Getting to Know the Unknown Unknowns: Destructive-Noise Resistant Boolean Matrix Factorization

Sanjar Karaev° Pauli Miettinen° Jilles Vreeken°,•

Abstract
Finding patterns in binary data is a classical problem in data mining, dating back to at least frequent itemset mining. More recently, approaches such as tiling and Boolean matrix factorization (BMF), have been proposed to find sets of patterns that aim to explain the full data well. These methods, however, are not robust against non-trivial destructive noise, i.e. when relatively many 1s are removed from the data: tiling can only model additive noise while BMF assumes approximately equal amounts of additive and destructive noise. Most real-world binary datasets, however, exhibit *mostly* destructive noise. In presence/absence data, for instance, it is much more common to fail to observe something than it is to observe a spurious presence. To address this problem, we take the recent approach of employing the Minimum Description Length (MDL) principle for BMF and introduce a new algorithm, `Nassau`, that directly optimizes the description length of the factorization instead of the reconstruction error. In addition, unlike the previous algorithms, it can adjust the factors it has discovered during its search. Empirical evaluation on synthetic data shows that `Nassau` excels at datasets with high destructive noise levels and its performance on real-world datasets confirms our hypothesis of the high numbers of missing observations in the real-world data.

Keywords: Boolean matrix factorization; minimum description length; unknown unknowns

1 Introduction

> [A]s we know, there are known knowns; there are things we know we know. We also know there are known unknowns; that is to say we know there are some things we do not know. But there are also unknown unknowns – the ones we don't know we don't know. And … it is the latter category that tend to be the difficult ones. [30]

While then-Secretary of Defense Donald Rumsfeld probably was not thinking about data mining when he made his famous comment about unknown unknowns, they are, arguably, the main problem data miners dealing with observation data have to face. A *known unknown*, in this setting, is an element for which we know we do not know its value. As an example, in a movie rating matrix we know which movies the users have not rated yet. In contrast, an *unknown unknown* is an element we have not observed, but we do not know if this is because it does not exist or because we have failed to observe it.

We assume that many real-world presence/absence (or *binary*) datasets contain unknown unknowns, and that it is much more common to fail to observe something that is than it is to observe something that is not there. In other words, we assume there is more *destructive* noise than there is *additive* noise – *without* having to pre-specify these amounts exactly.

In order to find interesting patterns from the data, we consider Boolean matrix factorization (BMF) [20], a technique that has been successfully applied to a variety of data mining problems considering binary data (e.g. [14, 16, 20, 22]). In its standard form, BMF aims to find a low-rank Boolean factorization of the data that is as close to the original as possible (i.e. minimizes the reconstruction error).

But here as well the unknown unknowns are the difficult ones. When we know which observations are unknown, we can ignore them, and fit the model (i.e. the Boolean matrix factorization) only to the observed parts of the data. But we cannot ignore all unobserved values, as that leaves only the observations (1s in the data) to work with. Still, we also have to be careful that we do not consider unobserved values as important to actual observations – as standard reconstruction error does – as this ignores our hypothesis that real data has more unobserved than spuriously observed values.

To address this problem, we use the Minimum Description Length principle (MDL) [11, 26], which is very useful in tackling the problem of the trade-off between fitting the data well while keeping the model simple. In general, the more complexity we allow in the model, the better we can fit the data. However, having high model order comes at the cost of fitting the noise. MDL identifies the optimal balance.

In this paper we present `Nassau`, a new BMF algorithm that is designed to directly minimize description length. A key aspect is that – unlike the majority of the previously proposed BMF algorithms – it can correct its previous mistakes. `Nassau` is quite robust to destructive noise, which is especially beneficial for real-world data as in many domains there are zeros simply due to the lack of observation.

2 Notation

In this paper we consider matrices of Boolean values. We denote a matrix by upper-case boldface letters ($\mathbf{A}$), and vectors by lower-case boldface letters ($\mathbf{a}$). If $\mathbf{A}$ is an n-by-m Boolean matrix, $|\mathbf{A}|$ denotes the number of 1s in it, i.e.,

°Max-Planck Institute for Informatics, Saarbrücken, Germany.
•Saarland University, Saarbrücken, Germany.
{skaraev,pmiettin,jilles}@mpi-inf.mpg.de

$|\mathbf{A}| = \sum_{i,j} a_{ij}$. For a matrix $\mathbf{A}$ we denote its ith row by $\mathbf{A}_i$ and its jth column by $\mathbf{A}^j$. The matrix obtained from $\mathbf{A}$ by removing its jth column (respectively ith row) is denoted by $\mathbf{A}^{-j}$ ($\mathbf{A}_{-i}$). Given a matrix $\mathbf{A}$ of size n-by-m and a column vector $\mathbf{c}$ of length n, or a row vector $\mathbf{r}$ of length m, we denote by $[\mathbf{A}, \mathbf{c}]$ and $\left[\begin{smallmatrix}\mathbf{A}\\ \mathbf{r}\end{smallmatrix}\right]$ the matrices obtained by concatenating $\mathbf{A}$ with $\mathbf{c}$ and $\mathbf{r}$, respectively.

Let $\mathbf{A} \in \{0,1\}^{n\times m}$, $\mathbf{B} \in \{0,1\}^{n\times k}$ and $\mathbf{C} \in \{0,1\}^{k\times m}$. We denote by $\mathbf{B} \circ \mathbf{C}$ the n-by-m *Boolean product* of matrices $\mathbf{B}$ and $\mathbf{C}$. The Boolean matrix product is defined like the normal product, but over the Boolean semiring, that is, $(\mathbf{B} \circ \mathbf{C})_{ij} = \bigvee_{\ell=1}^{k} \mathbf{B}_{i\ell}\mathbf{C}_{\ell j}$.

Let $\langle \mathbf{B}, \mathbf{C} \rangle$ be an (approximate) Boolean decomposition of $\mathbf{A}$, $\mathbf{A} \approx \mathbf{B} \circ \mathbf{C}$. We call $\mathbf{B}$ and $\mathbf{C}$ *factors* of this decomposition, and for any $1 \le l \le k$, we refer to the rank-1 matrix formed by the vector pair $\langle \mathbf{B}^l, \mathbf{C}_l \rangle$ as a *block*. If $\mathbf{X}$ and $\mathbf{Y}$ are n-by-m binary matrices, we use $\mathbf{X} \oplus \mathbf{Y}$ to denote their *element-wise exclusive or*. Finally, we denote by $L(\mathbf{A}, \mathbf{B}, \mathbf{C})$ the description length of $\mathbf{A}$ for factor matrices $\mathbf{B}$ and $\mathbf{C}$.

3 BMF with MDL

3.1 MDL: a brief introduction. The Minimum Description Length principle (MDL) [11] is a practical version of Kolmogorov Complexity. Both embrace the slogan *Induction by Compression*. This can be roughly described as follows: Given a set of models $\mathscr{M}$, the best model $M \in \mathscr{M}$ is the one that minimizes $L(M) + L(D \mid M)$, in which $L(M)$ is the length in bits of the description of M, and $L(D \mid M)$ is the length of the data when encoded with model M.

This is called two-part, or *crude*, MDL – as opposed to *refined* MDL, where model and data are encoded together [11]. We use two-part MDL because we are specifically interested in the model: the factors that give the best description of the data. Note that MDL requires the compression to be *lossless* in order to allow for fair comparison between different $M \in \mathscr{M}$, and that we are only concerned with code lengths, not actual code words.

3.2 BMF: a brief introduction. In Boolean matrix factorization, the goal is to (approximately) represent a Boolean matrix as the Boolean product of two Boolean matrices. That is, given Boolean matrix $\mathbf{A}$, find Boolean matrices $\mathbf{B}$ and $\mathbf{C}$ such that $\mathbf{A} \approx \mathbf{B} \circ \mathbf{C}$. In *minimum-error Boolean rank-k* decomposition one is given the n-by-m input matrix $\mathbf{A}$ and rank k, and the goal is to find n-by-k and k-by-m factor matrices $\mathbf{B}$ and $\mathbf{C}$ that minimize $|\mathbf{A} \oplus (\mathbf{B} \circ \mathbf{C})|$.

Using the Boolean decomposition instead of a normal (or non-negative) can give many advantages in data mining: the binary factor matrices are often easier to interpret [18, 20], or the binary factors might be required in the application (e.g. [14]), the Boolean factors can provide better input for subsequent algorithms (e.g. [1, 31]), and for sparse input data, the factors will be naturally sparse [19].

Unfortunately, computing the least-error BMF is NP-hard, and has strong inapproximability results [18]. But there is also another, more fundamental problem: the formulation of the minimum-error Boolean rank-k problem requires *a priori* knowledge on k, the Boolean rank of the decomposition. And as discussed in the Introduction, using the Hamming distance essentially assumes roughly equal distribution between destructive and additive noise – something that we hypothesize is not true. In order to solve these two problems, we will use the description length instead of the reconstruction error to measure the quality of our decomposition.

3.3 MDL for BMF and problem definition. To use MDL, we have to define what our models $\mathscr{M}$ are, how an $M \in \mathscr{M}$ describes a database, and how we encode these in bits. Miettinen and Vreeken [22] proposed a number of MDL objective functions for BMF. Here we use their so-called *typed data-to-model* encoding, which was shown to be both the most efficient as well as providing the best empirical performance. Below we give the main ideas of this encoding scheme. For further details we refer the reader to [22].

The description length of a Boolean n-by-m matrix $\mathbf{A}$ factorized into $\mathbf{B}$ and $\mathbf{C}$, such that $\mathbf{A} \approx \mathbf{B} \circ \mathbf{C}$, is defined as

$$L(\mathbf{A}, \mathbf{B}, \mathbf{C}) = \underbrace{L(\mathbf{B}) + L(\mathbf{C})}_{L(M)} + \underbrace{L(\mathbf{A} \mid \mathbf{B}, \mathbf{C})}_{L(D|M)} \quad , \tag{3.1}$$

where $L(\mathbf{B}) + L(\mathbf{C})$ are the description lengths of the factor matrices and $L(\mathbf{A} \mid \mathbf{B}, \mathbf{C})$ is the description length of the matrix $\mathbf{A}$ given this model. The columns of $\mathbf{B}$, and analogously the rows of $\mathbf{C}$, are encoded independently as binary vectors. One such vector can be identified by two integers: one encoding the number of nonzero elements in $\mathbf{B}^i$ (maximum n), and the other encoding the index of $\mathbf{B}^i$ among all binary strings having the same profile (maximum $\binom{n}{|\mathbf{B}^i|}$). The number of bits for encoding the k columns of $\mathbf{B}$ is hence

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

To reconstruct the data $\mathbf{A}$ given the factor matrices $\mathbf{B}$ and $\mathbf{C}$ means we need to describe the error of this factorization. We hence have to encode the exclusive OR of the data with the model, $\mathbf{E} = \mathbf{A} \oplus (\mathbf{B} \circ \mathbf{C})$. We can split $\mathbf{E}$ into unmodelled 1s, $\mathbf{E}^+$, and superfluous 1s, $\mathbf{E}^-$, where $\mathbf{E}^+_{ij} = 1$ if and only if $\mathbf{A}_{ij} > (\mathbf{B} \circ \mathbf{C})_{ij}$ and $\mathbf{E}^-_{ij} = 1$ if and only if $\mathbf{A}_{ij} < (\mathbf{B} \circ \mathbf{C})_{ij}$. To avoid rewarding structure in the error matrices $\mathbf{E}^+$ and $\mathbf{E}^-$ are encoded as binary strings in the same way as a column of $\mathbf{B}$. Combined, we have $L(\mathbf{A} \mid \mathbf{B}, \mathbf{C}) = L(\mathbf{E}^+) + L(\mathbf{E}^-)$, where

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

and

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

This concludes the definition of our MDL objective function. We can now define the problem we aim to solve in this paper.

PROBLEM 3.1. (MDLBMF) *Given a binary n-by-m matrix* **A**, *the Minimum Description Length Boolean Matrix Factorization (*MDLBMF*) problem is to find binary factor matrices* **B** *and* **C** *such that we minimize the total description length*

$$L(\mathbf{A},\mathbf{B},\mathbf{C}) \, . \tag{3.2}$$

Note that we do not require the rank of the decomposition as an input; instead we automatically find the rank (and decomposition) that minimizes the description length. Although the computational complexity of the MDLBMF problem is unknown, there are strong reasons to believe it is NP-hard [22]. Hence, we resort to heuristics.

4 Related Work

In BMF one decomposes a binary matrix into the Boolean product of two matrices while minimizing some cost function. Perhaps the intuitively most straightforward objective is to minimize the number of errors, i.e. the Frobenius norm of the residual. The Asso algorithm to solve BMF was proposed by [20]. Later, [13] proposed a heuristic based on a mixed-integer-programming formulation.

Error minimization is prone to overfitting, however, as more factors always allow better reconstruction. In practice, users thus have to choose the number of factors in advance. Moreover, this objective builds on the assumption that noise is equally likely to flip true 1s to 0s as it is to flip true 0s to 1s, which we argue here is not realistic.

As discussed by Faloutsos and Megalooikonomou [6], Kolmogorov Complexity, and its practical implementation, the Minimum Description Length principle [11, 26], are powerful, well-founded, and natural approaches to data mining, as they allow us to identify the most succinct and least redundant model for a dataset. MDL has been successfully employed for a wide range of data mining tasks, including discretization [7], outlier detection [28], classification [25], and clustering [17].

Miettinen and Vreeken [21, 22] recently formulated the BMF problem in terms of the Minimum Description Length (MDL) principle [11, 26] in order to solve the model order selection problem. Loosely speaking, this objective identifies the best factorization as the one that provides the best lossless compression of the data – thus automatically balancing the complexity of the factorization with the reconstruction error. The authors applied their score as post-processing for Asso to identify the optimal model order. In this paper we propose an algorithm that directly optimizes the description length, taking advantage of the fact that this objective can naturally handle different amounts of noise per type.

Tiling [10] is closely related to BMF, but aims at finding submatrices full of 1s. Xiang et al. [32] proposed an algorithm to mine noisy tiles – i.e. allowing 0s in the area covered by a tile. Given a collection of noisy tiles, Kontonasios and De Bie [12] iteratively discover the most interesting tile, defining interestingness through a local MDL score. Tatti and Vreeken [29] use MDL to hierarchically identify the most informative tile from data with ordered rows and columns.

Closer to our work is the Panda algorithm proposed by Lucchese et al. [15]. Panda performs Boolean matrix factorization, but instead of minimizing only the error, it minimizes the sum of Frobenius norms of the residual and the factor – by which the hidden assumption again is that false positives and false negatives are equally likely. Recently, the same authors proposed Panda+ [16] in which they optimize the TypedXOR encoding of Miettinen and Vreeken [21]. Panda+ will be the main competitor to Nassau.

Algorithm 1 Nassau

```
Input: $\mathbf{A} \in \{0,1\}^{n\times m}, t, \tau, \theta \in (0,1), M > 0$
Output: $\mathbf{B} \in \{0,1\}^{n\times k}$ and $\mathbf{C} \in \{0,1\}^{k\times m}$
 1: function Nassau($\mathbf{A}, t, \tau, \theta, M$)
 2:     $\mathbf{B}' \leftarrow 0^{n\times 0}$, $\mathbf{C}' \leftarrow 0^{0\times m}$
 3:     $Seeds \leftarrow$ GetSeeds($\mathbf{A}$)
 4:     repeat
 5:         $\mathbf{B} \leftarrow \mathbf{B}'$, $\mathbf{C} \leftarrow \mathbf{C}'$
 6:         $[\mathbf{b},\mathbf{c}] \leftarrow$ FindBlock($\mathbf{A},\mathbf{B},\mathbf{C}, Seeds$)
 7:         $\mathbf{B}' \leftarrow [\mathbf{B},\mathbf{b}]$, $\mathbf{C}' \leftarrow \left[\begin{smallmatrix}\mathbf{C}\\ \mathbf{c}\end{smallmatrix}\right]$
 8:         if $M$ rounds since last update then
 9:             $[\mathbf{B}',\mathbf{C}'] \leftarrow$ CyclUpd($\mathbf{A},\mathbf{B}',\mathbf{C}', Seeds, 0, \theta$)
10:     until $L(\mathbf{A},\mathbf{B}',\mathbf{C}') \geq L(\mathbf{A},\mathbf{B},\mathbf{C})$
11:     while not converged do
12:         $[\mathbf{B},\mathbf{C}] \leftarrow$ CyclUpd($\mathbf{A},\mathbf{B},\mathbf{C}, Seeds, t, \theta$)
13:         $t \leftarrow t \cdot \tau$
14:     return $\mathbf{B}$, $\mathbf{C}$
```

5 Algorithm

In this section we present Nassau, a new algorithm for heuristically solving the MDLBMF problem. The existing algorithms, Panda+ and Asso, never change an already-found factor. This property simplifies the search for the (locally) MDL-minimizing factorization [22], but has a drawback. The first few blocks blocks probably cover large parts of the data, but for higher-rank decompositions, these initial blocks can become too coarse-grained. Nassau, on the other hand, does iteratively refine previously discovered factors.

5.1 The main algorithm. Nassau (Algorithm 1) interleaves the addition of new blocks and updating of already-found ones. As new blocks add more fine-grained structure, the coarser older factors become obsolete and can be replaced.

The algorithm starts by finding a set of *seeds* that provide the starting point for finding the factorization (Line 3; see Section 5.2). After initialization, Nassau starts its first phase (Lines 4–10), where it repeatedly adds a new block to the existing factorization until the description length does not improve any more. The blocks are found using the FindBlock routine (Algorithm 2, see Section 5.3). To adjust the already-found factors, Nassau regularly calls CyclUpd (Algorithm 3, Section 5.4).

When additional factors do not decrease the objective further, we enter the last phase of the algorithm (Lines 11–13), which entails refinement of the discovered blocks. This step is explained in Section 5.4.

Nassau accepts several parameters that control its execution. Parameters t and τ control the simulated annealing by giving the initial temperature and update ratio, respectively, while M controls the frequency at which we update the found factors. Hence, at the expense of run time, higher t and lower τ and M potentially provide better results. Parameter θ controls the mining process and is explained in Section 5.3.

5.2 Finding seed columns. Finding a good block to add is a hard problem. Hence, we take an approach similar to [16, 20], and start by finding a good collection of *seed columns*. The seed column vectors are collected in the n-by-s matrix *Seeds*. Later, the FindBlock algorithm will use these seeds to build the final blocks. In principle, we can use any method to create these seeds, including those used in [16, 20], but here we present our approach, which is based on restarted random walks.

A good seed captures the correlation between data rows. Consider an n-by-m binary matrix $\mathbf{A}$ and the corresponding bipartite graph $G = (V_{\text{rows}} \cup V_{\text{cols}}, E)$. The correlated rows of $\mathbf{A}$ correspond to the highly interconnected nodes in V_{rows}, and if two nodes are correlated, then a short random walk starting from one of them is likely to frequently visit the other [27]. If we restart each walk from the same node frequently, the fraction of time we spend on each node indicates how related it is to the origin of the walk.

Let $P(i \mid j)$ be the probability of reaching node i from node j. We do one random walk for each node on the left part of the graph, and on every step we have a fixed probability ε to return back to the starting node. Otherwise, we go to one of the neighbouring nodes, selecting them uniformly at random. Now, the fraction of time we spend in node i in a random walk starting from node v is

$$\Pi_v(i) = \varepsilon \cdot \mathbf{1}(v = i) + (1 - \varepsilon) \sum_{(i,j) \in E} \Pi_v(j) P(i|j) \ .$$

The above equation can be solved using the dominant eigenvector, similar to [27]. We generate one seed for each data row v, and the seed has 1 on every row where the random walk starting from v spends sufficient time.

5.3 Finding the blocks. Given the input matrix and the current factorization, FindBlock (Algorithm 2) finds a new block to be added to the factorization.

Algorithm 2 FindBlock

```
Input: matrices $\mathbf{A} \in \{0,1\}^{n \times m}$, $\mathbf{B} \in \{0,1\}^{n \times k}$, $\mathbf{C} \in \{0,1\}^{k \times m}$,
    $Seeds \in \{0,1\}^{n \times s}$, $\theta \in (0,1)$
Output: block $\langle \mathbf{b}^*, \mathbf{c}^* \rangle$
 1: function FindBlock($\mathbf{A}, \mathbf{B}, \mathbf{C}, Seeds, \theta$)
 2:     $\mathbf{b}^* \leftarrow 0^n, \mathbf{c}^* \leftarrow 0^m$
 3:     for $i = 1$ to $s$ do
 4:         $\mathbf{b} \leftarrow Seeds^i$                    ▷ The $i$th seed column.
 5:         repeat
 6:             $\mathbf{c} \leftarrow \arg\max_{\mathbf{c} \in \{0,1\}^m} \texttt{cover}(\mathbf{A}, [\mathbf{B}, \mathbf{b}], \left[\begin{smallmatrix}\mathbf{C}\\ \mathbf{c}\end{smallmatrix}\right], \theta)$
 7:             $\mathbf{b} \leftarrow \arg\max_{\mathbf{b} \in \{0,1\}^n} \texttt{cover}(\mathbf{A}, [\mathbf{B}, \mathbf{b}], \left[\begin{smallmatrix}\mathbf{C}\\ \mathbf{c}\end{smallmatrix}\right], \theta)$
 8:         until stopping criteria are satisfied
 9:         if $L(\mathbf{A}, [\mathbf{B}, \mathbf{b}], \left[\begin{smallmatrix}\mathbf{C}\\ \mathbf{c}\end{smallmatrix}\right]) < L(\mathbf{A}, [\mathbf{B}, \mathbf{b}^*], \left[\begin{smallmatrix}\mathbf{C}\\ \mathbf{c}^*\end{smallmatrix}\right])$ then
10:             $\mathbf{b}^* \leftarrow \mathbf{b}, \mathbf{c}^* \leftarrow \mathbf{c}$
11:     return $\mathbf{b}^*, \mathbf{c}^*$
```

The algorithm tries every seed one by one, and uses alternating updates. To compute the updates, we ignore all the locations where the existing factorization has a non-zero, as we cannot change that value. Then, given a fixed column factor $\mathbf{b}$, we build the corresponding row factor $\mathbf{c}$ by testing, for every data column, if using $\mathbf{b}$ to cover that column would improve the cover function:

$$\begin{aligned}\texttt{cover}(\mathbf{A}, \mathbf{B}, \mathbf{C}, \theta) = \theta \left|\{(i,j) : a_{ij} = 1 \wedge (\mathbf{B} \circ \mathbf{C})_{ij} = 1\}\right| \\ - \left|\{(i,j) : a_{ij} = 0 \wedge (\mathbf{B} \circ \mathbf{C})_{ij} = 1\}\right| \ .\end{aligned}$$

We use the cover function as a surrogate to the full description length, partially to make this update step faster, and partially to avoid some local optima. Further local optima can be avoided by adjusting parameter θ; setting $\theta = 1$ returns locally optimal blocks, but we can introduce some locally nonoptimal decisions (that can yield to better global behaviour) by setting θ to a slightly larger (or smaller) value.

5.4 Updating the factors. As was mentioned above, blocks found earlier might become outdated – that is, they become less useful in covering data, or they can even become redundant. This happens if we have added new blocks that overlap with previous ones and cover the same data better. In these cases we can improve the factorization by updating the old blocks. CyclUpd (Algorithm 3) iteratively replaces blocks with better alternatives. This is performed by first removing a block, and then finding a replacement by calling FindBlock with the current factors (Line 4).

Note that the objective is not strictly decreasing: FindBlock is heuristic and is not guaranteed to improve the result. However, we might still want to keep the update to avoid getting stuck in a local minimum. In order to accomplish this, we apply a technique similar to simulated annealing. We use a temperature parameter t, which controls the probability of accepting a new block that does not improve the score. If a new block decreases the objective, we always accept it; otherwise we will only do so with a probability

Algorithm 3 `CyclUpd`

```
Input: matrices $\mathbf{A} \in \{0,1\}^{n\times m}$, $\mathbf{B} \in \{0,1\}^{n\times k}$, $\mathbf{C} \in \{0,1\}^{k\times m}$,
    $Seeds \in \{0,1\}^{n\times s}$, $t > 0$, $0 < \theta < 1$
Output: Factors $\mathbf{B}^* \in \{0,1\}^{n\times k}$ and $\mathbf{C}^* \in \{0,1\}^{k\times m}$
1: function CyclUpd($\mathbf{A}, \mathbf{B}, \mathbf{C}, Seeds, t, \theta$)
2:     $\mathbf{B}^* \leftarrow \mathbf{B}, \mathbf{C}^* \leftarrow \mathbf{C}$
3:     for $l = 1$ to $k$ do
4:         $[\mathbf{b}, \mathbf{c}] \leftarrow$ FindBlock($\mathbf{A}, \mathbf{B}^{-l}, \mathbf{C}_{-l}, Seeds, \theta$)
5:         $\mathbf{B}' \leftarrow [\mathbf{B}^{-l}, \mathbf{b}], \mathbf{C}' \leftarrow \left[\begin{smallmatrix}\mathbf{C}_{-l}\\ \mathbf{c}\end{smallmatrix}\right]$        ▷ Replace block
6:         if $L(\mathbf{A}, \mathbf{B}', \mathbf{C}') < L(\mathbf{A}, \mathbf{B}, \mathbf{C})$ then
7:             $\mathbf{B} \leftarrow \mathbf{B}', \mathbf{C} \leftarrow \mathbf{C}'$
8:         else
9:             $\mathbf{B} \leftarrow \mathbf{B}', \mathbf{C} \leftarrow \mathbf{C}'$ with probability $t$
10:        if $L(\mathbf{A}, \mathbf{B}', \mathbf{C}') < L(\mathbf{A}, \mathbf{B}^*, \mathbf{C}^*)$ then
11:            $\mathbf{B}^* \leftarrow \mathbf{B}, \mathbf{C}^* \leftarrow \mathbf{C}$
12:    return $\mathbf{B}^*, \mathbf{C}^*$
```

proportional to t. Notice that `Nassau` calls `CyclUpd` with non-zero t only in its last simulated annealing phase.

`Nassau` does most of the work inside `CyclUpd`, which is called every M iterations before all blocks are found (Line 9), and then once again when the number of blocks is fixed (Line 12), making a total of $k/M + 1$ calls. `CyclUpd` in turn calls `FindBlock` $O(k)$ times. `FindBlock` tries every seed vector, and for each one builds (approximately) a corresponding cover in time $O(nm)$. Given that we have n seeds, this yields the complexity of $O(kn^2m)/M$ for `CyclUpd`. Finding the seed columns is done using restarted random walks [27], which takes time $O(n(n+m))$. Thus the complexity of `Nassau` is $O(k/M)O(kn^2m) + O(n(n+m)) = O(k^2n^2m/M)$ (assuming $M < k$). It is worth noticing that the number of blocks k is rather small and rarely exceeds 100, which, combined with a relatively small constant hidden behind the asymptotics, makes the algorithm a practical choice for most of the real-world applications.

6 Experiments

In this section we empirically evaluate `Nassau` and compare its performance to that of two competing algorithms.

6.1 Algorithms and parameters. We used three algorithms, `Nassau`,[1] `Panda+` [16], and `Asso` [20]. For `Asso`, we followed the approach of [22] to compute the MDL-minimizing factorization. These algorithms take various parameters. For `Nassau` we used the same parameters in all experiments, and we set the initial annealing temperature $t = 0.8$, temperature scaling $\tau = 0.6$, weight $\theta = 1.1$, and launched the cyclic updates in every $M = 5$ rounds. `Asso` takes one parameter, the rounding threshold τ. For synthetic experiments, we tried values from 0 to 1 at increments of 0.1; for real-world experiments, we followed the setup of [22], and tried the values from 0.1 to 0.95 with increments of 0.025. In all cases, we selected the value of τ that yields the minimal MDL cost factorization. For `Panda+`, we minimize the description length (i.e. the J_E error function) and use randomization with 10 re-starts. Other parameters were left at their default values.

`Panda+` optimizes an encoding that differs slightly to the one we focus on. For fair comparison we thus ran a set of baseline experiments with both encodings. This, however, did not give results that were different in any significant way, and hence we do report these results separately – the fact that the encoding we use gives very similar results to the encoding used by `Panda+` was also noted in [22].

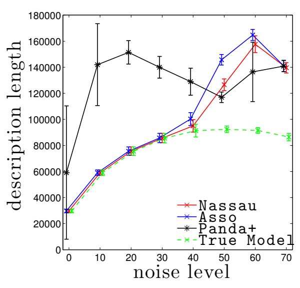
6.2 Synthetic data. We first use synthetic data to test the algorithms on data of known ground truth and characteristics. To that end, we generated 1000-by-800 matrices by first creating two binary factor matrices $\mathbf{B}^*$ and $\mathbf{C}^*$, computing their product $\mathbf{A} = \mathbf{B}^* \circ \mathbf{C}^*$, and applying noise to $\mathbf{A}$ in order to obtain the final input matrix $\tilde{\mathbf{A}}$. In addition to the results from the algorithms, we also report the results obtained with the original factor matrices $\mathbf{B}^*$ and $\mathbf{C}^*$. We call this the `True Model`. The amount of noise is measured in percentages of non-zeros in $\mathbf{A}$. That is, p% of destructive noise flips an expected $p\,|A|\,/100$ of the 1s of $\mathbf{A}$, while p% of additive noise flips an expected, $p\,|A|\,/100$ of the 0s.

When generating the matrices, we varied one parameter and kept the rest fixed. In particular, we evaluated different levels of additive and destructive noise, density of the matrix, and rank (model order). All results are averaged over 10 instances per configuration. When not varied, the rank was kept at 15, the density at 8%, and additive noise at 3%. To evaluate how well the algorithms perform with increasing amounts of destructive noise, we ran two sets of experiments, one with 15% destructive noise and one with 50%.

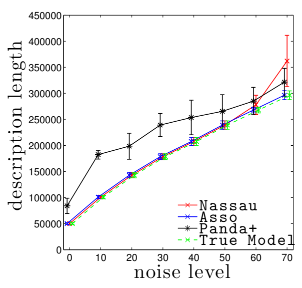
Destructive noise. We start by testing the effects of the destructive noise, shown in Figure 1a. Initially `Nassau` and `Asso` are indistinguishable from the true model, while `Panda+` has much higher description length. At around 40% destructive noise, however, `Nassau` starts to perform better than `Asso`, and continues in that way until the end. Meanwhile, we observe that while `Panda+` under-performs overall, it does relatively well when the data becomes very sparse (high levels of destructive noise). This may mean its search procedure is well-equipped to find blocks in sparse data.

Additive noise. The purpose of this test is to find how robust the algorithms are to additive noise, that is when 0s are turned to 1s. Keeping the destructive noise constant at 15%, we find that up to 60% additive noise both `Nassau` and `Asso` still find models virtually indistinguishable from the

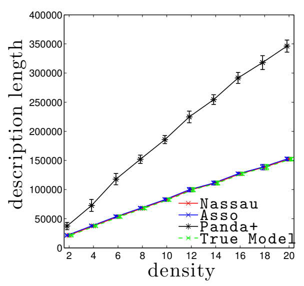
[1]The `Nassau` code is available for research purposes at `http://people.mpi-inf.mpg.de/~pmiettin/nassau/`; for the other two algorithms, we used code provided by the authors.



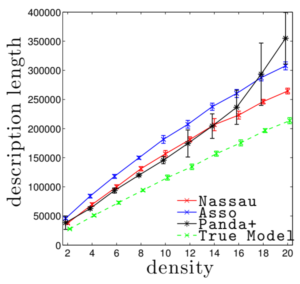
(a) Varying destructive noise.



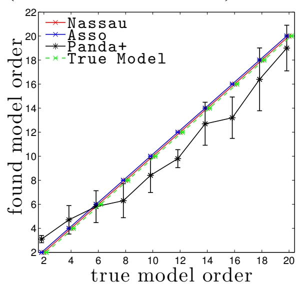
(b) Varying additive noise (15% destructive noise).



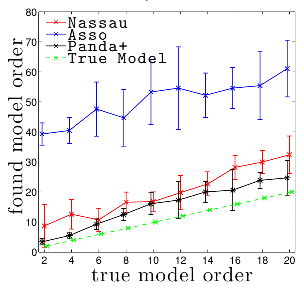
(c) Varying density (15% destructive noise).



(d) Varying density (50% destructive noise).



(e) True and estimated model order (15% destructive noise).



(f) True and estimated model order (50% destructive noise).

Figure 1: **Results on synthetic data.** Markers are mean values over 10 repetitions, error bars are twice the standard deviation. The noise level (both additive and destructive) is relative to the number of 1s.

true model, while Panda+ is consistently worse except at 70% noise (Figure 1b).

Varying density. We varied the density of the noise-free matrices from 2% to 20%. At 15% destructive noise (Fig. 1c), Nassau and Asso discover models on par with the true model, while Panda+ is clearly much worse. With 50% destructive noise (Fig. 1d), Nassau and Panda+ are virtually indistinguishable until the density reaches 16%, after which the performance of Panda+ starts to degrade.

Model order. Last, we varied the model order (rank) between 2 and 20. With 15% destructive noise, Nassau and Asso obtained exact results, while Panda+ was mostly under-estimating (Fig. 1e). With 50% destructive noise (Fig. 1f), Nassau over-estimates slightly more than Panda+, although for both methods we see occasional high variance.

6.3 Real-world datasets.

Next, we consider 10 real-world datasets, most of which are publicly available. Below we give a short description for each, and an overview in Table 1.

4News is a subset of the 20Newsgroups data, containing the usage of 800 words over 400 posts for 4 newsgroups.[2]

Abstracts represents the words in all abstracts of the papers accepted at the ICDM conference up to 2007, where the words have been stemmed and stop words removed [2].[3]

DBLP conf. contains records of which of the 19 conferences the 6980 authors had published in. The dataset is collected from the DBLP database and it is pre-processed as in [18].[4] *DBLP co-auth.* is a (symmetric) co-authorship matrix of a subset of the authors in the *DBLP conf.* data.

Dialect contains presence data of dialectical linguistic properties in 506 Finnish municipalities [3, 4].

DNA Amp. contains information on DNA copy number amplifications. Such copies activate oncogenes and are hallmarks of nearly all advanced tumours [24]. Amplified genes represent attractive targets for therapy and prognostics.

Firewall 2 describes the reachability between two IP addresses [5].[5] It has an exact Boolean decomposition of 10 factors [5], and hence should be highly compressible.

Mammals consists of presence records of European mammals within areas of 50 by 50 kilometers [23].[6]

Mushroom contains edibility records of mushrooms [9].

Lastly, *Paleo* consists of fossil records per location.[7]

6.4 Quantitative evaluation of real-world results.

We start by studying the compression ratios of the real-world data sets. By compression ratio, we mean the description

[2]The authors are grateful to Ata Kabán for pre-processing the data [18].

[3]Available upon request from the author [2]

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

[5]`http://www.hpl.hp.com/personal/Robert_Schreiber/`

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

[7]NOW 030717, `http://www.helsinki.fi/science/now/` [8].

Table 1: Real-world dataset overview. $L_\emptyset$ is the description length in bits with the empty model (without any factors).

Dataset	Rows	Columns	%1s	$L_\emptyset$
4News	400	800	3.5	70379
Abstracts	859	3933	1.2	319468
DBLP co-auth.	2345	2345	0.5	244754
DBLP conf.	6980	19	13.0	73785
Dialect	1334	506	16.1	430435
DNA Amp.	4590	392	1.5	199429
Firewall 2	325	590	19.0	134546
Mammals	2670	194	16.1	330302
Mushroom	8192	112	19.3	650373
Paleo	501	139	5.1	20223

Table 2: Compression ratio $L\% = 100 \times L/L_\emptyset$ (smaller is better) and model order k for the real-world datasets.

	`Nassau`		`Panda+`		`Asso`	
Dataset	$L\%$	k	$L\%$	k	$L\%$	k
4News	93.1	12	**92.7**	5	93.6	17
Abstracts	95.3	3	**86.7**	128	97.2	19
DBLP conf.	90.3	3	92.4	3	**90.0**	4
DBLP co-auth.	**94.1**	60	95.9	11	95.8	178
Dialect	**42.0**	30	57.3	17	48.8	37
DNA Amp.	**43.6**	100	63.4	20	49.8	58
Firewall 2	2.4	6	2.7	8	**1.7**	5
Mammals	**54.5**	29	66.8	8	64.6	17
Mushroom	72.6	4	63.6	15	**50.6**	59
Paleo	**89.7**	15	91.2	3	91.4	19

length L obtained by the algorithm divided by the description length of the data using an empty model, i.e. when no factors are used to represent it, $L_\emptyset$. The smaller this ratio is, the better the discovered factorization compresses the data. The results are presented in Table 2.

Out of the ten datasets in Table 2, `Nassau` obtains the best compression ratio in five – and in each of these cases it outperforms the second-best result by at least one percentage point. `Panda+` is the best for the two sparse text datasets, *4News* and *Abstracts*, although for *4News* `Nassau` is only 0.4 and `Asso` only 0.9 percentage points worse. For all practical purposes, we would consider *4News* a tie. Similarly, `Asso` is better in *DBLP conf.*, but only by 0.3 percentage points, more clearly in *Firewall 2*, and by a wide margin in *Mushroom*.

If we consider the two cases where `Nassau` is significantly worse than the best method, *Abstracts* (against `Panda+`) and *Mushroom* (against `Asso`), we notice that in both cases `Nassau` reports significantly smaller model order than the best method (3 vs. 128 and 4 vs. 59, respectively). It seems that in these cases the random walks were unable to find good seed vectors. Indeed, when we re-ran `Nassau` on *Mushroom* but generating the seeds as `Asso` does, we obtained a compression ratio of 56.4 with 44 factors – a significant improvement confirming that the random walks do not necessarily produce good seeds for every data set.

Scalability. The implementations of these methods are not fully comparable. `Panda+` is a full-C implementation, `Asso` is a mixture of C and Matlab, while `Nassau` is written purely in Matlab. Also, the different methods use different levels of parallelization, and the running time for all these algorithms is heavily influenced by the eventual model order. For `Asso`, the number of different rounding thresholds τ to try, and for `Panda+`, the number of random re-starts, also have obvious effects. That said, in our experiments, `Panda` was generally the fastest, followed by `Nassau` and `Asso`, but the order could vary from dataset to dataset. All methods were able to finish all datasets within 24 hours, except for `Asso` with *DBLP co-auth.*, which took more than four days.

6.5 Qualitative evaluation of real-world results. We studied the results on two datasets, *DBLP co-auth.* and *Mammals*, more carefully in order to understand the types of results we can obtain from `Nassau` (and its competitors). As Table 2 shows, *DBLP co-auth.* is not very compressible, while *Mammals* is reduced to almost half of its original size.

DBLP co-auth. The *DBLP co-auth.* data is a graph, i.e. a symmetric matrix, and while none of the methods tested here assume symmetry, we would expect that a good factorization would be (approximately) symmetric. This, however, is not the case for `Panda+` or `Asso`: in symmetric decompositions, the two factor matrices would be the same, but for `Asso`, 1.2% of the values differ, and for `Panda+`, 1.6% of the values in the factor matrices differ. For `Nassau`, only 0.3% of the values are different.

Looking at the factors, we observed that in particular `Asso` finds very skewed blocks that have many rows but few columns, or vice versa, being clearly unable to capitalize on the symmetry of the matrix. `Panda+` returns more square-like blocks, but they still have almost four times as many rows as columns (or vice versa) on average. The blocks `Nassau` returns have an average rows/columns ratio of 1.7, giving the most square-like blocks.

We show two exemplar factors found by `Nassau` in the *DBLP co-auth.* data. It shows the known collaborations between the authors in a factor. Here, `Nassau` has identified two (quasi-) cliques of famous data mining and machine learning researchers with strong collaboration patterns.

Mammals. Both `Panda+` and `Asso` perform so-called *hierarchical* factorization (rank-$(k-1)$ factorization is part of the rank-k factorization), while `Nassau` is designed to avoid that. We studied these different behaviours with the *Mammals* data and using the first four left-hand-side factors (i.e. columns of $\mathbf{B}$) for each method. As the rows of the data correspond to locations in Europe, we visualize the factors

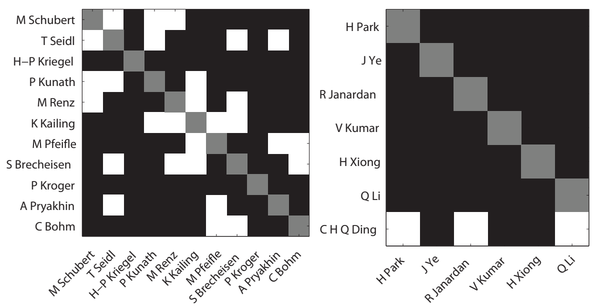


Figure 2: **We find intuitive quasi-cliques.** Example submatrices of *DBLP co-auth.* as selected by `Nassau`. Black cells indicate joint papers, grey cells stand for self-loops.

using maps in Figure 3. For `Nassau` the first four factors are very compact and correspond very closely to countries in Europe. For `Panda+`, the first factor covers almost the whole area of Europe that was part of the data, and the other three factors also cover large, not very well-defined areas. The same holds true for `Asso`, although to a slightly lesser extent. In other words, the results by `Nassau` in Figure 3 correspond to species that are very specific to certain areas, while both `Panda+` and `Asso` selected common species.

7 Discussion

The experiments show that `Nassau` performs well in practice. The evaluation on synthetic data shows it is resistant to high amounts of destructive noise, reliably discovering factorizations of the correct model order and of complexities close to the ground truth. For real-world data `Nassau` discovered the most succinct descriptions for five out of ten datasets, while being a close second for three more. Manual inspection showed that the factors `Nassau` finds are concise, clean, and more interpretable than those of `Asso` and `Panda+`. Overall, the experiments permit two conclusions: 1) `Nassau` performs very well when the data exhibits large amounts of destructive noise, and 2) our hypothesis that real-world data contains large amounts of destructive noise is warranted.

We see two main avenues for further research. First of all, the seed generation process of `Nassau` can be improved. Preliminary experiments suggest that by *adding* the seeds that `Asso` considers to `Nassau`, even better factorizations can be discovered – alternatively, it will be interesting to see if in addition the seed generation process of `Panda+` can be incorporated. The second line of research is optimization, of both the process and the implementation.

8 Conclusions

We studied the problem of Boolean matrix factorization under the assumption that in general it is more likely not to have observed the value of a cell than to have done so – and that the reliability of both observations differs. We formalized this problem by the Minimum Description Length principle, and proposed `Nassau`, a new algorithm that directly optimizes the description length of the factorization. `Nassau` is non-hierarchical and is capable of fixing errors it has previously made. Empirical evaluation on synthetic data shows that `Nassau` is very robust to high levels of destructive noise. Experiments on real-world data show it finds more compact and intuitive factors than the state of the art, which are more inclined to 'simply' cover large parts of the data.

Acknowledgements

S. Karaev is funded by the International Max Planck Research School for Computer Science and by the Saarbrücken Graduate School of Computer Science.

J. Vreeken is supported by the Cluster of Excellence "Multimodal Computing and Interaction" within the Excellence Initiative of the German Federal Government.

References

[1] E. Cergani and P. Miettinen. Discovering relations using matrix factorization methods. In *CIKM*, pages 1549–1552, 2013.

[2] T. De Bie. Maximum entropy models and subjective interestingness: An application to tiles in binary databases. *Data Min. Knowl. Disc.*, 23(3):407–446, 2011.

[3] S. M. Embleton and E. S. Wheeler. Finnish dialect atlas for quantitative studies. *J. Quant. Ling*, 4(1–3):99–102, 1997.

[4] S. M. Embleton and E. S. Wheeler. Computerized dialect atlas of Finnish: Dealing with ambiguity. *J. Quant. Ling*, 7(3):227–231, 2000.

[5] A. Ene, W. Horne, N. Milosavljevic, P. Rao, R. Schreiber, and R. E. Tarjan. Fast exact and heuristic methods for role minimization problems. In *SACMAT*, pages 1–10, 2008.

[6] C. Faloutsos and V. Megalooikonomou. On data mining, compression and Kolmogorov complexity. *Data Min. Knowl. Disc.*, 15(1):3–20, 2007.

[7] U. Fayyad and K. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. In *UAI*, pages 1022–1027, 1993.

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

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

[10] F. Geerts, B. Goethals, and T. Mielikäinen. Tiling databases. In *DS*, pages 278–289, 2004.

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

[12] K.-N. Kontonasios and T. De Bie. An information-theoretic approach to finding noisy tiles in binary databases. In *SDM*, pages 153–164. SIAM, 2010.

[13] H. Lu, J. Vaidya, and V. Atluri. Optimal Boolean matrix decomposition: Application to role engineering. In *ICDE*, pages 297–306, 2008.

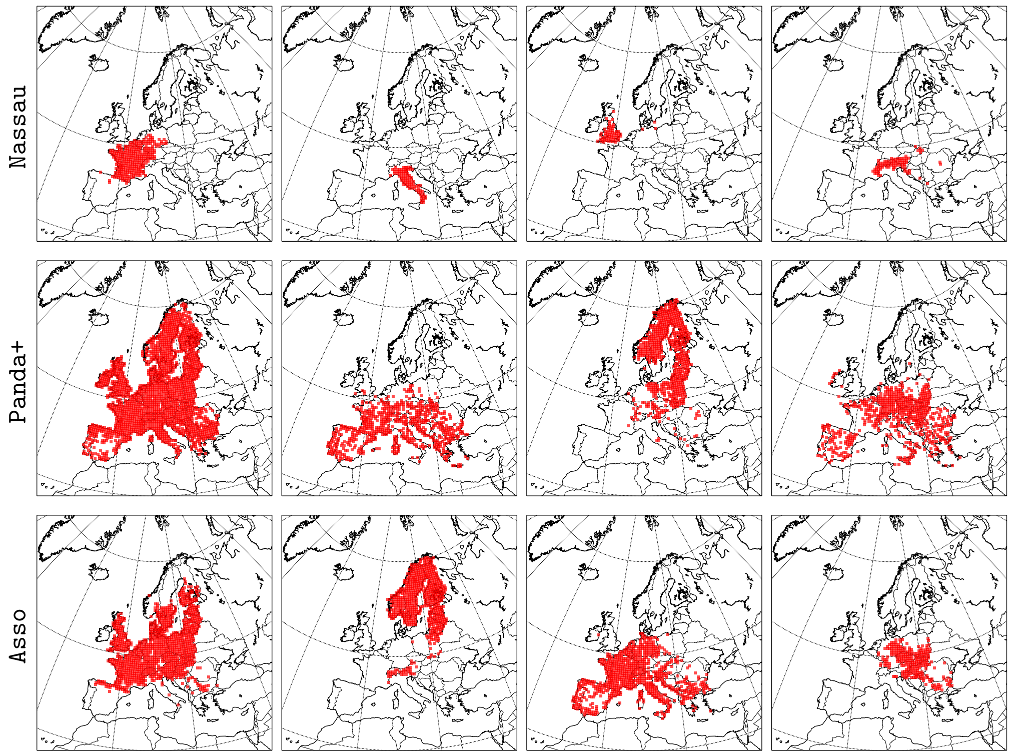


Figure 3: **We discover compact descriptions** The first four factors discovered on the *Mammals* data using, from top to bottom, `Nassau`, `Panda+`, and `Asso`.

[14] H. Lu, J. Vaidya, V. Atluri, and Y. Hong. Constraint-aware role mining via extended Boolean matrix decomposition. *IEEE Trans. Depend. Secure*, 9(5):655–669, 2012.

[15] C. Lucchese, S. Orlando, and R. Perego. Mining top-k patterns from binary datasets in presence of noise. In *SDM*, page 165–176, 2010.

[16] C. Lucchese, S. Orlando, and R. Perego. A unifying framework for mining approximate top-k binary patterns. *IEEE Trans. Knowl. Data En.*, 26(12):2900–2913, 2014.

[17] M. Mampaey and J. Vreeken. Summarising categorical data by clustering attributes. *Data Min. Knowl. Disc.*, 26(1):130–173, 2013.

[18] P. Miettinen. *Matrix Decomposition Methods for Data Mining: Computational Complexity and Algorithms*. PhD thesis, University of Helsinki, 2009.

[19] P. Miettinen. Sparse Boolean matrix factorizations. In *ICDM*, pages 935–940, 2010.

[20] P. Miettinen, T. Mielikäinen, A. Gionis, G. Das, and H. Mannila. The discrete basis problem. *IEEE Trans. Knowl. Data En.*, 20(10):1348–1362, Oct. 2008.

[21] P. Miettinen and J. Vreeken. Model order selection for Boolean matrix factorization. In *KDD*, page 51–59, 2011.

[22] P. Miettinen and J. Vreeken. MDL4BMF: Minimum description length for Boolean matrix factorization. *ACM Trans. Knowl. Discov. Data*, 8(4):A18:1–31, 2014.

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

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

[25] J. Quinlan. *C4.5: Programs for Machine Learning*. Morgan-Kaufmann, Los Altos, California, 1993.

[26] J. Rissanen. Modeling by shortest data description. *Automatica*, 14(1):465–471, 1978.

[27] D. Shahaf and C. Guestrin. Connecting two (or less) dots: discovering structure in news articles. *ACM Trans. Knowl. Discov. Data*, 5(4):A24:1–31, 2012.

[28] K. Smets and J. Vreeken. The odd one out: Identifying and characterising anomalies. In *SDM*, pages 804–815, 2011.

[29] N. Tatti and J. Vreeken. Discovering descriptive tile trees by fast mining of optimal geometric subtiles. In *ECML PKDD*. Springer, 2012.

[30] U.S. Department of Defense. DoD News Briefing — Secretary Rumsfeld and Gen. Myers, 12 Feb 2002. `http://www.defense.gov/transcripts/transcript.aspx?transcriptid=2636` Accessed 2 Oct 2014.

[31] J. Wicker, B. Pfahringer, and S. Kramer. Multi-label classification using Boolean matrix decomposition. In *SAC*, pages 179–186, 2012.

[32] Y. Xiang, R. Jin, D. Fuhry, and F. Dragan. Summarizing transactional databases with overlapped hyperrectangles. *Data Min. Knowl. Disc.*, 23:215–251, 2011.