Binary Matrix Factorization via Dictionary Learning

Ignacio Ramírez *Member, IEEE*

***Abstract*—Matrix factorization is a key tool in data analysis; its applications include recommender systems, correlation analysis, signal processing, among others. Binary matrices are a particular case which has received significant attention for over thirty years, especially within the field of data mining. Dictionary learning refers to a family of methods for learning overcomplete basis (also called frames) in order to efficiently encode samples of a given type; this area, now also about twenty years old, was mostly developed within the signal processing field. In this work we propose two binary matrix factorization methods based on a binary adaptation of the dictionary learning paradigm to binary matrices. The proposed algorithms focus on speed and scalability; they work with binary factors combined with bit-wise operations and a few auxiliary integer ones. Furthermore, the methods are readily applicable to online binary matrix factorization. Another important issue in matrix factorization is the choice of rank for the factors; we address this model selection problem with an efficient method based on the Minimum Description Length principle. Our preliminary results show that the proposed methods are effective at producing interpretable factorizations of various data types of different nature.**

***Index Terms*—binary matrix factorization, binary dictionary learning, data mining**

I. Introduction

We consider the problem of approximating a binary matrix $X \in \{0,1\}^{m\times n}$ as the product of two other binary matrices $U \in \{0,1\}^{m\times p}$ and $V \in \{0,1\}^{n\times p}$ plus a third *residual* matrix E,

$$X = UV^\intercal + E. \tag{1}$$

1.1 The problem of Binary Matrix Factorization (BMF) arises naturally in many problems, in particular in the so called "association matrices" where a $X_{ij} = 1$ indicates that some object i belongs to group j [1], or some entity (e.g., a gene) i is present in some species j [2], [3], or are related to some type of disease [4]. The latter examples show the increasing relevance of this type of data in genomics. Other similar, growing applications include recommender systems (client-product preferences, etc.). The BMF problem dates back to at least the 1960s [5] and has been treated extensively in the last three decades by various research communities, under quite different names. It was first studied as a combinatorial problem as a particular case of the classic *set covering* problem (see [6] and references therein). It then received great attention from 2.2-3 the data mining community. Long known to be an NP-hard problem [7], the earlier works in this field developed heuristics such as the *tiling* or *tile matching/searching* methods, where binary matrices are decomposed as Boolean or modulo-2 superpositions of rectangular tiles [8], [9]; the BMF problem was later formulated as a matrix factorization problem in [10], with several works following that line since then. Being an NP-hard problem, the quality of the decompositions (U, V) is commonly measured in terms of their *interpretability*, that 1.2 is whether the columns of U and V exhibit patterns that are intuitive in some sense, or reflect *a priori* knowledge about the problem. For example, in data mining problems, where the pairs (U_i, V_i) are interpreted as *association rules* (for example, a group of clients – indicated by non-zeros in U_i – prefers a certain group of products – indicated by non-zeros in V_i); the examples in this paper are designed to show this interpretability in a visual way, by studying the results on visual patterns obtained on sets of images. A thorough survey of BMF methods is beyond the scope of this paper; we refer the reader to [11] for a more in-depth review. We will however mention some works which are representative of the diversity of formulations and tools that surround the treatment of this problem, as well as the shortcomings that are common to the current state of the art and that motivate the development of the tools that we present in this work.

Departamento de Procesamiento de Señales, Instituto de Ingeniería Eléctrica, Facultad de Ingeniería, Universidad de la República, Uruguay.

A. Brief overview of Binary Matrix Factorization

We begin with the ASSO algorithm proposed in [7]. The method begins by constructing the so called *association matrix* C, which is a thresholded version of a particular normalization of the correlation matrix $X^\intercal X$. It then produces a series of increasing rank approximations by adding to U a column taken from C and searching for a corresponding binary row V_k whose outer product with U_k minimizes the number of non-zeros in the current approximation error E. This method can be efficiently implemented with bitwise and integer operations. It is also a popular and simple method with good performance. However, the complexity of each ASSO step is $O(kn^2m)$, so that it does not scale well in applications where n (the number of samples) is very large, something very common in current data science problems, which is the target of our work. 0.1

Many works approximate the BMF problem by a relaxed (non-convex) Non-negative Matrix Factorization (NMF) problem where U and V are allowed to take on real values, and 2.2-3 then map the resulting (approximate) solution to the binary domain using some predefined rule. Examples of this are [10], [12], [13]. In particular, the work [12] develops a set of BMF *identifiability* conditions, that is, conditions under which the binary factorization of X is unique (up to permutations). However, as in [10], [12], their solution is an approximation based on a local minima of the NMF problem, so there are no guarantees that the binarized pair (U, V) obtained coincides with the unique solution even if the identifiability conditions are satisfied. Being based on non-linear optimization, the NMF methods are significantly more computationally demanding than binary methods such as ASSO.

The work [14] stands out as an interesting alternative to BMF which formulates the decomposition of X as a Bayesian denoising problem with a particular prior on the unobserved *clean* matrix $\hat{X}$ ($X = \hat{X} + E$) and uses a *Message Passing* algorithm to find the maximum a posteriori estimation of $\hat{X}$. Message Passing is a mature technique which in the form presented in this work can be quite computationally demanding. Approximate Message Passing [15], [16] techniques have since been developed which may provide significant efficiency gains to the technique proposed in [14], but we are currently unaware of any development in this direction.

An example of a tile-searching heuristic is the Proximus method [8], which approximates the first principal left and right binary components of the binary matrix X. The method can be extended to produce a hierarchical representation of the matrix with further rank-1 components, although these do not coincide with additional factors in a rank-k factorization. Contrary to the above methods, the focus of Proximus is on speed an scalability, and indeed is much faster and scales better than any of the above methods. Also, as we will see in the next subsection, finding the first principal component of a binary matrix is closely related to a crucial step in one of the main dictionary learning methods. e will describe this method in detail later in this document.

B. Dictionary learning

Dictionary learning methods were first introduced in [17] and later adopted as a powerful extension of the *transform analysis* concept, ubiquitous in signal processing since the introduction of Fourier analysis and their related discrete variants (DFT, DCT). In this setting, the matrix $X \in \mathbb{R}^{m \times n}$ consists of n columns of dimension m, and the decomposition obtained $X = DA + E$ is comprised of a *dictionary* D, a matrix of *linear coefficients* A, plus a residual matrix E. Dictionary learning methods are adaptive: given sufficient data samples, they can be trained to efficiently represent such samples as a linear combination of very few basis elements or "atoms" (see [18] for a review). Despite coming from a very different community, dictionary learning methods can be seen as matrix factorization methods which are specially tailored to the where X is either extremely "fat" ($n \gg m$) or "tall" ($n \ll m$). Furthermore, many of these methods can be implemented online, that is, they can process new data samples as they arrive, and adapt the dictionary along the way [19]. Another important aspect of dictionary learning methods is that they are not restricted to low-rank decompositions; if fact, the solutions can be *overcomplete*, meaning that the number of columns p in D may be (much) larger than m, the dimension of the samples to be represented. **2.2-3**

As with all BMF methods, dictionary learning problems are non-convex and their solution cannot be obtained exactly. Nevertheless, similarly to what happens with BMF, their enormous practical success in a wide range of signal processing and machine learning problems, and their ability to produce human-interpretable patterns, has led to their widespread adoption.

C. Model selection

As with any statistical model, the problem of model selection, (in this case, choosing p) is of paramount importance to BMF. Various works [20]–[23] have addressed this particular problem. In particular, [20] and [21] are based on the Minimum Description Length (MDL) principle [24]–[26], which forms the basis of our model selection strategy as well. As a side note, the work [21] represents a recent example of the tiling approach. This problem has also been addressed in dictionary learning in [27].

D. Main contribution

To the best of our knowledge, the works in the existing literature on BMF make no particular assumptions on the *shape* of the matrices to be decomposed. In particular, many methods which are efficient for the $n \approx m$ case do not scale well for $n \gg m$ and vice versa. Also, most methods deal with the *offline* analysis of readily available matrices, which makes them unsuitable to many recent data processing tasks involving *online* adaptation of the models.

The main motivation behind this work is the ability of dictionary learning methods to cope with the challenges mentioned above. We propose two dictionary learning-based BMF methods which are particularly suited to the treatment of extremely fat (or tall) matrices, and which are also suitable to online processing of samples. The first one, named Method of Binary Directions (MOB), is an adaptation of the method proposed in [19], itself an adaptation of the Method of Optimum Directions (MOD) [28] (hence the name). The second method is based on the idea of sequential rank-one updates of the K-SVD method [29]; we adopt the Proximus [8] algorithm as a fast approximation to the mentioned operation; we thus bring together tools from two inherently related but otherwise disconnected fields to obtain a novel formulation to the binary matrix factorization. Finally, both methods rely on a third novel algorithm which we call Binary Matching Pursuit (BMP). This is a binary adaptation of the Matching Pursuit method [30] for approximating the sparsest solution to a least squares regression problem. This adaptation, although conceptually analogous to MP, is non-trivial, as its efficiency relies on a careful combination of different algebraic operations. **2.1**

Our methods construct binary dictionaries and binary coefficients matrices using efficient bitwise and a few integer operations. This is particularly relevant to the efficiency of our methods as recent processor architectures incorporate the ability to handle large number of bits through SIMD (Single Instruction Multiple Data) instructions. For example, a current off-the-shelf processor can perform a *popcount* instruction (which counts the number of 1s in a binary array) on a 256-bit register. This allows, for example, to compute the dot product between two binary vectors of dimension 256 with just two processor instructions. As we show in Section III, our methods are also linear both in n and m and thus scale well for large matrices. **2.4**

The main objectives of this paper are to present our proposed methods, assess their interpretability on different datasets, to analyze their computational properties such as

computational complexity and convergence rate, and to see how these properties are affected by the initial conditions. We thus focus our experiments on a small set of easily-interpretable datasets for which the patterns obtained can be easily recognized as salient features in the data.

The rest of this document is organized as follows: Section II-C provides the notation and background on the methods on which our methods are based. The proposed methods themselves are described in Section III. Section IV describes the proposed MDL-based model selection algorithm for searching the best model order p. We present and discuss our results in Section V, and provide concluding remarks in Section VI.

II. Background

A. Notation

We begin this section by establishing the notation to be used throughout the paper. Standard operations such as addition or subtraction are denoted as usual, $1+1=2$, $1-1=0$, etc. Given two binary values a and b, we use $a \wedge b$ to denote their logical AND (Boolean product), $a \vee b$ is their OR (Boolean sum), $a \oplus b$ is modulo-2 addition (Boolean eXclusive OR), and $\neg a$ is the 1's complement (Boolean negation) of a. The same notation is used for element-wise operations between vectors and matrices of the same dimension.

Let x and y be two binary vectors and A and B two binary matrices. We denote the standard inner and outer vector-vector, matrix-vector and matrix-matrix products as $x^\intercal y$, $xy^\intercal$, Ax and AB. The Boolean inner product between two vectors, $z = x^\intercal \circ y$ is defined as $\bigvee x_i \wedge y_i$. Similarly, the C_{ij} element of the matrix product $C = A \circ B$ is defined as the Boolean product between the i-th row of A, $A_{i:}$, and the j-th column of B, $B_{:j}$. We define the modulo-2 inner product of x and y as $x \otimes y = (x+y) \bmod 2$. Finally, similarly to the Boolean case the C_{ij} element of the modulo-2 product between two matrices, $C = A \otimes B$ is given by $A_{i:} \otimes B_{:j}$.

The cardinality of a set J is denoted by $|J|$. Given a set of indexes J, $B_{:J}$ denotes the sub-matrix of columns of B indexed by J, and $B_{J:}$ denotes a subset of its rows. These can be combined with single indexes or other sets, e.g., B_{iJ} contains the elements of row i and column indexes in J. For a vector x, its Hamming weight $h(x)$ is defined as the number of non-zero elements in x, that is $h(x) = |\{i : x_i \neq 0\}|$. The same notation $h(A)$ is applied to count the non-zero elements of matrices. The Hamming weight is usually referred to as the ℓ_0 pseudo-norm, $\|x\|_0 = h(x)$. Note that, for binary vectors, $h(x) = \|x\|_0 = \|x\|_1 = \|x\|_2^2$. Likewise, for binary matrices, $\|A\|_0 = \|A\|_{1,1} = \|A\|_F^2$. The function $\mathbf{1}(\cdot)$ is defined so that $\mathbf{1}(cond) = 1$ if $cond$ is true, and 0 otherwise.

B. A note on computational complexity

2.4

For each algorithm we provide a brief computational complexity analysis using a simplified form of the familiar "big O" notation. Here a procedure which has order $O(m)$ is said to have *linear complexity* in m, that is, it requires at most $am+b$ operations where a and b do not depend on m. Similarly $O(m^2)$ indicates *quadratic complexity*, and $O(m \log m)$ requires about $a(m \log m)+b$ operations in the worst case. This use is slightly different than the formal Bachmann-Landau definition of $O(\cdot)$, which is defined in asymptotic terms. For example, we might write $O(m \log m + p)$ as a shorthand to $O(m \log m) + O(p)$ to indicate that a function requires about $am \log m + bp + c$ operations. On the other hand, in all cases we will make a distinction as to the nature – floating point, integer, bitwise – of the operations, as the constants involved in each case can vary greatly. For example, as pointed out in the introduction, a bitwise inner product of two binary vectors of length 256 can computed in just *two* CPU instructions, whereas it might require over 512 instructions to compute the same operation on floating point or integer vectors.

C. Dictionary Learning and sparsity

As described before, dictionary learning methods seek a to decompose X as $X = DA + E$, where D is such that we can achieve $\|E_{:j}\| \ll \|X_{:j}\|$ by using just a few atoms of D, that is, with a number of non-zero elements in $A_{:j}$ much smaller than p. The latter requirement is called *sparsity*. Thus, dictionary learning is often also referred to as *sparse modeling*, and finding $A_{:j}$ given D as *sparse coding*. A typical approach to the dictionary Learning problem is to obtain a local solution by *alternate minimization* in D and A of a cost function $f(D,A) + g(A)$,

$$A^{(t+1)} = \arg\min_A \{f(D^{(t)}, A) + g(A)\} \quad (2)$$

$$D^{(t+1)} = \arg\min_D \{f(D, A^{(t+1)}) + g(A^{(t+1)})\}, \quad (3)$$

where $f(\cdot)$ is a *data fitting* term, usually $\|DA - X\|_F^2$ (here $\|X\|_F$ denotes the Frobenius norm of matrix X), and $g(\cdot)$ is a *regularization* term which promotes sparsity in the columns of A. We now describe the two methods on which our methods are inspired.

1) Method of Optimal Directions (MOD): For the case $f(D,A) = \|DA - X\|_2^2$ and $g(A) = \sum_j \|A_{:j}\|_1$ the *Method of Directions* (MOD) [28], is given by

$$A_j^{(t+1)} = \arg\min_{a \in \mathbb{R}^p} \{\|x_j - D^{(t)} a\|_2^2 + \|a\|_1\}, \quad (4)$$

$$D_r^{(t+1)} = U_r / \max\{1, \|U_r\|_2\},$$
$$U = X(A^{(t+1)})^\intercal \left(A^{(t+1)}(A^{(t+1)})^\intercal\right)^{-1}, \quad (5)$$

The first step corresponds to an ℓ_1-regularized least squares regression problem on each column of A, also known as LASSO [31], a non-differentiable convex problem whose solution has been extensively studied in recent years, with several efficient algorithms designed specifically for the task (see e.g. [32]). In the second step, each atom $D_{:r}$ of the dictionary corresponds to a normalized down version of the least squares solution u_r.

The MOD algorithm is well suited for online dictionary adaptation, as both $AA^\intercal$ and $XA^\intercal$ can be efficiently updated when new columns are added to X. Furthermore, if new samples arrive one at a time, the inverse of the Hessian matrix $(AA^\intercal)^{-1}$ can be efficiently updated via the Matrix Inversion Lemma [33]. Moreover, as shown in [19], excellent results can be still obtained if the Hessian is approximated by its diagonal (in which case computing its inverse requires just $O(p)$ operations).

Computational complexity: The offline version of the MOD dictionary update step presented in Algorithm 5 requires $O(mnp) + O(p^2 n) + O(p^3)$ floating point operations, which will generally be dominated by the $(p^2 n)$ term.

2) Matching Pursuit and the K-SVD algorithm: In this algorithm, proposed in [29], $f(E) = \|E\|_F^2$ and $g(A) = h(A)$. The columns of A are computed using a greedy method known as OMP (Orthogonal Matching Pursuit) [34], which under certain conditions can be shown to provide the actual solution to the corresponding ℓ_0-penalized least squares problem (see [35]). A simpler variant of this step uses the (non-orthogonal) Matching Pursuit (MP) [30], which is described next in Algorithm 1,

Data: vector to encode x, dictionary D, maximum residual norm ϵ, maximum coefficients weight $h_{\max}$
Result: Coefficients vector a
Set iteration $t \leftarrow 0$, residual $r^{(0)} \leftarrow x$, initial coefficients $a^{(0)} \leftarrow 0$;
Set $g^{(0)} \leftarrow D^\intercal r^{(0)}$, $G \leftarrow D^\intercal D$;
while $\left\|r^{(t)}\right\| \geq \epsilon$ *and* $h(a) < h_{\max}$ **do**
  $i = \arg\max \left\{g^{(t)}\right\}$;
  $\Delta \leftarrow D_{:i}^\intercal r^{(t)}$;
  $a_i^{(t+1)} \leftarrow a_i^{(t)} + \Delta$;
  $r^{(t+1)} \leftarrow r^{(t)} - \Delta D_{:i}$;
  $g^{(t+1)} \leftarrow g^{(t)} - \Delta G_{:i}$; /* (a) */
  $t \leftarrow t + 1$;
end
return $a \leftarrow a^{(t)}$;

Algorithm 1: Matching Pursuit

What MP does at each iteration is to project the residual onto the atom that is most correlated to it, and then remove the projection from the residual. For this to work well, the atoms must be normalized to have ℓ_2 norm 1. The vector g keeps track of the correlation between the residual r and the dictionary D. Its update (a) is derived as follows:

$$\begin{aligned} g^{(t+1)} &= D^\intercal r^{(t+1)} = D^\intercal (r^{(t)} - \Delta D_i) \\ &= g^{(t)} - \Delta D^\intercal D_{:i} = g^{(t)} - \Delta G_{:i}. \end{aligned} \tag{6}$$

Note that, by means of (6), updating g requires only $O(p)$ floating point operations, whereas the naïve update requires $O(mp)$ operations. This same trick, with a few modifications, will also be useful in the binary case. As the Gramm matrix is computed only once, the overall complexity of MP for processing all n samples is $O(p^2 m + p + kmn)$ floating point operations.

The dictionary update of K-SVD performs a simultaneous update of each atom $D_{:r}$ and the row of A associated to it. This update is described in Algorithm 2.

Computational complexity: Each of the p updates of D requires $O(mn)$ operations for updating the residual plus another $O(n^2)$ operations for computing the first pair of singular eigenvectors. This gives a total $O(pn^2) + O(pmn)$ operations, which in general will in our case ($n \gg m \approx p$) will be significantly more expensive than the $O(p^2 n) + O(pmn) + O(p^3)$ complexity of MOD.

Data: Current iterate $(D^{(t)}, A^{(t)})$
Result: Next iterate $(D^{(t+1)}, A^{(t+1)})$
for $r = 1, \ldots, p$ **do**
  $J \leftarrow \{j : A_{rj}^{(t)} \neq 0\}$;
  $R \leftarrow X_{:J} - D^{(t)} A_{:J}^{(t)} + D_{:r}^{(t)} (A_{rJ}^{(t)})$;
  $U \Sigma V^\intercal \leftarrow \mathrm{SVD}(R)$;
  $D_{:r}^{(t+1)} \leftarrow U_{:1}$;
  $A_{rJ}^{(t+1)} \leftarrow V_{:1}$;
end

Algorithm 2: K-SVD Dictionary update.

III. Binary Dictionary Learning

Below we describe our dictionary learning methods, which assume a given fixed dictionary size p and employ the traditional alternate descent approach to obtain the best model (D, A, E) for that p. We leave the description of the top-level model selection algorithm for choosing the best model order p to Section IV.

Both BMF algorithms share a common coefficients update step, the Binary Matching Pursuit (BMP) algorithm, and two choices for the dictionary update step: MOB (a binarized version of MOD) and K-PROX (combining ideas of K-SVD and Proximus); these are detailed next.

A. Coefficients update via Binary Matching Pursuit (BMP)

Data: sample to encode x, dictionary D, initial coefficients $a^{(0)}$, maximum coeffs. weight $h_{\max}$, maximum residual weight $w_{\max}$.
Result: Optimum coefficients for x, a
Set iteration $t = 0$, coefficients $a^{(0)} = a_0$,residual $r^{(0)} = x \oplus D \otimes a^{(0)}$;
Set modulo-2 Gramm matrix $G \leftarrow D^\intercal \otimes D$; /* (a) */
Set residual correlation $g^{(0)} \leftarrow D^\intercal r^{(0)}$; /* (b) */
while $h(r^{(t)}) \geq w_{\max}$ ***and*** $t < h_{\max}$ **do**
  $k \leftarrow \arg\max_l \{\, |g_l^{(t)}| \,/\, \|D_{:l}\|_0 \,\}$; /* (c) */
  if $g_k^{(t)} = 0$ **then**
    return $a \leftarrow a^{(t)}$;
  end
  $r^{(t+1)} \leftarrow r^{(t)} \oplus D_{:k}$;
  if $h(r^{(t+1)}) \geq h(r^{(t)})$ **then**
    return $a \leftarrow a^{(t)}$;
  end
  if $a_k^{(t+1)} = 1$ **then**
    $a_k^{(t+1)} \leftarrow 0$; $g^{(t+1)} \leftarrow g^{(t)} - G_{:k}$; /* (d) */
  else
    $a_k^{(t+1)} \leftarrow 1$; $g^{(t+1)} \leftarrow g^{(t)} + G_{:k}$; /* (d') */
  end
end
return $a \leftarrow a^{(t)}$;

Algorithm 3: Binary Matching Pursuit.

In essence, BMP is a binarized version of the Matching Pursuit Algorithm 1. For a given sample x, we begin $(t = 0)$ with an initial coefficients vector $a^{(0)} = a_0$, a residual

$r_j^{(0)} = x \oplus D \otimes a^{(0)}$ and an initial vector $g^{(0)} = D^\intercal r^{(0)}$ which keeps track of the correlation between the columns of D and $r^{(t)}$. Then, at each iteration t we determine the atom $D_{:k}$ which is most correlated to $r^{(t)}$. We then *toggle* the coefficient corresponding to that atom, $a_k^{(t)}$, and update $r^{(t)}$ and $g^{(t)}$ accordingly. The pseudocode is given in Algorithm 3. Some steps of this algorithm, marked as (a), (b), (c) and (d) (appearing twice) in Algorithm 3, are not obvious from the overall description of the algorithm and need to be clarified. In (a), the *modulo-2 Gramm matrix* of D is computed; this matrix is used in an analogous way in Algorithm 1 for the fast update of the correlations vector g as described in (6). In (b), we compute the standard correlation between the columns of D and r. In (c), since the atoms are not normalized, the best candidate is chosen using a form of normalized correlation called *association accuracy* [36], $g_l^{(t)}/\|D_{:l}\|_2^2 = g_l^{(t)}/\|D_{:l}\|_0$.

Finally, in (d) and (d'), despite the correlation vector g being initially computed using the *standard* matrix-vector product, its updated has exactly the same form as (6), but the Gramm matrix G is actually computed using the modulo-2; (d) corresponds to the case when $\Delta = 1$, that is, when a_k is switched from 0 to 1, and (d') corresponds to $\Delta = -1$, when a_k is switched off. (This curious result is easily verified by writing down the corresponding arithmetic.)

Computational complexity: The initialization of BMP requires $O(p^2m)$ binary operations for computing the Gramm matrix and integer ones $O(p^2m)$ for the correlation vector g. Then, for each sample $X_{:j}$ a maximum of $h_{\max}$ iterations can be required, each one requiring $O(p)$ floating point operations for finding the most correlated atom, another $O(p)$ integer operations for updating the residual r, and another $O(p)$ integer ones for updating g, for a total of $O(h_{\max}p)$ operations per sample. Overall, a whole pass over the n data samples requires $O(p^2m) + O(p) + O(h_{\max}pn)$ operations. **2.4**

B. MOB: Method Of Binary Directions

Here we want to update D so as to minimize the Hamming weight $h(E)$ of the residual matrix $E = X \oplus D \otimes A$. Suppose we want to update the r-th atom at iteration k. The affected columns will only be those for which the coefficients in A corresponding to that atom are non-zero. We define $J = \{j : A_{rj} \neq 0\}$ to be the set of indexes of those columns. What we want is to update $D_{:r}$ so that the weight of the columns of the residual affected by it is minimized,

$$\begin{aligned} D_{:r}^{(t+1)} &= \arg \min_{d \in \{0,1\}^m} \sum_{j \in J} h(E_{:j}^{(t)} \oplus d) \\ &= \arg \min_d \sum_{j \in J} (\sum_i E_{ij}^{(t)} + |J| d_i). \end{aligned} \tag{7}$$

According to (7), the optimization of $D_{:r}^{(t+1)}$ is separable in each of its elements,

$$\begin{aligned} D_{ir}^{(t+1)} &= \arg \min_{u \in \{0,1\}} \sum_j E_{ij}^{(t)} \oplus u \\ &= \arg \min_{u \in \{0,1\}} \left| \sum_{j \in J} E_{ij}^{(t)} - |J| u \right| \\ &= \arg \min_{u \in \{0,1\}} | \, h(E_{iJ}^{(t)}) - |J| u \, | \\ &= \mathbf{1}\left(\frac{h(E_{iJ}^{(t)})}{|J|} > \frac{1}{2} \right) \end{aligned} \tag{8}$$

Note that when $h(E_{iJ})/|J| = 1/2$ both 0 and 1 are optimum, in which case we use 0. Also, note that (8) can be rewritten as

$$D_{ir}^{(t+1)} = \mathbf{1}\left(\frac{h(EA_{r:}^\intercal)}{h(A_{r:})} > \frac{1}{2} \right).$$

As these statistics can be easily updated as new samples arrive, it follows that MOB is suited for fast online dictionary adaptation. (Our implementation of this feature is still work in progress.)

Computational complexity of MOB: Our current (offline) implementation requires $O(mnp)$ bitwise operations for the modulo-2 product $EA^\intercal$, $O(np)$ integer operations for computing the weights of the rows of A, and $O(mp)$ integer comparisons for updating D, for a total of $O(mnp)$ binary plus $O(mp)$ integer operations. **2.4**

C. K-PROX: Dictionary Update via Proximus

In this case, following the K-SVD concept, we want to obtain the best rank-one approximation to the residual E obtained after removing the contribution of $D_{:r}$. Let $J = \{j : A_{rj} \neq 0\}$ and $R_J = D_{:r} \otimes A_{rJ} \oplus E_J$. We then have,

$$(D_{:r}^{(t+1)}, A_{rJ}^{(t+1)}) = \arg \min_{u,v} h\left(R_{:J}^{(t)} \oplus uv^\intercal \right). \tag{9}$$

As the name K-PROX implies, our approximation to the NP-hard problem (9) is based on the Proximus algorithm [8], summarized in Algorithm 4, Interestingly, Algorithm 4 provides

Data: matrix $X \in \{0,1\}^{m \times n}$, $u^{(0)} \in \{0,1\}^m$, $v^{(0)} \in \{0,1\}^n$
Result: Vectors u, v so that $X \approx uv^\intercal$
Set iteration $k = 0$;
repeat
  $u_i^{(t+1)} \leftarrow \mathbf{1}\left(X_{i:}v^{(t)} > h(v^{(t)})/2\right)$, $i = 1, \ldots, m$;
  $v_j^{(t+1)} \leftarrow \mathbf{1}\left(X_{:j}^\intercal u^{(t+1)} > h(u^{(t+1)})/2\right)$, $j = 1, \ldots, n$;
  $k \leftarrow k + 1$;
until $u^{(t+1)}(v^{(t+1)})^\intercal = u^{(t)}(v^{(t)})^\intercal$;

Algorithm 4: Proximus

a local optimum to the rank-one approximation that we seek. This is stated in Proposition 1 below.

Proposition 1. The output (u, v) of the Proximus Algorithm 4 is a local optimum of the problem $\min \|X - uv^\intercal\|_0$.

Proof. Given $v^{(t)}$, it is easy to check that the update $u^{(t+1)}$ in Algorithm 4 is the value of u that *globally* minimizes

$\|X \oplus uv^{(t+1)}\|$ (if $s_i^{(t)} = w^{(t)}/2$, both 0 and 1 are equally optima; in such case, we default to 0). The same happens with the update $v^{(t+1)}$ given $u^{(t+1)}$. Therefore, $h(E^{(t)}) = h(X \oplus u^{(t)}(v^{(t)})^\intercal)$ cannot increase with k. As $h(E^{(t)}) \geq 0$ is bounded, non-increasing, and the iterates can take on a finite number of values, the sequence $h(E^{(t)})$ must converge after a finite number of steps. Let (u, v) be the arguments at which the stopping condition is satisfied. By definition of the algorithm, no change in u or v decreases the objective. This guarantees that (u, v) is a local minimum in a Hamming ball of radius at least 1.[1] □

Computational complexity of K-PROX: As with the K-SVD algorithm, we update D one atom at a time. This requires $O(mn)$ operations for computing the residual in (9) and running Algorithm 4, which takes a finite number of iterations requiring $O(mn) + O(np)$ bitwise operations and the same number of integer comparisons. **2.4**

D. Initialization

Initialization is of paramount importance to the success of any non-convex matrix factorization method. At the same time, there is no provably optimum way of doing so, otherwise we would be contravening the NP-hard nature of the factorization problem itself. We are left with heuristics based on intuition and prior information, if any. Ultimately, *initialization is an art*, and also an engineering decision which may depend on several aspects. As an example, we could use the resulting factorization obtained with *any* of the existing methods mentioned in Section I as an initial point. In this case, to maintain scalability and simplicity, we experiment with two simple but generally effective methods drawn also from dictionary learning literature: given dictionary size p, the first draws p atoms using a pseudo-random Bernoulli(θ) distribution; we use $\theta = 1/2$, but other values could be used to reflect prior information on the problem. The second method is to draw p columns from X at random and use them as the initial atoms. WE report on both strategies on Section V.

IV. Model selection

Beyond theoretical results and simulations, when confronted with real data, the true underlying model governing the generation of data is rarely available. In such situations, models can only be assumed to be tools for understanding the data at hand, and the best choice is dictated by how much regularity or structure each candidate model is capable to grasp from that data. In particular, the complexity of a model is limited by the amount of data available to estimate the different parameters of the model. An overly complex model will tend to *overfit* the data, whereas an overly simple one will miss important details. The problem of model selection is that of finding the best model for a given data set. Typically, this is done by seeking a balance between the *goodness of fit* of a model, usually expressed in terms of the likelihood of the data given the model, and a measure of the *model complexity*; some popular examples in this category are the Bayesian Information Criterion (BIC) [37], the Akaike's Information Criterion [38], and the Minimum Description Length (MDL) principle [24]–[26].

MDL translates the model selection problem as one of data compression, where both the data and the model have to be (hypothetically) transmitted and perfectly recovered using some encoding mechanism. The tension between complexity is then resolved by the number of bits (*codelength*) required to describe the data in terms of the model (*stochastic complexity*) and the number of bits required to describe the model itself (*model complexity*). The original version of MDL [24] made this division explicit, and is asymptotically equivalent to BIC. However, the modern version of MDL [25], [26] uses the more recent information-theoretic *universal coding theory* to make the aforementioned balance implicit by producing an optimum, joint description of both the model and the data which is furthermore independent of arbitrary choices of, for example, the way the model is parameterized. These advantages make MDL an appealing method for model selection.

In our case, the data X is described by the triplet (D, A, E). We describe each of these components separately using universal codes, so that $L(X) = L(D) + L(A) + L(E)$ is the total codelength of describing X. Here the tension between a good fit and a simple model is represented respectively by $L(E)$ and $L(D) + L(A)$.

A. Forward selection algorithm

Model selection is usually formulated as two nested problems. The inner problem is how to search for the best model (D, A, E) among all models of order p. The second problem is to choose the best model for X among all possible models of order $p \geq 0$. Given a codelength function $L(X)$, our method uses a *forward selection* strategy to sweep over all models. Starting with a initial order $p = p_0$, we approximate the best model given p using one of our dictionary learning algorithms, and then gradually increase p until the codelength $L(X)$ is no longer diminished. **2.5** **2.4**

In going from p to $p+1$ we employ a *warm restart* strategy, that is, atoms and coefficients learned for model order p are used as the starting point for learning the $p+1$-th order model. The overall procedure is summarized in Algorithm 5,

B. Codelength computation

It remains to describe how we compute $L(X)$. The universal compression of binary sources has been extensively studied in the literature. Moreover, we do not need to perform a real encoding; we need only to compute the codelength. One particularly simple method, for which the codelength is easy to compute, is *enumerative coding* [39]. Given a binary string x of length n and $r = h(x)$, its enumerative code is composed of two parts. The first one describes r with $\lceil \log_2(n) \rceil$ bits, and the second one describes the index of x in the lexicographically ordered list of all binary strings of length n and weight r, which requires $\lceil \log_2 \binom{r}{n} \rceil$ bits. The total codelength is then

$$L(x) = \lceil \log_2(n) \rceil + \left\lceil \log_2 \binom{r}{n} \right\rceil . \tag{10}$$

[1] We cannot guarantee that a simultaneous change in a single coordinate of u and a single coordinate of v will not decrease the cost function!.

Data: matrix $X \in \{0,1\}^{m\times n}$, initial order p_0 and model $(D^{(0)}, A^{(0)}, E^{(0)})$
Result: Selected model (D^*, A^*, E^*)
$p \leftarrow p_0$;
$(D^{(p)}, A^{(p)}, E^{(p)}) \leftarrow (D^{(0)}, A^{(0)}, E^{(0)})$;
$L^{(p)} \leftarrow L(D^{(p)}) + L(A^{(p)}) + L(E^{(p)})$;
repeat
  Initialize the rank-1 model (d, a) using $E^{(p)}$ as input ;
  $(\tilde{D}, \tilde{A}, \tilde{E}) \leftarrow ([D^{(p)}|d],\ [(A^{(p)})^\intercal | a^\intercal]^\intercal,\ E^{(p)} - da^\intercal)$;
  Adapt $(D^{(p+1)}, A^{(p+1)}, E^{(p+1)})$ using $(\tilde{D}, \tilde{A}, \tilde{E})$ as the starting point;
  $L^{(p+1)} \leftarrow L(D^{(p+1)}) + L(A^{(p+1)}) + L(E^{(p+1)})$;
  $p \leftarrow p + 1$;
until $L^{(p)} \geq L^{(p-1)}$;
$(D^*, A^*, E^*) \leftarrow (D^{(p-1)}, A^{(p-1)}, E^{(p-1)})$;

Algorithm 5: MDL-based Forward Selection Algorithm

(Note that $\log \binom{r}{n}$ can be accurately approximated by using Stirling's formula, i.e., requiring $O(1)$ operations.) As prior information, we expect the different columns of D to represent particular patterns in the data, the corresponding rows of A their appearance in X, and the different rows (corresponding to different variables of the data samples) of E to exhibit different patterns also. Accordingly, we encode each column of D and each row of E and A with its own code,

$$L(X) = \sum_{i=1}^{m} L(E_{i:}) + \sum_{k=1}^{p} L(D_{:k}) + \sum_{k=1}^{p} L(A_{k:}) \qquad (11)$$

Computational complexity: The cost of each forward selection step is clearly dominated by the dictionary learning algorithms. The codelength evaluation is negligible, requiring $O(mn) + O(mp) + O(np)$ operations to count the number of zeros in each component (D, E, A).

V. Results and Discussion

The main objectives of the following experiments are three. The first is to demonstrate the interpretability of the resulting models; we expect to obtain atoms which exhibit recognizable patterns of the input data. The second is to see the effect of different initialization strategies on the final result. The third is to study the numerical properties of the methods, in particular their convergence rate and computational cost.[2] The first dataset is a binarized version of MNIST [40], a set of $n = 10000$ 17×17 images of handwritten digits; here each columns of X contains the vectorized version of a digit ($m = 289$). The second consists of all the $m = 4088$ non-overlapping blocks of size 16×16 of the *halftone Einstein*, a binary 1160×896 image; the columns of X are the 4088 vectorized blocks ($m = 256$) of the image. As can be seen in Figure 1, both datasets have easily recognizable patterns.

[2]The version used in this paper can be downloaded from http://iie.fing.edu.uy/~nacho/bmf/bmf.zip; this includes scripts and data to reproduce all the results shown in this section and many more not discussed in this paper for lack of space. The latest version is available as a GIT repository https://gitlab.fing.edu.uy/nacho/bmf.

A. Interpretability of the resulting models

Figure 2 shows the MNIST model obtained using MOB and forward selection, The dictionary and a few columns from the coefficients and residual matrices A and E; atoms from D and samples from E are represented as mosaics with each sample/atom displayed as a tile. As can be seen, many atoms look like numbers; other atoms exhibit number-like silhouettes. The results obtained with K-PROX (shown in the supplementary material) are very similar in all aspects. Figure 3 shows the model obtained for the Einstein image using K-PROX; we show the dictionary, a few samples; again, the results are clearly interpretable in terms of the patterns that can be observed in the data. In this case too the results obtained with MOB (in the supplementary material) are very similar.

B. Sensitivity to initialization

Figures 4 and 5 show initial and final dictionaries of fixed size $p = 36$ for MNIST and Einstein, using both dictionary learning methods, but using each of the two initialization methods described in Section III-D. In this case, the random samples initialization does a good job in both cases, whereas the pseudo-random initialization fails miserably on the MNIST case. These results should be taken with a grain of salt, only to show how different initialization methods can work (or fail) under different circumstances.

C. Numerical results

In this set of experiments we are interested mainly in three aspects of the proposed methods. First, the convergence of the forward selection method in terms of overall cost function, the convergence rate of the dictionary learning algorithms, and the empirical computational complexity of both the selection and learning methods as measured in running time (seconds), both per iteration and accumulative.[3] Figure 6 shows that forward selection converges exponentially to a local minimum which is very close in cost function for both MOB and K-PROX. The forward selection mechanics shown in the breakdown of the arguments are typical of methods such as MDL, which shows the correctness of the procedure. Figure 7 shows that both MOB and K-PROX converge quickly both in cost function and the respective optimization variables (we recall that convergence is exact here – there is no further change in the arguments). This same behavior was observed for all model sizes in Figure 6. In terms of execution time, both Figure 6 and 7 show MOB as the fastest of the two methods in terms of computing speed. It should be noted however that our implementation for MOB is reasonably optimized, whereas that of K-PROX is not. By comparing the average time for learning a fixed order model in figures 6 and 7 it should be clear that the warm-restarts strategy used is crucial for efficiently searching through the different model sizes.

[3]The timing results were obtained on a Lenovo V310-14IKB notebook with an Intel i5-7200U (4 cores) processor, 8GB of RAM, running Lubuntu 18.04 64bits, with executables compiled from C++ code using GCC 7.3.0 with maximum optimization ("-O3"), multicore support ("-fopenmp") and all SIMD functions enabled.

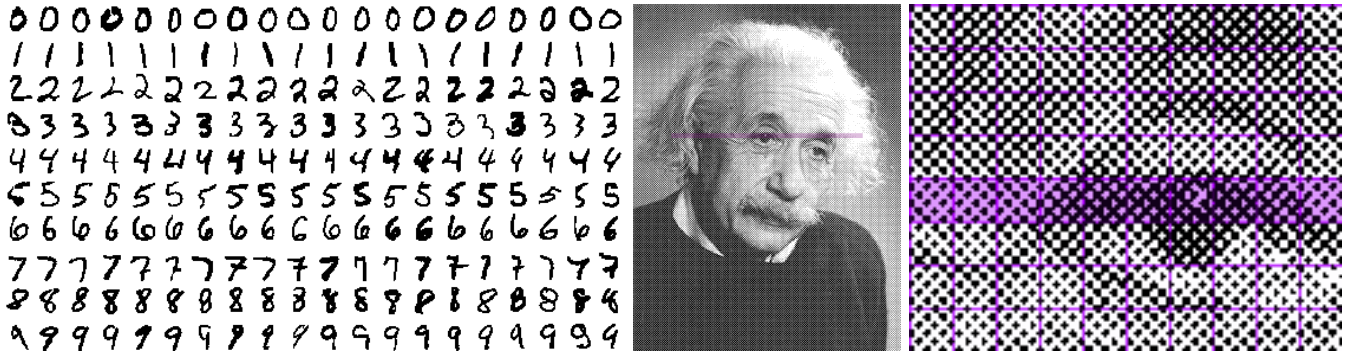

Figure 1: Left to right: a few samples of the MNIST dataset; halftone image of Einstein with the stripe used in Figure 3 highlighted in magenta; detail of Einstein's left eye and the 16×16 blocks partition; including part of the stripe. the first case, we expect the atoms in the final dictionary to resemble numbers of different shapes (see e.g. [41]). In the case of Einstein, we expect the dictionary atoms to resemble the halftoning patterns observable in the 16×16 blocks shown on the right picture.

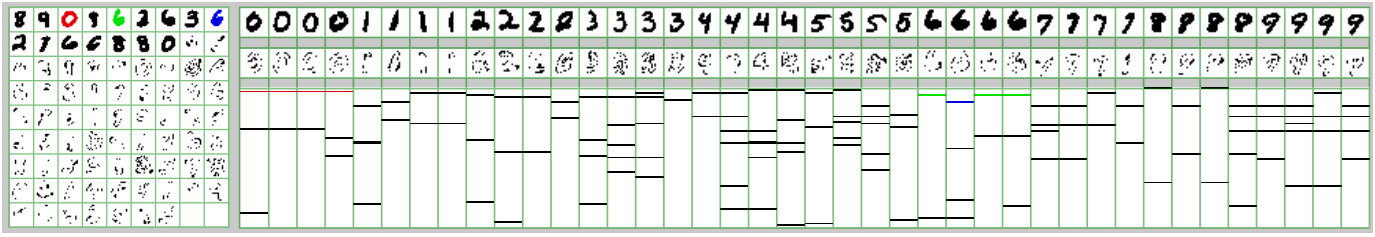

Figure 2: MNIST model obtained using Forward selection with MOB at each step. The model was initialized with $p_0 = 16$ random samples. The final dictionary, with $p = 79$ atoms, is shown to the left as a mosaic from top to bottom and left to right. The three rows to the right of the dictionary show 40 samples from X (top), their final residual from E (middle), and the corresponding coefficients from A (bottom). Each column of A is represented as a vertical pattern of 79 bands: the top band corresponds to the first atom, and the bottom one to the 79th; a black band on the i-th row indicates that atom i is being used to represent the sample on top of the same column, giving rise to the residual shown in the middle. In general, coefficients are not easy to interpret. However, in some cases the correspondence is clear. In this case the four digits "0" use the third atom (painted in red). Something similar happens with the "6"s: three of them use the 5th atom (green) and one uses the 9th (blue). The corresponding coefficients are marked as red, green and blue bands in the bottom-right picture.

VI. Concluding remarks

In this paper we have presented two novel and efficient Binary Factorization Methods based in dictionary learning techniques through the combination of three novel algorithms, BMP for learning coefficients, and MOB and K-PROX for updating the dictionaries. We have provided theoretical guarantees on their convergence to local minima, and a simplified but rigorous analysis of their computational complexity. Through experimentation on two very different datasets, we have demonstrated that our methods produce interpretable results in both cases, requiring very few iterations to converge, and with an overall computational complexity which is very competitive (even though we did not employ any speed-up strategies such as online or mini-batch learning), requiring as little as 0.1s to learn a complete dictionary on a dataset of 10000 289-dimensional samples from scratch. We have also shown the scalability of our model in terms of growing model size, allowing us to search over a large family (hundreds) of candidate models (dictionaries) in as little as 4 seconds on a modest notebook. In a follow up of this work, which is currently underway, we will present on-line implementations of the MOB and K-PROX algorithms developed here, with the objective of employing them on huge genomic datasets. Other possible lines of work include finding conditions under which our methods (or some of them) can be guaranteed to recover a given underlying model.

References

[1] I. Ramírez and M. Tepper, "Bi-clustering via mdl-based matrix factorization," in *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications*, J. Ruiz-Shulcloper and G. Sanniti di Baja, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 230–237.

[2] A. Moya-Hernndez, E. Bosquez-Molina, A. Serrato-Daz, G. Blancas-Flores, and F. Alarcn-Aguilar, "Analysis of genetic diversity of cucurbita ficifolia bouch from different regions of mexico, using aflp markers and study of its hypoglycemic effect in mice," *South African Journal of Botany*, vol. 116, pp. 110–115, 2018.

[3] U. Kulsum, A. Kapil, H. Singh, and P. Kaur, "Ngspanpipe: A pipeline for pan-genome identification in microbial strains from experimental reads," *Advances in Experimental Medicine and Biology*, vol. 1052, pp. 39–49, 2018.

[4] A. Hujdurovic, U. Kacar, M. Milanic, B. Ries, and A. Tomescu, "Complexity and algorithms for finding a perfect phylogeny from mixed tumor samples," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 15, no. 1, pp. 96–108, 2018.

[5] M. Mickey, P. Mundle, and L. Engelman, *BMDP Statistical Software Manual*. University of California Press, 1990, vol. 2.

[6] S. D. Monson, N. J. Pullman, and R. Rees, "A survey of clique and biclique coverings and factorizations of (0, 1)-matrices," *Bull. Inst. Combin. Appl.*, vol. 14, pp. 17–86, 1995.

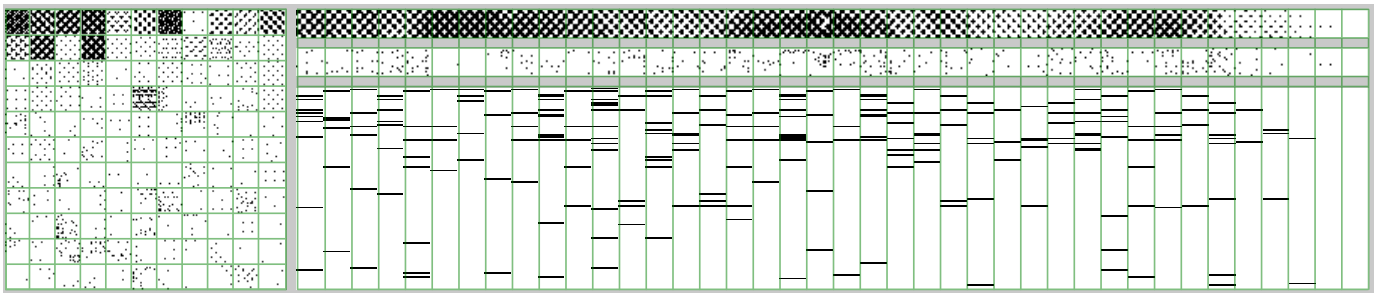

Figure 3: Einstein model obtained using forward selection and K-PROX. Here too we used random samples for the initialization. As in Figure 2, we show the resulting dictionary on the left, and the samples (top), residuals (middle) and coefficients (bottom) for the patches contained in the stripe highlighted in Figure 1.

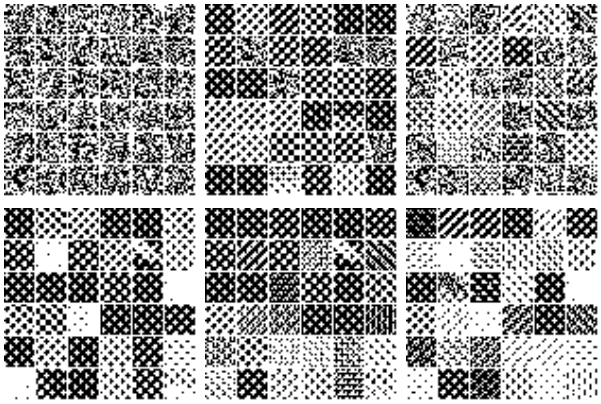

Figure 4: Effect of initialization on Einstein. Top row: initial dictionary drawn from a Bernoulli$(1/2)$ process (left), resulting MOB (center) and K-PROX (right) models for $p = 36$. Bottom row: initial dictionary using random columns from X (left), resulting MOB (center) and K-PROX (right) dictionaries. Both initialization methods worked well in this case using both MOB and K-PROX. In all cases, the dictionaries evolved so that some atoms resemble halftoning patterns. Other atoms were not changed, indicating that they were never used.

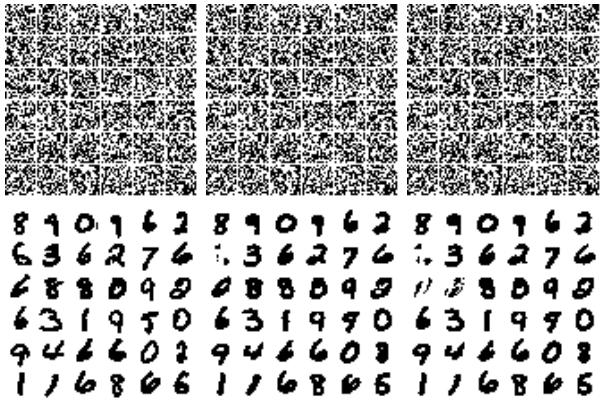

Figure 5: Effect of initialization on MNIST. Top row: initial pseudo-random dictionary using a Bernoulli$(1/2)$ model (left), result of MOB (center) and of K-PROX (right) for $p = 36$. Bottom row: initial dictionary using random samples from the dataset (left), MOB (center) and K-PROX (right) dictionaries. Here the pseudo-random initialization does not work; no atom is ever used, and so both algorithms stop at iteration 1. In the case of the samples-based initialization, both final models show some adaptation.

[7] P. Miettinen, T. Mielikinen, A. Gionis, G. Das, and H. Mannila, "The discrete basis problem," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 20, pp. 335–346, 09 2006.

[8] M. Koyutrk and A. Grama, "PROXIMUS: A framework for analyzing very high dimensional discrete-attributed datasets," in *KDD 2003*. IEEE Computer Society, 2003, pp. 147–156.

[9] F. Geerts, B. Goethals, and T. Mielikäinen, "Tiling databases," *Discovery Science*, vol. 3245, pp. 278–289, 2004.

[10] Z. Zhang, T. Li, C. Ding, and X. Zhang, "Binary matrix factorization with applications," in *Seventh IEEE International Conference on Data Mining (ICDM 2007)*, Oct 2007, pp. 391–400.

[11] E. Bartl, R. Belohlavek, P. Osicka, and H. ezankov, "Dimensionality reduction in boolean data: Comparison of four bmf methods," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 7627, pp. 118–133, 2015.

[12] M. Slawski, M. Hein, and P. Lutsik, "Matrix factorization with binary components," in *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, ser. NIPS'13. USA: Curran Associates Inc., 2013, pp. 3210–3218.

[13] M. Diop, A. Larue, S. Miron, and D. Brie, "A post-nonlinear mixture model approach to binary matrix factorization," vol. 2017-January. Institute of Electrical and Electronics Engineers Inc., 2017, pp. 321–325.

[14] S. Ravanbakhsh, B. Póczos, and R. Greiner, "Boolean matrix factorization and noisy completion via message passing," in *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ser. ICML'16. JMLR.org, 2016, pp. 945–954.

[15] D. L. Donoho, A. Maleki, and A. Montanari, "Message-passing algorithms for compressed sensing," *Proc. Natl. Acad. Sci. U.S.A.*, vol. 106, no. 45, pp. 18 914–18 919, Nov 2009, 0069[PII].

[16] S. Rangan, "Generalized approximate message passing for estimation with random linear mixing," in *2011 IEEE International Symposium on Information Theory Proceedings*, July 2011, pp. 2168–2172.

[17] B. Olshausen and D. Field, "Sparse coding with an overcomplete basis set: A strategy employed by V1?" *Vision Research*, vol. 37, pp. 3311–3325, 1997.

[18] R. Rubinstein, A. Bruckstein, and M. Elad, "Dictionaries for sparse representation modeling," *Proceedings of the IEEE*, vol. 98, no. 6, pp. 1045–1057, June 2010.

[19] J. Mairal, F. Bach, J. Ponce, and G. Sapiro, "Online dictionary learning for sparse coding," in *ICML 2009, Montreal, Canada*, 2009.

[20] P. Miettinen and J. Vreeken, "Model order selection for boolean matrix factorization," in *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '11. New York, NY, USA: ACM, 2011, pp. 51–59.

[21] S. Hess, K. Morik, and N. Piatkowski, "The PRIMPING routinetiling through proximal alternating linearized minimization," *Data Mining and Knowledge Discovery*, vol. 31, no. 4, pp. 1090–1131, 2017.

[22] C. Liu, L. Feng, R. Fujimaki, and Y. Muraoka, "Scalable model selec-

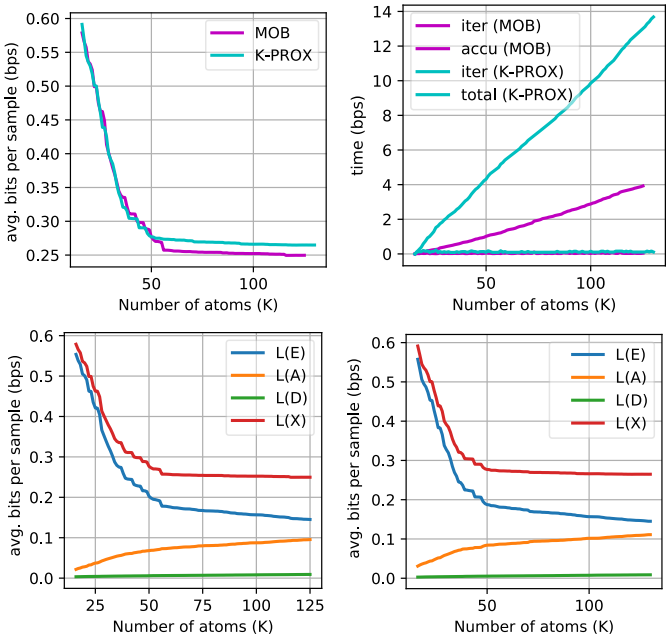

Figure 6: Convergence of Forward Selection on Einstein. Top to bottom, left to right: MDL cost function (in avg. bits per sample) for MOB and K-PROX; computational cost (in seconds) for both variants; break-down of the cost function in its three parts (E, A and D) for MOB; same for K-PROX. Both MOD and K-PROX produced similar models in the end. K-PROX, however, required more time to run (14s) than MOB (4s). Thanks to the warm-restart strategy, the cost per forward selection step is small (0.11s for MOB and 0.03s for K-PROX) and approximately constant despite the growing model size p. Finally, both break-downs show typical MDL curves: as p increases, the stochastic complexity ($L(E)$) decreases while the model cost $L(A) + L(D)$ increases.

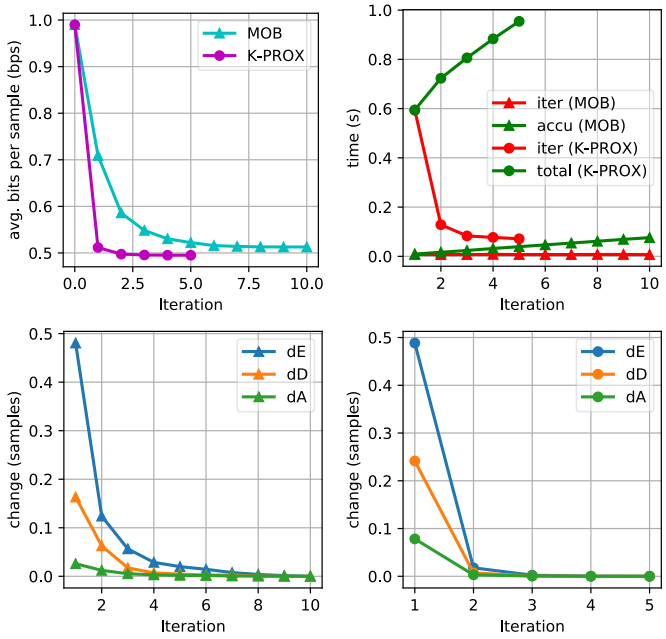

Figure 7: Convergence of MOB and K-PROX on Einstein. Top to bottom, left to right: MDL cost function; execution time; change in the arguments A,D and E for MOB; same for K-PROX. Both methods converge quickly, with K-PROX reaching its near-optimum in just one iteration. In terms of execution time, however, just one iteration of K-PROX required 0.6s (this is much larger than the 0.11s reported in Figure 6 since the model has to be learned from scratch), whereas all 10 MOB iterations take an accumulated time smaller than 0.1s. Note that the convergence is exact, not asymptotic: both algorithms stop when the change of all arguments is 0.

tion for large-scale factorial relational models," in *32nd International Conference on Machine Learning, ICML 2015*, vol. 2. International Machine Learning Society (IMLS), 2015, pp. 1227–1235.

[23] C. Lucchese, S. Orlando, and R. Perego, "Mining Top-K patterns from binary datasets in presence of noise," in *Proceedings of the 2010 SIAM International Conference on Data Mining*, 2010, pp. 165–176.

[24] J. Rissanen, "Universal coding, information,prediction, and estimation," *IEEE Trans. Inf. Theory*, vol. 30, no. 4, pp. 629–636, 1984.

[25] ——, *Stochastic complexity in statistical inquiry*. Singapore: World Scientific, 1992.

[26] A. Barron, J. Rissanen, and B. Yu, "The minimum description length principle in coding and modeling," *IEEE Trans. Inf. Theory*, vol. 44, no. 6, pp. 2743–2760, 1998.

[27] I. Ramirez and G. Sapiro, "An MDL framework for sparse coding and dictionary learning," *IEEE Trans. Signal Process.*, vol. 60, no. 6, pp. 2913–2927, June 2012.

[28] K. Engan, S. Aase, and J. Husoy, "Multi-frame compression: Theory and design," *Signal Process.*, vol. 80, no. 10, pp. 2121–2140, Oct. 2000.

[29] M. Aharon, M. Elad, and A. Bruckstein, "The K-SVD: An algorithm for designing of overcomplete dictionaries for sparse representations," *IEEE Trans. Signal Process.*, vol. 54, no. 11, pp. 4311–4322, Nov. 2006.

[30] S. Mallat and Z. Zhang, "Matching pursuit in a time-frequency dictionary," *IEEE Trans. Signal Process.*, vol. 41, no. 12, pp. 3397–3415, 1993.

[31] R. Tibshirani, "Regression shrinkage and selection via the LASSO," *Journal of the Royal Statistical Society: Series B*, vol. 58, no. 1, pp. 267–288, 1996.

[32] A. Beck and M. Teboulle, "A fast iterative shrinkage-thresholding algorithm for linear inverse problems," *SIAM Journal on Imaging Sciences*, vol. 2, no. 1, pp. 183–202, 2009.

[33] W. W. Hager, "Updating the inverse of a matrix," *SIAM Review*, vol. 31, no. 2, pp. 221–239, 1989.

[34] G. Davis, S. Mallat, and M. Avellaneda, "Greedy adaptive approximation," *Constr. Approx*, vol. 13, pp. 57–98, 1997.

[35] J. A. Tropp and A. C. Gilbert, "Signal recovery from random measurements via orthogonal matching pursuit," *IEEE Trans. Inf. Theor.*, vol. 53, no. 12, pp. 4655–4666, Dec. 2007.

[36] R. Agrawal, T. Imieliński, and A. Swami, "Mining association rules between sets of items in large databases," in *Proceedings of the 1993 ACM SIGMOD Intl. Conf. on Management of Data*. New York, NY, USA: ACM, 1993, pp. 207–216.

[37] G. Schwartz, "Estimating the dimension of a model," *Annals of Statistics*, vol. 6, no. 2, pp. 461–464, 1978.

[38] H. Akaike, "A new look at the statistical model identification," *IEEE Trans. Autom. Control*, vol. 19, pp. 716–723, 1974.

[39] T. Cover, "Enumerative source encoding," *IEEE Transactions on Information Theory*, vol. 19, no. 1, pp. 73–77, January 1973.

[40] Y. Lecun, "The MNIST database of handwritten digits," http://yann.lecun.com/exdb/mnist/.

[41] I. Ramírez, P. Sprechmann, and G. Sapiro, "Classification and clustering via dictionary learning with structured incoherence and shared features," in *CVPR*, June 2010.