$ and an n-by-k Boolean matrix $\mathbf{B}$, it is* NP*-hard to find a k-by-m Boolean matrix $\mathbf{C}$ such that the Naïve Indices encoding length (Section 5.2.2) is minimized.*

PROOF. We reduce the problem of finding $\mathbf{C}$ that minimizes the error to that of finding $\mathbf{C}$ that minimizes the encoding length (under the NI encoding scheme). Assume we are given n-by-m and n-by-k Boolean matrices $\mathbf{A}$ and $\mathbf{B}$ as in Theorem 5.10. For the reduction, we construct new Boolean matrices, $\mathbf{A}_r$ and $\mathbf{B}_r$, as follows. Matrix $\mathbf{A}_r$ is

αn-by-m and matrix $\mathbf{B}_r$ is αn-by-k, where α is a nonnegative integer to be decided later. Matrix $\mathbf{A}_r$ simply contains α copies of $\mathbf{A}$ stacked on top of each other, and $\mathbf{B}_r$ is built similarly with copies of $\mathbf{B}$.

We now claim that any k-by-m Boolean matrix $\mathbf{C}^*$ that minimizes the NI encoding of decomposition $\mathbf{A}_r \approx \mathbf{B}_r \circ \mathbf{C}^*$ also minimizes the error $|\mathbf{A} \oplus (\mathbf{B} \circ \mathbf{C}^*)|$. To that end, notice first that any $\mathbf{C}$ that minimizes the error $|\mathbf{A}_r \oplus (\mathbf{B}_r \circ \mathbf{C})|$ minimizes also the error $|\mathbf{A} \oplus (\mathbf{B} \circ \mathbf{C})|$ (specifically, $|\mathbf{A}_r \oplus (\mathbf{B}_r \circ \mathbf{C})| = \alpha|\mathbf{A} \oplus (\mathbf{B} \circ \mathbf{C})|$). Hence, it suffices to show that $\mathbf{C}^*$ minimizes the error with $\mathbf{A}_r$ and $\mathbf{B}_r$.

Consider the difference between the maximum and minimum encoding lengths of $\mathbf{C}$, $\arg\max_{\mathbf{C}\in\{0,1\}^{k\times m}} L(\mathbf{C}) - \arg\min_{\mathbf{C}\in\{0,1\}^{k\times m}} L(\mathbf{C})$. The minimum is obtained when $\mathbf{C}$ is monochromatic (full of 1s or 0s), while the maximum happens when each row of $\mathbf{C}$ contains exactly $\frac{m}{2}$ 1s. The difference between these two is

$$\begin{aligned}
&\arg\max_{\mathbf{C}\in\{0,1\}^{k\times m}} L(\mathbf{C}) - \arg\min_{\mathbf{C}\in\{0,1\}^{k\times m}} L(\mathbf{C}) \\
&\quad = k\log(m) + \sum_{j=1}^{k}\big(-m/2\log(1/2) - m/2\log(1/2)\big) \\
&\qquad - \left(k\log(m) + \sum_{j=1}^{k}(-0\log 0 - 0\log 1)\right) \\
&\quad = km.
\end{aligned} \tag{14}$$

Now, let $\alpha = km$. As NI spends $\log(\alpha n) + \log(m)$ bits for each error in $\mathbf{A}_r$, and as every error $\mathbf{C}^*$ does in one copy of $\mathbf{A}$ is multiplied α times in $\mathbf{A}_r$, each error $\mathbf{C}^*$ does increases the encoding length by $\alpha(\log(\alpha n) + \log(m)) = km(\log(kmn) + \log(m))$. But this is more than the largest possible increase in the encoding length and so reducing the error always reduces the encoding length. Therefore, for $\mathbf{C}^*$ to minimize the encoding length, it must minimize the error, concluding the proof. □

The aforementioned proof is based on construction where minimizing the error is equivalent on minimizing the MDL cost. While this is not true in general, we consider the minimization of error a good heuristic to minimizing the MDL cost, essentially ignoring the cost of the model when building the factorization. When evaluating the factorization, we naturally compute the full MDL cost.

5.4. Merging MDL and Asso

The most straightforward way to decide the model order, given an encoding method, is to compute the BMF for each value of k, and to report the k at which $L(\mathbf{A}, H)$ was found to be minimal. This naïve strategy, however, may be very slow in practice, as, if we do not set a maximum k ourselves, we would have to restart $\min(n, m)$ times—with n and m in the order of thousands, this would mean a significant computational task. But here the hierarchical nature of Asso comes to good use: instead of having to restart every time, we can start by computing the first factor, compute its description length, add the second factor, compute the new encoded length, and so on. To compute $L(\mathbf{A}, H)$, we can use the fact that at any point we know the cost of encoding the previous $k-1$ factors, and thus only have to compute the cost for the new factor. Further, as Asso calculates per step how much it reduces the error, we can use this information when encoding the error matrix.

As we minimize the error, we cannot guarantee that the description length is a convex function with respect to k. Nevertheless, if we assume the cost to be approximately convex , we can implement an early-stopping criterion: stop the algorithm if compression has not improved during the last c steps.

The benefits of using MDL with Asso are not restricted to only selecting the model order. Recall that Asso requires a user-provided parameter τ to generate the candidate factors. Selecting the value for this parameter can be a daunting task. Here, MDL also helps: by computing the decompositions for different values of τ, we can select the pair (k, τ) that minimizes the total description length. Hence, we can use MDL not only for BMF model order selection in general but also for BMF model selection.

6. EXPERIMENTS

In this section, we experimentally evaluate how well our description length functions identify the correct model orders. We first investigate how well our encodings identify correct model orders with Asso on synthetic data. Second, on the same data, we compare to alternative scoring and factorization methods. Third, we evaluate performance on real datasets.

While, naturally, we will investigate the factors discovered at the identified model orders, note that a qualitative evaluation of factor extraction is not specifically the topic of this article; as opposed to evaluating the performance of Asso in extracting correct factors, our main concern here is identifying the correct model order.

We implemented the Asso algorithm and our scoring models in Matlab/C, and provide the source code for research purposes together with the generator for the synthetic data.[3] For PaNDa, we used the publicly available implementation by Lucchese et al. [2010], and the implementation of Transfer Cost was provided to us by the authors of Frank et al. [2011].

6.1. Comparing Encodings

Before comparing between different methods, we first investigate whether our encodings work at all for identifying correct model orders—and whether we can identify which encoding works best to this end. Therefore, we here restrict ourselves to Asso and use synthetic data.

We perform three different experiments. First, we evaluate the detection of the correct model order under noise. Second, we consider different model orders while keeping the data density equal. Third, we consider data of varying model orders, where we vary the data density. For each experiment, we report the averaged results over five independent runs.

For each of these settings, we generate binary 8,000-by-100 matrices, in which we randomly plant k itemsets, and apply both additive noise (flipping 0s into 1s) as well as subtractive noise (flipping 1s into 0s), with respective probabilities n^+ and n^-.

Varying Model Orders. We first investigate the detection of different model orders. To this end, we generate data in which we respectively plant $k = 2$, 5, 10, 15, or 20 random itemsets of random uniform cardinality $[2, 10]$, and of random uniform frequencies between 10% and 40%. We keep the amounts of additive and subtractive noise fixed to $n^+ = 10\%$, and $n^- = 5\%$. We refer to this setup as **Setting 1**.

We run Asso on each dataset, sweeping for k over 1 to 100, and for τ from 0.1 to 0.9 in increments of 0.025. For each of the $100 \times 33 = 3\,300$ returned factorizations per dataset, we calculate $L(\mathbf{A}, H)$ for each encoding, and we record those values for k associated with the discovered minimal description lengths.

We plot the averaged model order estimates in Figure 3. We see that the Factor and Indices encodings strongly overestimate. The more efficient XOR encodings, on the other hand, consistently give good estimates. Only for 20 planted itemsets do we see a slight underestimation, of which inspection shows is due to overlap: due to chance, and the small number of columns (i.e., 100), many of the planted itemsets overlap and

[3]`http://people.mpi-inf.mpg.de/~pmiettin/mdl4bmf/`.

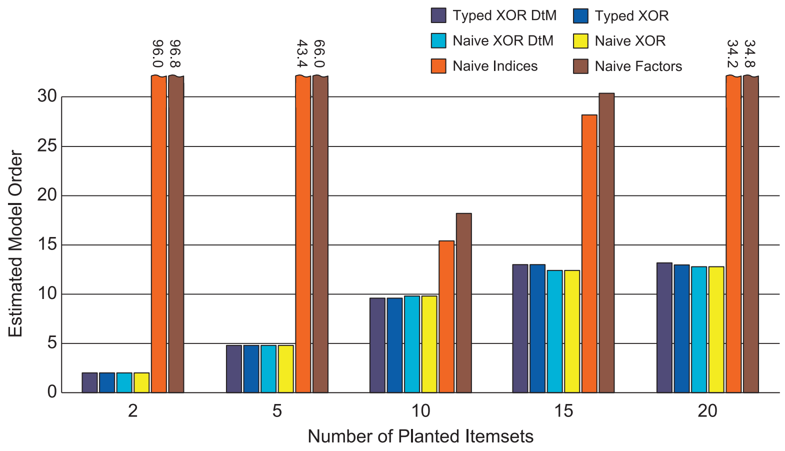

Fig. 3. **Comparing between encodings**. Model order estimates for varying true model orders, averaged over five independently generated datasets. For all datasets, width and frequencies of the planted itemsets are equal on expectation. The bars represent the estimates obtained when using Asso with, from left to right, TXD, NXD, TX, NX, NI, and NF.

co-occur. By it's greedy strategy, Asso groups some of these together. For $k = 10$, we note that NI and NF are either correct *or* strongly overestimate—for all other settings, the standard deviations are very small.

Next, we evaluate a highly similar setting, which we will refer to as **Setting 2**. Instead of generating more and more dense data at higher values of k, we now aim at keeping the density the same, on expectation, to that of the $k = 10$ setting. That is, for $k < 10$ we generate itemsets of higher cardinality, and at $k > 10$ of lower cardinality. Specifically, we generate itemsets with cardinalities randomly drawn from $[10, 40]$, $[5, 15]$, $[2, 5]$, and $[2, 3]$ for $k = 2, 5, 15, 20$, respectively. For the rest, we proceed as noted earlier.

Figure 4 shows the results. For values of k below 10, performance is virtually equal to that of Figure 3. Clearly, Asso has no problem in identifying the correct factors at $k \leq 10$. For high values of k, corresponding to many small itemsets, Asso does not consider the correct model. The efficient XOR encodings all work as intended; as desired, overfitting is punished. The DtM variants slightly outperform the standard XOR encodings, and overall TXD works best by a small margin.

While underestimation is clearly preferred to overestimation—we do not want to model noise—it does raise the question whether this is due to the search or the scoring function; by iteratively minimizing error, Asso may simply not consider any H of correct k that remotely resemble the true model, hence making it impossible to detect the correct model order.

To see what is the case, we manually compare the encoded sizes of the true model to that of the best model found by Asso. The results are clear: for low noise, Asso finds models that compress as well as the true model, sometimes even better (e.g., by not to modeling damaged parts of a structure). Unsurprisingly, the discovered itemsets match the underlying model very well. For high noise levels, on the other hand, we see that Asso returns models that compress much worse, typically requiring 10% more bits than the generating model. The itemsets Asso discovers in these settings, however, do consist of combinations of the true itemsets, specifically those with relatively large overlap in items and rows. So, while the true itemsets are in fact detected, by the iterative minimization of error, Asso does not report them separately. As such, the scoring functions perform rather well; even for the highest levels of noise, the true

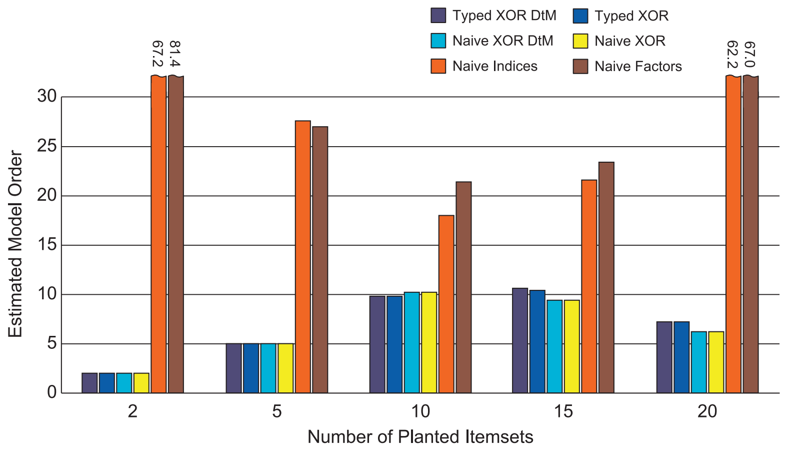

Fig. 4. **Comparing between encodings**. Model order estimates for varying true model orders, averaged over five independently generated datasets. Average density of the datasets kept equal on expectation, by changing the width of the planted itemsets (i.e., higher number of planted itemsets corresponds to smaller itemsets). The bars represent the estimates obtained when using ASSO with, from left to right, TXD, NXD, TX, NX, NI, and NF.

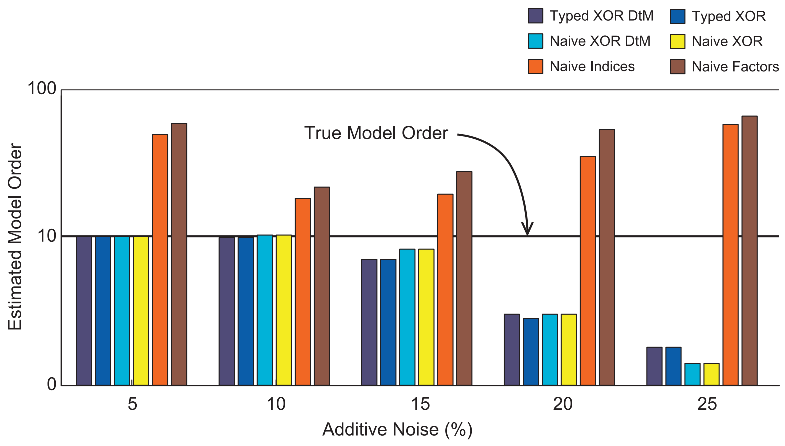

Fig. 5. **Comparing between encodings**. Model order estimates for varying amounts of additive noise (true $k = 10$), averaged over five independently generated datasets. The bars represent the estimates obtained when using ASSO with, from left to right, TXD, NXD, TX, NX, NI, and NF. The y-axis has logarithmic scale. All values averaged over five datasets. Note that beyond 12.5% noise, there are more 1s in the data due to noise than to structure.

model compresses much better, and hence, if it (or any model remotely resembling it) would be considered, the model order would be identified correctly.

Varying Noise Levels. Last, we investigate sensitivity to noise. We generate data varying n^+ from 5% to 25% in steps of 5%, and in each dataset, we plant $k = 10$ randomly generated itemsets, each of cardinality uniformly randomly drawn from $[4, 6]$, with random frequency drawn from $[10\%, 40\%]$. We fix n^- to 5%. As such, for values of n^+ above 12.5%, there are more 1s in the data due to noise than to structure—at 20% and 25% even twice as many. We will refer to this data setting as **Setting 3**.

We proceed as as noted earlier and run ASSO on each dataset, scoring every factorization by each of our encodings. We plot average estimated model orders in Figure 5.

The bar plot shows us that both NF and NI overestimate k strongly. The XOR encodings all perform highly similar and provide very good estimates: exact at 5% and 10%, start to underestimate when the majority of 1s are due to noise, yet still perform very well at 15%. Overall, the Typed XOR variants slightly outperform the Naïve variants, with TXD as the overal best performer.

These three experiments show that the XOR encodings provide highly accurate BMF model order estimates as well as graceful degradation at very large amounts of noise and for densely populated matrices. The subpar performance of the NF and NI encodings show the need of balancing the encoding of the model and error. Overall, TXD and NXD perform best, and we will use these for the remainder of this section.

6.2. Comparing between Methods

Next, we compare to the model order selection performance on our synthetic data of three alternative methods: Cross-validation, Transfer Cost, and PaNDa.

Cross-validation, or CV for short, is perhaps the most well-known and commonly used approach for automatically tuning parameters. As our goal is to validate the discovered factors, instead of trying to predict missing values, we use columns of the original data as holdout data.

As we briefly discussed in Section 1, although CV is an appealing and simple approach, it has one major drawback in the context of BMF: generalizing the column factors to the hold-out data is NP-hard. That is, finding the best possible way to express a given column by the provided factors is called the *Basis Usage Problem*. This problem is known to be NP-hard, even for to be approximated within a super-polylogarithmic factor [Miettinen 2009]. As a practical solution, we here use the same greedy process as employed by Asso (see Section 4.2). We average the overall error for each k and t over the 10 folds and select that combination of parameters for which we observe the smallest average error. Note that we cannot use Owen and Perry's [2009]] bi-cross-validation approach, as their method to generalize learned factors to the test data does not apply to BMF.

Transfer Cost (T-Cost) in essence is a more practical variant of CV, as it provides a way to score how well factors generalize the data. As such, it is a direct alternative to our MDL approach. We apply T-Cost to score the factorizations of Asso. With PaNDa, we also compare to a different algorithm for extracting factors, as well as automatically identifying the model order. As discussed in Section 2, while PaNDa takes cues from MDL, it employs a lossy and rather crude encoding.

We compare the performance of these methods in detecting model orders to Asso in detail using the same synthetic data as mentioned earlier. We run each of the methods, and record the reported model order estimates, for each of the datasets of both settings. As mentioned, all results are averaged over five independently generated sets of databases. We observe that CV always estimates k to full-rank and, therefore, conclude that "vanilla" CV is not a good choice for model order selection in BMF, and we do not further report on it. We do provide a discussion of why CV fails in Section 7.

We start by considering **Settings 1** and **2**, in which we consider different true model orders. Figures 6 and 7 show the corresponding bar plots of the relative error of the model order estimates. Both show the same trend. Overall, PaNDa suffers from severe overestimation, finding hundreds of factors instead of tens. Asso with TXD, NXD or T-Cost, on the other hand, provides good estimates. For T-Cost, the estimates do deteriorate toward overestimation for higher k, whereas our MDL score tend to slightly underestimate. For brevity, we forgo detailed discussion of **Setting 3**, besides remarking that Asso with either NXD, TXD, or T-Cost all provide correct estimates at the noise levels up to 15%, and then deteriorate. As described, PaNDa strongly overestimates.

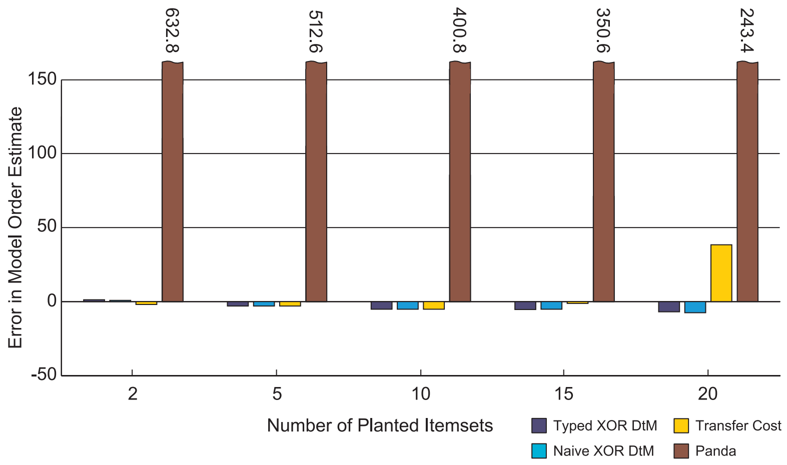

Fig. 6. **Comparing between methods**. Error in model order estimates, averaged over five independently generated datasets, for varying true model orders. For all datasets, width and frequencies of the planted itemsets are equal on expectation. The bars represent the estimates obtained with, from left to right, using ASSO with, respectively, TXD, NXD, and T-Cost; and PaNDa.

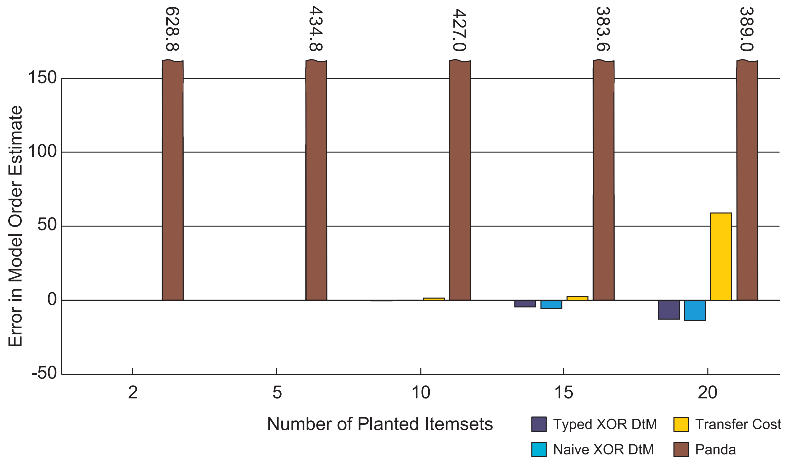

Fig. 7. **Comparing between methods**. Error in model order estimates, averaged over five independently generated datasets, for varying true model orders. Average density of the datasets kept equal on expectation, by changing the width of the planted itemsets (i.e., a higher number of planted itemsets corresponds to smaller itemsets). The bars represent the estimates obtained with, from left to right, using ASSO with TXD, NXD, and T-Cost; and PaNDa.

6.3. Real Data

Next, we consider eight real datasets, most of which are publicly available. In the following, we give a short description per dataset and an overview in Table I.

Abstracts represents the words for all abstracts of the accepted papers at the ICDM conference up to 2007, where the words have been stemmed and stop words removed[4] [De Bie 2011].

[4]Available upon request from the author De Bie [2011].

Table I. Dataset Overview
Shown are, per dataset, the number of rows, n, and the number of columns, m, as well as the density of the matrix, the relative number of 1s. Further, for Asso with T-Cost, PaNDa, and Asso with TXD, the model order estimates.

	Base Statistics			T-Cost	PaNDa	Asso
Dataset	n	m	%1s	k	k	k
Abstracts	859	3,933	1.2	–	168	19
DBLP	6,980	19	13.0	19	15	4
Dialect	1,334	506	16.1	389	56	37
DNA Amp.	4,590	392	1.5	365	39	54
Mammals	2,183	124	20.5	122	50	13
Mushroom	8,192	112	19.3	112	175	59
Newsgroups	400	800	3.5	398	17	17
NSF Abstracts	12,841	4,894	0.9	–	1,835	2,098
Paleo	501	139	5.1	46	96	19

DBLP contains records of in which of the 19 conferences the 6,980 authors had published. The dataset is collected from the DBLP database,[5] and it is preprocessed as by Miettinen [2009].

Dialect contains presence data of dialectical linguistic properties in 506 Finnish municipalities [Embleton and Wheeler 1997, 2000].

DNA Amplification contains information on DNA copy number amplifications. Such copies activate oncogenes and are hallmarks of nearly all advanced tumors [Myllykangas et al. 2006]. Amplified genes represent attractive targets for therapy, diagnostics, and prognostics. This dataset exhibits a banded structure [Garriga et al. 2011].

Mammals consists of presence data[6] consists of presence records of European mammals within geographical areas of 50×50 kilometers [Mitchell-Jones et al. 1999].

Mushroom is a standard benchmark dataset from the UCI repository [Frank and Asuncion 2010] and consists of properties of edible and poisonous mushrooms.

Newsgroups is a subset of the well-known 20Newsgroups dataset,[7] containing, for 400 posts from four newsgroups,[8] the usage of 800 words.

NSF Abstracts contains the occurrence of terms in abstracts of successful NSF grant applications.[9] The preprocessing follows Miettinen [2009]. The resulting data is extremely sparse (0.9% of elements are 1s).

Last, *Paleo* consists of fossil records per location.[10]

We ran each method on these datasets and give the returned model orders in Table I. Transfer cost is computationally rather expensive, in particular for the larger and more sparse datasets, and did not finish within reasonable time for *Abstracts* and *NSF*.

When we investigate the model orders, we see an interesting reversal compared to the synthetic data. Here, PaNDa does not strongly overestimate, but T-Cost does: many of the estimates obtained by T-Cost are full-rank or close to it. While for these datasets there is no ground truth, these scores are beyond what would be inspectable by hand.

The estimates provided by PaNDa, on the other hand, seem more realistic. For *Abstracts*, *Mammals*, and *Mushroom*, however, PaNDa estimates the model order much higher than Asso.

[5] `http://www.informatik.uni-trier.de/~ley/db/`.
[6] Available for research purposes from the Societas Europaea Mammalogica at `http://www.european-mammals.org`.
[7] `http://people.csail.mit.edu/jrennie/20Newsgroups/`.
[8] The authors are grateful to Ata Kabán for preprocessing the data. The exact preprocessing is detailed in Miettinen [2009].
[9] `http://kdd.ics.uci.edu/databases/nsfabs/nsfawards.html`.
[10] NOW public release 030717, available from `http://www.helsinki.fi/science/now/` [Fortelius et al. 2003].

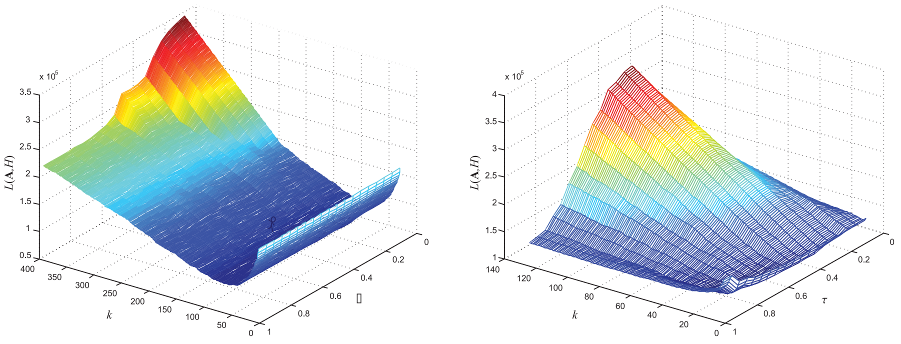

Fig. 8. 3D plots of the number of bits (y-axis), for varying values of k (x-axis) and τ (z-axis) for Asso with Typed XOR DtM encoding. Left: The *DNA* dataset. Minimal description length attained at $k = 54$ and $\tau = 0.35$. Right: The *Mammals* dataset. Minimal description length attained at $k = 13$ and $\tau = 0.65$.

For Asso, with the exception of *NSF*, the factorizations of the identified model orders are all such that a data analyst can inspect by hand. We discuss the results of Asso in closer detail in the following text.

First, however, we investigate the sensitivity to Asso's parameter τ for the model order estimates on real data. In Figure 8, we plot two typical examples of total encoded lengths, $L(\mathbf{A}, H)$, for Asso with TXD and using different values of k and τ. In the left figure, for the *DNA* data, we see the landscape to be a valley, with extreme high values for overly complex and overly simplistic models. The value of k at which Asso minimizes $L(\mathbf{A}, H)$ is found in the distinct valley, at $k = 54$. The shape of the valley tells us that for this relatively structured and dense dataset, the value chosen for parameter τ does not (strongly) influence the detected model order k.

In the right plot of Figure 8, we show the total encoded lengths obtained by Asso on the *Mammals* dataset. Although slightly less well expressed, we observe the same general behavior as for *DNA*; high values for k lead to overly complex models, and hence overly long total encoded lengths. We again observe the distinct peak at high k and low τ, settings at which Asso is allowed to find highly noisy factors. The landscape shows a distinct local valley, or "pit," around $k \in [10, 15]$ and $\tau \in [0.6, 0.85]$, at which the encoded lengths are noticeably lower than elsewhere. The minimal observed description length attained at $k = 13$ and $\tau = 0.65$ is found in this local valley.

Over all other considered datasets, the landscapes are of similar shape, with the exception of the highly noisy synthetic data. This shows that in practice we can sweep over τ in coarse steps, possibly with local optimization. Next, we discuss the near-convexity over values of k.

In Figure 9, we show, for each of the datasets, the total encoded size per k, fixing τ to the value at which the minimum description length was found. The identified model orders (i.e., values for k at which the minimum was reached) are given in each figure. As the plots show, our description length function is close-to-convex for Asso's greedy heuristic search. This means that the early-stop criterion c can validly be employed. It also suggests that binary search might be a valid search strategy for nonhierarchical algorithms.

As mentioned earlir, the purpose of these experiments is to assess the quality of our model order selection approach, not that of BMF or Asso per se. Nevertheless, we need to analyze whether the selected model orders indeed make sense and can best do this by investigating the factors Asso discovers at the identified model orders.

For the *DBLP* dataset, our method proposes $k = 4$. This yields four disjoint sets of conferences as row factors: (1) SIGMOD, VLDB, ICDE; (2) SODA, FOCS, STOC; (3) KDD, PKDD,

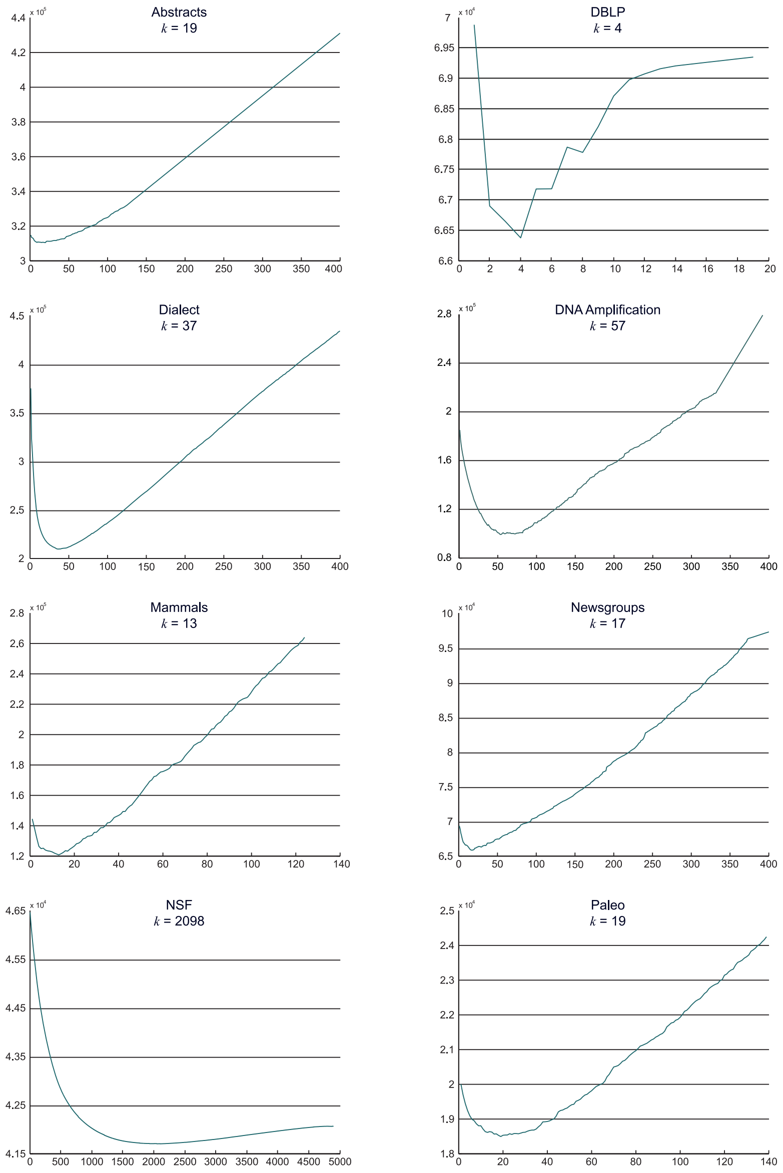

Fig. 9. MDL score per k obtained by Asso for eight datasets. Value for τ fixed per dataset to the value at which the minimal description length is found over all τ and k. For all datasets, we considered values for k up to the data dimensions, with 400 as maximum.

ICDM; and (4) ICML, ECML. These four factors clearly correspond to four different fields of computer science, namely (1) databases, (2) theoretical computer science, (3) data mining, and (4) machine learning.

The suggested number of factors for *Newsgroups*, 17, might seem high given that the data is on only four newsgroups. However, it would be overly simplistic to assume that each newsgroup could be represented well by just one (or even two) topic(s). Instead, there are some general topics (i.e., factors containing words appearing across the data, such as "question") and three to four subtopics per newsgroup.

The estimate for *DNA*, $k = 54$, returns factors that model the known chromosomal regions (or bands [Garriga et al. 2011]) of interest very well, only missing some of the smallest itemsets [Myllykangas et al. 2006].

For the paleontological data, *Paleo*, we identify $k = 19$ as the best model order. Interestingly, Tatti et al. [2006] computed the normalized correlation dimension for this data to be 15. While this normalized correlation dimension has no direct connection to BMF, we consider it interesting that these two methods designed for Boolean data give rather similar results. Even more so, Tatti et al. [2006] report that explaining 90% of the variance with PCA requires 79 factors.

We also checked the error curves for these datasets. In most of the cases, error decreases smoothly as k increases, and hence we cannot find any "elbow" that would suggest the model order.

Further, note that for some datasets, such as *Abstracts* and *Dialect*, there is a range of k that could be considered as the "correct" model order. This is not surprising, as these datasets are very sparse, and hence factorizations of slightly varying complexity and error may exist that model the data equally well. Also, due to the heuristic search of Asso, it cannot be guaranteed that the best factorization out of all valid Boolean matrix factorizations is actually considered.

For *NSF*, our model selection procedure resulted in values of $\tau = 0.8$ and $k = 2,098$. This is one of the few datasets where PaNDa returned a lower estimate (1,835 factors). In both cases, the returned values for k are of course too large for humans to manually interpret the factors. Yet, it seems that the data indeed requires quite large k, as the model cost decreases steeply when k is increased—as is clear from the plot in Figure 9. Furthermore, using PCA, 1,848 factors explain only about 69% of the variance, and in order to explain at least 90% of the variance, PCA needs $3,399$ factors. While these numbers are not directly comparable, they do indicate that the data has a very fine-grained structure (i.e., very little overlap between the rows, many words that (co-) occur only in few documents). This, at least partly, explains the need for high k when factorizing the data by BMF.

Last, though we introduce a model selection criterion, and not a new algorithm, we remark on the recorded runtimes. On the data we consider here, PaNDa finishes within minutes, while for a full sweep over k and t Asso requires between minutes and hours—depending mostly on the density of the data (see Miettinen et al. [2008] for a discussion). Standard Asso returns all counts our scores require, and hence calculating the score is trivial. The evaluation of T-Cost, on the other hand, is computationally costly, and takes up to a few days for a full sweep over k and t for one dataset.

7. DISCUSSION

The experiments show our approach to solve the BMF model order selection problem by MDL works very well. By the MDL principle, and hence by employing the most efficient encoding for the error matrix, TXD, we find highly accurate estimates.

Our approach compares favorably to PaNDa and T-Cost. While the former is fast, for more dense data, it severely overestimates the model order. This was particularly apparent in the synthetic data, but similar behavior was observed for the *Mammals* and *Mushroom* datasets. T-Cost is a powerful mechanism and provides very good model order estimates on the synthetic data. On real data, however, it returns very high model order estimates. Moreover, computing the generalization of a factorization is computationally highly taxing.

We also applied cross-validation. While intuitively appealing, it failed to produce any reasonable results. Why was that? There are (at least) two possible reasons. First, generalizing the learned factors to new columns is a hard problem, as we already mentioned. But it might be that cross-validation fails even if we could generalize in an optimal way. The problem is that when we generalize to new columns (or rows), we do not have to use all factors. The consequence is most obvious in hierarchical decompositions, such as those returned by Asso: adding a new factor reduces the error on training data, but it will never increase the error on test data (if it would, we would not use it to explain the test data). This is not special to Boolean matrix factorizations, as similar behavior has been reported with PCA [dos S. Dias and Krzanowski 2003].

Of our error matrix encoding strategies, the inefficient NF and NI do not work well in practice—except when noise is low and a model closely resembling the true model is offered. The XOR encoding variants, on the other hand, consistently provide highly accurate model order estimates. Neither overestimate model complexity, and naturally provide underestimation in high-noise situations; highly desirable properties for model (order) selection. As it is the most efficient encoding, our overall recommendation is to use TXD to encode $\mathbf{E}$.

Experiments on real data show that meaningful factors are discovered at the selected model orders, as well as that the score is near-convex. This means that, even for the heuristic BMF algorithm we employed in our experiments, we can confidently employ a greedy early-stopping criterion. Further, it suggests binary-search strategy might make a viable strategy for selecting the correct model order if nonhierarchical algorithms are used.

In this article, we proposed to use MDL for identifying the model order in Boolean matrix factorization. To this end, we developed MDL objective functions that can be used to evaluate the complexity of any Boolean matrix factorization—and are hence applicable to any BMF algorithm. As such, contrary to most data mining papers, we did not introduce a novel algorithm.

In order to evaluate the performance of our scores, we applied them to identify the best out of the factorizations discovered by Asso. The empirical evaluation shows that our scores, and TXD in particular, provide very high-accuracy model order estimates. It is important to note that Asso is an existing algorithm for heuristically mining error-minimizing BMF. Though error minimization and entropy minimization are intuitively linked in this setting, it would be informative if this link can be formally established: *under what circumstances and what encoding is the minimum error BMF also the minimum description length BMF?*

Though Asso gives good results, the development of an efficient BMF algorithm directly optimizing our MDL score will make for important future research—given the complexity of the BMF problem in general, and in particular the nonlinear behaviour of the TXD score, smart heuristics will be required. Cues may be taken from recent developments in pattern set mining, where algorithms have been proposed that can mine high-quality results directly from data [Smets and Vreeken 2012; Akoglu et al. 2012; Mampaey et al. 2012].

Finally, we note that our encoding length functions are easily adaptable for different variations of BMF, such as Boolean column subset-selection [Miettinen 2009] and dominated BMF [Miettinen 2010]. Indeed, after the publication of the preliminary version of this article, the proposed encoding length functions have already been adapted to find model orders in joint subspace BMF [Miettinen 2012] and dynamic BMF [Miettinen 2013]. Furthermore, extending our encoding methods to other models for representing binary matrices as a product of binary factor matrices is straightforward. An example of such model is the binary matrix factorization [Zhang et al. 2010] that uses normal arithmetic instead of the Boolean one. The factor matrices can be encoded using exactly the same techniques as presented here, but as the representation is not limited to binary matrices, the error matrix needs to encode not just the location of the errors, but also the correct value. Of the presented encoding schemes, the NI encoding, for example, would require some modifications, but the TXD encoding would work essentially as is. It would be interesting to study how well these encodings would work on these different scenarios.

8. CONCLUSION

We proposed a general solution to the model order selection problem for BMF. By the MDL principle, we formulate the number of bits required to lossless encode the model and the error it introduces, and report the order of the model for which this sum is minimized. We discussed how to construct an appropriate encoding for BMF, starting by introducing simple and intuitive approaches, and progressing to a highly efficient typed DtM-based encoding. We showed that minimizing the MDL score is already NP-hard for the relatively simple NI encoding.

We empirically evaluated its performance in combination with the Asso BMF algorithm. The experiments show that the correct model orders are reliably selected for varying amounts of noise and model orders, and moreover, that the models at which the MDL score is minimized consist of meaningful factors.

Future work includes the development of good heuristics that optimize the MDL score directly, instead of the error, when finding BMFs of a given rank, as well as further analysis of the complexity of problem.

REFERENCES

Leman Akoglu, Hanghang Tong, Jilles Vreeken, and Christos Faloutsos. 2012. CompreX: Compression based Anomaly Detection. In *Proceedings of the 21st ACM Conference on Information and Knowledge Management (CIKM'12)*. ACM, 415–424.

R. Bayardo. 1998. Efficiently mining long patterns from databases. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD'98)*. 85–93.

Radim Belohlavek and Vilém Vychodil. 2010. Discovery of optimal factors in binary data via a novel method of matrix decomposition. *Journal of Computing System Science* 76, 1 (2010), 3–20.

Toon Calders and Bart Goethals. 2007. Non-derivable itemset mining. *Data Mining and Knowledge Discovery* 14, 1 (2007), 171–206.

R. Cattell. 1966. The scree test for the number of factors. *Multivariate Behavioral Research* 1 (1966), 245–276.

Varun Chandola and Vipin Kumar. 2007. Summarization—compressing data into an informative representation. *Knowledge and Information Systems* 12, 3 (2007), 355–378.

Rudi Cilibrasi and Paul Vitányi. 2005. Clustering by Compression. *IEEE Transactions on Information Technology* 51, 4 (2005), 1523–1545.

Thomas M. Cover and Joy A. Thomas. 2006. *Elements of Information Theory*. Wiley-Interscience, New York.

Tijl De Bie. 2011. Maximum entropy models and subjective interestingness: An application to tiles in binary databases. *Data Mining and Knowledge Discovery* 23, 3 (2011), 407–446.

Paul De Boeck and Seymour Rosenberg. 1988. Hierarchical classes: Model and data analysis. *Psychometrika* 53, 3 (Sept. 1988), 361–381.

Carlos T. dos S. Dias and Wojtek J. Krzanowski. 2003. Model selection and cross validation in additive main effect and multiplicative interaction models. *Crop Science* 43 (2003), 865–873.

Sheila M. Embleton and Eric S. Wheeler. 1997. Finnish dialect atlas for quantitative studies. *Journal of Quantitative Linguistics* 4, 1–3 (1997), 99–102.

Sheila M. Embleton and Eric S. Wheeler. 2000. Computerized dialect atlas of Finnish: Dealing with ambiguity. *Journal of Quantitative Linguistics* 7, 3 (2000), 227–231.

Christos Faloutsos and Vasilis Megalooikonomou. 2007. On data mining, compression and Kolmogorov Complexity. *Data Mining and Knowledge Discovery* 15 (2007), 3–20. Issue 1.

Usama Fayyad and K. Irani. 1993. Multi-Interval discretization of continuous-valued attributes for classification learning. In *Proceedings of the 9th International Conference on Uncertainty in Artificial Intelligence (UAI'93)*. 1022–1027.

Mikael Fortelius and others. 2003. Neogene of the Old World Database of Fossil Mammals (NOW). Available at `http://www.helsinki.fi/science/now/`.

A. Frank and A. Asuncion. 2010. UCI Machine Learning Repository. Available at `http://archive.ics.uci.edu/ml`.

Mario Frank, Morteza Haghir Chehreghani, and Joachim M. Buhmann. 2011. The minimum transfer cost principle for model-order selection. In *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD'11)*. 423–438.

Gemma C. Garriga, Esa Junttila, and Heikki Mannila. 2011. Banded structure in binary matrices. *Knowledge and Information Systems* 28, 1 (2011), 197–226.

Floris Geerts, Bart Goethals, and Taneli Mielikäinen. 2004. Tiling databases. In *Proceedings of Discovery Science*. 278–289.

Gene H. Golub and Charles F. Van Loan. 1996. *Matrix Computations*. Johns Hopkins University Press.

Peter Grünwald. 2007. *The Minimum Description Length Principle*. MIT Press.

Hannes Heikinheimo, Jilles Vreeken, Arno Siebes, and Heikki Mannila. 2009. Low-entropy set selection. In *Proceedings of the 9th SIAM International Conference on Data Mining (SDM'09)*. 569–580.

Ruoming Jin, Yang Xiang, and Lin Liu. 2009. Cartesian contour: A concise representation for a collection of frequent sets. In *Proceedings of the 15th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD'09)*. ACM, 417–426.

Kleanthis-Nikolaus Kontonasios and Tijl De Bie. 2010. An information-theoretic approach to finding noisy tiles in binary databases. In *Proceedings of the 10th SIAM International Conference on Data Mining (SDM'10)*. SIAM, 153–164.

Laks V. S. Lakshmanan, Raymond T. Ng, Christine Xing Wang, Xiaodong Zhou, and Theodore Johnson. 2002. The generalized MDL approach for summarization. In *Proceedings of the 28th International Conference on Very Large Data Bases (VLDB'02)*. 766–777.

Matthijs Leeuwenvan Leeuwen, Jilles Vreeken, and Arno Siebes. 2009. Identifying the components. *Data Mining and Knowledge Discovery* 19, 2 (2009), 173–292.

M. Li and P. Vitányi. 1993. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer.

Haibing Lu, Jaideep Vaidya, and Vijayalakshmi Atluri. 2008. Optimal Boolean matrix decomposition: Application to role engineering. In *Proceedings of the 24th International Conference on Data Engineering (ICDE'08)*. 297–306.

Claudio Lucchese, Salvatore Orlando, and Raffaele Perego. 2010. Mining top-k patterns from binary datasets in presence of noise. In *Proceedings of the 10th SIAM International Conference on Data Mining (SDM'10)*. 165–176.

Claudio Lucchese, Salvatore Orlando, and R. Perego. 2014. A unifying framework for mining approximate top-k binary patterns. *IEEE Transactions on Knowledge and Data Engineering* (2014). In press.

Michael Mampaey, Jilles Vreeken, and Nikolaj Tatti. 2012. Summarizing data succinctly with the most informative itemsets. *ACM Transactions on Knowledge Discovery from Data* 6, 4 (2012), 1–44.

Pauli Miettinen. 2008. On the positive-negative partial set cover problem. *Information Processing Letters* 108, 4 (2008), 219–221.

Pauli Miettinen. 2009. *Matrix Decomposition Methods for Data Mining: Computational Complexity and Algorithms*. Ph.D. Dissertation. University of Helsinki.

Pauli Miettinen. 2010. Sparse boolean matrix factorizations. In *Proceedings of the 10th IEEE International Conference on Data Mining (ICDM'10)*. 935–940.

Pauli Miettinen. 2012. On finding joint subspace Boolean matrix factorizations. In *Proceedings of the 12th SIAM International Conference on Data Mining (SDM'12)*. 954–965.

Pauli Miettinen. 2013. Fully dynamic quasi-biclique edge covers via Boolean matrix factorizations. In *Proceedings of the 1st ACM SIGMOD Workshop on Dynamic Network Management and Mining (DyNetMM'13)*.

Pauli Miettinen, Taneli Mielikäinen, Aristides Gionis, Gautam Das, and Heikki Mannila. 2008. The discrete basis problem. *IEEE Transactions on Knowledge and Data Engineering* 20, 10 (2008), 1348–1362.

Pauli Miettinen and Jilles Vreeken. 2011. Model order selection for Boolean matrix factorization. In *Proceedings of the 17th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD'11)*. 51–59.

T. P. Minka. 2001. Automatic choice of dimensionality for PCA. In *Proceedings of the 13th Annual Conference on Neural Information Processing Systems (NIPS'01)*. 598–604.

A. J. Mitchell-Jones, G. Amori, W. Bogdanowicz, B. Krystufek, P. J. H. Reijnders, F. Spitzenberger, M. Stubbe, J. B. M. Thissen, V. Vohralik, and J. Zima. 1999. *The Atlas of European Mammals*. Academic Press.

Fabian Moerchen, Michael Thies, and Alfred Ultsch. 2011. Efficient mining of all margin-closed itemsets with applications in temporal knowledge discovery and classification by compression. *Knowledge and Information Systems* 29, 1 (2011), 55–80.

Sylvia D. Monson, Norman J. Pullman, and Rolf Rees. 1995. A survey of clique and biclique coverings and factorizations of (0, 1)-matrices. *Bulletin of the ICA* 14 (1995), 17–86.

Samuel Myllykangas, J. Himberg, T. Böhling, B. Nagy, Jaakko Hollmén, and S. Knuutila. 2006. DNA copy number amplification profiling of human neoplasms. *Oncogene* 25, 55 (2006), 7324–7332.

Dana S. Nau, George Markowsky, Max A. Woodbury, and D. Bernard Amos. 1978. A mathematical analysis of human leukocyte antigen serology. *Mathematical Biosciences* 40 (1978), 243–270.

Dianne P. O'leary and Shmuel Peleg. 1983. Digital image compression by outer product expansion. *IEEE Transactions on Communications* 31, 3 (1983), 441–444.

Art B. Owen and Patrick O. Perry. 2009. Bi-cross-validation of the SVD and the nonnegative matrix factorization. *Annals of Applied Statistics* 3, 2 (June 2009), 564–594.

Pentti Paatero and Unto Tapper. 1994. Positive matrix factorization: A non-negative factor model with optimal utilization of error estimates of data values. *Environmetrics* 5 (1994), 111–126.

Nicolas Pasquier, Yves Bastide, Rafik Taouil, and Lotfi Lakhal. 1999. Discovering frequent closed itemsets for association rules. In *Proceedings of the 7th International Conference on Database Theory (ICDT'99)*. ACM, 398–416.

V. Pestov. 2008. An axiomatic approach to intrinsic dimension of a dataset. *Neural Networks* 21, 2–3 (2008), 204–213.

J. Ross Quinlan and Ronald L. Rivest. 1989. Inferring decision trees using the minimum description length principle. *Information and Computation* 80, 3 (1989), 227–248.

Jorma Rissanen. 1978. Modeling by shortest data description. *Automatica* 14, 1 (1978), 465–471.

Jorma Rissanen. 1983. Modeling by shortest data description. *The Annals of Statistics* 11, 2 (1983), 416–431.

Andrew I. Schein, Lawrence K. Saul, and Lyle H. Ungar. 2003. A generalized linear model for principal component analysis of binary data. In *Proceedings of the 9th International Workshop on Artificial Intelligence and Statistics*.

M. Schmidt, O. Winther, and L. Hansen. 2009. Bayesian non-negative matrix factorization. In *Proceedings of International Conference on Independent Component Analysis and Signal Separation.* Vol. 5411. 540–547.

Gideon Schwarz. 1978. Estimating the dimension of a model. *The Annals of Statistics* 6, 2 (1978), 461–464.

Hao Shao, Bin Tong, and Einoshin Suzuki. 2013. Extended MDL principle for feature-based inductive transfer learning. *Knowledge and Information Systems* 35, 2 (2013), 365–389. DOI:http://dx.doi.org/10.1007/s10115-012-0505-x

Arno Siebes, Jilles Vreeken, and Matthijs van Leeuwen. 2006. Item sets that compress. In *Proceedings of the 6th SIAM International Conference on Data Mining (SDM'06)*. SIAM, 393–404.

David Skillicorn. 2007. *Understanding Complex Datasets: Data Mining with Matrix Decompositions*. Chapman & Hall/CRC Press.

Koen Smets and Jilles Vreeken. 2012. SLIM: Directly mining descriptive patterns. In *Proceedings of the 12th SIAM International Conference on Data Mining (SDM'12)*. SIAM, 236–247.

Andreas Streich, Mario Frank, David Basin, and Joachim Buhmann. 2009. Multi-assignment clustering for Boolean data. In *Proceedings of the 26th International Conference on Machine Learning (ICML'09)*. 969–976.

Nikolaj Tatti, Taneli Mielikäinen, Aristides Gionis, and Heikki Mannila. 2006. What is the dimension of your binary data? In *Proceedings of the 6th IEEE International Conference on Data Mining (ICDM'06)*. 603–612.

Nikolaj Tatti and Jilles Vreeken. 2008. Finding good itemsets by packing data. In *Proceedings of the 8th IEEE International Conference on Data Mining (ICDM'08).* IEEE, 588–597.

Jaideep Vaidya, Vijayalakshmi Atluri, and Qi Guo. 2007. The role mining problem: Finding a minimal descriptive set of roles. In *Proceedings of the 12th ACM International Symposium on Access Control Models and Technologies (SACMAT'07)*. 175–184.

N. K. Vereshchagin and P. M. B. Vitanyi. 2004. Kolmogorov's structure functions and model selection. *IEEE Transactions on Information Technology* 50, 12 (2004), 3265–3290.

Jilles Vreeken and Arno Siebes. 2008. Filling in the blanks: Krimp minimisation for missing data. In *Proceedings of the 8th IEEE International Conference on Data Mining (ICDM'08).* IEEE, 1067–1072.

Jilles Vreeken, Matthijs van Leeuwen, and Arno Siebes. 2011. KRIMP: Mining itemsets that compress. *Data Mining and Knowledge Discovery* 23, 1 (2011), 169–214.

C. S. Wallace. 2005. *Statistical and Inductive Inference by Minimum Message Length*. Springer-Verlag.

Chao Wang and Srinivasan Parthasarathy. 2006. Summarizing itemset patterns using probabilistic models. In *Proceedings of the 12th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD'06)*. ACM, 730–735.

Jianyong Wang and George Karypis. 2004. SUMMARY: Efficiently summarizing transactions for clustering. In *Proceedings of the 4th IEEE International Conference on Data Mining (ICDM'04)*. IEEE, 241–248.

Y. Xiang, R. Jin, D. Fuhry, and F. Dragan. 2010. Summarizing transactional databases with overlapped hyperrectangles. *Data Mining and Knowledge Discovery* 23, 2 (2010), 215–251.

Yang Xiang, Ruoming Jin, David Fuhry, and Feodor F. Dragan. 2008. Succinct summarization of transactional databases: An overlapped hyperrectangle scheme. In *Proceedings of the 14th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD'08)*. ACM, 758–766.

K. A. Yeomans and P. A. Golder. 1982. The Guttman–Kaiser criterion as a predictor of the number of common factors. *The Statistician* 31, 3 (1982), 221–229.

Zhong-Yuan Zhang, Tao Li, Chris Ding, Xian-Wen Ren, and Xiang-Sun Zhang. 2010. Binary matrix factorization for analyzing gene expression data. *Data Mining and Knowledge Discovery* 20, 1 (2010), 28–52.

M. Zhu and A. Ghodsi. 2006. Automatic dimensionality selection from the scree plot via the use of profile likelihood. *Computational Statistics and Data Analysis* 51, 2 (2006), 918–930.

Received June 2012; revised May 2013; accepted December 2013