MDL4BMF:Minimum Description Length for Boolean Matrix Factorization

Pauli Miettinen
Jilles Vreeken

MPI–I–2012–5-001 June 2012

Authors' Addresses

Pauli Miettinen
Max-Planck-Institut für Informatik
Campus E1.4
D-66123 Saarbrücken
Germany

Jilles Vreeken
Departement Wiskunde en Informatica
Universiteit Antwerpen
Middelheimlaan 1
BE-2020 Antwerp
Belgium

Abstract

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 where fine-grained structure stops, and noise starts, i.e., what is the proper size of the factor matrices.

Boolean matrix factorization (BMF)—where data, factors, and matrix product are Boolean—has received increased attention from the data mining community in recent years. The technique has desirable properties, such as high interpretability and natural sparsity. However, so far no method for selecting the correct model order for BMF has been available. In this paper we propose to use the Minimum Description Length (MDL) principle for this task. Besides solving the problem, this well-founded approach has numerous benefits, e.g., 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.

Keywords

Boolean Matrix Factorization, Model Order Selection, Model Selection, Pattern Sets, Summarization, Minimum Description Length Principle, MDL, Parameter Free

Contents

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 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 transactions. Such 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 Non-negative Matrix Factorization (NMF). SVD and PCA restrict the factor matrices to be orthogonal, while NMF requires the data and the factor matrices to be non-negative.

When the input data is Boolean, (that is, 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 structure and what is noise.

Whereas various methods have been proposed to answer this question for non-Boolean matrix factorization, varying from statistical methods based on likelihood scores (such as the Bayesian Information Criterion, BIC) 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 paper, 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. 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 paper 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 choice, which by MDL, we can make in a principled manner: fewer bits are better. We will start by considering simple, 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 in order to obtain good model order estimates.

A preliminary version of this paper was published as Miettinen and Vreeken (2011). In this extended version, we improve in a number of ways. First of all, we have refined our encodings, and introduce an additional encoding for BMF that employs highly efficient data-to-model codes. Second, we provide much more extensive experimental evaluation of the proposed encodings, as well as comparison to two recent proposals, i.e., PaNDa (Lucchese et al., 2010) and minimum transfer cost principle (Frank et al., 2011). Third, we present a proof that optimizing a non-trivial 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.

The remainder of this paper is organized as follows. First, in Section 2 we discuss related work. Then, in Section 3 we give the preliminaries and notation used throughout the paper. Section 5 gives a short primer on both Boolean matrix factorization, and the Minimum description length principle, before developing description length functions for BMF, and discussing computational complexity of the minimisation 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 conclude the paper with Section 8.

2 Related Work

Matrix factorization methods such as Singular Value Decomposition (SVD) (Golub and Van Loan, 1996) or Non-negative Matrix Factorization (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.

Boolean matrix factorizations (BMF) 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). There exist many data mining problems and techniques related to BMF. We give an overview below, but refer to Miettinen (2009) for a more extensive discussion on methods related to BMF. Outside data mining, Boolean factorizations have found application in, e.g., 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. At 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).

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 Fifties (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 methods. For example, in a probabilistic framework one can use Bayesian model selection (e.g. Minka (2001); 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 applied to the noise removal. 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, hold-out 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 over-fitting problems as there is no penalty associated on having more factors.

To overcome this, Owen and Perry (2009) proposed a method to leave out a sub-matrix. 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 hold-out set consist of rows (or columns) of the data matrix withhold from the algorithm. But to compute the test error, they map each row in the hold-out 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 dimension of the data is related to the model order. While often the intrinsic dimension 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.

As discussed by Faloutsos and Megalooikonomou (2007), Kolomogorov Complexity, or its practical implementation, the Minimum Description Length principle (Grünwald, 2007), are powerful, well-founded, and natural approaches to data mining, as they allows 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), and clustering (Cilibrasi and Vitányi, 2005; Keogh et al., 2004; van Leeuwen et al., 2009).

Basically, a Boolean matrix factorization returns a group of patterns (the left-hand matrix $\mathbf{B}$), and the occurrences per pattern (the right-hand matrix $\mathbf{C}$). As such, BMF essentially describes the data with a *set of patterns*. Therefore, pattern set mining techniques are related.

Krimp (Vreeken et al., 2011; Siebes et al., 2006) pioneered the use of MDL for identifying good pattern sets, and selects that group of frequent itemsets that describes the data best. LESS (Heikinheimo et al., 2009) and Pack (Tatti and Vreeken, 2008) follow a similar approach, but respectively describe data using low-entropy sets and decision trees. A major difference between these methods and BMF is that rows are only covered 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 three typically return many more, and much more specific, patterns than BMF.

Summarization, proposed by Chandola and Kumar (2007), is a compression-based approach that identifies a group of itemsets such that each transaction is summarized by one itemset with as little loss of information as possible. Wang and Karypis (2004) find summary sets, i.e., sets of itemsets such that each transaction is (partially) covered by the largest itemset that is frequent. In BMF, however, we do not require every row to be modeled by at least one factor.

More closely related to BMF is Tiling (Geerts et al., 2004), which essentially employs the well known greedy set-cover algorithm to iteratively cover the data with that itemset that covers the most uncovered 1s in the data. 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, however, is the PaNDa algorithm proposed by Lucchese et al. (2010). PaNDa is doing Boolean matrix factorization, but instead of minimizing just the error, it tries to minimize the error and the complexity of the factors. There are few important differences to present work. First, PaNDa uses different approach on doing the 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. 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.

3 Notation

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

We identify datasets as Boolean matrices. Matrices are denoted by uppercase 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 $z_{ij} = \bigvee_{l=1}^{k} x_{il} y_{lj}$, that is, Boolean matrix product is the normal matrix product using the Boolean addition.

The *Boolean rank* of an n-by-m Boolean matrix $\mathbf{A}$, $\mathrm{rank}_B(\mathbf{A})$, 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 a 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

$$|\mathbf{A} \oplus (\mathbf{B} \circ \mathbf{C})| . \tag{4.1}$$

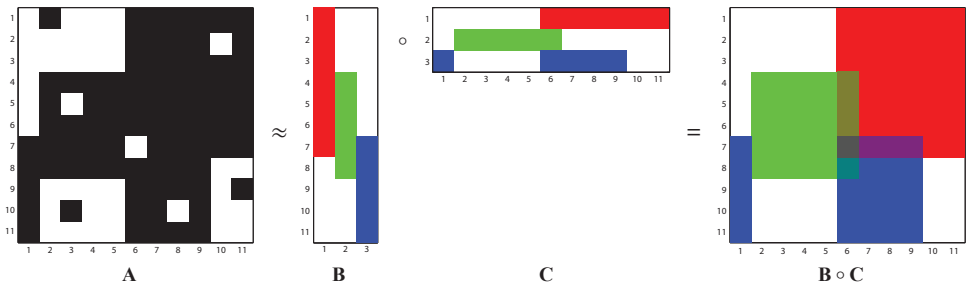

Figure 4.1: An approximate Boolean matrix factorization of an 11-by-11 matrix $\mathbf{A}$ (left) into Boolean rank-3 factor matrices $\mathbf{B}$ and $\mathbf{C}$ (middle), and the Boolean matrix product $\mathbf{B} \circ \mathbf{C}$ (right). The colors denote the different factors; white matrix elements are 0 and non-white elements are 1.

Example 1. Figure 4.1 shows an approximate Boolean matrix factorization of 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 non-binary. 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 the structure/noise interpretation above, 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 paper 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, i.e., 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 explanation 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 not-used 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:

$$\begin{aligned} \mathsf{cover}(\mathbf{A}, \mathbf{B}, \mathbf{C}) = {} & |\{(i,j) : a_{ij} = 1 \wedge (\mathbf{B} \circ \mathbf{C})_{ij} = 1\}| \\ & - |\{(i,j) : a_{ij} = 0 \wedge (\mathbf{B} \circ \mathbf{C})_{ij} = 1\}| \,. \end{aligned} \tag{4.2}$$

The pseudo code 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 $\mathbf{A}$.

function $\texttt{Asso}(\mathbf{A}, k, \tau, w)$
  $\mathbf{X} = (x_{ij}), \quad x_{ij} = \mathbf{1}(\langle \mathbf{a}_i, \mathbf{a}_j \rangle \,/\, \langle \mathbf{a}_j, \mathbf{a}_j \rangle \geq \tau)$
  $\mathbf{B} \leftarrow [\,], \mathbf{C} \leftarrow [\,]$ ▷ $\mathbf{B}$ and $\mathbf{C}$ are empty matrices.
  for $l = 1, \ldots, k$ **do** ▷ Select the k basis vectors from $\mathbf{X}$.
    $(i, \mathbf{c}) \leftarrow \arg\max\{\mathsf{cover}(\mathbf{A}, [\mathbf{B}\ \mathbf{x}^i], \left[\begin{smallmatrix}\mathbf{C}\\ \mathbf{c}\end{smallmatrix}\right]) : i \in \{1, \ldots, n\}, \mathbf{c} \in \{0,1\}^{1 \times m}\}$
    $\mathbf{B} \leftarrow \left[\mathbf{B}\ \mathbf{x}^i\right]$
    $\mathbf{C} \leftarrow \left[\begin{smallmatrix}\mathbf{C}\\ \mathbf{c}\end{smallmatrix}\right]$
  end for
  return $\mathbf{B}$ and $\mathbf{C}$
end function

5 MDL for BMF

In this section we give our approach for selecting model orders for BMF by the Minimum Description Length principle.

5.1 MDL, a brief primer

The MDL (Minimum Description Length) (Rissanen, 1978; Grünwald, 2007) principle, like its close cousin MML (Minimum Message Length) (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[1] $\mathcal{H}$, the best model $H \in \mathcal{H}$ is the one that minimizes

$$L(H) + L(D \mid H)$$

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, however, that with MDL we are only interested in the length of the description, and not in the encoded data itself. That is, we are only concerned with the length of the used codes, and do not have to materialize the codes themselves.

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

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, whether or not 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, between the original data $\mathbf{A}$ and its approximation as given by the Boolean product of its factor matrices $\mathbf{B}$ and $\mathbf{C}$, i.e., $\mathbf{B} \circ \mathbf{C}$. That is, we define error matrix $\mathbf{E}$ for $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$, to be the unique Boolean matrix of dimensions n-by-m such that

$$\mathbf{E} = \mathbf{A} \oplus (\mathbf{B} \circ \mathbf{C}) \,. \tag{5.1}$$

Vice versa, when we are given matrices $\mathbf{B}$, $\mathbf{C}$, and $\mathbf{E}$ we can reconstruct $\mathbf{A}$ without loss.

Example 2. Figure 5.1 shows matrix $\mathbf{E}$ for the example Boolean matrix factorization we discussed in Example 1. From left to right, we have our Boolean matrix $\mathbf{A}$, the Boolean product of two factor matrices $\mathbf{B} \circ \mathbf{C}$, and the resulting error matrix $\mathbf{E}$. If we transmit $\mathbf{B}$, $\mathbf{C}$, and $\mathbf{E}$, the recipient can reconstruct $\mathbf{A}$ without loss by taking the exclusive-OR between $\mathbf{B} \circ \mathbf{C}$ and $\mathbf{E}$.

The approximation of $\mathbf{A}$ by our example factorization leads to 7 errors. Four of these errors are *subtractive*, ones covered by the approximation but not present in $\mathbf{A}$; the black squares in $\mathbf{E}$ within the grey outline of $\mathbf{B} \circ \mathbf{C}$. The remaining three errors are *additive*, ones in $\mathbf{A}$ that are not covered by the approximation. $\lozenge$

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

$$L(\mathbf{A}, H) = L(H) + L(\mathbf{E}) \,, \tag{5.2}$$

where $\mathbf{E}$ follows from $\mathbf{A}$ and H, using Eq. 5.1. Following the MDL principle, the best factorization for $\mathbf{A}$ is found by minimizing Eq. 5.2. As we will discuss later, this is not as simple as it sounds. But let us first discuss how we encode H and $\mathbf{E}$, 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 $\mathbf{B}$ and $\mathbf{C}$ respectively, as

$$L(H) = L_{\mathbb{N}}(n) + L_{\mathbb{N}}(m) + L(k) + L(\mathbf{B}) + L(\mathbf{C}) \,. \tag{5.3}$$

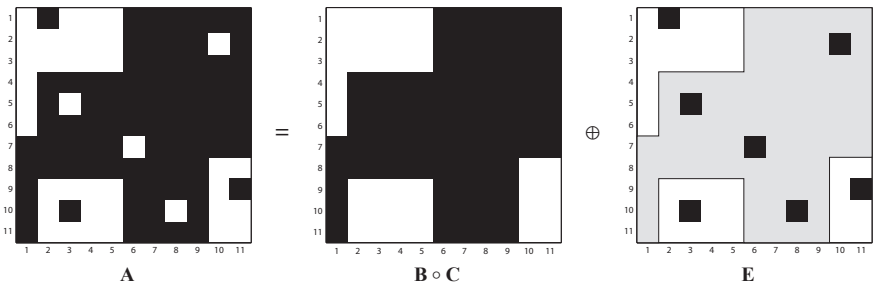

Figure 5.1: Example of a Boolean matrix $\mathbf{A}$, and a lossless representation thereof by the exclusive-OR between an approximation of $\mathbf{A}$ by the Boolean product of two factor matrices, $\mathbf{B} \circ \mathbf{C}$, and the error matrix or residual $\mathbf{E}$. Note that $\mathbf{E}$ consists of both additive and subtractive errors, resp. 1s that exist in $\mathbf{A}$, but not in $\mathbf{B} \circ \mathbf{C}$, and vice-versa.

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 $\mathbf{B}$ and $\mathbf{C}$ using an optimal prefix code (Cover and Thomas, 2006).

To encode m and k, 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

$$L_{\mathbb{N}}(n) = \log^*(n) + \log(c_0)\,, \tag{5.4}$$

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.

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 the encoding of the value of k—we do not encode k using $L_{\mathbb{N}}$ (Eq. 5.4). Instead, we use a fixed number of bits, i.e., block-encoding, which gives us

$$L(k) = \log(\min(m, n))\,. \tag{5.5}$$

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 row and $\mathbf{C}$ per column. 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,

$$L(\mathbf{B}) = k \log(n) + \sum_{i=1}^{k} \left(|\mathbf{b}_i| l_1^{\mathbf{b}_i} + (n - |\mathbf{b}_i|) l_0^{\mathbf{b}_i} \right) \tag{5.6}$$

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

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

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 $\mathbf{C}$, of k-by-m, per row and have

$$L(\mathbf{C}) = k \log(m) + \sum_{j=1}^{k} \left(|\mathbf{c}_j| l_1^{\mathbf{c}_j} + (m - |\mathbf{c}_j|) l_0^{\mathbf{c}_j} \right) \tag{5.7}$$

with

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

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

With the above 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 $\mathbf{A}$. We will now discuss four increasingly involved alternatives for encoding $\mathbf{E}$; we will explore their quality experimentally in Section 6.

Example 3. As an example, let us compute $L(H)$ for the factor matrices $\mathbf{B}$ and $\mathbf{C}$ of Examples 1 and 2. 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 $k \log(n) = 3 \log 11 = 10.38$ bits. The first column of $\mathbf{B}$ consists of seven 1s and four 0s. Hence, as probabilities we resp. have $p_1^{\mathbf{b}_1} = 7/11$, and $p_0^{\mathbf{b}_1} = 4/11$. The corresponding code lengths to encode a value then resp. are $l_1^{\mathbf{b}_1} = -\log(p_1^{\mathbf{b}_1}) = 0.65$ bits, and $l_0^{\mathbf{b}_1} = 1.46$ bits. Encoding the values of the first column of $\mathbf{B}$ hence takes $7 \times 0.65 + 4 \times 1.46 = 10.40$ bits. Skipping the details for the second and third column, we have for $\mathbf{B}$ 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 $L(H) = 2 \times 7.61 + 3.46 + 42.65 + 43.18 = 104.51$ bits.

$\Diamond$

5.2.1 Encoding E: Naïve Factors

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

$$L_f(\mathbf{E}) = \log(mn) + |\mathbf{E}| \Big(-(n-1)\log\left(\frac{n-1}{n}\right) - \log\left(\frac{1}{n}\right) - (m-1)\log\left(\frac{m-1}{m}\right) - \log\left(\frac{1}{m}\right)\Big) \tag{5.8}$$

in which we essentially use the same encoding for the factors as in Eq. 5.6 and 5.7 (but with the extra knowledge that every row (resp., column) in the factor matrices contains only one 1). We refer to this encoding as the Naïve Factors encoding for $\mathbf{E}$.

Example 4. Continuing from Example 3, we here encode $\mathbf{E}$ using Naïve Factors. Encoding one factor of length 11, with only one 1, we require 4.83 bits. For $L_f(\mathbf{E})$ we then have $log(11 \times 11) + 7(4.83 \times 2) = 74.59$ bits. Putting this together with the $L(H)$ as we obtained in Example 3, for Naïve Factors we have a total encoded size, $L(\mathbf{A}, H)$ of 172.19 bits. $\Diamond$

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 $\mathbf{E}$.

5.2.2 Encoding E: Naïve Indices

This observation suggests that we should simply transmit the coordinates of the 1s in $\mathbf{E}$. Clearly, this takes $\log m + \log n$ bits per entry. Then,

$$L_i(\mathbf{E}) = \log(mn) + |\mathbf{E}| \left(\log m + \log n\right) , \tag{5.9}$$

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

Example 5. Continuing from Example 4, we here encode $\mathbf{E}$ using Naïve Indices. Encoding one error, one 1 in $\mathbf{E}$, takes $\log 11 + \log 11 = 6.92$ bits. For $L_i(\mathbf{E})$ we then have $\log(11 \times 11) + 7(6.92) = 55.35$. Putting this together with the $L(H)$ as we obtained in Example 3, for Naïve Indices we have a total encoded size, $L(\mathbf{A}, H)$ of 152.94 bits—which is much cheaper than we saw for Naïve Factors in Example 4. ◊

Naïve Indices saves bits compared to Naïve Factors 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 E: Naïve Exclusive-Or

Although perhaps counter-intuitive, 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 $\mathbf{E}$, i.e., $|\mathbf{E}|$, in $\log mn$ bits, which allows the receiver to calculate the probability $p_1^{\mathbf{E}} = \frac{|\mathbf{E}|}{mn}$, 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 Naïve XOR, is given by

$$L_n(\mathbf{E}) = \log(mn) + |\mathbf{E}|\, l_1 + (mn - |\mathbf{E}|) l_0 , \tag{5.10}$$

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

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

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 Boolean inverse—a cost that is dwarfed by the total number of bits to describe $\mathbf{E}$. We will refer to this encoding as Naïve XOR.

Example 6. Continuing from Example 5, we here instead encode $\mathbf{E}$ using Naïve XOR. 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 $L_n(\mathbf{E})$ 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 3, for Naïve XOR we have a total encoded size, $L(\mathbf{A}, H)$ of 150.01 bits—which in turn is cheaper than with Naïve Indices. $\Diamond$

In general, the gain of Naïve XOR over Naïve Factors and Naïve Indices is substantial.

5.2.4 Encoding 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 $\mathbf{A}$, i.e., 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 Typed XOR.

We encode each of these two parts analogously to Naïve XOR, 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 $\mathbf{E}$ by the Typed XOR encoding as

$$L_x(\mathbf{E}) = L(\mathbf{E}^+) + L(\mathbf{E}^-) \tag{5.11}$$

where

$$\begin{aligned} L(\mathbf{E}^+) = \log(mn - |\mathbf{B} \circ \mathbf{C}|) + \left|\mathbf{E}^+\right| l_1^+ \\ + (mn - |\mathbf{B} \circ \mathbf{C}| - \left|\mathbf{E}^+\right|) l_0^+ \end{aligned}$$

and

$$L(\mathbf{E}^-) = \log(|\mathbf{B} \circ \mathbf{C}|) + \left|\mathbf{E}^-\right| l_1^- + (|\mathbf{B} \circ \mathbf{C}| - \left|\mathbf{E}^-\right|) l_0^-$$

respectively give the number of bits to encode the additive and subtractive parts of $\mathbf{E}$. We calculate l_1^+, l_0^+, l_1^-, l_1^- analogous to how we calculate l for Naïve XOR.

Example 7. Continuing from Example 6, we here encode $\mathbf{E}$ using Typed XOR. In Figure 5.1 the area of $\mathbf{E}$ 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 Naïve XOR, we skip the details. In total, for encoding $\mathbf{E}$ by Typed XOR we require $L_x(\mathbf{E}) = 49.89$ bits. Note that this is more than Naïve XOR; this is because the distribution of errors within $\mathbf{E}^+$ and $\mathbf{E}^-$ here are very similar. As such, Typed XOR 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 Typed XOR obtains large gains over Naïve XOR. For Typed XOR, we here have a total encoded size, $L(\mathbf{A}, H)$ of 154.4 bits. $\Diamond$

Like Naïve XOR, 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 $\mathbf{A}$. 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 $\mathbf{B}$, $\mathbf{C}$, and $\mathbf{E}$ 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 above. 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 a data-to-model 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 data-to-model encoding for the factor matrices, and the error matrix. We start with the factor matrices. As above, we assume independence between factors, and hence will encode the factors independently of each other. This gives us

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

and

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

for the $\mathbf{B}$ and $\mathbf{C}$ factor matrices respectively. We will use this encoding for both of the two strategies for encoding $\mathbf{E}$.

Example 8. Continuing from Example 7, we now calculate the description length of our running example model H by the data-to-model approach, i.e. $L^{dtm}(H)$.

For $\mathbf{B}$, 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 eleven (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 $\mathbf{B}$. Analog for $\mathbf{C}$, we require $L^{dtm}(\mathbf{C}) = 36.93$ bits. Wrapping things up, we have $L(H) = 92.06$ when using the data-to-model 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 3. ◊

For the error matrix $\mathbf{E}$ we construct encodings alike to the Exclusive-Or encodings above, by which we have

$$L_n^{dtm}(\mathbf{E}) = \log(mn) + \log\left(\binom{mn}{|\mathbf{E}|}\right)$$

for the Data-to-Model equivalent of Naïve XOR. We will refer to this encoding as Naïve XOR DtM.

Next, for the Typed Exclusive-Or encoding, when we follow a data-to-model approach, we have

$$L_x^{dtm}(\mathbf{E}) = L^{dtm}(\mathbf{E}^+) + L^{dtm}(\mathbf{E}^-)$$

where

$$L^{dtm}(\mathbf{E}^+) = \log(mn - |\mathbf{B} \circ \mathbf{C}|) + \log\left(\binom{mn - |\mathbf{B} \circ \mathbf{C}|}{|\mathbf{E}^+|}\right)$$

and

$$L^{dtm}(\mathbf{E}^-) = \log(|\mathbf{B} \circ \mathbf{C}|) + \log\left(\binom{|\mathbf{B} \circ \mathbf{C}|}{|\mathbf{E}^-|}\right) .$$

We refer to this encoding as Typed XOR DtM.

Example 9. Continuing from Example 8, we can now calculate for our running example the encoded size of $\mathbf{E}$ using resp. Naïve XOR DtM and Typed XOR DtM. For $L_n^{dtm}(\mathbf{E})$ we arrive at a cost of 42.80 bits, and for Typed XOR DtM we require $L_n^{dtm}(\mathbf{E}) = 45.47$ bits.

For the total encoded sizes, $L(\mathbf{A}, H)$, using the data-to-model encoding for H, we then require 134.86 and 137.54 bits for resp. Naïve XOR DtM and Typed XOR DtM.

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.

Like in Example 7 we see that for our running example the typed encoding does not provide a gain, although the difference between Naïve XOR DtM and Typed XOR DtM is already smaller. As we will see in the experiments, for real data Typed XOR DtM does obtain a strong lead over Naïve XOR DtM. $\Diamond$

By making a choice for one of the six above 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, Typed XOR DtM 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 non-computable. But even when we try to minimize some given encoding, like one of the above, 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 1. 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 $\mathbf{C}$ such that $L(\mathbf{A}, (\mathbf{B}, \mathbf{C}))$ is minimized.

Proving proposition 1 is straight forward: 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 1 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 1 ((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 2. *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 Naïve Indices encoding scheme). Assume we are given n-by-m and n-by-k Boolean matrices $\mathbf{A}$ and $\mathbf{B}$ as in Theorem 1. 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 non-negative 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 build similarly with copies of $\mathbf{B}$.

We now claim that any k-by-m Boolean matrix $\mathbf{C}^*$ that minimizes the Naïve Indices 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

$$\begin{aligned}
&\operatorname*{arg\,max}_{\mathbf{C}\in\{0,1\}^{k\times m}} L(\mathbf{C}) - \operatorname*{arg\,min}_{\mathbf{C}\in\{0,1\}^{k\times m}} L(\mathbf{C}) \\
&= k\log(m) + \sum_{j=1}^{k} \left(-m/2\log(1/2) - m/2\log(1/2)\right) \\
&\quad - \left(k\log(m) + \sum_{j=1}^{k} \left(-0\log 0 - 0\log 1\right)\right) \\
&= km\ .
\end{aligned} \tag{5.12}$$

Now, let $\alpha = km$. As Naïve Indices 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 above 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 re-start $\min(n, m)$ 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 re-start 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 Asso with MDL 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 paper; 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.[1] 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.

[1] `http://www.mpi-inf.mpg.de/~pmiettin/mdl4bmf/`

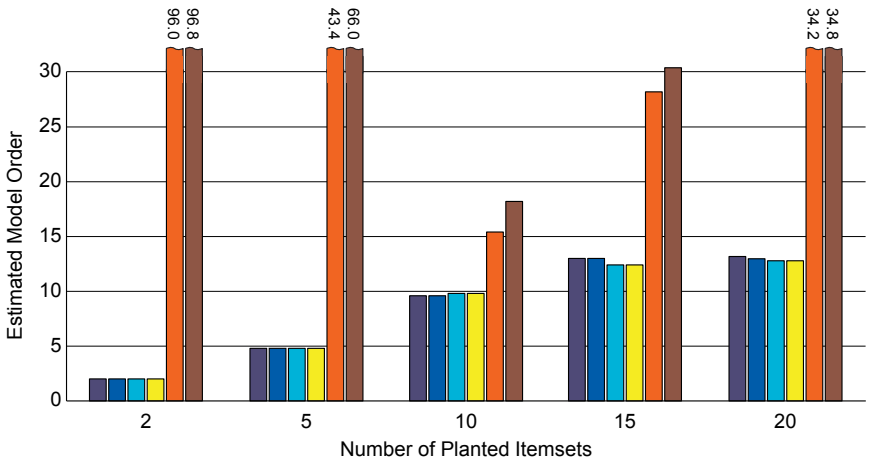

Figure 6.1: **Comparing between encodings**. Model order estimates for varying true model orders, averaged over five independently generated datasets. For all datasets, width and frequencies of the planted itemsets are equal on expectation. The bars represent the estimates obtained when using Asso with, from left to right, Typed XOR DtM, Naïve XOR DtM, Typed XOR, Naïve XOR, Naïve Indices, and Naïve Factors.

For each of these settings, we generate binary 8000-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 set-up 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 6.1. 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 we see a slight under-estimation, 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 co-occur. By it's greedy strategy, Asso groups some of these together. For $k = 10$, we note that Naïve Indices and Naïve Factors 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, for $k = 2, 5, 15, 20$ from resp. $[10, 40]$, $[5, 15]$, $[2, 5]$, and $[2, 3]$. For the rest, we proceed as above.

Figure 6.2 shows the results. For values of k below 10, performance is virtually equal to that of Fig 6.1. 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 Typed XOR DtM works best by a small margin.

While under-estimation is clearly preferred to over-estimation—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, and 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 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

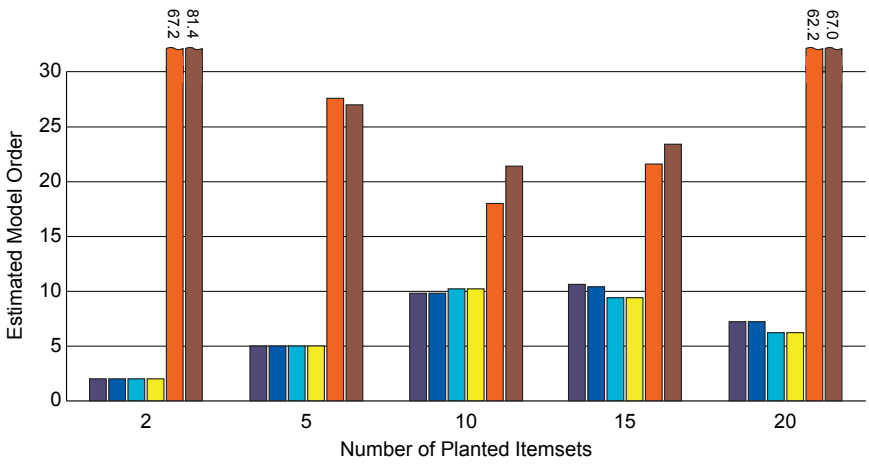

Figure 6.2: **Comparing between encodings**. Model order estimates for varying true model orders, averaged over five independently generated datasets. Average density of the datasets kept equal on expectation, by changing the width of the planted itemsets, i.e. higher number of planted itemsets corresponds to smaller itemsets. The bars represent the estimates obtained when using Asso with, from left to right, Typed XOR DtM, Naïve XOR DtM, Typed XOR, Naïve XOR, Naïve Indices, and Naïve Factors.

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 above, and run Asso on each dataset, scoring every factorization by each of our encodings. We plot average estimated model orders in Figure 6.3. The bar plot shows us that both Naïve Factors and Naïve Indices over-estimate 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 Typed XOR DtM 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 sub-par performance of the Naïve Factors and Naïve Indices encodings show the need of balancing the encoding of the model and error. Overall, Typed XOR DtM

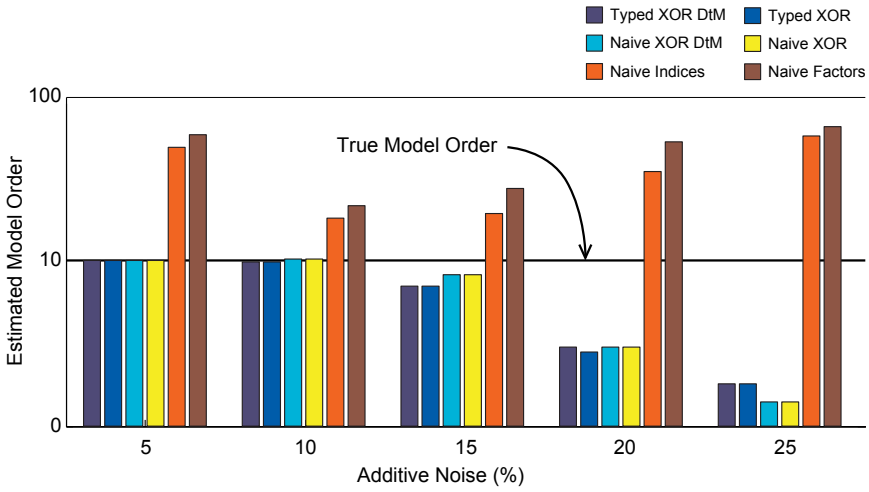

Figure 6.3: **Comparing between encodings**. Model order estimates for varying amounts of additive noise (true $k = 10$), averaged over five independently generated datasets. The bars represent the estimates obtained when using Asso with, from left to right, Typed XOR DtM, Naïve XOR DtM, Typed XOR, Naïve XOR, Naïve Indices, and Naïve Factors. All values averaged over 5 datasets. Note that beyond 12.5% noise there are more 1s in the data due to noise than to structure.

and Naïve XOR DtM 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 (T-Cost), and PaNDa.

Cross validation 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 hold-out data.

As we briefly discussed in Section 1, while 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 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 bi-cross-validation approach (Owen and Perry, 2009), as their method to generalize learned factors to the test data does not apply to BMF.

T-Cost in essence is a more practical variant of cross-validation, 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 above. We run each of the methods, and record the reported model order estimates, for each of the datasets of both settings. As above, all results are averaged over five independently generated sets of databases. We observe that cross-validation always estimates k to full-rank, and therefore conclude that 'vanilla' CV is not a good choice for model order selection in BMF, and 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. Fig. 6.4 and Fig. 6.5 show the corresponding bar plots of the relative error of the model order estimates. Both show the same trend. Overall PaNDa suffers from severe over-estimation, finding hundreds of factors instead of tens. Asso with Typed XOR DtM, Naïve XOR DtM or T-Cost, on the other hand, provides good estimates. For T-Cost the estimates do deteriorate towards over estimation 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 Naïve XOR DtM, Typed XOR DtM, or T-Cost all provide correct estimates at the noise levels up to 15%, and then deteriorate. As above, PaNDa strongly over-estimates.

6.3 Real Data

Next, we consider 8 real datasets, most of which are publicly available. Below we give a short description per dataset, and an overview in Table 6.1.

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

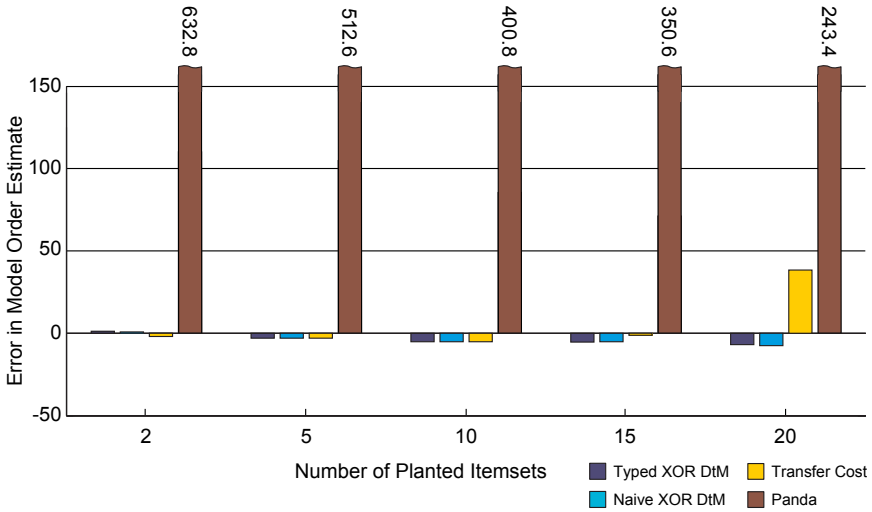

Figure 6.4: **Comparing between methods**. Error in model order estimates, averaged over five independently generated datasets, for varying true model orders. For all datasets, width and frequencies of the planted itemsets are equal on expectation. The bars represent the estimates obtained with, from left to right, using Asso with Typed XOR DtM, Naïve XOR DtM, T-Cost, and PaNDa.

Table 6.1: 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 Typed XOR DtM, 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	–	1835	1950
Paleo	501	139	5.1	46	96	19

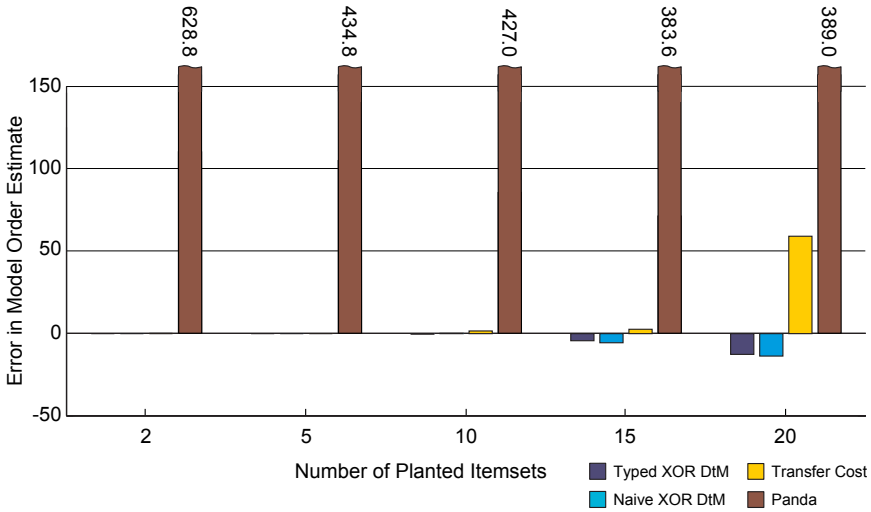

Figure 6.5: **Comparing between methods**. Error in model order estimates, averaged over five independently generated datasets, for varying true model orders. Average density of the datasets kept equal on expectation, by changing the width of the planted itemsets, i.e. higher number of planted itemsets corresponds to smaller itemsets. The bars represent the estimates obtained with, from left to right, using Asso with Typed XOR DtM, Naïve XOR DtM, T-Cost, and PaNDa.

stop words removed[2] (De Bie, 2011).

DBLP contains records of in which of the 19 conferences the 6980 authors had published. The dataset is collected from the DBLP database[3] and it is pre-processed as in (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[4] consists of presence records of European mammals within geographical areas of 50×50 kilometers (Mitchell-

[2]Available upon request from the authors of De Bie (2011)

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

[4]Available for research purposes from the Societas Europaea Mammalogica at `http://www.european-mammals.org`

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,[5] containing, for 400 posts from 4 newsgroups[6], the usage of 800 words.

NSF Abstracts contains the occurrence of terms in abstracts of successful NSF grant applications.[7] The pre-processing is explained in (Miettinen, 2009). The resulting data is extremely sparse (0.9% 1s).

Last, *Paleo* consists of fossil records per location.[8]

We ran each method on these datasets, and give the returned model orders in Table 6.1. 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.

For Asso, with the exception of *NSF*, the factorisations 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 below.

First, however, we investigate the sensitivity to Asso's parameter τ for the model order estimates on real data. In Figure 6.6, we plot two typical examples of total encoded lengths, $L(\mathbf{A}, H)$, for Asso with Typed XOR DtM 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 6.6, we show the total encoded lengths obtained

[5] `http://people.csail.mit.edu/jrennie/20Newsgroups/`

[6] The authors are grateful to Ata Kabán for pre-processing the data. The exact pre-processing is detailed in (Miettinen, 2009).

[7] `http://kdd.ics.uci.edu/databases/nsfabs/nsfawards.html`

[8] NOW public release 030717, available from `http://www.helsinki.fi/science/now/` (Fortelius et al., 2003).

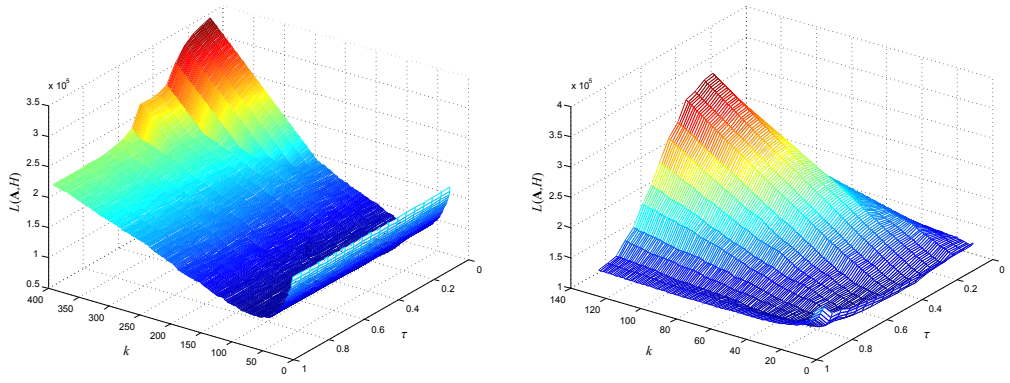

Figure 6.6: 3D plots of the number of bits (y-axis), for varying values of k (x-axis) and τ (z-axis) for Asso with Typed XOR DtM encoding. (left) The *DNA* dataset. Minimal description length attained at $k = 54$ and $\tau = 0.35$. (right) The *Mammals* dataset. Minimal description length attained at $k = 13$ and $\tau = 0.65$.

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 6.7 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 non-hierarchical algorithms.

As mentioned above, 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, ICDM, and (4) ICML, ECML. These four factors clearly correspond to four different field 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 (Myllykangas et al., 2006), only missing some of the smallest itemsets.

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, as Tatti et al. (2006) report that explaining 90% of the variance with PCA requires 79 factors.

We also checked the error curves for these data sets. 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 small 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 $k = 1950$. This is one of the few datasets where Panda returned a lower estimate, of 1835 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 6.7. Furthermore, using PCA, 1848 factors explain only about 69% of the variance, and in order to explain at least 90% of the variance, PCA needs to have 3399 factors. While

these numbers are not directly comparable, they do indicate that the data has very complex structure (e.g., very little overlap between the rows). This can, at least partly, explain also the need for high k with BMF.

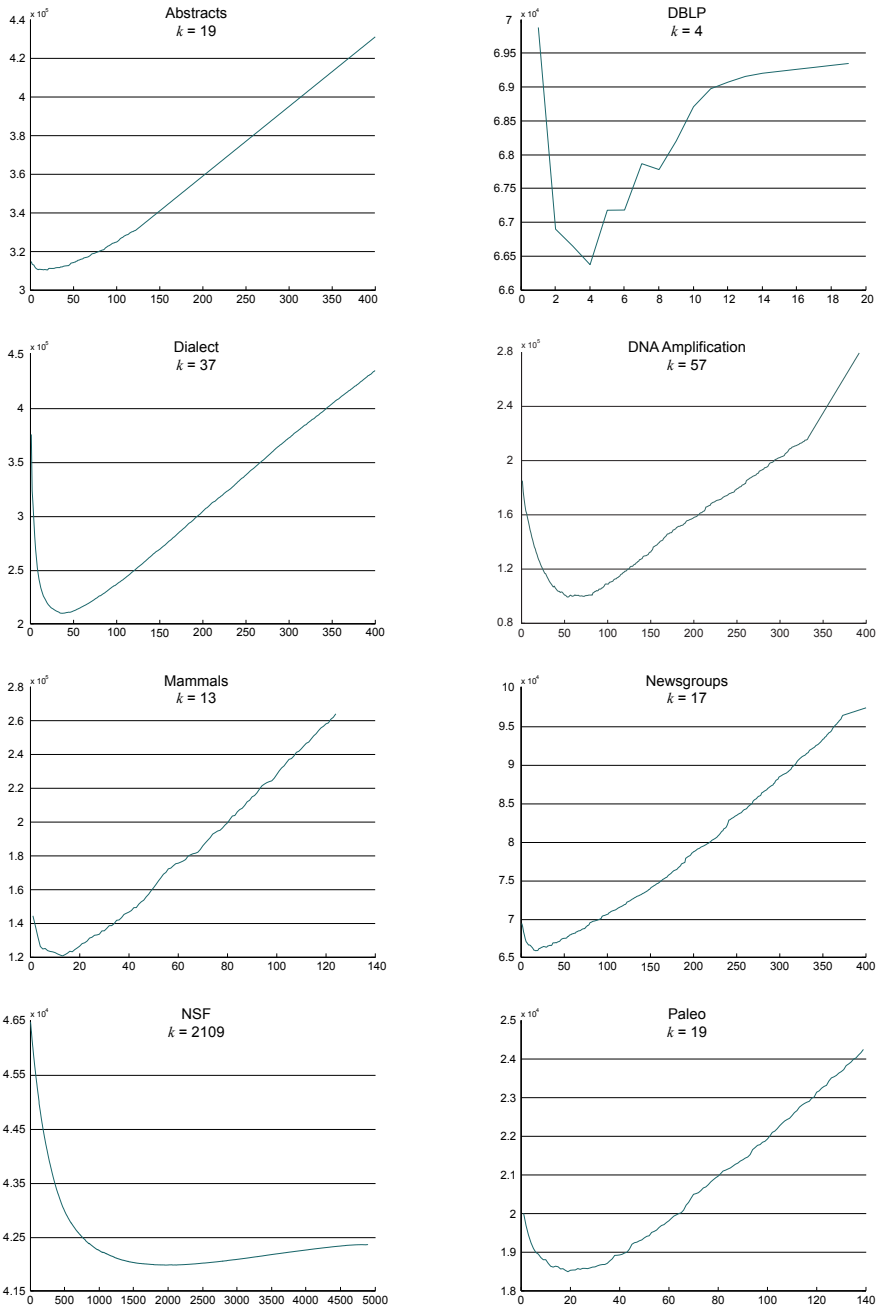

Figure 6.7: MDL score per k obtained by Asso for 8 datasets. Value for τ fixed per dataset to the value at which the minimal description length is found over all τ and k. For all datasets we considered values for k up to the data dimensions, with 400 as maximum.

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, Typed XOR DtM, 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 over-estimates the model order; which was particularly apparent in the synthetic data, but similarly 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 Naïve Factors and Naïve Indices 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 Typed XOR DtM to encode **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 non-hierarchical algorithms are used.

The development of an efficient BMF algorithm that optimizes the MDL score would make for important future research—given the complexity of BMF, and that of MDL, very smart heuristics will be required for any algorithm that combines these in order to find the BMF that directly minimizes the description length.

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 paper, the proposed encoding length functions have already been adapted to find model orders in joint subspace Boolean matrix factorization (Miettinen, 2012).

8 Conclusion

We proposed a general solution to the model order selection problem for Boolean matrix factorization. By the Minimum Description Length 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 data-to-model based encoding. We showed that minimizing the MDL score is already NP-hard for the relatively simple Naïve Indices 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 Boolean matrix factorizations of a given rank, as well as further analysis of the complexity of problem.

Bibliography

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

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

Chandola, V. and Kumar, V. 2007. Summarization – compressing data into an informative representation. *Knowledge and Information Systems 12,* 3, 355–378.

Cilibrasi, R. and Vitányi, P. 2005. Clustering by compression. *IEEE Transactions on Information Technology 51,* 4, 1523–1545.

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

De Bie, T. 2011. Maximum entropy models and subjective interestingness: an application to tiles in binary databases. *Data Mining and Knowledge Discovery 23,* 3, 1–40.

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

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

Embleton, S. M. and Wheeler, E. S. 2000. Computerized dialect atlas of finnish: Dealing with ambiguity. *Journal of Quantitative Linguistics 7,* 3, 227–231.

Faloutsos, C. and Megalooikonomou, V. 2007. On data mining, compression and Kolmogorov complexity. In *Data Mining and Knowledge Discovery*. Vol. 15. Springer-Verlag, 3–20.

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

Fortelius, M. et al. 2003. Neogene of the old world database of fossil mammals (NOW). http://www.helsinki.fi/science/now/.

Frank, A. and Asuncion, A. 2010. UCI machine learning repository. http://archive.ics.uci.edu/ml.

Frank, M., Chehreghani, M. H., and Buhmann, J. M. 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), Athens, Greece*. 423–438.

Garriga, G. C., Junttila, E., and Mannila, H. 2011. Banded structure in binary matrices. *Knowledge and Information Systems 28,* 1, 197–226.

Geerts, F., Goethals, B., and Mielikäinen, T. 2004. Tiling databases. In *Proceedings of Discovery Science*. 278–289.

Golub, G. H. and Van Loan, C. F. 1996. *Matrix Computations*. JHU Press.

Grünwald, P. 2007. *The Minimum Description Length Principle*. MIT Press.

Heikinheimo, H., Vreeken, J., Siebes, A., and Mannila, H. 2009. Low-entropy set selection. In *Proceedings of the 9th SIAM International Conference on Data Mining (SDM), Sparks, NV*. 569–580.

Keogh, E., Lonardi, S., and Ratanamahatana, C. A. 2004. Towards parameter-free data mining. In *Proceedings of the 10th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), Seattle, WA*. 206–215.

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

van Leeuwen, M., Vreeken, J., and Siebes, A. 2009. Identifying the components. *Data Mining and Knowledge Discovery 19,* 2, 173–292.

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

Lu, H., Vaidya, J., and Atluri, V. 2008. Optimal Boolean matrix decomposition: Application to role engineering. In *Proceedings of the 24th International Conference on Data Engineering (ICDE), Cancun, Mexico*. 297–306.

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

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

Miettinen, P. 2009. Matrix decomposition methods for data mining: Computational complexity and algorithms. Ph.D. thesis, University of Helsinki.

Miettinen, P. 2010. Sparse Boolean matrix factorizations. In *Proceedings of the 10th IEEE International Conference on Data Mining (ICDM), Sydney, Australia*. 935–940.

Miettinen, P. 2012. On Finding Joint Subspace Boolean Matrix Factorizations. In *Proceedings of the 12th SIAM International Conference on Data Mining (SDM), Anaheim, CA*. 954–965.

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

Miettinen, P. and Vreeken, J. 2011. Model order selection for boolean matrix factorization. In *Proceedings of the 17th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), San Diego, CA*. 51–59.

Minka, T. P. 2001. Automatic choice of dimensionality for PCA. In *Proceedings of the 13th Annual Conference on Neural Information Processing Systems (NIPS)*. 598–604.

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

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

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

Nau, D. S., Markowsky, G., Woodbury, M. A., and Amos, D. B. 1978. A mathematical analysis of human leukocyte antigen serology. *Mathematical Biosciences 40*, 243–270.

Owen, A. B. and Perry, P. O. 2009. Bi-cross-validation of the SVD and the nonnegative matrix factorization. *Annals of Applied Statistics 3,* 2, 564–594.

Paatero, P. and Tapper, U. 1994. Positive matrix factorization: A nonnegative factor model with optimal utilization of error estimates of data values. *Environmetrics 5*, 111–126.

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

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

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

Schmidt, M., Winther, O., and Hansen, L. 2009. Bayesian nonnegative matrix factorization. In *Proceedings of International Conference on Independent Component Analysis and Signal Separation*. LNCS Series, vol. 5411. 540–547.

Siebes, A., Vreeken, J., and van Leeuwen, M. 2006. Item sets that compress. In *Proceedings of the 6th SIAM International Conference on Data Mining (SDM), Bethesda, MD.* SIAM, 393–404.

Streich, A., Frank, M., Basin, D., and Buhmann, J. 2009. Multi-assignment clustering for Boolean data. In *Proceedings of the 26th International Conference on Machine Learning (ICML), Montreal, Canada.* 969–976.

Tatti, N., Mielikainen, T., Gionis, A., and Mannila, H. 2006. What is the dimension of your binary data? In *Proceedings of the 6th IEEE International Conference on Data Mining (ICDM), Hong Kong, China.* 603–612.

Tatti, N. and Vreeken, J. 2008. Finding good itemsets by packing data. In *Proceedings of the 8th IEEE International Conference on Data Mining (ICDM), Pisa, Italy.* 588–597.

Vaidya, J., Atluri, V., and Guo, Q. 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).* 175–184.

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

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

Vreeken, J., van Leeuwen, M., and Siebes, A. 2011. Krimp: Mining itemsets that compress. *Data Mining and Knowledge Discovery 23,* 1, 169–214.

Wallace, C. 2005. *Statistical and inductive inference by minimum message length.* Springer-Verlag.

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

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

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

Below you find a list of the most recent research reports of the Max-Planck-Institut für Informatik. Most of them are accessible via WWW using the URL `http://www.mpi-inf.mpg.de/reports`. Paper copies (which are not necessarily free of charge) can be ordered either by regular mail or by e-mail at the address below.

Max-Planck-Institute für Informatik
– Library and Publications –
Campus E 1 4

D-66123 Saarbrücken

E-mail: `library@mpi-inf.mpg.de`

MPI-I-2012-RG1-001	M. Suda, C. Weidenbach	Labelled superposition for PLTL
MPI-I-2012-4-001	J. Kerber, M. Bokeloh, M. Wand, H. Seidel	Symmetry detection in large scale city scans
MPI-I-2011-5-002	B. Taneva, M. Kacimi, G. Weikum	Finding images of rare and ambiguous entities
MPI-I-2011-5-001	A. Anand, S. Bedathur, K. Berberich, R. Schenkel	Temporal index sharding for space-time efficiency in archive search
MPI-I-2011-4-005	A. Berner, O. Burghard, M. Wand, N.J. Mitra, R. Klein, H. Seidel	A morphable part model for shape manipulation
MPI-I-2011-4-001	M. Granados, J. Tompkin, K. In Kim, O. Grau, J. Kautz, C. Theobalt	How not to be seen inpainting dynamic objects in crowded scenes
MPI-I-2010-RG1-001	M. Suda, C. Weidenbach, P. Wischnewski	On the saturation of YAGO
MPI-I-2010-5-008	S. Elbassuoni, M. Ramanath, G. Weikum	Query relaxation for entity-relationship search
MPI-I-2010-5-007	J. Hoffart, F.M. Suchanek, K. Berberich, G. Weikum	YAGO2: a spatially and temporally enhanced knowledge base from Wikipedia
MPI-I-2010-5-006	A. Broschart, R. Schenkel	Real-time text queries with tunable term pair indexes
MPI-I-2010-5-005	S. Seufert, S. Bedathur, J. Mestre, G. Weikum	Bonsai: Growing Interesting Small Trees
MPI-I-2010-5-004	N. Preda, F. Suchanek, W. Yuan, G. Weikum	Query evaluation with asymmetric web services
MPI-I-2010-5-003	A. Anand, S. Bedathur, K. Berberich, R. Schenkel	Efficient temporal keyword queries over versioned text
MPI-I-2010-5-002	M. Theobald, M. Sozio, F. Suchanek, N. Nakashole	URDF: Efficient Reasoning in Uncertain RDF Knowledge Bases with Soft and Hard Rules
MPI-I-2010-5-001	K. Berberich, S. Bedathur, O. Alonso, G. Weikum	A language modeling approach for temporal information needs
MPI-I-2010-1-001	C. Huang, T. Kavitha	Maximum cfardinality popular matchings in strict two-sided preference lists
MPI-I-2009-RG1-005	M. Horbach, C. Weidenbach	Superposition for fixed domains
MPI-I-2009-RG1-004	M. Horbach, C. Weidenbach	Decidability results for saturation-based model building
MPI-I-2009-RG1-002	P. Wischnewski, C. Weidenbach	Contextual rewriting
MPI-I-2009-RG1-001	M. Horbach, C. Weidenbach	Deciding the inductive validity of $\forall\exists^*$ queries
MPI-I-2009-5-007	G. Kasneci, G. Weikum, S. Elbassuoni	MING: Mining Informative Entity-Relationship Subgraphs
MPI-I-2009-5-006	S. Bedathur, K. Berberich, J. Dittrich, N. Mamoulis, G. Weikum	Scalable phrase mining for ad-hoc text analytics
MPI-I-2009-5-005	G. de Melo, G. Weikum	Towards a Universal Wordnet by learning from combined evidenc
MPI-I-2009-5-004	N. Preda, F.M. Suchanek, G. Kasneci, T. Neumann, G. Weikum	Coupling knowledge bases and web services for active knowledge
MPI-I-2009-5-003	T. Neumann, G. Weikum	The RDF-3X engine for scalable management of RDF data

MPI-I-2009-5-003	T. Neumann, G. Weikum	The RDF-3X engine for scalable management of RDF data
MPI-I-2009-5-002	M. Ramanath, K.S. Kumar, G. Ifrim	Generating concise and readable summaries of XML documents
MPI-I-2009-4-006	C. Stoll	Optical reconstruction of detailed animatable human body models
MPI-I-2009-4-005	A. Berner, M. Bokeloh, M. Wand, A. Schilling, H. Seidel	Generalized intrinsic symmetry detection
MPI-I-2009-4-004	V. Havran, J. Zajac, J. Drahokoupil, H. Seidel	MPI Informatics building model as data for your research
MPI-I-2009-4-003	M. Fuchs, T. Chen, O. Wang, R. Raskar, H.P.A. Lensch, H. Seidel	A shaped temporal filter camera
MPI-I-2009-4-002	A. Tevs, M. Wand, I. Ihrke, H. Seidel	A Bayesian approach to manifold topology reconstruction
MPI-I-2009-4-001	M.B. Hullin, B. Ajdin, J. Hanika, H. Seidel, J. Kautz, H.P.A. Lensch	Acquisition and analysis of bispectral bidirectional reflectance distribution functions
MPI-I-2008-RG1-001	A. Fietzke, C. Weidenbach	Labelled splitting
MPI-I-2008-5-004	F. Suchanek, M. Sozio, G. Weikum	SOFI: a self-organizing framework for information extraction
MPI-I-2008-5-003	G. de Melo, F.M. Suchanek, A. Pease	Integrating Yago into the suggested upper merged ontology
MPI-I-2008-5-002	T. Neumann, G. Moerkotte	Single phase construction of optimal DAG-structured QEPs
MPI-I-2008-5-001	G. Kasneci, M. Ramanath, M. Sozio, F.M. Suchanek, G. Weikum	STAR: Steiner tree approximation in relationship-graphs
MPI-I-2008-4-003	T. Schultz, H. Theisel, H. Seidel	Crease surfaces: from theory to extraction and application to diffusion tensor MRI
MPI-I-2008-4-002	D. Wang, A. Belyaev, W. Saleem, H. Seidel	Estimating complexity of 3D shapes using view similarity
MPI-I-2008-1-001	D. Ajwani, I. Malinger, U. Meyer, S. Toledo	Characterizing the performance of Flash memory storage devices and its impact on algorithm design
MPI-I-2007-RG1-002	T. Hillenbrand, C. Weidenbach	Superposition for finite domains
MPI-I-2007-5-003	F.M. Suchanek, G. Kasneci, G. Weikum	Yago : a large ontology from Wikipedia and WordNet
MPI-I-2007-5-002	K. Berberich, S. Bedathur, T. Neumann, G. Weikum	A time machine for text search
MPI-I-2007-5-001	G. Kasneci, F.M. Suchanek, G. Ifrim, M. Ramanath, G. Weikum	NAGA: searching and ranking knowledge
MPI-I-2007-4-008	J. Gall, T. Brox, B. Rosenhahn, H. Seidel	Global stochastic optimization for robust and accurate human motion capture
MPI-I-2007-4-007	R. Herzog, V. Havran, K. Myszkowski, H. Seidel	Global illumination using photon ray splatting
MPI-I-2007-4-006	C. Dyken, G. Ziegler, C. Theobalt, H. Seidel	GPU marching cubes on shader model 3.0 and 4.0
MPI-I-2007-4-005	T. Schultz, J. Weickert, H. Seidel	A higher-order structure tensor
MPI-I-2007-4-004	C. Stoll, E. de Aguiar, C. Theobalt, H. Seidel	A volumetric approach to interactive shape editing
MPI-I-2007-4-003	R. Bargmann, V. Blanz, H. Seidel	A nonlinear viseme model for triphone-based speech synthesis
MPI-I-2007-4-002	T. Langer, H. Seidel	Construction of smooth maps with mean value coordinates
MPI-I-2007-4-001	J. Gall, B. Rosenhahn, H. Seidel	Clustered stochastic optimization for object recognition and pose estimation
MPI-I-2007-2-001	A. Podelski, S. Wagner	A method and a tool for automatic veriication of region stability for hybrid systems
MPI-I-2007-1-003	A. Gidenstam, M. Papatriantafilou	LFthreads: a lock-free thread library